J Supercomput (2018) 74:3278–3294
https://doi.org/10.1007/s11227-018-2383-9

A heuristic search algorithm based on subspaces for PageRank computation

Takafumi Miyata[1]

Published online: 23 April 2018

Abstract We studied a fast algorithm for the large-scale computation of PageRank. PageRank is what the Google search engine uses to simulate the importance of web pages. It is defined by the eigenvector of a particular stochastic matrix related to the graphs of web pages. The power method is the typical means to compute the eigenvector, while the Krylov subspace method shows faster convergence, which can be regarded as a two-step algorithm. The first step predicts the eigenvector, and the second step corrects the predicted result. More precisely, the power method is first iterated to compute the eigenvector approximately. Secondly, a Krylov subspace spanned by the approximations is searched for a better approximate eigenvector in terms of minimizing a residual. To get a better approximation efficiently, we consider using subspaces not only at the second step but also at the first step. Specifically, a Krylov subspace is first used to compute an approximate eigenvector, by which another subspace is expanded. Secondly, this non-Krylov subspace is searched for a better approximate eigenvector that minimizes its residual over the subspace. This paper describes a heuristic search algorithm iterating the two steps alternately and presents its efficient implementation. Experimental results with huge Google matrices illustrate improvements in performance of the algorithm.

Keywords PageRank · Google matrix · Power iteration · Krylov subspace · Residual minimization · Parallel computing

This work was supported by KAKENHI Grant No. 18K11343.

✉ Takafumi Miyata
miyata@fit.ac.jp

[1] Department of Computer Science and Engineering, Fukuoka Institute of Technology, 3-30-1 Wajiro-higashi, Higashi-ku, Fukuoka 811-0295, Japan

1 Introduction

PageRank is for ranking web pages and plays an important role in the Google search engine [1]. The large-scale computation of PageRank is an attractive research topic in scientific computing and is extended to a variety of applications [2–8]. The PageRank computation is searching for the eigenvector of a huge sparse matrix called the Google matrix, which is an irreducible, positive, and stochastic matrix associated with the graphs of web pages. Specifically, the target is the positive eigenvector corresponding to the largest eigenvalue in absolute value, which is simple and known as 1. When the eigenvector is normalized by the 1-norm, it is called a PageRank vector and the vector elements give rankings of web pages. To get the eigenvector of the large matrix, iterative methods are feasible. The power method is the typical means to compute the eigenvector. The method is accelerated by several techniques with parallel computing such as adaptive computation, extrapolation, relaxation, and Krylov subspaces, see, for example, [9–15] and references therein. This paper describes a heuristic algorithm searching a non-Krylov subspace for the eigenvector, which was motivated as described next.

The power method iterates matrix–vector multiplications, and the current multiplication is simply used as an approximate eigenvector. Instead, a Krylov subspace spanned by the old vectors as well as the current vector is searched in the Arnoldi-type method [12] for a possibly better approximate eigenvector in terms of minimizing a residual. This method can be regarded as a two-step algorithm. The first step predicts the eigenvector by the power method, and the second step corrects the current prediction using the Krylov subspace. We consider using subspaces not only at the second step but also at the first step to improve the prediction and get a better approximation efficiently. Specifically, a Krylov subspace is first used to compute an approximate eigenvector, by which another subspace is expanded. Secondly, this non-Krylov subspace is searched for a better approximate eigenvector that minimizes its residual over the subspace. The purpose of the study was to investigate the performance of a heuristic search algorithm iterating the two steps alternately.

The remainder of this paper is organized as follows. The PageRank problem is described in Sect. 2. Several existing algorithms are discussed in Sect. 3. In Sect. 4, an efficient implementation of the proposed algorithm is described in detail. Experimental results are reported in Sect. 5. Concluding remarks are described in Sect. 6. Notations used throughout the paper are as follows.

- For a given sequence of vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$, $U_k := [\mathbf{u}_1, \ldots, \mathbf{u}_k]$.
- A subset of $\mathbb{R}^n$ is called a subspace. Specifically, the set of all linear combinations of the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k \in \mathbb{R}^n$ is a subspace denoted by $\mathrm{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$:

$$\begin{aligned} \mathrm{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\} &:= \left\{ \sum_{j=1}^{k} y_j \mathbf{u}_j : y_j \in \mathbb{R} \right\} \\ &= \left\{ U_k \mathbf{y} : \mathbf{y} \in \mathbb{R}^k \right\}. \end{aligned}$$

– A k-dimensional Krylov subspace with respect to an $n \times n$ matrix A and a vector $\mathbf{u}_0 \in \mathbb{R}^n$ is denoted by $\mathcal{K}_k(A, \mathbf{u}_0)$ and is defined as follows:

$$\mathcal{K}_k(A, \mathbf{u}_0) := \text{span}\left\{\mathbf{u}_0, A\mathbf{u}_0, \ldots, A^{k-1}\mathbf{u}_0\right\}.$$

– The 1-norm and 2-norm of a vector $\mathbf{u} \in \mathbb{R}^n$ are, respectively, defined as

$$\|\mathbf{u}\|_1 := \sum_{j=1}^{n} |u_j|,$$

$$\|\mathbf{u}\|_2 := \sqrt{\sum_{j=1}^{n} u_j^2}.$$

– For a given matrix, a set of its smallest singular value $\sigma \geq 0$ and the corresponding left and right singular vectors $\mathbf{y}_\mathrm{L}$ and $\mathbf{y}_\mathrm{R}$ with $\|\mathbf{y}_\mathrm{L}\|_2 = \|\mathbf{y}_\mathrm{R}\|_2 = 1$ is the smallest singular triple denoted by $(\sigma, \mathbf{y}_\mathrm{L}, \mathbf{y}_\mathrm{R})$. This triple can be computed by using the singular value decomposition algorithms [16], which are available from the library [17,18].
– MV stands for a matrix–vector multiplication involving a Google matrix A to be defined in (2). This is the only computation accessing the huge matrix in the algorithms we will discuss in Sects. 3 and 4.
– Subscripts of a matrix and a vector are abbreviated when their elements are denoted. For example, the (i, j) element of a matrix $H_{k+1,k}$ is denoted by h_{ij} and the jth element of $\mathbf{u}_0$ is denoted by u_j.
– Let I_k denote the $k \times k$ identity matrix. In particular, the $n \times n$ identity matrix is denoted by I.
– Let $\mathbf{0} := [0, \ldots, 0]^\mathrm{T}$ and $\mathbf{e} := [1, \ldots, 1]^\mathrm{T}$ with appropriate lengths in the context.

2 PageRank

This section describes the definition of PageRank following [5,7].

