Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 8(10): 25104–25130.
DOI: 10.3934/math.20231282
Received: 23 May 2023
Revised: 25 July 2023
Accepted: 04 August 2023
Published: 29 August 2023

Research article

Recursive reordering and elimination method for efficient computation of PageRank problems

Zhao-Li Shen[1,2,*], Yu-Tong Liu[1], Bruno Carpentieri[3], Chun Wen[4,*] and Jian-Jun Wang[1,*]

[1] College of Science, Sichuan Agricultural University, Ya'an, Sichuan 625000, China

[2] Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, 9700 AK Groningen, The Netherlands

[3] Faculty of Engineering, Free University of Bozen-Bolzano, 39100 Bolzano, Italy

[4] School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China

* **Correspondence:** szlxiaoyao@163.com, wchun17@163.com, wangjianjun02@163.com.

Abstract: The PageRank model is widely utilized for analyzing a variety of scientific issues beyond its original application in modeling web search engines. In recent years, considerable research effort has focused on developing high-performance iterative methods to solve this model, particularly when the dimension is exceedingly large. However, due to the ever-increasing extent and size of data networks in various applications, the computational requirements of the PageRank model continue to grow. This has led to the development of new techniques that aim to reduce the computational complexity required for the solution. In this paper, we present a recursive 5-type lumping algorithm combined with a two-stage elimination strategy that leverage characteristics about the nonzero structure of the underlying network and the nonzero values of the PageRank coefficient matrix. This method reduces the initial PageRank problem to the solution of a remarkably smaller and sparser linear system. As a result, it leads to significant cost reductions for computing PageRank solutions, particularly in scenarios involving large and/or multiple damping factors. Numerical experiments conducted on over 50 real-world networks demonstrate that the proposed methods can effectively exploit characteristics of PageRank problems for efficient computations.

Keywords: PageRank model; elimination; reordering; Krylov subspace methods; ILU factorizations
Mathematics Subject Classification: 65C40, 65F10

1. Introduction

Google proposed the PageRank model [1] to assess the relevance of web pages that match a user-defined search query. Since then, PageRank has been widely used in various fields that go beyond search engines, see for instances [2–10]. This widespread use is attributed to the simplicity of the model and the universality of its underlying principle. PageRank is often used to evaluate the centralities or importances of network nodes, which are measured by the stationary probability vector of a random process that visits nodes through their directed links. In detail, an adjacency matrix $G \in \mathbb{N}^{n\times n}$ stores link structure of the network nodes with quantity n, where $G(i, j) = 1$ when there is a directed edge from the j_{th} node to the i_{th} node. The transition probability matrix $P \in \mathbb{R}^{n\times n}$ of the random process is formed by

$$P(i,j) = \begin{cases} \frac{1}{\sum\limits_{k=1}^{n} G(k,j)}, & \text{when } G(i,j) = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{1.1}$$

To guarantee the existence of a unique stationary distribution and the convergence of the Power iteration of the random walk process [11,12], typically matrix P is transformed into a stochastic matrix A through

$$A = \alpha\widetilde{P} + (1-\alpha)ve^T. \tag{1.2}$$

In (1.2), $\widetilde{P} = P + vd^T$, where the binary vector $d \in \mathbb{N}^{n\times 1}$ has value 1 at the indexes with no out-link, $0 \le v \in \mathbb{R}^{n\times 1} \le 1$, $\|v\|_1 = 1$, e is a vector of all ones, $0 < \alpha < 1$ is the damping factor and assigns the probability that the traveler transfers through following directed links, while ve^T represents probabilities of transferring in other ways such as jumping to a random unlinked node. The PageRank model then can be regarded as computing the eigenvector x satisfying the equation

$$Ax = x, \quad \|x\|_1 = 1, \quad x > 0. \tag{1.3}$$

As only the ratio between the importances of nodes are needed, the relative importances can also be obtained by solving the following linear equations [13]

$$(I - \alpha P)x = v. \tag{1.4}$$

Since the PageRank model was proposed, the development of high-performance solvers for systems (1.3)–(1.4) has attracted much study attention. Classical iteration algorithms, e.g., the Power method, often prove effective when the damping factor equals to a moderate value (e.g., Google recommended to use $\alpha = 0.85$ for search engine ranking). On the other hand, when α increases to 1, often needed by some applications [14]), the speed that stationary iterative methods converge deteriorates sharply, necessitating the exploitation of more robust solvers. Extrapolation algorithms [15–18], adaptive methods [19], multigrid solvers [20,21], inner-outer strategies [22–26], Krylov subspace algorithms [27–30] and methods that combine stationary iterations and Krylov subspace algorithms [31–34] are used in the quest of robust solvers. It has been demonstrated that in some problems involving large damping factors, Krylov subspace methods can save over 50% CPU time cost when compared with classical iterative methods [35]. Nonetheless, due to the ever-increasing extent and size of data networks in applications, the computational demands of the PageRank model continue to grow, motivating the development of new techniques that can reduce the computational complexity required for the solution.

One relevant research direction classifies network nodes into family of special types, e.g., those with and without out-links, and reorders them accordingly so that only the PageRank values of one type are the solution of a linear system, while the PageRank values of the remaining nodes can be obtained efficiently using vector-vector and matrix-vector operations. If the linear system corresponding to one type of nodes (called the "kernel linear system") is significantly smaller than the original system, the costs associated with the solution of the PageRank model can be greatly decreased. Some of the relevant methods reorder the nodes into two types [13], three types [36], five types [37] and include recursive variants [38]. A different research direction exploits the similarities between rows in the adjacency matrix G to either devise elimination algorithms [39] that increases the sparsity of the PageRank problem (1.4), or find suitable permutations that reveal some off-diagonal low-rank blocks in the coefficient matrix $I - \alpha P$ [40, 41], leading to highly efficient compressed matrix representations of (1.4). The effectiveness of these strategies depends on the structural characteristics of the underlying network. However, they proved useful in simplifying realistic PageRank calculations in many contexts. A deeper understanding of the structural properties of existing networks arising from different backgrounds, the development of new strategies to further exploit these properties, and the design of fast solvers for the simplified problem are some issues that still require additional research.

In this paper, we present a recursive 5-type reordering approach and a two-stage elimination algorithm that combines information of both the nonzero structure of the underlying network and the numerical properties of the PageRank model to reduce the computational complexity. Meanwhile, the effect of the elimination strategy on changing the spectrum distribution is demonstrated to be controlled. The remaining content is structured as follows. Section 2 covers the most generally used methods for classifying graph nodes into different types. Section 3 presents a recursive 5-type reordering algorithm. Section 4 introduces a new two-stage elimination strategy which extends the work proposed in [39], and analyzes its impact on the eigenvalues distribution of the PageRank linear system. Section 5 presents real-world network simulations. Finally, Section 6 discusses some remarks arising from this study and perspective of future works. The MATLAB notation [42] is used throughout the paper.

2. Existed lumping algorithms for computing PageRank

2.1. *Lumping the graph nodes into two types (Lump-2)*

Current approaches to classifying nodes in the web adjacency graph begin with Lee, Golub, and Zenios' work [43], which takes advantage of the presence of dangling nodes, i.e., nodes with 0 out-degree. The method aggregates all dangling nodes into one node, and then computes the PageRank values by solving two smaller linear systems. Langville and Meyer refined this initial idea [13]. They classify the nodes of the web adjacency graph into dangling nodes (D) and nondangling nodes (ND) [13]; we refer to this classification scheme as Lump-2. They also reorder the nodes into two subvectors corresponding to dangling and nondangling nodes, and permute the linear system symmetrically accordingly. The new system is shown as:

$$\tilde{x}^T \left(1 - \alpha \tilde{P}\right) = \tilde{v}^T, \tag{2.1}$$

where

$$\tilde{P} = \Pi P \Pi^T = \begin{array}{c c} & \begin{array}{cc} ND & D \end{array} \\ \begin{array}{c} ND \\ D \end{array} & \begin{bmatrix} \tilde{P}_{11} & \tilde{P}_{12} \\ 0 & 0 \end{bmatrix} \end{array}, \tag{2.2, 2.3}$$

and Π is the permutation matrix[*], $\tilde{x}^T = x^T \Pi^T$, $\tilde{v} = v^T \Pi^T$. Note that,

$$\left(1 - \alpha\tilde{P}\right)^{-1} = \begin{bmatrix} \left(1 - \tilde{P}_{11}\right)^{-1} & \alpha\left(1 - \tilde{P}_{11}\right)^{-1} \tilde{P}_{12} \\ 0 & I \end{bmatrix}. \tag{2.4}$$

Thus, the permuted PageRank vector $\tilde{x}^T = \tilde{v}^T \left(1 - \alpha\tilde{P}\right)^{-1}$ can be formed as:

$$\tilde{x}^T = \left(\tilde{v}_1^T \left(1 - \tilde{P}_{11}\right)^{-1}, \alpha\tilde{v}_1^T \left(1 - \tilde{P}_{11}\right)^{-1} \tilde{P}_{12} + \tilde{v}_2^T\right). \tag{2.5}$$

After splitting $\tilde{x}^T$ into $[\tilde{x}_1, \tilde{x}_2]^T$, where $\tilde{x}_1^T$ and $\tilde{x}_2^T$ store the PageRank values of nondangling and dangling nodes, respectively, the solution procedure can be implemented as

$$\tilde{x}_1^T = \tilde{v}_1^T \left(1 - \alpha\tilde{P}_{11}\right)^{-1}, \tag{2.6}$$

$$\tilde{x}_2^T = \alpha\tilde{x}_1^T \tilde{P}_{12} + \tilde{v}_2^T. \tag{2.7}$$

Iterative algorithms, such as Krylov subspace methods, need only solve the smaller linear system (2.6) (which we refer to as the "kernel linear system") to compute the PageRank values of nondangling nodes, i.e., subvector $\tilde{x}_1^T$. Then, the PageRank values of dangling nodes, i.e., subvector $\tilde{x}_2^T$, can be obtained at the negligible cost of one matrix-vector multiplication and one vector-vector addition. Thus, if the number of nondangling-nodes is substantially smaller than the total dimension, the time needed to solve the PageRank problem can be drastically reduced. This method can provide a fivefold performance speedup over the Power method when more than 80% of the total nodes are dangling nodes.

Following this idea, a number of enhancements are suggested to further reduce the size of the kernel system. They can be classified in one of two ways: either categorizing the nodes into more classes, or applying the reordering recursively to a given class. For these two distinct research directions, we review the representative methods briefly before introducing a new reordering algorithm that combines the different strategies.