Let us consider a portion of the Web and denote by n the number of web pages. Let S be an $n \times n$ connectivity matrix, where $s_{ij} = 1$ if there is a hyperlink from the jth page to the ith page and $s_{ij} = 0$ otherwise. The number of web pages n can be large, while each web page often has a few hyperlinks. Thus, the matrix S is large and sparse. Let c_j be the number of outgoing links of the jth page, i.e., $c_j := \sum_i s_{ij}$. Let A be an $n \times n$ real nonsymmetric matrix whose (i, j) element is

$$a_{ij} := \begin{cases} \alpha s_{ij}/c_j + (1-\alpha)/n & (c_j \neq 0), \\ 1/n & (c_j = 0), \end{cases} \tag{1}$$

where α is called the damping factor, which is a positive parameter smaller than 1. This parameter is the probability that a web surfer follows a link. Then, $1 - \alpha$ is the

probability that a web surfer will jump to a random page without following a link and thus $(1-\alpha)/n$ is the probability that a particular random page is chosen. The (i, j) element of A shows the probability that a web surfer moves from the jth page to the ith page. If the jth page has no outgoing link, $c_j = 0$, a random page is chosen with a uniform probability of $1/n$. In matrix form, A is expressed by

$$A = \alpha SD + \mathbf{e}\mathbf{b}^{\mathrm{T}}, \tag{2}$$

where D is an $n \times n$ diagonal matrix whose jth diagonal element is

$$d_{jj} := \begin{cases} 1/c_j & (c_j \neq 0), \\ 0 & (c_j = 0), \end{cases}$$

and $\mathbf{b}$ is an n-vector whose jth element is

$$b_j := \begin{cases} (1-\alpha)/n & (c_j \neq 0), \\ 1/n & (c_j = 0). \end{cases}$$

The matrix A is never formed explicitly and is used through the MV (defined in Sect. 1). In this way, the sparsity of S can be utilized in iterative methods. The second term in (2) accounts for the random choices of web pages. As a consequence of this term, A is irreducible, positive ($a_{ij} > 0$), and column stochastic ($\mathbf{e}^{\mathrm{T}}A = \mathbf{e}$). From the Perron–Frobenius theorem, it follows that the largest eigenvalue of A in absolute value is simple, which is equal to 1, and its corresponding positive eigenvector exists. This eigenpair is denoted by $(\lambda, \mathbf{x})$, where $\lambda = 1$ and $x_j > 0$. If the eigenvector is normalized as $\|\mathbf{x}\|_1 = 1$, then $\mathbf{x}$ defines a PageRank vector, where x_j shows the ranking of the jth page. It concludes that the target is $\mathbf{x}$ satisfying $A\mathbf{x} = \mathbf{x}$, which is the eigenvector corresponding to the known eigenvalue $\lambda = 1$ of the huge matrix A.

The important point to note here is the influence of the damping factor α on the computation of the PageRank vector $\mathbf{x}$. Rankings of web pages are dependent on α. As shown in (1), the closer α is to 1, the closer A is to S that is the original link structure of web pages, i.e., the closer $\mathbf{x}$ is to the PageRank vector of the original structure. Thus, higher values of α will give true PageRank vectors [2]. Meanwhile, α determines the difficulty of computing $\mathbf{x}$. The second largest eigenvalue $\tilde{\lambda}$ of A in absolute value has the property that $|\tilde{\lambda}| \leq \alpha$ [19]. When α is close to 1, λ will not be well separated from other eigenvalues. Consequently, more iterations are taken to compute $\mathbf{x}$ corresponding to λ, and thus fast algorithms are needed [6,9–15].

In the following sections, we focus on algorithms for PageRank computation associated with high values of α.

3 The algorithms for PageRank computation

We discuss existing algorithms to compute the eigenvector $\mathbf{x}$ corresponding to the eigenvalue $\lambda = 1$ of the Google matrix A. The power method and the Arnoldi method

for standard eigenvalue problems [20–22] were sophisticated for PageRank computation. We focus on these methods to present a heuristic search algorithm related to them in the next section. Other existing works include extrapolation methods, see, for example, [6,10,11,13–15]. Experimental results to be described in Sect. 5 also include comparisons with these effective methods.

The power method is the typical means and successively computes MVs:

$$\mathbf{u}_0, A\mathbf{u}_0, \ldots, A^{k-1}\mathbf{u}_0, \tag{3}$$

where $\mathbf{u}_0 \in \mathbb{R}^n$ is a starting vector and $k \in \mathbb{N}$. Suppose $\mathbf{u}_0$ is nonnegative ($u_j \geq 0$) and normalized by the 1-norm. Since $A\mathbf{u}_0$ is also nonnegative and A is column stochastic, $\|A\mathbf{u}_0\|_1 = \mathbf{e}^{\mathrm{T}} A\mathbf{u}_0 = \mathbf{e}^{\mathrm{T}}\mathbf{u}_0 = 1$. This property removes the normalization after each MV, which is usually needed to avoid underflow and overflow, and makes the algorithm simpler [5]. Here, $\mathbf{p}_k := A^{k-1}\mathbf{u}_0$ is simply chosen as the kth approximate eigenvector. The iteration proceeds until the relative residual 1-norm $\|A\mathbf{p}_k - \mathbf{p}_k\|_1 / \|\mathbf{p}_k\|_1 = \|A\mathbf{p}_k - \mathbf{p}_k\|_1$ becomes smaller than a given threshold ε. The algorithm is summarized in Algorithm 1. As iterated, $\mathbf{p}_k$ (denoted by $\mathbf{p}$ in Algorithm 1) linearly converges to $\mathbf{x}$ with the rate $|\tilde{\lambda}/\lambda| = |\tilde{\lambda}| \leq \alpha$. If α is close to 1, the method shows slow convergence.

Algorithm 1 : Power method

```
1: input w, ε                          ▷ Put a nonnegative initial guess into w
2: w := w/‖w‖_1
3: repeat
4:     p := w
5:     w := Ap
6: until ‖w − p‖_1 < ε                 ▷ The relative residual is checked since ‖p‖_1 = 1
7: output p                            ▷ p is an approximate PageRank vector
```

The Arnoldi-type method [12] searches the Krylov subspace spanned by the vectors in (3) for a possibly better approximate eigenvector than what the power method computes. Specifically, an approximate eigenvector $\mathbf{u} \in \mathbb{R}^n$ that minimizes the relative residual 2-norm is computed as follows:

$$\min_{\mathbf{u}\in\mathbb{R}^n} \frac{\|A\mathbf{u} - \mathbf{u}\|_2}{\|\mathbf{u}\|_2}, \quad \mathbf{u} \in \mathcal{K}_k(A, \mathbf{u}_0), \tag{4}$$

where k is a fixed small integer ($k \ll n$) and $\mathbf{u}_0$ is an initial vector. Although the 1-norm may be a natural choice, the 2-norm is used since its minimization can be efficiently computed as follows. Let $\mathbf{u}_1, \ldots, \mathbf{u}_k \in \mathbb{R}^n$ be an orthonormal basis of $\mathcal{K}_k(A, \mathbf{u}_0)$. As defined in Sect. 1, $U_k := [\mathbf{u}_1, \ldots, \mathbf{u}_k]$. The k-step Arnoldi procedure generates U_k (see lines 3–13 in Algorithm 2), which satisfies

$$AU_k = U_{k+1} H_{k+1,k}. \tag{5}$$

In this expression, $H_{k+1,k}$ is a $(k+1) \times k$ upper Hessenberg matrix with entries $h_{ij} = 0$ if $i > j + 1$. From (5), called the Arnoldi decomposition, the approximate

eigenvector $\mathbf{u}$ is obtained as follows. Since $\mathbf{u} \in \mathcal{K}_k(A, \mathbf{u}_0)$, $\mathbf{u} = U_k\mathbf{y}$, where $\mathbf{y} \in \mathbb{R}^k$. Here, $\|\mathbf{y}\|_2 = 1$ and thus $\|\mathbf{u}\|_2 = 1$. From (4) and (5), it follows that

$$\min_{\mathbf{u}\in\mathcal{K}_k} \frac{\|A\mathbf{u} - \mathbf{u}\|_2}{\|\mathbf{u}\|_2} = \min_{\|\mathbf{y}\|_2=1} \|(A - I)U_k\mathbf{y}\|_2 \tag{6}$$

$$= \min_{\|\mathbf{y}\|_2=1} \| \left(H_{k+1,k} - [I_k, \mathbf{0}]^{\mathrm{T}}\right) \mathbf{y}\|_2. \tag{7}$$