2.2. *Lumping the web matrix into three types (Lump-3)*

Lin et al. further classify the nodes in the web adjacency graph into three categories: strongly nondangling nodes (S), weakly nondangling nodes (W), and dangling nodes (D), with the new type W representing nodes that only point to dangling nodes [36]. We refer to Lin et al.'s method as Lump-3. By reordering the nodes as $[S, W, D]$ and permuting the problem symmetrically, the matrix $\tilde{P}$ in the permuted PageRank linear system (2.1) becomes

$$\tilde{P} = \Pi P \Pi^T = \begin{array}{c} S \\ W \\ D \end{array} \begin{bmatrix} \tilde{P}_{11} & \tilde{P}_{12} & \tilde{P}_{13} \\ 0 & 0 & \tilde{P}_{23} \\ 0 & 0 & 0 \end{bmatrix}, \tag{2.8}$$

[*] Π is obtained through reordering the rows and columns of an identity matrix according to the lumping algorithm.

while $\tilde{x}$ and $\tilde{v}$ are partitioned according to node type: $\tilde{x} = \left[\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T\right]^T, \tilde{v} = \left[\tilde{v}_1^T, \tilde{v}_2^T, \tilde{v}_3^T\right]^T$. Then, this permuted problem is expressed as

$$\left[\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T\right]\begin{bmatrix} I - \alpha\tilde{P}_{11} & -\alpha\tilde{P}_{12} & -\alpha\tilde{P}_{13} \\ 0 & I & -\alpha\tilde{P}_{23} \\ 0 & 0 & I \end{bmatrix} = \left[\tilde{v}_1^T, \tilde{v}_2^T, \tilde{v}_3^T\right], \tag{2.9}$$

and $\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T$ can be obtained using the formulas

$$\tilde{x}_1^T = \tilde{v}_1^T\left(1 - \alpha\tilde{P}_{11}\right)^{-1}; \tag{2.10}$$

$$\tilde{x}_2^T = \alpha\tilde{x}_1^T\tilde{P}_{12} + \tilde{v}_2^T; \tag{2.11}$$

$$\tilde{x}_3^T = \alpha\tilde{x}_1^T\tilde{P}_{13} + \alpha\tilde{x}_2^T\tilde{P}_{23} + \tilde{v}_3^T. \tag{2.12}$$

Iterative methods are only used to solve the linear system (2.10) for the subvector of PageRank values of S, which is not larger than the set ND in the Lump-2 method; therefore, it may be less expensive to solve.

2.3. The recursive lumping method for PageRank (Lump-R)

Langville and Meyer generalize the Lump-2 and Lump-3 algorithms to a recursive version that recursively applies Lump-2 reordering to the upper-left corner of the permuted transition matrix $\tilde{P}$ [38]. We refer to this method as Lump-R. The structure of the transition matrix after $k-1$ reordering is described below:

$$\tilde{P} = \Pi P \Pi^T = \begin{bmatrix} \tilde{P}_{11} & \tilde{P}_{12} & \tilde{P}_{13} & \cdots & \tilde{P}_{1k} \\ & 0 & \tilde{P}_{23} & \cdots & \tilde{P}_{2k} \\ & & 0 & \cdots & \tilde{P}_{3k} \\ & & & \ddots & \vdots \\ & & & & 0 \end{bmatrix}. \tag{2.13}$$

Meanwhile, $\tilde{x}$ and $\tilde{v}$ are permuted accordingly as follows: $\tilde{x} = \left[\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T, \cdots, \tilde{x}_k^T\right]^T, \tilde{v} = \left[\tilde{v}_1^T, \tilde{v}_2^T, \tilde{v}_3^T, \cdots, \tilde{v}_k^T\right]^T$. From (2.13), $\tilde{x}$ can be computed as follows:

$$\tilde{x}_1^T = \tilde{v}_1^T\left(1 - \alpha\tilde{P}_{11}\right)^{-1}. \tag{2.14}$$

$$\text{For} \quad i = 2, 3, \cdots k, \quad \tilde{x}_i^T = \alpha\sum_{j=1}^{i-1}\tilde{x}_j^T\tilde{P}_{ji} + \tilde{v}_i^T. \tag{2.15}$$

The Lump-2 and Lump-3 techniques are instances of this recursive reordering procedure for $k = 2$ and $k = 3$, respectively. This method is useful for uncovering the network structure in greater depth and it may reduce the dimension of the "kernel linear system" further as k increases, but the reordering phase requires more time [38]. As a result, it is recommended to terminate the recursive process when the upper-left submatrix $\tilde{P}_{11}$ contains few dangling nodes.

2.4. *Lumping the web matrix into five types (Lump-5)*

Similarly, the previously discussed classification for nodes with zero outlinks can be applied to nodes with zero inlinks, i.e., nodes that are not referenced by other nodes. These are referred to as unreferenced nodes (UR), in contrast to referenced nodes (R), which are referenced by other nodes. Taking R and UR categories into consideration, Yu et al. further classify the Lump-3 method's types into 5 categories: strong nondangling & referenced nodes ($S\&R$), strong nondangling & unreferenced nodes ($S\&UR$), dangling & referenced nodes ($D\&R$), dangling & unreferenced nodes ($D\&UR$), weakly nondangling nodes (W). This technique is known as Lump-5. Figure 1 depicts the relationship between these five categories.

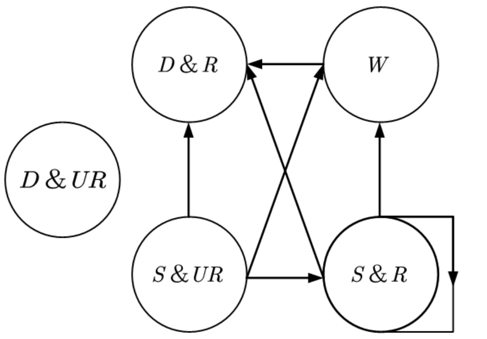


Figure 1. Relationship between the five distinct types of nodes.

By reordering the nodes as $[S\&R, S\&UR, W, D\&R, D\&UR]$ and permuting the problem symmetrically, the matrix $\tilde{P}$ in the permuted PageRank linear system (2.1) becomes

$$\tilde{P} = \Pi P \Pi^T = \begin{array}{r} S\&R \\ S\&UR \\ W \\ D\&R \\ D\&UR \end{array} \begin{bmatrix} \tilde{P}_{11} & 0 & \tilde{P}_{13} & \tilde{P}_{14} & 0 \\ \tilde{P}_{21} & 0 & \tilde{P}_{23} & \tilde{P}_{24} & 0 \\ 0 & 0 & 0 & \tilde{P}_{34} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{2.16}$$

and $\tilde{x}$ and $\tilde{v}$ are reordered accordingly as $\tilde{x} = \left[\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T, \tilde{x}_4^T, \tilde{x}_5^T\right]^T, \tilde{v} = \left[\tilde{v}_1^T, \tilde{v}_2^T, \tilde{v}_3^T, \tilde{v}_4^T, \tilde{v}_5^T\right]^T$. The permuted problem can be written

$$\left[\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T, \tilde{x}_4^T, \tilde{x}_5^T\right] \begin{bmatrix} I - \alpha\tilde{P}_{11} & 0 & -\alpha\tilde{P}_{13} & -\alpha\tilde{P}_{14} & 0 \\ -\alpha\tilde{P}_{21} & I & -\alpha\tilde{P}_{23} & -\alpha\tilde{P}_{24} & 0 \\ 0 & 0 & I & -\alpha\tilde{P}_{34} & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} = \left[\tilde{v}_1^T, \tilde{v}_2^T, \tilde{v}_3^T, \tilde{v}_4^T, \tilde{v}_5^T\right], \tag{2.17}$$

and $\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T, \tilde{x}_4^T, \tilde{x}_5^T$ can be calculated as follows:

$$\tilde{x}_1^T = \left(\tilde{v}_1^T + \alpha\tilde{v}_2^T\tilde{P}_{21}\right)\left(1 - \alpha\tilde{P}_{11}\right)^{-1}; \tag{2.18}$$

$$\tilde{x}_2^T = \tilde{v}_2^T; \tag{2.19}$$

$$\tilde{x}_3^T = \alpha\left(\tilde{x}_1^T\tilde{P}_{13} + \tilde{x}_2^T\tilde{P}_{23}\right) + \tilde{v}_3^T; \tag{2.20}$$

$$\tilde{x}_4^T = \alpha\left(\tilde{x}_1^T\tilde{P}_{14} + \tilde{x}_2^T\tilde{P}_{24} + \tilde{x}_3^T\tilde{P}_{34}\right) + \tilde{v}_4^T; \tag{2.21}$$

$$\tilde{x}_5^T = \tilde{v}_5^T. \tag{2.22}$$

This method detects more network structural features than the prior reordering algorithm and generates a smaller "kernel linear system" for test problems in [37]. However, its performance on really large networks has to be shown yet. Furthermore, it will be interesting to see if a recursive implementation of this method can further reduce the dimension of the "kernel linear system".

3. A recursive 5-types lumping algorithm for Google's PageRank

This section presents the Lump-R5 algorithm, which applies Lump-5 recursively to the current "kernel linear system" until the dimension cannot be reduced further or the algorithm is executed a predetermined number of times. Throughout each phase of the recursive procedure, the nodes can be classified into five distinct categories. Although this algorithm can be conveniently described as a recursive call to itself, we provide formulas for calculating the PageRank value of each node category. The purpose of these formulas is to shed light on efficient implementations of this algorithm, as opposed to a more intuitive but less efficient recursive implementation, and to demonstrate how the method can be adapted to solve PageRank linear systems with multiple damping factors.

In the permuted problem (2.1), after k iterations of Lump-5 reordering, there are $n = 4k + 1$ classes of nodes, and $\tilde{x}$ and $\tilde{v}$ are reordered as follows: $\tilde{x} = \left[\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T, \cdots, \tilde{x}_n^T\right]^T, \tilde{v} = \left[\tilde{v}_1^T, \tilde{v}_2^T, \tilde{v}_3^T, \cdots, \tilde{v}_n^T\right]^T$. Below are the formulas used to compute $\tilde{x}_1^T, \tilde{x}_2^T, \tilde{x}_3^T, \cdots, \tilde{x}_n^T$. Note that $\tilde{x}_1$ are the only type of nodes that require solving a linear system:

$$\tilde{x}_1^T = \left(\tilde{v}_1^T + \alpha\sum_{j=0}^{k-1}\tilde{x}_{4j+2}^T\tilde{P}_{4j+2,1}\right)\left(I - \alpha\tilde{P}_{11}\right)^{-1}. \tag{3.1}$$

To express the computation formulas for the PageRank of the remaining nodes, we introduce an integer $l\ (k-1 \geqslant l \geqslant 0)$:

$$\tilde{x}_i^T = \alpha\sum_{j=1}^{k-l-1}\tilde{x}_{4j+i}^T\tilde{P}_{4j+i,i} + \tilde{v}_i^T, \quad \text{when} \quad i = 4l+2 \leqslant n; \tag{3.2}$$

$$\tilde{x}_i^T = \alpha\sum_{j=1}^{i-1}\tilde{x}_j^T\tilde{P}_{ji} + \alpha\sum_{j=1}^{k-l-1}\tilde{x}_{4j+i-1}^T\tilde{P}_{4j+i-1,i} + \tilde{v}_i^T, \quad \text{when} \quad i = 4l+3 \leqslant n; \tag{3.3}$$

$$\tilde{x}_i^T = \alpha\sum_{j=1}^{i-1}\tilde{x}_j^T\tilde{P}_{ji} + \alpha\sum_{j=1}^{k-l-1}\tilde{x}_{4j+i-2}^T\tilde{P}_{4j+i-2,i} + \tilde{v}_i^T, \quad \text{when} \quad i = 4l+4 \leqslant n; \tag{3.4}$$

$$\tilde{x}_i^T = \alpha\sum_{j=1}^{k-l-1}\tilde{x}_{4j+i-3}^T\tilde{P}_{4j+i-3,i} + \tilde{v}_i^T, \quad \text{when} \quad i = 4l+5 \leqslant n. \tag{3.5}$$

According to (3.2), we first compute $\tilde{x}_{4l+2}^T$ in decreasing order of l. Then, $\tilde{x}_1^T$ and $\tilde{x}_{4l+5}^T$ can be computed using Eqs (3.1) and (3.5), where (3.1) is solved by iterative methods. Following that, $\tilde{x}_{4l+3}^T$ and $\tilde{x}_{4l+4}^T$ can be calculated using Eqs (3.3) and (3.4). The PageRank vectors of n nodes are concatenated and permuted at this stage, to arrive at the required solution x^T. The computation of $\tilde{x}^T_{4l+2}$, $\tilde{x}^T_{4l+3}$, $\tilde{x}^T_{4l+3}$ and $\tilde{x}^T_{4l+4}$ can be done in parallel. In our cases, where l is not excessively large, the cost of computing these quantities is negligible compared to the cost of solving for $\tilde{x}^T_1$. Therefore, we will not discuss implementation optimization in detail here.

In conclusion, the Lump-R5 algorithm is depicted in Algorithm 1 shown below.

Algorithm 1 Recursive Lump-5 reordering method (Lump-R5).

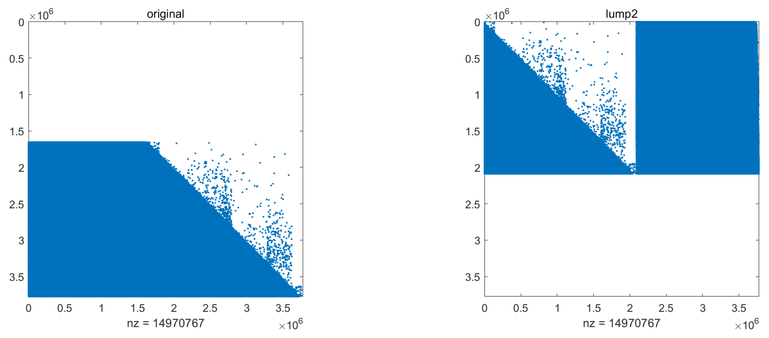
```
Input:  Damping factor $\alpha$, convergence torenlcen $tol$, number of recursive steps $k$.
 1: Permute the transition matrix $P$ in the original linear system, so that the same class of nodes are
    clustered.
 2: Let $l = k - 1$.
 3: while $l \neq 0$ do
 4:     Computing $\tilde{x}^T_{4l+2}$ through (3.2).
 5:     $l = l - 1$.
 6: end while
 7: Use iterative algorithms to solve $\tilde{x}^T_1$ according to (3.1).
 8: Computing directly by (3.1) - (3.5) for the PageRank value of the remaining nodes.
 9: Set $\tilde{x} = \left[\tilde{x}^T_1, \tilde{x}^T_2, \tilde{x}^T_3, \cdots, \tilde{x}^T_n\right]^T$,compute the PageRank vector $x^T = \tilde{x}^T\Pi$,and normalize $x^T =$
    $x^T / \left\|x^T\right\|_1$
10: return $x$;
```

Here, the computational costs of the lumping algorithms are compared approximately. Because Lump-2 and Lump-3 algorithms are instances of the Lump-R algorithm, and Lump-5 is an instance of Lump-R5, we only compare Lump-R5 and Lump-R. The cost of each recursive step for these two recursive methods comes mainly from classifying nodes into special categories. Note that the classification can be conducted using the binary information contained in the adjacency matrix G. Lump-R requires finding the dangling nodes, which necessitates the computation of the number of nonzero elements in each row of G. This costs $s_1 \in [nnz(G) - n, nnz(G)]$ additions of integers. For Lump-R5, it is necessary to identify the dangling nodes, unreferenced nodes, and weakly dangling nodes, and then the required 5 categories of nodes are determined by performing some straightforward intersection operations. Finding unreferenced nodes requires calculating the number of non-zero elements in each column of G, which costs $s_2 \in [nnz(G) - n, nnz(G)]$ additions of integers. Finally, finding the weakly dangling nodes requires computing the row sums of the submatrix $G(nondangling, nondangling)$ corresponding to the nondangling nodes, which costs at least $s_3 \in [nnz(G(nondangling, nondangling)) - length(nondangling), nnz(G(nondangling, nondangling))]$ operations. In general, (1) the computational cost of this implementation is less than that of finding the dangling nodes, and the greater the proportion of dangling nodes, the lower the computational cost; (2) a recursive step of Lump-R5 requires approximately two to three times as many integer additions as Lump-R. Note, additionally, that: (1) the computational cost of these algorithms depends on the number of nonzero elements and the number of dangling nodes of the left-top block at each recursive step, and thus is problem-dependent; (2) the storage format affects the computational cost in actual computer implementations, e.g., computing column sums on the MATLAB platform takes significantly less time than computing row sums; and (3) data extraction and redistribution operations also incur significant costs that are platform-dependent and difficult to account for.

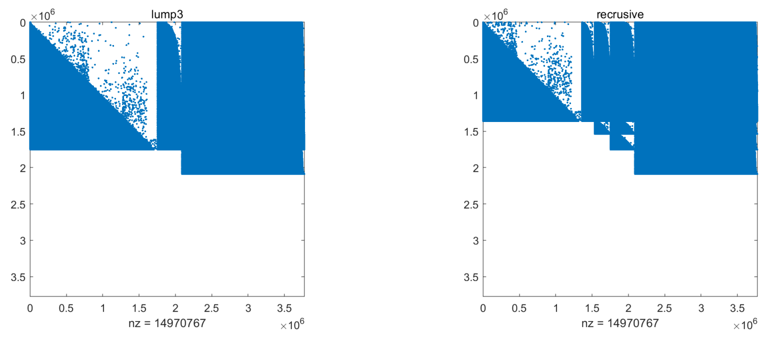
3.1. A numerical experiment

We provide an insight on the merits of the Lump-2, Lump-3, Lump-R, Lump-5 and the proposed Lump-R5 methods to reduce the dimension of the PageRank linear system on a 3774768×3774768 adjacency matrix named "patents" downloaded from https://sparse.tamu.edu/, using 4 recursive steps for Lump-R and Lump-R5.

Figures 2 and 3 of the original and reordered matrix patterns for this problem reveal the block structures produced by the five reordering methods. Nondangling nodes represent the largest proportion, while dangling nodes represent the second largest proportion (see Figure 3 (left)). There are many weakly nondangling nodes and unreferenced dangling nodes, suggesting that the dimension of the "kernel linear system" can be decreased even further when only the dangling nodes are considered. Indeed, it is not necessary to split dangling nodes into dangling & referenced nodes and dangling & unreferenced nodes when only the "kernel linear system" dimension is considered. Figure 3 (right) shows that the proportion of strong non-dangling & referenced nodes decreases with each recursive step, and the speed of such a decrease is diminishing with the recursive process proceeds. Similar observation of the decrease in the proportion of the non-dangling nodes can be found in Figure 2. It is interesting to confirm this phenomenon in general. Figures 2 and 3 indicate that the Lump-R5 method produces the smallest upper left block of the five reordering methods, which is a significant advantage from a computational viewpoint. In Section 5, we test the effectiveness of these methods on a large set of network matrices. However, some general observations can be made based on this analysis.



Structures of the original (left) and Lump-2 reordered (right) patent matrix.



Structures of the patent matrix after Lump-3 reordering (left) and recursive reordering (right).

Figure 2. Structures of the original and of three reordered "patent" matrices.

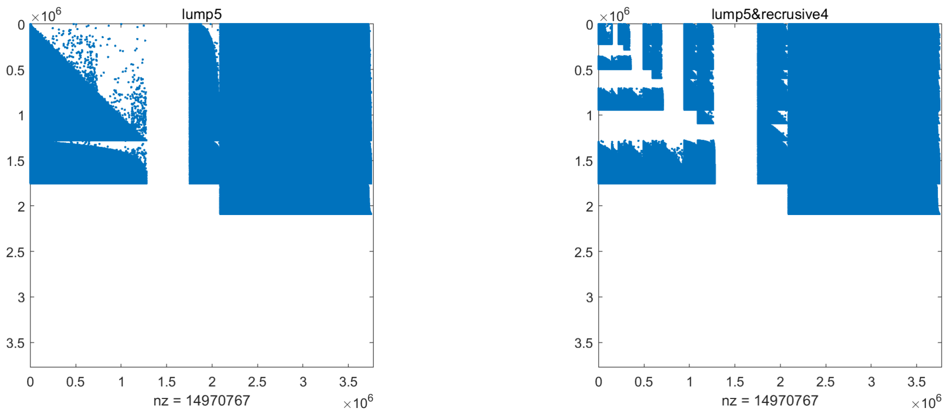


Figure 3. Structure of "patent" matrix after reordering using Lump-5 (left) and the recursive Lump-5 lumping methods (right).