Let us denote by $(\sigma, \mathbf{y}_{\mathrm{L}}, \mathbf{y}_{\mathrm{R}})$ the smallest singular triple of the shifted Hessenberg matrix $H_{k+1,k} - [I_k, \mathbf{0}]^{\mathrm{T}}$. The norm in (7) is minimized when $\mathbf{y} = \mathbf{y}_{\mathrm{R}}$, and thus the eigenvector is approximated as $\mathbf{u} := U_k\mathbf{y}_{\mathrm{R}}$. Note that (5) enables us to compute $\mathbf{u}$ using (7) instead of (6). If $\mathbf{u}$ is computed via (6), the singular triple of the $n \times k$ matrix $(A - I)U_k$ is required and its computational cost is $O(nk^2)$ [16]. In contrast, the computation of $\mathbf{u}$ via (7) can be performed in $O(k^3)$, because it requires only the singular triple of the small $(k + 1) \times k$ matrix $H_{k+1,k} - [I_k, \mathbf{0}]^{\mathrm{T}}$ as described above. Therefore, (5) enables the efficient computation of the approximate eigenvector $\mathbf{u}$ minimizing (4). The residual vector corresponding to $\mathbf{u}$ is given by

$$\mathbf{r} := A\mathbf{u} - \mathbf{u} = \sigma U_{k+1}\mathbf{y}_{\mathrm{L}}. \tag{8}$$

As well as the power method, the Arnoldi-type method stops if the relative residual 1-norm $\|\mathbf{r}\|_1/\|\mathbf{u}\|_1$ becomes smaller than a given threshold ε. Otherwise, the method restarts: The current approximation $\mathbf{u}$ becomes a new starting vector $\mathbf{u}_0$, and a Krylov subspace starts from the beginning. Then, the method searches this new subspace for a possibly better approximation. The algorithm is summarized in Algorithm 2. Note that putting $\mathbf{r}$ and $\mathbf{u}$ (computed at the lines 15 and 16 in Algorithm 2) into $\mathbf{u}_{k+1}$ and $\mathbf{u}_1$, respectively, can save storage for them. Then, storage is approximately $n \times (k+1)$ real entries for U_{k+1}.

Algorithm 2 : Arnoldi-type method

1: **input** $\mathbf{u}, k, \varepsilon$ ▷ Put an initial guess into $\mathbf{u}$
2: **repeat**
3: $\quad\mathbf{u}_1 := \mathbf{u}/\|\mathbf{u}\|_2$
4: $\quad$**for** $j = 1, \ldots, k$ **do**
5: $\quad\quad\mathbf{u}_{j+1} := A\mathbf{u}_j$
6: $\quad\quad$**for** $i = 1, \ldots, j$ **do**
7: $\quad\quad\quad h_{ij} := \mathbf{u}_i^{\mathrm{T}}\mathbf{u}_{j+1}$
8: $\quad\quad\quad\mathbf{u}_{j+1} := \mathbf{u}_{j+1} - h_{ij}\mathbf{u}_i$
9: $\quad\quad$**end for**
10: $\quad\quad h_{j+1,j} := \|\mathbf{u}_{j+1}\|_2$
11: $\quad\quad$Stop **if** $h_{j+1,j} = 0$
12: $\quad\quad\mathbf{u}_{j+1} := \mathbf{u}_{j+1}/h_{j+1,j}$
13: $\quad$**end for**
14: $\quad$Compute the smallest singular triple $(\sigma, \mathbf{y}_{\mathrm{L}}, \mathbf{y}_{\mathrm{R}})$ of $H_{k+1,k} - [I_k, \mathbf{0}]^{\mathrm{T}}$
15: $\quad\mathbf{r} := \sigma U_{k+1}\mathbf{y}_{\mathrm{L}}$
16: $\quad\mathbf{u} := U_k\mathbf{y}_{\mathrm{R}}$
17: **until** $\|\mathbf{r}\|_1/\|\mathbf{u}\|_1 < \varepsilon$
18: **output** $\mathbf{u} := \mathbf{u}/\|\mathbf{u}\|_1$ ▷ $\mathbf{u}$ is an approximate PageRank vector

4 Heuristic search algorithm

As described in Sect. 1, we present a heuristic search algorithm for PageRank computation using two subspaces: One is a Krylov subspace to predict the eigenvector, and the other is a non-Krylov subspace to correct the predicted result and get a possibly better approximation to be shown in (22).

Let us assume that a small $(m-1)$-dimensional subspace $\mathcal{V}_{m-1}$ is given, and the approximate eigenvector generated in $\mathcal{V}_{m-1}$ needs to be improved. To get a better approximation, we consider expanding the subspace to

$$\mathcal{V}_m = \mathcal{V}_{m-1} + \mathrm{span}\{\mathbf{u}\}, \quad \mathbf{u} \in \mathcal{K}_k(A, \mathbf{v}_0), \tag{9}$$

where $\mathbf{u}$ is an approximate eigenvector computed by the Arnoldi-type method, and the dimension $k \in \mathbb{N}$ and the initial vector $\mathbf{v}_0 \in \mathbb{R}^n$ of the Krylov subspace are both dependent on m, as described later. In $\mathcal{V}_m$, we compute a new approximate eigenvector $\mathbf{v} \in \mathbb{R}^n$ so that it minimizes a relative residual:

$$\min_{\mathbf{v} \in \mathbb{R}^n} \frac{\|A\mathbf{v} - \mathbf{v}\|_2}{\|\mathbf{v}\|_2}, \quad \mathbf{v} \in \mathcal{V}_m. \tag{10}$$

The main difference between our method and the Arnoldi-type method is the subspace to search for the eigenvector: Our method uses $\mathcal{V}_m$ expanded by (9), instead of the Krylov subspace. Since the useful decomposition in (5) does not hold for the orthonormal basis of $\mathcal{V}_m$, we now consider an efficient way to compute $\mathbf{v}$ minimizing (10) at the following three steps.

Step 1. An iteration of Algorithm 2 is performed to compute the approximate eigenvector $\mathbf{u} \in \mathcal{K}_k(A, \mathbf{v}_0)$ with $\|\mathbf{u}\|_2 = 1$ and $\mathbf{r} = A\mathbf{u} - \mathbf{u}$ with $\sigma = \|\mathbf{r}\|_2$, see (8). As described later, k and $\mathbf{v}_0$ are both dependent on m.
Let us denote by $\mathbf{v}_1, \ldots, \mathbf{v}_{m-1} \in \mathbb{R}^n$ an orthonormal basis of $\mathcal{V}_{m-1}$. From (9), $\mathbf{u}$ is orthonormalized against $\mathbf{v}_1, \ldots, \mathbf{v}_{m-1}$ to compute the mth basis vector and expand $\mathcal{V}_{m-1}$ to $\mathcal{V}_m$ as follows:

$$\mathbf{v}_m = \tilde{\mathbf{v}}_m / \|\tilde{\mathbf{v}}_m\|_2, \quad \tilde{\mathbf{v}}_m = \left(I - \mathbf{v}_{m-1}\mathbf{v}_{m-1}^{\mathrm{T}}\right) \cdots \left(I - \mathbf{v}_1\mathbf{v}_1^{\mathrm{T}}\right) \mathbf{u}. \tag{11}$$

The orthonormalization procedure gives not only $\mathbf{v}_m$ but also

$$\beta := \|\tilde{\mathbf{v}}_m\|_2, \quad \mathbf{f} := V_{m-1}^{\mathrm{T}}\mathbf{u}. \tag{12}$$

These results are used at the next step.