- Classifying nodes into several types requires searching and permutation operations. Typically, these costs are small compared to the cost of solving the PageRank linear system. Therefore, if the reordering procedure significantly reduces the size of the "kernel linear system", its implementation is justified especially in the case multiple instances of the same network is solved.
- For the PageRank problem on a fixed network with multiple values of the damping factor, the Lump-2, Lump-3, and Lump-R methods produce "kernel linear systems" of the shifted form $(I - \alpha_i P)x = v$ with collinear right-hand sides (see Eqs (2.6), (2.10) and (2.14)), and these shifted systems can be simultaneously solved using Krylov subspace algorithms. This observation is important for an efficient implementation. Although, on the other hand, Lump-5 and Lump-R5 do not naturally maintain the collinearity of the right-hand sides, as shown by Eqs (2.18) and (3.1)), the sequence of shifted linear systems with s damping factors $\tilde{x}_1^T = \left(\tilde{v}_1^T + \alpha_i {\tilde{v}_2}^T P_{21}\right)\left(1 - \alpha_i \tilde{P}_{11}\right)^{-1}$ $1 \leq i \leq s$ can be transformed into a sequence of shifted linear systems with two separate right-hand sides $[\tilde{x}_{11}^T, \tilde{x}_{12}^T] = \left[\tilde{v}_1^T, \tilde{v}_2^T \tilde{P}_{21}\right]\left(1 - \alpha_i \tilde{P}_{11}\right)^{-1}$ so that block shifted Krylov methods can be applied to solve them simultaneously.
- If networks lack a notable special structure that could yield a significantly smaller "kernel linear system", it can be worth considering some combined strategy that splits the matrix into a linear combination of two matrices, one of which can be reordered effectively.
- All of the reordering methods discussed in this section produce the "kernel linear system" of the form
$$\tilde{x}_1^T \left(I - \alpha \tilde{P}_{11}\right) = b_1^T, \tag{3.6}$$
where $\tilde{P}_{11}$ is the upper-left block of the permuted transition matrix $\tilde{P}$, and b_1^T denotes the right hand side, e.g., $b_1^T = \tilde{v}_1^T$ for Lump-2, Lump-3 and Lump-R, and $b_1^T = \left(\tilde{v}_1^T + \alpha {\tilde{v}_2}^T \tilde{P}_{21}\right)^T$ for Lump-5. The following theorem shows that some similarities exist between the spectrum ranges and the matrix types of the original PageRank system and its "kernel linear system".

Theorem 3.1. *The coefficient matrix A of the original PageRank system (1.4) and the coefficient matrix $I - \alpha \tilde{P}_{11}$ of the "kernel linear system" (3.6) are both nonsingular M-matrices. The modules of the eigenvalues of these two matrices are in the interval* $[1 - \alpha, 1 + \alpha]$.

Proof. According to (1.1), $P \geq 0$ and $\rho(P) \leq \|P\|_1 = max_j(\sum_{i=1}^n P(i, j)) \leq 1$. Because $0 < \alpha < 1$, the PageRank coefficient matrix $A = I - \alpha P$ is a nonsingular M-matrix, and the modules of its eigenvalues are in the interval $[1-\alpha, 1+\alpha]$. As $\tilde{P}$ and $\tilde{A}$ are the symmetrically permuted versions of P and A, respectively, the above analysis and conclusion also apply to $\tilde{P}$ and $\tilde{A}$. Because $\tilde{P}_{11}$ in the "kernel linear system" (3.6) is a sub-matrix of $\tilde{P}$, we get $\tilde{P}_{11} > 0$ and $\rho(\tilde{P}_{11}) \leq \|\tilde{P}_{11}\|_1 = max_j(\sum_{i=1}^{n} \tilde{P}_{11}(i,j)) \leq max_j(\sum_{i=1}^{n} P(i,j)) = \|P\|_1 \leq 1$. Therefore, the coefficient matrix $I - \alpha\tilde{P}_{11}$ of the "kernel linear system" remains a nonsingular M-matrix, and modules of its eigenvalues continue to lie in the interval $[1-\alpha, 1+\alpha]$. □

Therefore, any efficient solver for the original PageRank system (1.4) can be used to solve the "kernel linear system" corresponding to any of the described reordering schemes.

- Some bioinformatics networks have special structures that have the potential to accelerate PageRank computations but have not yet been fully utilized. Figure 4 depicts the pattern structure of the protein-protein interaction networks of Human, fruit fly, mouse and yeast downloaded from the Molecular INTeraction Database https://mint.bio.uniroma2.it/index.php/download/. It is clear that such structure is preferable for constructing approximate inverses or preconditioners. In addition, the zoom-in section demonstrates the existence of nested-block structures and similar row sparsity-patterns, which can be exploited using methods such as [44] and [39, 40], respectively.

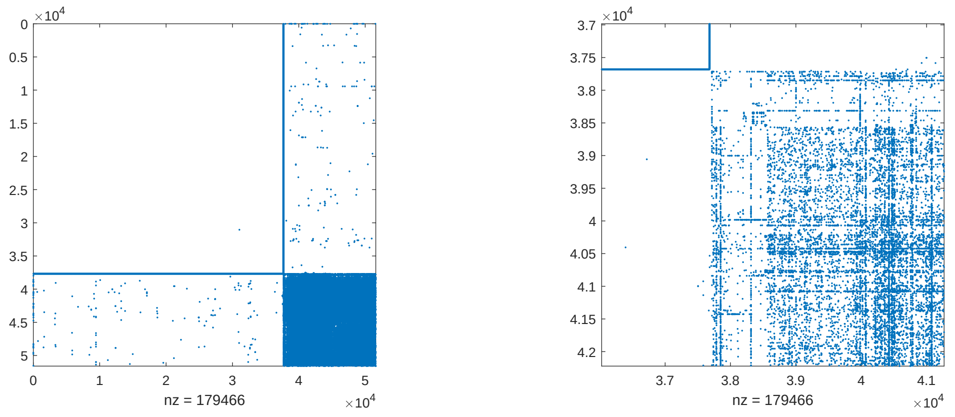


Figure 4. Pattern structure of Human (Homo Sapiens) protein-protein network (left) and zoom-in of the right-bottom part (right).

4. Two-stages elimination of PageRank linear systems

4.1. *The single stage elimination strategy (SSES)*

The two essential components of iterative solvers for PageRank linear systems are an accelerator, which is any algorithm that accelerates the convergence of the basic Power method, and a preconditioner, which modifies the initial system to make it more amenable to the iterative solution. Iterative methods are able to overcome the memory constraints of direct methods because they mainly require matrix-vector multiplications. When sparse storage format is used, the computational complexity of matrix-vector product operations is nearly proportional to the number of non-zero elements in the matrix, i.e., $O(nnz)$. In addition, this quantity also has a significant impact on the construction cost of matrix factorization methods such as incomplete lower-upper triangular decompositions (which are commonly used as preconditioners). As a result, reducing the density of the left-upper block is a critical issue for improving the performance of a PageRank solution. The elimination strategy [39] and the off-diagonal low-rank factorization [40] are two examples of techniques that achieve this goal by utilizing some properties of PageRank matrices. These properties are recalled below.

Property 4.1. *[39] In the PageRank transition matrix P, $P(j,i) = P(k,i)$ when these are nonzero values for $1 \leq i, j, k \leq n$.*

Property 4.2. *[39] For the network adjacency matrix G and the PageRank transition matrix P,*

$$P(i,S) = P(j,S) \quad when \quad G(i,S) = G(j,S),$$

where $1 \leq i, j \leq n$, $S \subseteq \{1, 2, \cdots, n\}$.

Due to these properties, it is possible to determine the degree of similarity, or the identity, of two rows of P by simply comparing their sparsity patterns. If a group of rows have very similar non-zero patterns, many of their non-zero values may be annihilated by row subtraction. Note, however, that this operation may generate fill-ins outside the common non-zero pattern between two rows; therefore, the row reduction should only be executed if more non-zero values are eliminated than those that are filled in. In general, the criterion to guide the decision if the row subtraction row j – row i should be performed or not is

$$comnnz(i, j) > rowsum(i) - comnnz(i, j), \tag{4.1}$$

where *rowsum* denotes the number of nonzero elements of a certain row of G and P, and $comnnz(i, j)$ denotes the quantity of non-zero values in the common sparsity pattern of rows i and j.

The elimination operation between two rows can be expressed as $E_{ij}P$, where matrix $E_{ij} = \{e_{kl}\}$ formed by subtracting the i_{th} row from the j_{th} row on the identity matrix of dimension n.

Note that, due to the possibility of introducing negative values in row j, $E_{ij}P$ may lose Properties 4.1–4.2, as the previous example demonstrates. Once a row has been eliminated, it is marked and skipped in the subsequent stages of the elimination algorithm. The main framework of this strategy is concluded as below:

(i) select sequentially row i from the unmarked rows to be the *reference* row, or generate complete elimination operator $E \leftarrow \prod E_{ij}$, which accounts for all row subtractions if all rows are marked;

(ii) traverse subsequent unmarked rows and reduce them if criterion (4.1) is satisfied, generating the corresponding operator matrix E_{ij};

(iii) mark row i and the eliminated rows, then return back to step (i).

The following two improvements are introduced in this framework to enhance the overall efficiency in the elimination strategy that we published in [39].

- Since eliminating very sparse rows may not be very efficient, it is suggested to initially permute rows in *rowsum*-increasing order and skip in the search those that have cumulative *rowsum* smaller or equal than θ times the total number of nonzeros in G. For instance, when θ is set to 25%, we only consider the rows with largest densities that contain at most 75% nonzero values of P.

- In the elimination, for each reference row, only at most ω (which is typically a small number, such as 100) unmarked rows are traversed.

In web page networks, the proportion of nodes with k in-links almost equals to $1/k^{\gamma}$, and the most recent estimate of γ is 2.1 [45]; the network structure also exhibits a nested block characteristic [44]. The above scheme can significantly reduce the time costs when dealing with web link networks based on these characteristics, while a little sacrifice in the effect of eliminating the nonzero values. A detailed algorithm description of this elimination process can be found in Algorithm 1 in [39], and we call this algorithm the **first-step elimination algorithm**.

In every i_{th} column of the elimination operator E, the indexes of negative values refer to the rows reduced using the i_{th} reference row. Meanwhile, each row of E has at most 1 negative element because it can be eliminated at most once. Therefore, $nnz(E) < 2n$ and $inv(E) = abs(E)$ [39]. The transformed system after elimination of (1.4) can be expressed as

$$A_E x = v_E, \text{ with } A_E = EA, \ v_E = Ev, \tag{4.2}$$

where A_E and P_E are expected to be sparser than A and P, respectively.

Note that, the eliminated matrix A_E is expected to be different with A in terms of eigenvalues, and it may be less favorable for iterative solvers. Fortunately, it has been demonstrated theoretically and confirmed experimentally that the perturbation is bounded by a moderate amount [39].

4.2. *Two-stage elimination strategy*

Although the reduced matrix EP may no longer satisfy Properties 4.1 in general, the following result provides some hints for continuing the elimination.

Theorem 4.1. *In the eliminated PageRank transition matrix EP, $P(j,i) = \pm P(k,i)$ when these are nonzero values for $1 \le i, j, k \le n$.*

Proof. Because of Property 1, elimination modifies a non-zero element $P(i, j)$ of P to 0 when the i_{th} row of P is subtracted from a reference k_{th} row that has a non-zero value $P(k, j)$, otherwise $P(i, j)$ remains unchanged. Meanwhile a zero element $P(i, j) = 0$ in the same j_{th} column changes to $-P(k, j)$, and $P(k, j) = P(i, j)$ due to Property 1. In conclusion, the non-zero values in the j_{th} column are either equal or opposite to one another. □

Example 4.1. Given a PageRank transition matrix P, the eliminated matrix EP generated after the first-step elimination algorithm will be

$$P = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{2} \end{pmatrix} \longrightarrow EP = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{3} & \frac{1}{2} \\ 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{3} & 0 \end{pmatrix}.$$

The non-zero elements in each column of EP have either same values or opposite values. It is evident that matrix EP can be eliminated further via basic row operations, e.g., subtracting the 2_{nd} row from the 3_{rd} row decreases the number of non-zero values by 1 in row 3, and adding the 4_{th} row to the 1_{st} row reduces the non-zero elements of row 1 by 1. □

In order to continue elimination on EP, we need to accurately determine whether a row addition/subtraction row $j \pm$ row i will decrease or increase the density. Consequently, criterion 4.1 must be updated to take into account the new structure of EP provided by Theorem 4.1. Specifically, fixing a reference row i, we eliminate row j if one of the two operations row $j :=$ row $j +$ row i or row $j :=$ row $j -$ row i decrease the density in row j. Both operations have the potential to achieve the goal. For example, if row $j = (a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ and row $i = (a_1, a_2, a_3, a_4, -a_5, -a_6, -a_7)$ with $a_i > 0$ $(i = 1, 2, \cdots, 7)$, both row $j :=$ row $j +$ row i and row $j :=$ row $j -$ row i reduce the number of non-zeros in row j, but the latter should be preferred as it annihilates more entries. If we denote as $nnz(j)_+$ and $nnz(j)_-$ the numbers of non-zero elements of row $j +$ row i and row $j -$ row i, respectively, we decide to reduce row j when $min(nnz(j)_+, nnz(j)_-) < nnz(j)$, and we record the type of operation (either addition or subtraction). We outline the second-step elimination algorithm as Algorithm 2. The whole two-stage elimination strategy sequentially implements the first-step elimination algorithm and then the second-step elimination algorithm.

Note that, the sequence of row-operations performed during elimination is stored in the variables m, $group_i$ and $operation_i$, $i = m + 1, m + 2, \ldots, n$ at lines 26 – 29 of Algorithm 2. These steps are straightforward to implement using a high-level programming language such as MATLAB. For the sake of clarity, we will briefly describe the implementation of line 27 in the example that follows.

Example 4.2. Suppose G is of dimension 5, $m = 1$, $group_2 = \{2, 3\}$, $group_2 = \emptyset$, $group_4 = \{4, 5\}$, $operation_2 = \{1, -1\}$, $operation_3 = \{1\}$, $operation_4 = \{1, 1\}$, $operation_5 = \{1\}$. Then the line 27 can be implemented by:
(1) row(i).index=i:i+length($group_i$)-1, col(i).index=i*ones(1,length($group_i$)), $i = 2, 3, 4, 5$;
(2) whole_row=[row(2:5).index], whole_col=[col(2:5).index];
(3) value=[$operation_2, \cdots, operation_5$];
(4) EC=sparse(whole_row,whole_col,value,4,4);
(5) E=blkdiag(speye(5-4),EC); □

The main difference between the second-step elimination algorithm (Algorithm 2) and the single-stage elimination algorithm [39] from the standpoint of computational cost is that: for each candidate reference rows i, the former algorithm requires to assess the effect on another row j of two elimination operations, namely row $j :=$ row $j +$ row i and row $j :=$ row $j -$ row i, while the latter of only one operation, row $j :=$ row $j -$ row i. According to the analysis presented in [39], the time required to compute the elimination effect varies by problem and parameter. In general, it is not possible to predict which algorithm will be the most cost-efficient. For the analysis of the computational cost of a single-stage elimination algorithm, one can refer to [39], as it is also highly applicable to the second-stage elimination algorithm.

Next, we examine the impact of the two-stage elimination algorithm on the eigenvalues distribution of the Google matrix A, which may determine to a large extent the convergence speed of iterative solvers applied to (1.4). We denote by E_1 and E_2 the 1-step and 2-step elimination operators, respectively. We will begin by introducing the following theorems.

Theorem 4.2. *Let $A_{E_1} = E_1 \cdot A$ and $A_{E_2} = E_2 \cdot A_{E_1} = E_2 \cdot E_1 \cdot A$ be the PageRank coefficient matrices after the first-step elimination and then the second-step elimination (Algorithm 2), respectively. We establish the following bounds:*

$$\|E_2E_1\|_1 = \|(E_2E_1)^{-1}\|_1 \leq (1+2\cdot\omega), \quad \|E_2E_1\|_\infty = \|(E_2E_1)^{-1}\|_\infty \leq 3, \quad \|A_{E_2}\|_1 \leq (1+\alpha)(1+2\cdot\omega). \tag{4.3}$$

Algorithm 2 Second-step elimination process for PageRank models.

```
Input:  $EG$, $\theta$, $\omega$
 1: Compute the numbers of nonzero entries in rows of $EG$ by $rowsum = sum(EG, 2)$.
 2: Reorder $rowsum$ to a value increasing order, reorder rows of $EG$ accordingly.
 3: Compute the smallest $s$ satisfying $rowsum(1 : s) \geq \theta nnz(EG)$.
 4: Set $mark = [0, 0, \cdots, 0] \in \mathbb{R}^{1\times n}$
 5: for $i = s + 1 \ : \ n$ do
 6:     $group_i = \emptyset$, $operation_i = \{1\}$;
 7:     if $mark(i) = 0$ then
 8:         $group_i = \{i\}$;
 9:         for $j = (i + 1) \ : \ min(i + \omega, n)$ do
10:             if $mark(j) = 0$ then
11:                 Compute $nnz(j)_+$ and $nnz(j)_-$.
12:                 Compute $[M, ind] = min(nnz(j)_+, nnz(j)_-)$
13:                 if $M < nnz(j)$ then
14:                     $group_i = \{group_i, j\}$
15:                     if $ind == 1$ then
16:                         $operation_i = \{operation_i, 1\}$
17:                     else
18:                         $operation_i = \{operation_i, -1\}$
19:                     end if
20:                     Set $mark(j) = 1$.
21:                 end if
22:             end if
23:         end for
24:         Set $mark(i) = 1$.
25:     end if
26: end for
27: Construct the permutation $perm = \{1 : m, group_{m+1}, group_{m+2}, \cdots, group_n\}$
28: Generate the elimination operator $E$ by modifying the $m + 1 \ : \ n$ columns of identity matrix
    according to $operation_i$, $i = m + 1, m + 2, \ldots, n$.
29: Permute $E$ using the inverse perm of $perm$.
30: Reorder the rows of $EG$ and $E$ back to the original order.
31: return $E$;
```

Proof. In the 1-step elimination procedure, we carry out the operation row j := row j – row i only if it decreases the density of row j. In both elimination steps, this operation can be carried out at most once on row j. In the 2-step elimination algorithm, the operation row j := row j + row i sets row j back to its state before 1-step elimination, so it will actually increase the density of row j. On the other hand, the operation row j := row j – row i in the 2-step elimination will also increase the density of row j because the zero elements reduced by the 1-step elimination will change to negative values, the elements with negative values will be modified to their doubleness, and the remaining elements will remain unchanged. As a result, neither of these two operations will be implemented in the 2-step elimination. In summary, if the row j is reduced using the reference row i in the first stage of elimination, it will no longer be reduced using row i in the 2-step elimination process.

Therefore matrix E_2E_1 has the following form: the diagonal elements are 1, the off-diagonal elements are either 1 or -1, the number of nonzero elements in each column is less than $1+2\omega$ because a reference row may reduce at most ω rows in each elimination process. Thus $\|E_2E_1\|_1 \leq 1 + 2\omega$, and $\|A_{E_2}\|_1 \leq \|A\|_1\|E_2E_1\|_1 \leq (1+\alpha)(1+2\omega)$.

Besides, as a row may be reduced both in the 1-step and 2-step elimination, the number of non-zero elements in each row of E_2E_1 is at most 3 including the diagonal element. We conclude that $\|E_2E_1\|_\infty \leq 3$.

By reverting the operations performed during the elimination, namely adding or subtracting each reference row from its reduced rows, the final eliminated system can be transformed back to the original system. As a result, changing the values of the off-diagonal elements (1 and -1) in E_2E_1 to their opposite number yields $(E_2E_1)^{-1}$. Thus $\|(E_2E_1)^{-1}\|_1 = \|E_2E_1\|_1 \leq 2\cdot\omega + 1$, $\|(E_2E_1)^{-1}\|_\infty = \|E_2E_1\|_\infty \leq 3$. □

Lemma 4.1. *[46] If matrix $C \in \mathbb{R}^{n\times n}$ is strictly diagonally dominant by columns, denoted by*

$$\delta = \min_k(|c_{kk}| - \sum_{j\neq k}|c_{jk}|),$$

there must be $\|C^{-1}\|_1 \leq 1/\delta$.

Theorem 4.3. *Let λ_{E_2} be an eigenvalue of A_{E_2}, then*

$$\frac{1-\alpha}{2\cdot\omega+1} \leq |\lambda_{E_2}| \leq (1+\alpha)(2\cdot\omega+1).$$

Proof. According to Theorem 4.2, $|\lambda_{E_2}| \leq \rho(A_{E_2}) \leq \|A_{E_2}\|_1 \leq (1+\alpha)(1+2\cdot\omega)$. As A is diagonally dominant by columns, therefore $\|A^{-1}\|_1 \leq \frac{1}{\min_k(|A_{kk}|-\sum_{j\neq k}|A_{jk}|)} \leq \frac{1}{1-\alpha}$, according to Lemma 4.1. Then we get $\|A_{E2}^{-1}\|_1 \leq \|(E_2E_1)^{-1}\|_1\|A^{-1}\|_1 \leq \frac{1+2\cdot\omega}{1-\alpha}$ and, consequently, $|\lambda_{E_2}| \geq \frac{1-\alpha}{1+2\cdot\omega}$. □

For the case where matrix $A = I - \alpha P$ is symmetric, we derive stricter bounds for the eigenvalues of A_E through the following lemma and theorem, where $\rho(\cdot)$ denotes the spectral radius.

Lemma 4.2. *[47] If matrix A is normal,*

$$\rho(XA) \leq \rho(A)\|X\| \quad \text{and} \quad \rho(AY) \leq \rho(A)\|Y\|,$$

for any matrices X and Y.