Step 2. As the preliminary for computing $\mathbf{v}$, we consider a QR decomposition

$$(A - I)V_m = Q_m R_m, \tag{13}$$

where $Q_m = [\mathbf{q}_1, \ldots, \mathbf{q}_m]$ is an $n \times m$ orthogonal matrix and R_m is a small $m \times m$ upper triangular matrix. Suppose that Q_{m-1} and R_{m-1} are precom-

puted, where R_{m-1} is the $(m-1)\times(m-1)$ upper left block of R_m. Here, $\mathbf{q}_m$ and the last column of R_m need to be computed.
The mth basis vector $\mathbf{q}_m$ is obtained by orthonormalizing $(A-I)\mathbf{v}_m$ against $\mathbf{q}_1,\ldots,\mathbf{q}_{m-1}$. From (11) and (12), it follows that

$$\begin{aligned}(A-I)\,\mathbf{v}_m &= (A-I)\,\tilde{\mathbf{v}}_m/\beta\\ &= \left((A-I)\,\mathbf{u}-(A-I)\,V_{m-1}V_{m-1}^{\mathrm{T}}\mathbf{u}\right)/\beta\\ &= \left(\mathbf{r}-Q_{m-1}R_{m-1}\mathbf{f}\right)/\beta. \end{aligned}\tag{14}$$

This equation indicates that $\mathbf{q}_m$ can be computed via the orthonormalization of $\mathbf{r}$, instead of $(A-I)\mathbf{v}_m$:

$$\mathbf{q}_m=\tilde{\mathbf{q}}_m/\|\tilde{\mathbf{q}}_m\|_2,\quad \tilde{\mathbf{q}}_m=\left(I-\mathbf{q}_{m-1}\mathbf{q}_{m-1}^{\mathrm{T}}\right)\cdots\left(I-\mathbf{q}_1\mathbf{q}_1^{\mathrm{T}}\right)\mathbf{r}.\tag{15}$$

In this way, we can efficiently compute $\mathbf{q}_m$ without the MV. Through the orthonormalization, we get

$$\hat{\beta}:=\|\tilde{\mathbf{q}}_m\|_2,\quad \hat{\mathbf{f}}:=Q_{m-1}^{\mathrm{T}}\mathbf{r}.\tag{16}$$

In the usual manner, the last column of R_m is obtained by

$$\left[r_{1m},\ldots,r_{m-1,m}\right]^{\mathrm{T}}=Q_{m-1}^{\mathrm{T}}(A-I)\mathbf{v}_m,\tag{17}$$

$$r_{mm}=\|\left(I-\mathbf{q}_{m-1}\mathbf{q}_{m-1}^{\mathrm{T}}\right)\cdots\left(I-\mathbf{q}_1\mathbf{q}_1^{\mathrm{T}}\right)(A-I)\mathbf{v}_m\|_2.\tag{18}$$

This computation requires the MV in both (17) and (18). Using (12)–(16) in (17) and (18), we can efficiently compute the last column of R_m without the MVs as follows:

$$\left[r_{1m},\ldots,r_{m-1,m}\right]^{\mathrm{T}}=\left(\hat{\mathbf{f}}-R_{m-1}\mathbf{f}\right)/\beta,\tag{19}$$

$$r_{mm}=\hat{\beta}/\beta.\tag{20}$$

Step 3. Finally, we compute $\mathbf{v}$ by using the singular triple of the small $m\times m$ matrix R_m as follows. Since $\mathbf{v}\in\mathcal{V}_m$, $\mathbf{v}=V_m\hat{\mathbf{y}}$, where $\hat{\mathbf{y}}\in\mathbb{R}^m$. Here, $\|\hat{\mathbf{y}}\|_2=1$ and thus $\|\mathbf{v}\|_2=1$. From (10) and (13), it follows that

$$\begin{aligned}\min_{\mathbf{v}\in\mathcal{V}_m}\frac{\|A\mathbf{v}-\mathbf{v}\|_2}{\|\mathbf{v}\|_2} &= \min_{\|\hat{\mathbf{y}}\|_2=1}\|(A-I)V_m\hat{\mathbf{y}}\|_2\\ &= \min_{\|\hat{\mathbf{y}}\|_2=1}\|R_m\hat{\mathbf{y}}\|_2.\end{aligned}$$

Let $\left(\hat{\sigma},\hat{\mathbf{y}}_{\mathrm{L}},\hat{\mathbf{y}}_{\mathrm{R}}\right)$ be the smallest singular triple of R_m. Then, the approximate eigenvector is $\mathbf{v}:=V_m\hat{\mathbf{y}}_{\mathrm{R}}$, and its corresponding residual vector is

$$\mathbf{s}:=A\mathbf{v}-\mathbf{v}=\hat{\sigma}\,Q_m\hat{\mathbf{y}}_{\mathrm{L}}.\tag{21}$$

From (9), (21), and the minimal residual condition over $\mathcal{V}_m$, it follows that

$$\frac{\|A\mathbf{v} - \mathbf{v}\|_2}{\|\mathbf{v}\|_2} = \hat{\sigma} \leq \sigma = \frac{\|A\mathbf{u} - \mathbf{u}\|_2}{\|\mathbf{u}\|_2}. \tag{22}$$

Thus, we can get the possibly better approximate eigenvector $\mathbf{v}$ in terms of minimizing the relative residual 2-norm.

The method stops if the relative residual 1-norm $\gamma := \|\mathbf{s}\|_1/\|\mathbf{v}\|_1$ becomes smaller than a given threshold ε. Otherwise, the next iteration will start after updating the two inputs at Step 1: the dimension k and the starting vector $\mathbf{v}_0$ of the Krylov subspace as described below.

The Arnoldi-type method used at Step 1 requires a real $n \times (k+1)$ array to store the Krylov subspace basis U_{k+1} in column. In addition, the two n-vectors $\mathbf{v}_m$ and $\mathbf{q}_m$ need to be stored additionally per iteration. To keep the storage requirement under control, we change k per iteration as $k := k_{\max} - 2(m-1)$, where $k_{\max}$ is a positive integer. Then, storage is kept fixed as $n \times (k_{\max}+1)$ real entries, though the Krylov subspace reduces per iteration and will deteriorate the approximation $\mathbf{u}$. We compensate for this by another setting as follows.