Proof. Because $\|\cdot\|$ is an induced matrix norm, we can write $\rho(XA) \leq \|XA\| \leq \|A\|\|X\|$ and $\rho(AY) \leq \|AY\| \leq \|A\|\|Y\|$. As A is normal, $\|A\| = \rho(A)$. Therefore, the inequalities are demonstrated. □

Theorem 4.4. *If the PageRank coefficient matrix $A = I - \alpha P$ is symmetric, and λ_{E_2} is an eigenvalue of A_{E_2}, then*

$$\frac{1-\alpha}{\sqrt{3(1+2\omega)}} \leq |\lambda_{E_2}| \leq \sqrt{3(1+2\omega)}(1+\alpha).$$

Proof. It is clear that symmetric matrices A and A^{-1} are normal. According to Lemma 4.2, we get

$$|\lambda_{E_2}| \le \rho(A_{E_2}) \le \|E_2E_1\|\rho(A) \le \sqrt{\|E_2E_1\|_1\|E_2E_1\|_\infty} \le \sqrt{3(1+2\cdot\omega)}(1+\alpha),$$

$$\rho(A_{E_2}^{-1}) = \rho(A^{-1}(E_2E_1)^{-1}) \le \rho(A^{-1})\|(E_2E_1)^{-1}\| \le \sqrt{\|(E_2E_1)^{-1}\|_1\|(E_2E_1)^{-1}\|_\infty}\rho(A^{-1}) \le \frac{\sqrt{3(1+2\omega)}}{1-\alpha}.$$

Consequently, $|\lambda_{E_2}| \ge \frac{1}{\rho(A_{E_2}^{-1})} \ge \frac{1-\alpha}{\sqrt{3(1+2\omega)}}$. □

Note that all of the above analyses provide bounds based on the worst-case scenarios. Our numerical experiments (even with non-symmetric A) show that the bounds on the spectrum radius of A_{E_2} stated in the preceding theorem may be loose in practice. Indeed, the spectral distribution of A_{E_2} can be expected to be only marginally less favourable than those of A for the iterative solution. In addition, efficient preconditioners (such as incomplete LU factorizations) can be used to effectively enhance the spectral distribution at a moderate cost, while still benefiting from the decrease in density caused by elimination, as shown in [39].

In order to reduce the computational complexity of the PageRank problem, we propose integrating elimination strategies with lumping reordering techniques. The combination should be carried out as follows:

(1) first, the PageRank linear system is reordered to produce a more compact "kernel linear system" $(I-\alpha\tilde{P}_{11})\tilde{x}_1 = b_1$ where the matrix $\tilde{P}_{11}$ and the coefficient matrix $(I-\alpha\tilde{P}_{11})$ inherit Properties 4.1-4.2;

(2) second, the elimination technique is applied to the "kernel linear system" to generate a reduced (i.e., more sparse) system $E(I-\alpha\tilde{P}_{11})\tilde{x}_1 = Eb_1$;

(3) then, the eliminated "kernel linear system" is solved for $\tilde{x}_1$ iteratively;

(4) finally, using $\tilde{x}_1$, the PageRank values of the remaining nodes excluding $\tilde{x}_1$ are computed by a few vector-vector and matrix-vector operations.

Note that the theoretical results in this section related with the values of matrix norms and the modules of eigenvalues also hold when replacing the PageRank linear system (1.4) with the "kernel linear system" (3.6), and because $I-\alpha P$ and $I-\alpha\tilde{P}_{11}$ are both strictly diagonally dominant by columns, P and $\tilde{P}_{11}$ are both non-negative with 1-norm values no larger than 1.

5. Numerical experiments

We assess and compare the effectiveness of the previously discussed reordering and elimination strategies for the solution of a large number of PageRank linear systems originating from various fields. All of the tests are implemented using MATLAB R2022a on a Windows 10 computer with an AMD Ryzen7 4800u CPU and 16 GB RAM. The selected networks represented by binary matrices are downloaded from the University of Florida matrix repository [48] and the Web Algorithmics Laboratory [49–51]. We report on their characteristics in Table 1, where the matrix problems are classified according to their group name in the matrix repositiory, *num* represents the number of matrices tested in the group, n is the average dimension, nnz denotes the average number of non-zero entries per matrix and $den = nnz/n^2$ denotes the average density. In all of our experiments, the personalization vector v of the PageRank problem and the initial guess x_0 are $v = [1, 1, \cdots, 1]^T/n$ and $x_0 = v$, respectively.

Table 1. Characteristics of the binary matrices of each group.

Name	num	n	nnz	den
SNAP	25	2,502,735	8,539,049	1.36×10^{-6}
LAW	9	10,416,785	228,858,878	2.11×10^{-6}
Pajek	10	386,009	1,572,415	1.06×10^{-5}
cit	3	1,279,028	5,764,444	3.52×10^{-6}
Gleich	6	3,666,739	34,738,258	2.58×10^{-6}
Kamvar	2	482,675	4,947,937	2.12×10^{-5}

Performances of the lumping algorithms

We test the Lump-2, Lump-3, Lump-5 and its recursive version Lump-R5 algorithms. We vary the number of recursive steps for the recursive orderings from 1 to 7, and we record the result of 7 recursive iterations reporting two important metrics for each run: D_{ratio} represents the ratio between the dimension of the final "kernel linear system" corresponding to the upper left block and the dimension of the original adjacency matrix; CPU is the elapsed CPU time cost (in seconds) required to execute the lumping algorithm on our computer. In Table 2, we present the mean value of these two metrics for each group of matrices.

Table 2. Comparison results between various reordering techniques for reducing the PageRank problem's dimension.

Methods	Lump-2		Lump-3		Lump-R		Lump-5		Lump-R5	
Groups	D_{ratio}	CPU	D_{ratio}	CPU	D_{ratio}	CPU	D_{ratio}	CPU	D_{ratio}	CPU
SNAP	66.4%	0.03	63.9%	0.13	63.5%	0.63	56.5%	0.17	55.7%	1.42
LAW	81.6%	0.27	79.7%	3.24	79.3%	10.56	78.0%	3.42	77.3%	26.70
Pajek	74.78%	0.04	67.01%	0.10	54.17%	0.32	57.76%	0.15	38.58%	0.30
cit	79.56%	0.07	72.44%	0.21	57.13%	0.64	57.86%	0.28	31.92%	0.59
Gleich	91.42%	0.34	90.74%	1.40	90.56%	7.00	67.99%	1.89	66.41%	15.29
Kamvar	91.42%	0.02	89.48%	0.06	89.32%	0.36	89.11%	0.11	88.78%	0.92

Table 2 shows that Lump-R>Lump-3>Lump-2 in terms of their ability to reduce the problem dimension. It should be noted that Lump-2, Lump-3 and Lump-R can be viewed as instances of Lump-R using 1, 2 and 7 recursive steps, respectively. Therefore, these results are consistent with theory and our expectations. The "kernel system" produced by Lump-2 is significantly smaller than the original one for SNAP, LAW, Pajek and cit groups, while Lump-3 never significantly reduces the problem dimension further than Lump-2. In these categories of networks, the proportion of dangling nodes is important, whereas the proportion of weakly dangling nodes is modest. Lump-5 outperforms Lump-3 at a modest CPU time increase, and Lump-R5 outperforms Lump-5. Lump-5 is clearly more effective than Lump-3 at reducing problem size for SNAP, Pajek, cit and Gleich groups, whereas Lump-R outperforms Lump-3 for Pajek and cit groups only. Dangling nodes in these tests have a weak recursive structure, and the proportion of unreferenced nodes is not small (although it is smaller than that of dangling nodes). Therefore, excluding the unreferenced nodes provides greater advantages than recursively excluding the dangling nodes. Lump-R5 significantly reduces the problem size further than Lump-5 for groups Pajek and cit. In comparison to the little benefit of only recursively removing dangling nodes, also recursively excluding the unreferenced nodes brings some improvements on reducing the problem dimension. As a result, we can say that the unreferenced nodes in these networks have some recursive structure. Another interesting observation is that, for the recursive-type methods Lump-R and Lump-R5, the recursive process significantly increases the time cost when it has little effect on reducing the problem dimension, whereas this increase is much gentler when it has a significant effect on reducing the problem dimension. This suggests the strategy of detecting the dimension of the upper-left linear system at each step of the recursive process; if it is not significantly reduced relative to the input matrix at this step, the permutation will not be implemented, and the recursive process should terminate.

We conclude, based on the results of this experiment, that the recursive 5-type reordering method outperforms the other lumping algorithms for our PageRank problems.

Performances of elimination algorithms

This section tests the combined effect of the proposed two-stage elimination strategy with the 5-type lumping reordering in reducing the number of non-zero elements of the PageRank linear system. We assess and compare the single-stage elimination strategy (SSES) from [39], the two-stage elimination strategy (TSES) proposed by this paper, and both elimination methods executed on the "kernel linear system" of the five-type reordering technique (called 5-re-SSES and 5-re-TSES, respectively). The latter approach is implemented in two distinct stages: first, the transition matrix P is permuted using 5-type lumping, and then elimination is applied to the upper-left block. The parameters in the elimination algorithms are set as $\theta = 0.3$ and $\omega = 100$ as suggested in [39].

The results are presented in Table 3, where NNZ_r represents the ratio of the amount of non-zero coefficients in the eliminated "kernel linear system" to that of the original PageRank system, and CPU represents the total elapsed CPU time cost (in seconds) required by lumping plus elimination on our computer.

Table 3. Comparison results between different methods for reducing the density of the PageRank linear system.

Methods	SSES		TSES		5-re-SSES		5-re-TSES	
Problems	NNZ_r	CPU	NNZ_r	CPU	NNZ_r	CPU	NNZ_r	CPU
cnr-2000	52.9%	0.77	48.8%	2.58	46.6%	0.74	42.6%	1.98
eu-2005	53.5%	4.32	43.5%	9.27	50.8%	3.76	41.6%	8.00
in-2004	42.0%	3.43	36.7%	11.34	37.2%	3.43	32.4%	9.40
indochina-2004	33.3%	59.90	23.8%	96.32	30.9%	73.01	21.5%	102.75
web-BerkStan	49.7%	1.78	46.1%	5.60	44.6%	2.11	41.9%	5.71
web-Stanford	67.5%	1.14	63.9%	2.89	63.0%	1.29	59.9%	2.99
web-NotreDame	66.6%	1.46	63.5%	3.80	40.5%	0.56	38.6%	1.41
Stanford	78.5%	14.11	74.8%	22.78	73.5%	13.50	70.2%	21.56
Stanford-Berkeley	55.7%	28.20	46.4%	40.40	50.9%	27.10	42.7%	38.29
patents	99.9%	21.09	99.9%	47.25	34.1%	8.19	34.1%	16.20
EVA	100.0%	0.07	99.8%	0.12	1.6%	0.006	1.6%	0.007
uk-2007-100000	45.9%	0.36	39.1%	0.80	44.9%	0.51	38.0%	0.92

Table 3 shows that the TSES strategy presented in this work is more effective than SSES at reducing the linear system density by 4.58% to 28.53%. The reason for this is that TSES can make further use of the negative values in the matrix, and continue elimination. Similarly, 5-re-TSES outperforms 5-re-SSES. Meanwhile, 5-re-SSES and 5-re-TSES reduce problem density better than SSES and TSES, respectively. The reason for this is that the top left corner block produced by the Lump-5 method contains fewer non-zero elements and is generally denser than the original matrix. Among the tested methods, the 5-re-TSES outperforms the other methods at reducing problem density, as expected. The resulting "kernel linear system" has only 1.6% to 70.6% of the initial number of non-zeros in the original system. For 10 of the 12 studied problems, 5-re-TSES decreases the density by more than 50%, while TSES does this for 7 of the 12 problems, and SSES only for 4 of them. We can conclude from our results that the networks analysed exhibit a high degree of repetitive row sparsity patterns and the two-stage elimination strategy is very effective to exploit this structural property, especially when used in combination with the 5-type reordering.

In terms of computational efficiency, 5-re-SSES and 5-re-TSES have lower CPU time costs than their counterparts SSES and TSES for two-thirds of the problems. For some adjacency matrices with a modest proportion of $S\&R$ nodes, the left-top block is small, and the elimination algorithm is fast since it traverses fewer rows. Furthermore, 5-re-SSES and 5-re-TSES clearly outperform SSES and TSES on the matrices "patents" and "EVA", because repetitive row sparsity patterns are not important in these matrices, but the proportion of $S\&R$ nodes is small, and as a result the problem density is decreased primarily with the help of the 5-type reordering approach rather than elimination procedures.

We conclude that the 5-re-SSES method is effective at reducing the density of the PageRank problems studied, and it can be efficiently implemented. It should be noted that the time cost of this pre-processing process, which includes the reordering and elimination stages, may be amortized during the solution, particularly when dealing with sequences of PageRank problems with multiple damping factors or multiple personalization vectors since it is performed only once.

This section examines the performance of SSES, TSES, 5-re-SSES, 5-re-TSES and Lump-5 methods on accelerating the convergence of Krylov subspace methods for solving the initial PageRank problem. We solve the linear system corresponding to the left-top block with or without elimination strategies using the ILU-preconditioned GMRES solver [52]. The ILU factorization and iterative solution are implemented by the ILUPACK package [53]. Note that ILPUACK can be only used in Linux-based operating systems. Each iteration begins with the zero vector $x_0 = 0$, the maximum dimension of the Krylov subspace is $m = 10$, and the iterations are terminated when the approximate solution x_i satisfies $\frac{\|(x_i - \alpha P x_i) - v\|_2}{\|v\|_2} < 10^{-8}$ or the number of restarts reaches 1000. The memory costs are quantified as

$$mem = \begin{cases} \frac{nnz(LU_A)+nnz(A)}{nnz(A)} & \text{Lump-5,} \\ \frac{nnz(LU_{A_E})+nnz(A_E)}{nnz(A)} & \text{the eliminated system by SSES,} \\ \frac{nnz(LU_{A_{TE}})+nnz(A_{TE})-nnz(A_T)+nnz(A)}{nnz(A)} & \text{the eliminated system by 5-re-SSES,} \\ \frac{nnz(LU_{A_{TE_2}})+nnz(A_{TE_2})-nnz(A_T)+nnz(A)}{nnz(A)} & \text{the eliminated system by 5-re-TSES,} \end{cases} \tag{5.1}$$

where symbol $nnz(LU_A)$ represents the number of the non-zeros in all the multilevel ILU factors of A, while A_T denotes the coefficient matrix of the upper-left "kernel system" after 5-type lumping, A_{TE} and A_{TE_2} denote the eliminated matrix of A_T by the SSES strategy and the TSES strategy, respectively. For fair comparisons, the parameters of elimination algorithms are tuned to get good performance. We tune the 'droptol' parameter that affects ILUPACK accuracy to ensure that the memory costs of the various tested solvers are comparable. However, methods that incorporate elimination reduce memory costs to such a great extent that other methods often cannot guarantee convergence at the same memory footprint. Therefore, the elimination cases are tuned to have the same memory cost, the non-elimination cases are tuned to have another level of memory cost.

The comparison results are presented in Table 4, where T_p, T_f and T_s denote the CPU time (in seconds) needed by the pre-processing including reordering and/or elimination, the factorization step and the Krylov subspace solver, respectively, and T_{total} denotes the total CPU time. Symbol '-' represents no corresponding metric value.

Table 4. Numerical results for the solution of PageRank problems with $\alpha = 0.995$.

Problem&Method	D_{ratio}	NNZ_r	Mem	T_p	T_f	T_s	T_{total}
cnr-2000							
ILUPACK	100.0%	100.0%	2.22	-	8.92	0.60	9.52
SSES	100.0%	52.9%	1.60	0.95	2.80	0.27	4.03
Lump-5	74.2%	90.8%	2.13	0.13	4.21	0.36	4.70
5-re-SSES	74.2%	46.6%	1.61	0.50	2.20	0.07	2.76
5-re-TSES	74.2%	42.6%	1.60	1.06	2.06	0.05	3.17
web-BerkStan							
ILUPACK	100.0%	100.0%	2.41	-	26.49	4.96	31.44
SSES	100.0%	49.7%	1.59	1.72	9.84	0.32	11.89
Lump-5	89.2%	86.9%	2.39	0.32	21.12	0.61	22.06
5-re-SSES	89.2%	44.6%	1.59	1.65	7.87	0.31	9.83
5-re-TSES	89.2%	41.9%	1.61	3.49	8.13	0.24	11.85
web-Stanford							
ILUPACK	100.0%	100.0%	2.02	-	5.68	4.06	9.74
SSES	100.0%	67.5%	1.60	1.26	3.07	2.13	6.46
Lump-5	92.7%	92.5%	1.99	0.15	3.56	1.61	5.32
5-re-SSES	92.7%	63.0%	1.62	1.21	2.94	1.19	5.34
5-re-TSES	92.7%	59.9%	1.60	1.81	2.68	0.39	4.88
eu-2005							
ILUPACK	100.0%	100.0%	2.04	-	53.52	15.76	69.29
SSES	100.0%	53.5%	1.62	3.78	44.77	1.51	50.06
Lump-5	91.2%	93.9%	2.02	0.79	75.00	7.14	82.93
5-re-SSES	91.2%	50.8%	1.59	3.86	37.54	1.47	42.87
5-re-TSES	91.2%	41.6%	1.61	6.19	36.26	1.40	43.85
in-2004							
ILUPACK	100.0%	100.0%	2.00	-	25.31	15.54	40.85
SSES	100.0%	42.0%	1.23	3.25	12.76	0.69	16.70
Lump-5	77.9%	92.2%	2.01	0.74	21.12	2.32	24.18
5-re-SSES	77.9%	37.2%	1.24	2.60	9.87	0.36	12.84
5-re-TSES	77.9%	32.4%	1.00	5.71	8.06	0.45	14.23
uk-2007-100000							
ILUPACK	100.0%	100.0%	2.04	-	8.19	0.72	8.91
SSES	100.0%	45.9%	1.60	0.28	6.31	0.20	6.79
Lump-5	94.6%	98.0%	2.12	0.11	7.82	0.71	8.63
5-re-SSES	94.6%	44.9%	1.59	0.28	5.77	0.18	6.23
5-re-TSES	94.6%	38.0%	1.60	0.33	4.05	0.09	4.47

As shown in Table 4, SSES and Lump-5 can effectively lower the factorization time T_f and the solving process time T_s, with the preprocessing process adding a small amount of additional time cost T_p. The reason is clearly that the quantity of non-zero values and/or the size of the "kernel system" have been reduced. These two methods increase the total computational efficiency with decreased total time costs, T_{total} as a result. The SSES method outperforms the Lump-5 method because it reduces T_f, T_s and T_{total} by a greater amount. When both are combined, the proposed 5-re-SSES method can further reduce the factorization time T_f and the solving process time T_s and achieve a higher level of computational efficiency with a lower total time cost T_{total}.

Compared with the 5-re-SSES method, the use of the proposed two-stage elimination strategy generally further decreases the factorization time and the solving time. However, because the two-stage elimination strategy may significantly increase the pre-processing time compared to the one-stage elimination process (note, however, that the code has not been fully optimized), the resulting 5-re-TSES method does not outperform the 5-re-SSES method in terms of total time costs. As stated previously, the pre-processing procedure can be implemented only once and used to solve multiple PageRank problems on the same network graph. In such situations, the solution time T_s becomes the most important metric, followed by the factorization time T_f. We conclude that 5-re-TSES will be preferable in terms of time consumption for solving multiple PageRank problems on the same network graph, whereas 5-re-SSES should be preferred for solving a single PageRank linear system. In terms of memory requirements, TSES-type methods are more efficient that SSES-type methods.

6. Conclusions

For the solution of large PageRank models, in this paper we have described a recursive 5-type lumping algorithm combined with a two-stage elimination strategy that integrates information about the nonzero structure of the underlying network and the nonzero values of the PageRank coefficient matrix to reduce the dimension and the density of the relevant PageRank system to solve. Numerical experiments on over 50 real-world networks demonstrate that 1) real networks often have deeper structural characteristics than those already reported in the literature, and these can be used for faster PageRank computations; 2) the structural properties of data networks tend to be related to their application backgrounds; and 3) the proposed methods can exploit these properties effectively and have the potential to decrease significantly the costs of PageRank solutions especially for the case of large and/or multiple damping factors.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (12101433), the Sichuan Science and Technology Program (2023NSFSC1136) and the Two-Way Support Programs of Sichuan Agricultural University, China (1921993077). Bruno Carpentieri belong to the Gruppo Nazionale per il Calcolo Scientifico (GNCS) of the Istituto Nazionale di Alta Matematica (INdAM) and this work was partially supported by INdAM-GNCS under Progetti di Ricerca 2022. Furthermore Bruno Carpentieri was funded by Provincia autonoma di Bolzano/Alto Adige - Ripartizione Innovazione, Ricerca, Universit*à*e Musei (contract nr. 19/34).

Conflict of interest

The authors declare no conflict of interest.

References

1. S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, *Comput. Netw. ISDN Syst.*, **30** (1998), 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X
2. T. Zhou, E. Martinez-Baez, G. Schenter, A. E. Clark, PageRank as a collective variable to study complex chemical transformations and their energy landscapes, *J. Chem. Phys.*, **150** (2019), 134102. https://doi.org/10.1063/1.5082648
3. B. Liu, S. Jiang, Q. Zou, Hits-pr-hhblits: Protein remote homology detection by combining pagerank and hyperlink-induced topic search, *Brief. Bioinformatics*, **21** (2020), 298–308. https://doi.org/10.1093/bib/bby104
4. M. Rafiei, A. A. Kardan, A novel method for expert finding in online communities based on concept map and pagerank, *Hum. Cent. Comput. Inf. Sci.*, **5** (2015), 10. https://doi.org/10.1186/s13673-015-0030-5
5. F. A. Massucci, D. Docampo, Measuring the academic reputation through citation networks via pagerank, *J. Informetr.*, **13** (2019), 185–201. https://doi.org/10.1016/j.joi.2018.12.001
6. M. Zhang, X. Li, L. Zhang, S. Khurshid, Boosting spectrum-based fault localization using Pagerank, In: *Proceedings of the 26th ACM SIGSOFT international symposium on software testing and analysis*, 2017, 261–272. https://doi.org/10.1145/3092703.3092731
7. A. Bojchevski, J. Gasteiger, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, et al., Scaling graph neural networks with approximate pagerank, In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, 2464–2473. https://doi.org/10.1145/3394486.3403296
8. E. Chien, J. Peng, P. Li, O. Milenkovic, Adaptive universal generalized pagerank graph neural network, *arXiv preprint*, 2020. https://doi.org/10.48550/arXiv.2006.07988
9. A. Roth, T. Liebig, Transforming pagerank into an infinite-depth graph neural network, In: *Joint European conference on machine learning and knowledge discovery in databases*, 2022, 469–484. https://doi.org/10.1007/978-3-031-26390-3_27
10. D. F. Gleich, PageRank beyond the web, *SIAM Rev.*, **57** (2015), 321–363. https://doi.org/10.1137/140976649
11. R. A. Horn, S. Serra-Capizzano, A general setting for the parametric Google matrix, *Internet Math.*, **3** (2008), 385–411. https://doi.org/10.1080/15427951.2006.10129131
12. S. Serra-Capizzano, Jordan canonical form of the Google matrix: A potential contribution to the PageRank computation, *SIAM J. Matrix Anal. Appl.*, **27** (2005), 305–312. https://doi.org/10.1137/S0895479804441407
13. A. Langville, C. Meyer, *Google's PageRank and beyond: The science of search engine rankings*, Princeton: Princeton University Press, 2006. https://doi.org/10.1515/9781400830329

14. P. G. Constantine, D. F. Gleich, Random alpha PageRank, *Internet Math.*, **6** (2009), 189–236. https://doi.org/10.1080/15427951.2009.10129185

15. S. D. Kamvar, T. H. Haveliwala, C. D. Manning, G. H. Golub, Extrapolation methods for accelerating PageRank computation, In: *Proceedings of the 12th international conference on World Wide Web*, (2003), 261–270. https://doi.org/10.1145/775152.775190

16. X. Tan, A new extrapolation method for PageRank computations, *J. Comput. Appl. Math.*, **313** (2017), 383–392. https://doi.org/10.1016/j.cam.2016.08.034

17. C. Brezinski, M. Redivo-Zaglia, S. Serra-Capizzano, Extrapolation methods for PageRank computations, *CR Math.*, **340** (2005), 393–397. https://doi.org/10.1016/j.crma.2005.01.015

18. A. Cicone, S. Serra-Capizzano, Google PageRanking problem: The model and the analysis, *J. Comput. Appl. Math.*, **234** (2010), 3140–3169. https://doi.org/10.1016/j.cam.2010.02.005

19. S. D. Kamvar, T. H. Haveliwala, G. H. Golub, Adaptive methods for the computation of the PageRank, *Linear Algebra Appl.*, **386** (2004), 51–65. https://doi.org/10.1016/j.laa.2003.12.008

20. H. D. Sterck, T. A. Manteuffel, S. F. McCormick, Q. Nguyen, J. Ruge, Multilevel adaptive aggregation for Markov chains, with application to web ranking, *SIAM J. Sci. Comput.*, **30** (2008), 2235–2262. https://doi.org/10.1137/070685142

21. Z. L. Shen, T. Z. Huang, B. Carpentieri, C. Wen, X. M. Gu, Block-accelerated aggregation multigrid for Markov chains with application to PageRank problems, *Commun. Nonlinear Sci. Numer. Simul.*, **59** (2018), 472–487. https://doi.org/10.1016/j.cnsns.2017.11.031

22. D. F. Gleich, A. P. Gray, C. Greif, T. Lau, An inner-outer iteration for computing PageRank, *SIAM J. Sci. Comput.*, **32** (2010), 349–371. https://doi.org/10.1137/080727397

23. C. Q. Gu, F. Xie, K. Zhang, A two-step matrix splitting iteration for computing PageRank, *J. Comput. Appl. Math.*, **278** (2015), 19–28. https://doi.org/10.1016/j.cam.2014.09.022

24. C. Wen, T. Z. Huang, Z. L. Shen, A note on the two-step matrix splitting iteration for computing PageRank, *J. Comput. Appl. Math.*, **315** (2017), 87–97. https://doi.org/10.1016/j.cam.2016.10.020

25. Z. L. Tian, Y. Liu, Y. Zhang, Z. Y. Liu, M. Y. Tian, The general inner-outer iteration method based on regular splittings for the PageRank problem, *Appl. Math. Comput.*, **356** (2019), 479–501. https://doi.org/10.1016/j.amc.2019.02.066

26. M. Y. Tian, Y. Zhang, Y. D. Wang, A general multi-splitting iteration method for computing PageRank, *Comput. Appl. Math.*, **38** (2019), 1–29. https://doi.org/10.1007/s40314-019-0830-8

27. G. H. Golub, C. Greif, An Arnoldi-type algorithm for computing pagerank, *BIT Numer. Math.*, **46** (2006), 759–771. https://doi.org/10.1007/s10543-006-0091-y

28. J. F. Yin, G. J. Yin, M. Ng, On adaptively accelerated Arnoldi method for computing PageRank, *Numer. Linear Algebra Appl.*, **19** (2012), 73–85. https://doi.org/10.1002/nla.789

29. Z. L. Shen, H. Yang, B. Carpentieri, X. M. Gu, C. Wen, A preconditioned variant of the refined arnoldi method for computing PageRank eigenvectors, *Symmetry*, **13** (2021), 1327. https://doi.org/10.3390/sym13081327

30. H. F. Zhang, T. Z. Huang, C. Wen, Z. L. Shen, FOM accelerated by an extrapolation method for solving PageRank problems, *J. Comput. Appl. Math.*, **296** (2016), 397–409. https://doi.org/10.1016/j.cam.2015.09.027

31. G. Wu, Y. Wei, A power-Arnoldi algorithm for computing pagerank, *Numer. Linear Algebra Appl.*, **14** (2007), 521–546. https://doi.org/10.1002/nla.531

32. C. Q. Gu, X. L. Jiang, C. C. Shao, Z. B. Chen, A GMRES-Power algorithm for computing PageRank problems, *J. Comput. Appl. Math.*, **343** (2018), 113–123. https://doi.org/10.1016/j.cam.2018.03.017

33. Q. Y. Hu, C. Wen, T. Z. Huang, Z. L. Shen, X. M. Gu, A variant of the Power-Arnoldi algorithm for computing PageRank, *J. Comput. Appl. Math.*, **381** (2021), 113034. https://doi.org/10.1016/j.cam.2020.113034

34. C. Q. Gu, W. W. Wang, An Arnoldi-Inout algorithm for computing PageRank problems, *J. Comput. Appl. Math.*, **309** (2017), 219–229. https://doi.org/10.1016/j.cam.2016.05.026

35. D. F. Gleich, L. Zhukov, P. Berkhin, Fast parallel pagerank: A linear system approach, 2005.

36. Y. Lin, X. Shi, Y. Wei, On computing PageRank via lumping the Google matrix, *J. Comput. Appl. Math.*, **224** (2009), 702–708. https://doi.org/10.1016/j.cam.2008.06.003

37. Q. Yu, Z. Miao, G. Wu, Y. Wei, Lumping algorithms for computing Google's PageRank and its derivative, with attention to unreferenced nodes, *Inf. Retr.*, **15** (2012), 503–526. https://doi.org/10.1007/s10791-012-9183-2

38. A. N. Langville, C. D. Meyer, A reordering for the PageRank problem, *SIAM J. Sci. Comput.*, **27** (2006), 2112–2120. https://doi.org/10.1137/040607551

39. Z. L. Shen, T. Z. Huang, B. Carpentieri, X. M. Gu, C. Wen, An efficient elimination strategy for solving PageRank problems, *Appl. Math. Comput.*, **298** (2017), 111–122. https://doi.org/10.1016/j.amc.2016.10.031

40. Z. L. Shen, T. Z. Huang, B. Carpentieri, C. Wen, X. M. Gu, X. Y. Tan, Off-diagonal low-rank preconditioner for difficult PageRank problems, *J. Comput. Appl. Math.*, **346** (2019), 456–470. https://doi.org/10.1016/j.cam.2018.07.015

41. Z. L. Shen, B. Carpentieri, Multi-Step Low-Rank Decomposition of Large PageRank Matrices, In: *The 7th international conference on fuzzy systems and data mining*, **340** (2021), 397–404. https://doi.org/10.3233/FAIA210212

42. D. J. Higham, N. J. Higham, *MATLAB guide*, SIAM press, 2016.

43. C. P. Lee, G. H. Golub, S. A. Zenios, Partial state space aggregation based on lumpability and its application to PageRank, *Tech. Rep. Stanford Univ.*, 2003.

44. S. D. Kamvar, T. H. Haveliwala, C. D. Manning, G. H. Goloub, Exploiting the block structure of the web for computing PageRank, *Tech. Rep. Stanford Univ.*, 2003.

45. A. Scime, *Web mining: Applications and techniques*, IGI Global Press, 2005. https://doi.org/10.4018/978-1-59140-414-9

46. Y. P. Hong, C. T. Pan, A lower bound for the smallest singular value, *Linear Algebra Appl.*, **172** (1992), 27–32. https://doi.org/10.1016/0024-3795(92)90016-4

47. O. Axelsson, M. Neytcheva, A general approach to analyse preconditioners for two-by-two block matrices, *Numer. Linear Algebra Appl.*, **20** (2013), 723–742. https://doi.org/10.1002/nla.830

48. T. A. Davis, Y. Hu, The University of Florida sparse matrix collection, *ACM Trans. Math. Softw.*, **38** (2011), 1–25.

49. P. Boldi, S. Vigna, The webgraph framework I: Compression techniques, In: *Proceedings of the 13th international conference on World Wide Web*, 2004, 595–602. https://doi.org/10.1145/988672.988752

50. P. Boldi, M. Rosa, M. Santini, S. Vigna, Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks, In: *Proceedings of the 20th international conference on World Wide Web*, 2011, 587–596. https://doi.org/10.1145/1963405.1963488

51. P. Boldi, B. Codenotti, M. Santini, S. Vigna, Ubicrawler: A scalable fully distributed Web crawler, *Softw. Pract. Exp.*, **34** (2004), 711–726. https://doi.org/10.1002/spe.587

52. Y. Saad, M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Comput.*, **7** (1986), 856–869. https://doi.org/10.1137/0907058

53. M. Bollhöefer, Y. Saad, O. Schenk, ILUPACK-preconditioning software package, 2010.