After $m_{\max} := \lfloor k_{\max}/2 \rfloor$ iterations, the method restarts with $m = 1$. To this end, a new starting vector $\mathbf{v}_0$ is needed. Although $\mathbf{v}$ is a simple choice, we use another choice: $\mathbf{v}_0 := A^{\ell}\mathbf{v}$, where ℓ is a positive integer. In other words, the starting vector is improved by the power method to compensate for the reduction described above. No additional storage for this computation is needed by using a part of the storage for V_m and Q_m temporarily. On the parameter ℓ, there is a trade-off between the possibly faster convergence and the increase of MVs per iteration. We adopt the variable choice: Let ℓ grow if the convergence is unacceptable. To be more precise, ℓ is increased by a predefined integer ℓ_{add} if $\gamma > \delta\gamma_{\text{pre}}$, where γ is the relative residual 1-norm, γ_{pre} is the previous one, and δ is a predefined threshold. Note that the first MV can be saved via $A\mathbf{v} = \mathbf{v} + \mathbf{s}$. Thus, the number of MVs per iteration is $k + \ell - 1 = k_{\max} + \ell + 1 - 2m$ in total, where ℓ is adaptively changed. As the subspace size $k_{\max}$ increases, the algorithm will show faster convergence but requires more storage and more computations per iteration.

We summarize the heuristic search method in Algorithm 3 and comment on some parts of it referring to line numbers in Algorithm 3 as follows.

- 4–8: A loop of the Arnoldi-type method is used. At lines 4 and 5, the dimension and the starting vector of the Krylov subspace are updated, respectively. At lines 7 and 8, the residual vector $\mathbf{r}$ and the eigenvector $\mathbf{u}$ are computed and put into $\mathbf{q}_m$ and $\mathbf{v}_m$, respectively.
- 9–14: The vector $\mathbf{u}$ stored in $\mathbf{v}_m$ is orthonormalized against $\mathbf{v}_1, \ldots, \mathbf{v}_{m-1}$ by (11). The resulting vector is put into $\mathbf{v}_m$.
- 15–20: The vector $\mathbf{r}$ stored in $\mathbf{q}_m$ is orthonormalized against $\mathbf{q}_1, \ldots, \mathbf{q}_{m-1}$ by (15). The resulting vector is put into $\mathbf{q}_m$.
- 21–22: The last column of R_m is computed via (19) and (20).
- 24–25: The approximate eigenvector $\mathbf{v}$ and the residual vector $\mathbf{s}$ are computed. We do not need the additional storage for these vectors: We can put $\mathbf{v}$ and $\mathbf{s}$ into $\mathbf{v}_{m+1}$ and $\mathbf{v}_{m+2}$, respectively, to store them temporarily. Then, storage is kept

Algorithm 3 : Heuristic search method

1: **input** $\mathbf{v}, k_{\max}, \varepsilon, \ell_{\text{start}}, \ell_{\text{add}}, \ell_{\max}, \delta$ ▷ Put an initial guess into $\mathbf{v}$
2: $m := 1,\ m_{\max} := \lfloor k_{\max}/2 \rfloor,\ \gamma_{\text{pre}} := 1,\ \ell := \ell_{\text{start}}$
3: **repeat**
4: $k := k_{\max} - 2(m-1)$ ▷ Update the dimension of the Krylov subspace
5: $\mathbf{u}_1 := \mathbf{v}/\|\mathbf{v}\|_2$ ▷ Update the starting vector of the Krylov subspace
6: Execute the lines 4–14 in Algorithm 2 ▷ Use the Arnoldi-type method
7: $\mathbf{q}_m := \sigma U_{k+1}\mathbf{y}_{\text{L}}$ ▷ Compute $\mathbf{r}$ and put it into $\mathbf{q}_m$
8: $\mathbf{v}_m := U_k\mathbf{y}_{\text{R}}$ ▷ Compute $\mathbf{u}$ and put it into $\mathbf{v}_m$
9: **for** $j = 1, \ldots, m-1$ **do**
10: $f_j := \mathbf{v}_j^{\text{T}}\mathbf{v}_m$
11: $\mathbf{v}_m := \mathbf{v}_m - f_j\mathbf{v}_j$
12: **end for**
13: $\beta := \|\mathbf{v}_m\|_2$
14: $\mathbf{v}_m := \mathbf{v}_m/\beta$
15: **for** $j = 1, \ldots, m-1$ **do**
16: $\hat{f}_j := \mathbf{q}_j^{\text{T}}\mathbf{q}_m$
17: $\mathbf{q}_m := \mathbf{q}_m - \hat{f}_j\mathbf{q}_j$
18: **end for**
19: $\hat{\beta} := \|\mathbf{q}_m\|_2$
20: $\mathbf{q}_m := \mathbf{q}_m/\hat{\beta}$
21: $[r_{1m}, \ldots, r_{m-1,m}]^{\text{T}} := (\hat{\mathbf{f}} - R_{m-1}\mathbf{f})/\beta$ if $m \neq 1$
22: $r_{mm} := \hat{\beta}/\beta$
23: Compute the smallest singular triple $(\hat{\sigma}, \hat{\mathbf{y}}_{\text{L}}, \hat{\mathbf{y}}_{\text{R}})$ of R_m
24: $\mathbf{v} := V_m\hat{\mathbf{y}}_{\text{R}}$
25: $\mathbf{s} := \hat{\sigma}\,Q_m\hat{\mathbf{y}}_{\text{L}}$
26: $\gamma := \|\mathbf{s}\|_1/\|\mathbf{v}\|_1$ ▷ Compute the relative residual 1-norm
27: **if** $\gamma > \varepsilon$ **then**
28: **if** $\ell < \ell_{\max}$ and $\gamma > \delta\gamma_{\text{pre}}$ **then**
29: $\ell := \ell + \ell_{\text{add}}$ ▷ Update ℓ if the convergence is unacceptable
30: **end if**
31: $\gamma_{\text{pre}} := \gamma$
32: $\mathbf{v} := \mathbf{v} + \mathbf{s}$
33: **for** $j = 1, \ldots, \ell - 1$ **do** ▷ Compute $A^{\ell}\mathbf{v}$ and put it into $\mathbf{v}$
34: $\mathbf{s} := A\mathbf{v}$
35: $\mathbf{v} := \mathbf{s}$
36: **end for**
37: **if** $m < m_{\max}$ **then**
38: $m := m + 1$
39: **else**
40: $m := 1$ ▷ Restart
41: **end if**
42: **end if**
43: **until** $\gamma < \varepsilon$
44: **output** $\mathbf{v} := \mathbf{v}/\|\mathbf{v}\|_1$ ▷ $\mathbf{v}$ is an approximate PageRank vector

as $n \times (k_{\max} + 1)$ real entries. To this end, some modifications are needed at the following lines.

1: **input** $\mathbf{v}_1, \ldots$ (Put an initial guess into $\mathbf{v}_1$);
5: $\mathbf{u}_1 := \mathbf{v}_m/\|\mathbf{v}_m\|_2$;
24: $\mathbf{v}_{m+1} := V_m\hat{\mathbf{y}}_{\text{R}}$;
25: $\mathbf{v}_{m+2} := \hat{\sigma}\,Q_m\hat{\mathbf{y}}_{\text{L}}$;
26: $\gamma := \|\mathbf{v}_{m+2}\|_1/\|\mathbf{v}_{m+1}\|_1$;

Table 1 The order n and the number of nonzeros nz of the test matrices

No.	Matrix	n	nz
1	Stanford Berkeley	683,446	7,583,376
2	Wikipedia-20051105	1,634,989	19,753,078
3	Wikipedia-20070206	3,566,907	45,030,389
4	Indochina-2004	7,414,866	194,109,311
5	UK-2002	18,520,486	298,113,762
6	UK-2005	39,459,925	936,364,282
7	Webbase-2001	118,142,155	1,019,903,190

32: $\mathbf{v}_{m+1} := \mathbf{v}_{m+1} + \mathbf{v}_{m+2}$;
34: $\mathbf{v}_{m+2} := A\mathbf{v}_{m+1}$;
35: $\mathbf{v}_{m+1} := \mathbf{v}_{m+2}$;
—: Insert $\mathbf{v}_1 := \mathbf{v}_{m+1}$ between the lines 39 and 40;
44: $\mathbf{v}_{m+1} := \mathbf{v}_{m+1}/\|\mathbf{v}_{m+1}\|_1$.

27–42: These lines are skipped if the stopping criterion, $\gamma < \varepsilon$, is accepted.

28–30: The parameter ℓ is adaptively determined. The other parameters are empirically given as $\ell_{\text{start}} = 10$, $\ell_{\text{add}} = 5$, $\ell_{\max} = 100$, and $\delta = 0.9$.

32–36: The new starting vector $A^{\ell}\mathbf{v}$ is computed. At line 32, the first multiplication is computed via $A\mathbf{v} = \mathbf{v} + \mathbf{s}$ without the MV. The rest $\ell - 1$ MVs are computed through the loop at lines 33–36, where $\mathbf{v}$ and $\mathbf{s}$ are used as temporal storage. At the end of this loop, $\mathbf{v}$ is overwritten by $A^{\ell}\mathbf{v}$.

5 Experimental results

The performance of the heuristic search algorithm was tested in Fortran 95 double precision arithmetic run under Linux with an Intel Xeon E5-1680V3 octacore operating up to 3.8 GHz. Our codes were compiled by Intel Fortran version 15.0.3.

Table 1 presents the test matrices obtained from the collection [23], which provide S in (2). Note that the matrix No. 1 in the collection provides S; the remaining matrices provide S^{T}.

The effect of the damping factor α on the convergence of the algorithm was evaluated on the smaller matrices Nos. 1–3. Recall that α determines the Google matrix A in (2). The results of the heuristic search algorithm and the most basic power algorithm are compared in Sect. 5.1.

The heuristic search algorithm and four existing algorithms were tested on the larger matrices Nos. 4–7: the power algorithm, the Arnoldi-type algorithm [12], the Arnoldi extrapolation (ArnoldiEx) algorithm [14], and the heuristic relaxed and extrapolated (HRELEXT) algorithm [15]. The comparisons were made in parallel computing and are reported in Sect. 5.2.

In all the experiments, an initial guess of the eigenvector was given by $\mathbf{e}$, and the stopping criterion was the relative residual 1-norm below $\varepsilon = 10^{-7}$.

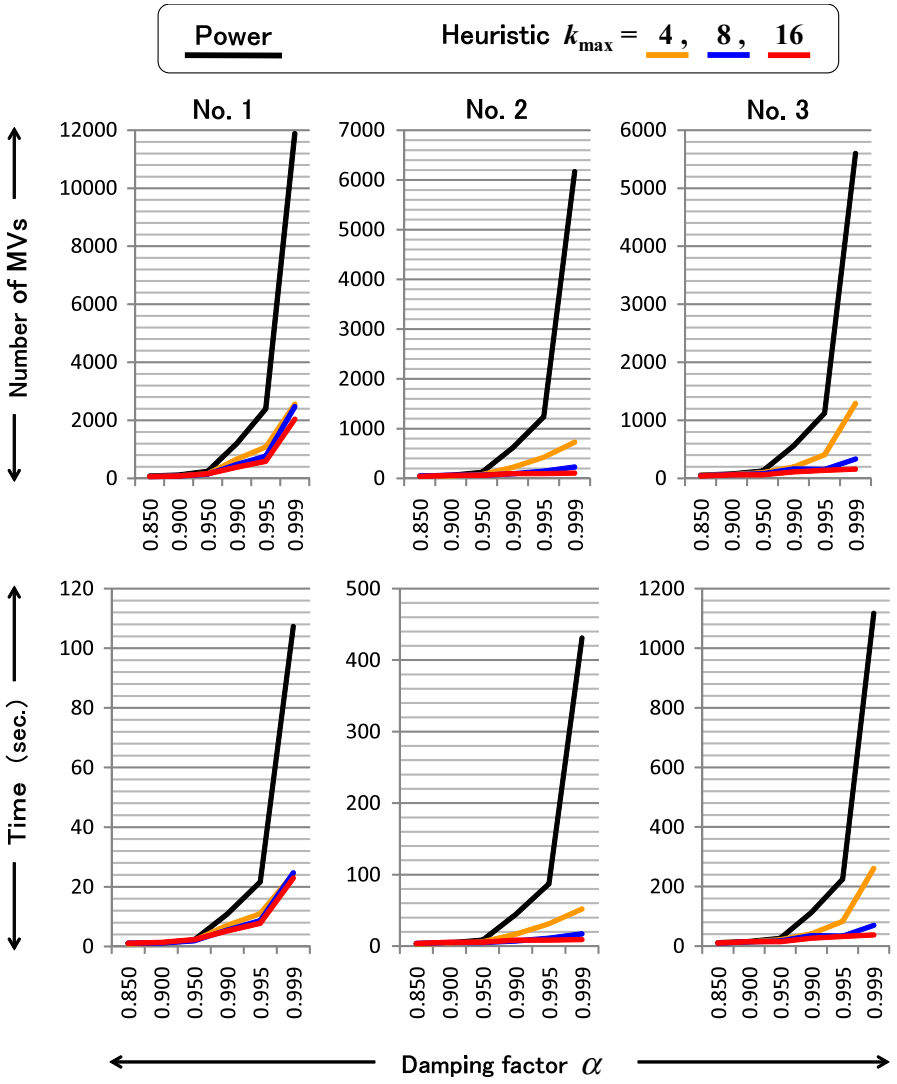

Fig. 1 Number of MVs required for convergence (top) and computational time (bottom) versus the damping factor α in the power algorithm and the heuristic search algorithm with different subspace sizes ($k_{\max} = 4$, 8, and 16). The algorithms are evaluated on the matrices Nos. 1–3 (left/center/right)

5.1 Effect of damping factor on convergence

We first examined the effect of the damping factor α on the convergence behavior of the heuristic search algorithm with several sizes $k_{\max}$ of the subspace.

Figure 1 shows the relationship between α and the performance of the algorithm. As α increased, the computational time increased because more iterations were required for convergence. This behavior was described in Sect. 2. The heuristic search algorithm

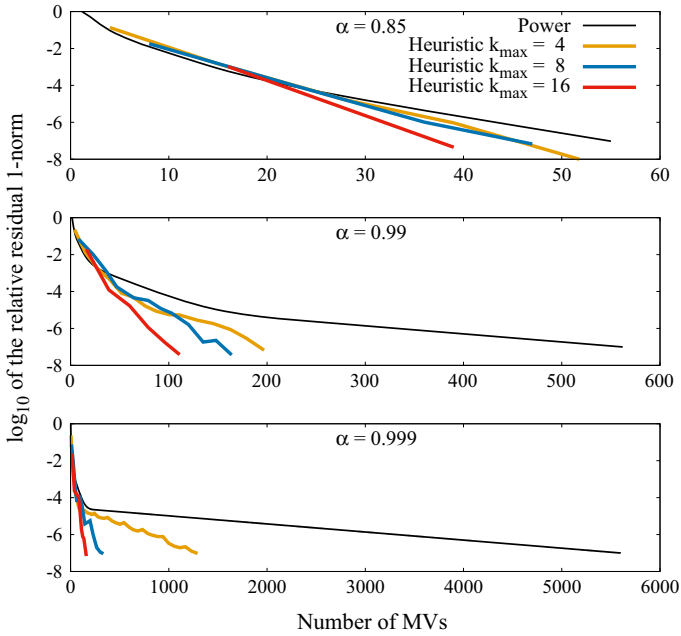

Fig. 2 Convergence behaviors of the power algorithm and the heuristic search algorithm with different subspace sizes ($k_{\max} = 4$, 8, and 16) and damping factors α. The algorithms are evaluated on the matrix No. 3

converged within fewer MVs and within computational time lesser than that required by the power algorithm, particularly for high α. In addition, searching larger subspaces improved the performance of the heuristic search algorithm in these cases.

Figure 2 shows the convergence behaviors of the algorithms. The convergences of both algorithms were similar for $\alpha = 0.85$, but when α was increased to 0.99, our algorithm with $k_{\max} = 4$, 8, and 16 converged within approximately 63.5, 68.5, and 76.7% fewer MVs than the power algorithm, respectively. When α was further increased to 0.999, these reductions improved to 76.6, 93.7, and 96.6%, respectively.

5.2 Performance comparisons in parallel computing

Next, we examined the parallel performance of the heuristic search algorithm using OpenMP [24]. In the algorithm, the MV was parallelized so that each thread handled the same number of nonzero elements. The inner product of vectors and the axpy operation were also parallelized. For a fair comparison, the existing algorithms (the power algorithm, Arnoldi-type algorithm, ArnoldiEx algorithm, and HRELEXT algorithm) were parallelized in the same manner. In the Arnoldi-type algorithm and its variant, the ArnoldiEx algorithm, the subspace size was set to $k = 6$ or 10. Accordingly, the subspace size in the heuristic search algorithm was set to $k_{\max} = 6$ or 10. As tested in [14], the ArnoldiEx algorithm performed the extrapolation process every 50 power iterations until the relative residual 1-norm falls below 10^{-5}.

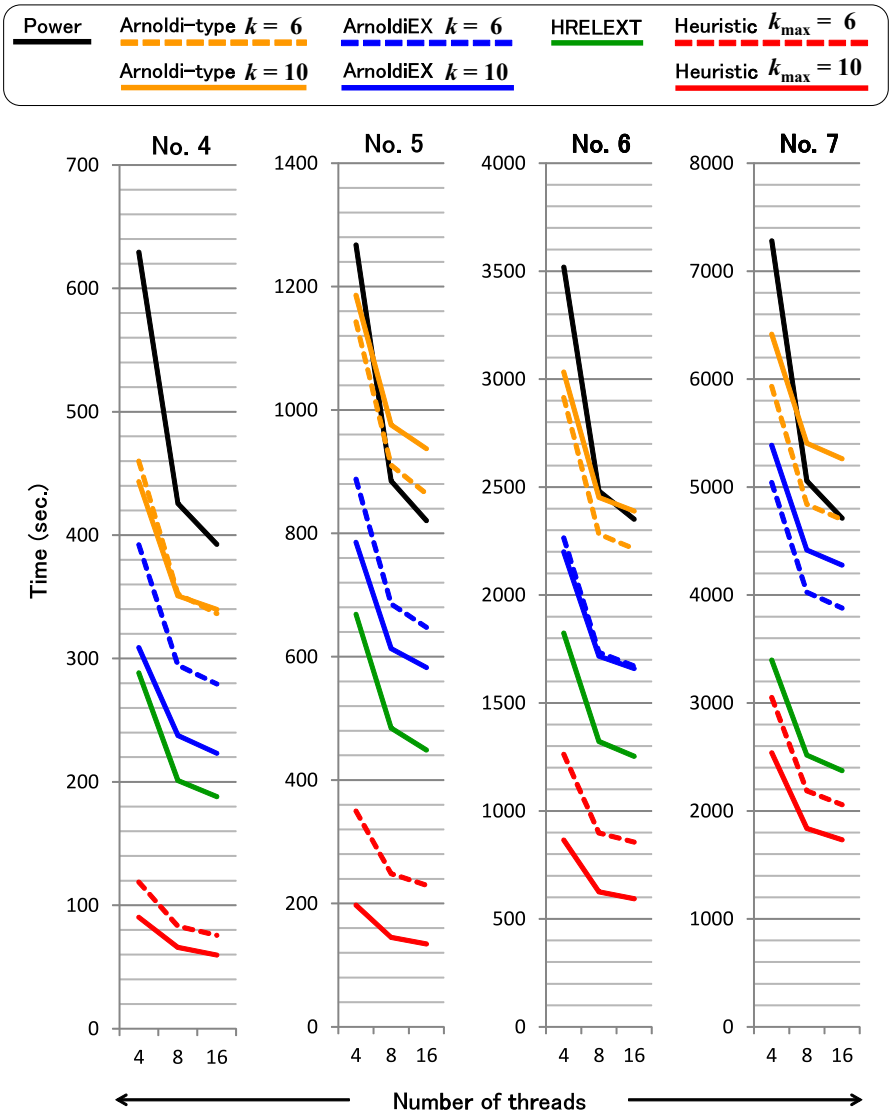

Fig. 3 Computational times of parallel algorithms evaluated on the matrices Nos. 4–7 with the damping factor $\alpha = 0.999$. Results are plotted for the power algorithm, the Arnoldi-type algorithm with the subspace size $k = 6$ and 10, the ArnoldiEx algorithm with the subspace size $k = 6$ and 10, the HRELEXT algorithm, and the heuristic search algorithm with the subspace size $k_{\max} = 6$ and 10

Figure 3 relates the number of threads to the computational time in each algorithm. Increasing the thread number reduced the computational time. The heuristic search algorithm consistently required less computational time than the other algorithms.

Figure 4 shows the convergence behaviors of the algorithms. These results were invariant with increasing number of threads. The ArnoldiEx algorithm with $k = 10$

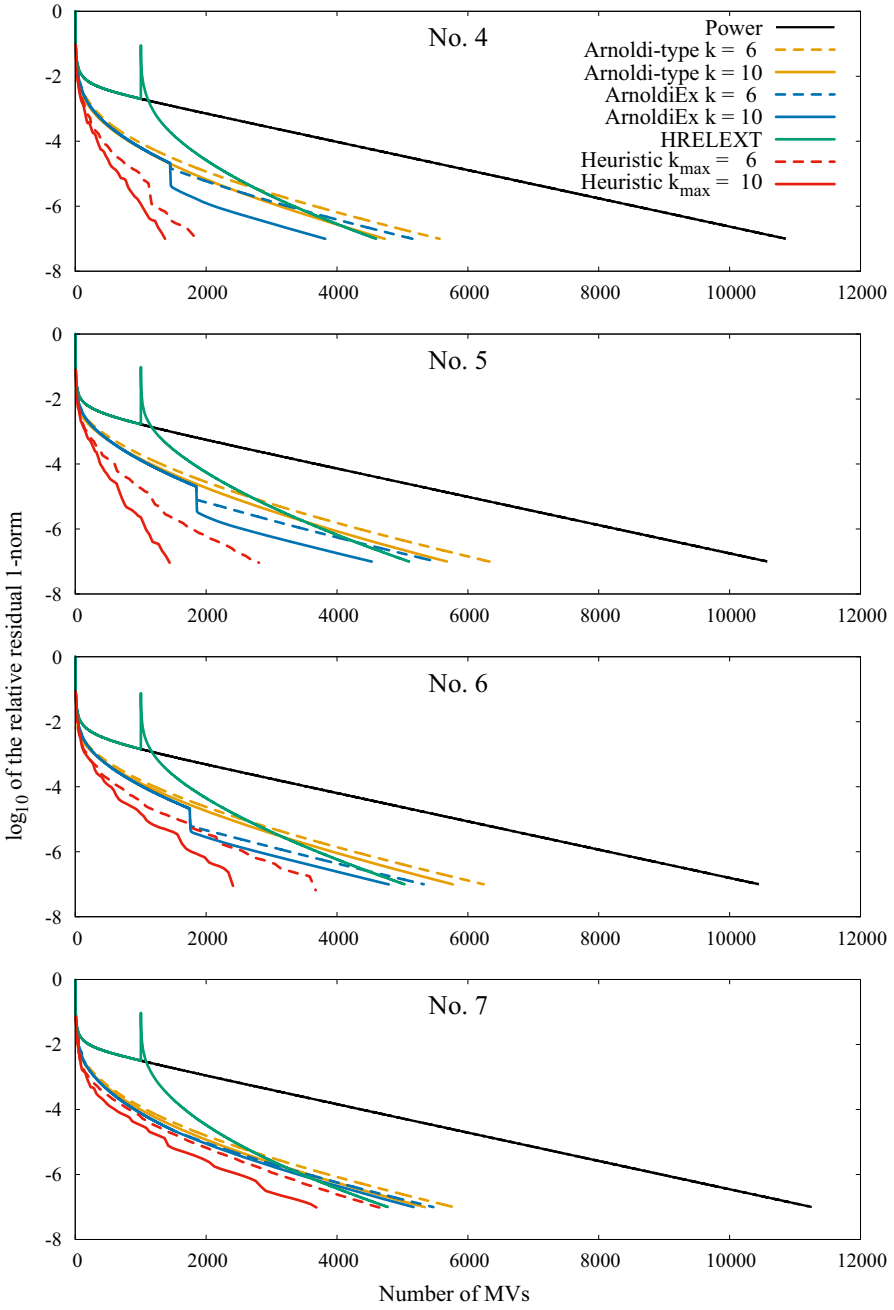

Fig. 4 Convergence behaviors of the algorithms evaluated on the matrices Nos. 4–7 with the damping factor $\alpha = 0.999$. Results are plotted for the power algorithm, the Arnoldi-type algorithm with the subspace size $k = 6$ and 10, the ArnoldiEx algorithm with the subspace size $k = 6$ and 10, the HRELEXT algorithm, and the heuristic search algorithm with the subspace size $k_{\max} = 6$ and 10

converged slightly faster than the HRELEXT algorithm on the matrices Nos. 4–6, but required more computations for generating the subspaces than HRELEXT. For this reason, the HRELEXT algorithm required the least computational time among the existing algorithms (see Fig. 3). The heuristic search algorithm converged faster than any of the existing algorithms. Moreover, both the convergence and computational time of the heuristic algorithm were more enhanced by searching the larger dimensional subspace $k_{\max} = 10$ than by searching the smaller dimensional subspace $k_{\max} = 6$ in these cases (see Fig. 3).

6 Conclusion

We introduced a heuristic search algorithm for the eigenvector computation arising from PageRank. In this algorithm, a Krylov subspace is used to compute an approximate eigenvector, by which another subspace is expanded iteratively. In this non-Krylov subspace, a new approximate eigenvector is generated so that it minimizes a residual over the subspace. This computation corrects the approximate eigenvector predicted in the Krylov subspace and enables us to get a possibly better approximation. The algorithm showed better performance, particularly for damping factors close to 1.

References

1. Page L, Brin S, Motwani R, Winograd T (1999) The PageRank citation ranking: bringing order to the web. Stanford University Technical Report 1999-66
2. Langville AN, Meyer CD (2003) Deeper inside PageRank. Internet Math 1(3):335–380
3. Langville AN, Meyer CD (2006) Google's PageRank and beyond: the science of search engine rankings. Princeton University Press, Princeton
4. Eldén L (2006) Numerical linear algebra in data mining. Acta Numer 15:327–384
5. Eldén L (2007) Matrix methods in data mining and pattern recognition. SIAM, Philadelphia
6. Kamvar SD (2010) Numerical algorithms for personalized search in self-organizing information networks. Princeton University Press, Princeton
7. Moler C (2011) Experiments with MATLAB. Electronic edition published by MathWorks. http://www.mathworks.com/moler
8. Gleich DF (2015) PageRank beyond the web. SIAM Rev 57(3):321–363
9. Kamvar SD, Haveliwala TH, Golub GH (2003) Adaptive methods for the computation of PageRank. Stanford University Technical Report 2003-26
10. Kamvar SD, Haveliwala TH, Manning CD, Golub GH (2003) Extrapolation methods for accelerating PageRank computations. In: Proceedings of the 12th International Conference on World Wide Web
11. Haveliwala TH, Kamvar SD, Klein D, Manning CD, Golub GH (2003) Computing PageRank using power extrapolation. Stanford University Technical Report 2003-45
12. Golub GH, Greif C (2006) An Arnoldi-type algorithm for computing page rank. BIT 46(4):759–771
13. Arnal J, Migallón H, Migallón V, Palomino JA, Penadés J (2014) Parallel relaxed and extrapolated algorithms for computing PageRank. J Supercomput 70(2):637–648
14. Tan X (2017) A new extrapolation method for PageRank computations. J Comput Appl Math 313:383–392
15. Migallón H, Migallón V, Palomino JA, Penadés J (2016) A heuristic relaxed extrapolated algorithm for accelerating PageRank. Adv Eng Softw. https://doi.org/10.1016/j.advengsoft.2016.01.024
16. Golub GH, Loan CFV (2012) Matrix computations. SIAM, Philadelphia
17. LAPACK—Linear Algebra PACKage. http://www.netlib.org/lapack/
18. Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999) LAPACK users' guide. SIAM, Philadelphia

19. Haveliwala TH, Kamvar SD (2003) The second eigenvalue of the Google matrix. Stanford University Technical Report 2003-20
20. Arnoldi WE (1951) The principle of minimized iterations in the solution of the matrix eigenvalue problem. Q Appl Math 9(1):17–29
21. Wilkinson JH (1988) The algebraic eigenvalue problem. Oxford University Press, Oxford
22. Bai Z, Demmel J, Dongarra J, Ruhe A, Vorst H (2000) Templates for the solution of algebraic eigenvalue problems: a practical guide. SIAM, Philadelphia
23. Davis TA, Hu Y (2011) The university of Florida sparse matrix collection. ACM Trans Math Softw 38(1):1–25. Available as the SuiteSparse matrix collection. http://www.cise.ufl.edu/research/sparse/matrices/
24. OpenMP application programming interface examples ver. 4.5.0. http://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf