Methods for solving PageRank with multiple damping factors

Luca Lombardo

Abstract

Since its publication in 1998, the PageRank model has undergone extensive study and has been adapted for various forms and applications. The goal of this project is to implement a modified version of the Power method for solving the PageRank problem with multiple damping factors, as proposed in [1]. An algorithm for solving the PageRank problem with multiple damping factors using the Shifted `GMRES` method will also be proposed, although this has not yet been fully implemented and therefore no numerical results are presented.

Contents

1 Introduction

The PageRank algorithm is a method developed by Google to determine the relevance of web pages to specific keywords. It is used to rank search results based on the importance of the pages, as determined by the probability that a web surfer will visit them. The algorithm works by representing the links between web pages as a directed graph, with each page represented by a vertex and each link represented by an edge. The importance of a page is then determined by the number of links pointing to it and the importance of the pages that link to it. The PageRank algorithm is based on the idea that a page is more likely to be important if it is linked to by other important pages, and it is represented mathematically by a transition probability matrix $P \in \mathbb{R}^{n\times n}$, which can be calculated using the formula in equation 1, where we consider its adjacency matrix $G \in \mathbb{N}^{n\times n}$, where n is the number of pages and G_{ij} is nonzero (being 1) only if the *j-th* page has a hyperlink pointing to the *i-th* page.

$$P(i,j) = \begin{cases} \dfrac{1}{\sum_{k=1}^{n} G_{kj}} & \text{if } G_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The entire random process needs a unique stationary distribution. To ensure this propriety is satisfied, the transition matrix P is usually modified to be an irreducible stochastic matrix A (called the Google matrix) as follows:

$$A = \alpha \tilde{P} + (1-\alpha) v e^T \tag{2}$$

In 2 we have defined a new matrix called $\tilde{P} = P + vd^T$ where $d \in N^{n\times 1}$ is a binary vector tracing the indices of the damping web pages with no hyperlinks, i.e., $d(i) = 1$ if the *i-th* page has no hyperlink, $v \in \mathbb{R}^{n\times 1}$ is a probability vector, $e = [1,1,...,1]^T$ and $0 < \alpha < 1$, the so-called damping factor that represents the probability in the model that the surfer transfer by clicking a hyperlink rather than other ways. Mathematically, the PageRank model can be formulated as the problem of finding the positive unit eigenvector x (the so-called PageRank vector) such that

$$Ax = x, \quad \|x\| = 1, \quad x > 0 \tag{3}$$

equivalently, the problem of finding the solution of the linear system

$$(I - \alpha \tilde{P})x = (1-\alpha)v \tag{4}$$

In recent years, there has been a lot of interest in finding efficient ways to solve problems 3 and 4, especially when the number of variables (denoted by n) is very large. For moderate values of the damping factor (e.g., $\alpha = 0.85$, as suggested by Google for search engine rankings), the Power method has proven to be a reliable solution. However, when the damping factor gets closer to 1, which is necessary in some cases, traditional iterative methods like the Power method may not work as well and more robust algorithms may be required. This point is emphasized in the paper [1].

In the reference paper that we are using for this project, the authors focus their attention in the area of PageRank computations with the same network structure but multiple damping factors. For example, in the Random Alpha PageRank model used in the design of anti-spam mechanism

[2], the rankings corresponding to many different damping factors close to 1 need to be computed simultaneously. They explain that the problem can be expressed mathematically as solving a sequence of linear systems

$$(I - \alpha_i \tilde{P}) x_i = (1 - \alpha_i) v \quad \alpha_i \in (0,1) \quad \forall i \in \{1,2,...,s\} \tag{5}$$

Traditionally, PageRank algorithms applied to problem 5 would involve solving multiple linear systems independently. While this process can be parallelized to some extent, it can still be computationally intensive for high-dimensional problems. In an effort to find more efficient methods with lower algorithmic and memory complexity, the authors of the paper searched for alternative approaches that would allow them to solve larger problems with moderate computing resources. They proposed expressing the PageRank problem with multiple damping factors given in 5 as a series of shifted linear systems, in the form described in the following equation. This approach aims to reduce the computational demands of the problem.

$$\left(\frac{1}{\alpha_i} I - \tilde{P}\right) x^{(i)} = \frac{1 - \alpha_i}{\alpha_i} v \quad \forall i \in \{1,2,...,s\} \quad 0 < \alpha_i < 1 \tag{6}$$

It has been previously noted in the literature that the Shifted Krylov methods may have slow convergence when the damping factor gets close to 1, requiring a larger search space to achieve satisfactory speed. In order to address this issue, the authors of [1] suggest combining stationary iterative methods with shifted Krylov subspace methods. They present an implementation of the Power method that can solve the PageRank problem with multiple damping factors in approximately the same amount of time as the standard Power method for solving a single system. They also show that this shifted Power method generates collinear residual vectors. Based on this result, they use the shifted Power iterations to provide smooth initial solutions for running shifted Krylov subspace methods such as `GMRES`. In addition, they discuss how techniques such as seed system choosing and extrapolation can be used to further accelerate the iterative process.

1.1 Overview of the classical PageRank problem

The Power method is a popular algorithm for solving either the eigenvalue problem in equation 3 or the linear system in equation 4, which were originally used to calculate PageRank by Google. It works by iteratively applying the matrix A to an initial estimate of the solution, using the following formula:

$$x_{(k+1)} = A x_k = \alpha \tilde{P} x_{(k)} + (1 - \alpha) v \tag{7}$$

The convergence behavior is determined mainly by the ratio between the two largest eigenvalues of A. When α gets closer to 1, though, the convergence can slow down significantly.

As stated in [1] The number of iterations required to reduce the initial residual down to a tolerance τ, measured as $\tau = \|Ax_k - x_k\| = \|x_{k+1} - x_k\|$ can be estimated as $\frac{\log_{10}\tau}{\log_{10}\alpha}$. The authors provide an example: when $\tau = 10^{-8}$ the Power method requires about 175 steps to converge for $\alpha = 0.9$ but the iteration count rapidly grows to 1833 for $\alpha = 0.99$. Therefore, for values of the damping parameter very close to 1 more robust alternatives to the simple Power algorithm should be used.

2 The shifted power method for PageRank computations

This section presents the adaptations of stationary iterative methods for solving PageRank problems with multiple damping factors, as described in [1]. The goal is to determine if there are implementations of these methods that have a computational cost similar to that of solving a standard PageRank problem with a single damping factor when applied to the problem with multiple damping factors. In other words, we want to know if these methods are efficient for solving PageRank problems with multiple damping factors.

2.1 The implementation of the shifted power method

The authors of [1] were motivated by the idea that shifted Krylov subspaces can save computational cost by reducing duplications in the calculations of multiple linear systems. They therefore sought to determine if there were similar opportunities for optimization in the case of stationary iterative methods applied to the PageRank problem with multiple damping factors. To do this, they used a dynamic programming approach, in which they analyzed the Power method applied to the sequence of linear systems in equation 4. This standard method computes approximate solutions $x_k^{(i)} (1 \leq i \leq s)$ at the k^{th} iteration of the form:

$$x_k^{(i)} = \alpha_i \tilde{P} x_{k-1} + (1 - \alpha_i) v \tag{8}$$

after k iterations of the Power Method, we obtain

$$x_k^{(i)} = \alpha_i^k \tilde{P}^k x_0^{(i)} + (1 - \alpha_i^k) \sum_{j=0}^{k-1} \alpha_i^j \tilde{P}^j v \tag{9}$$

Using the same initial[1] approximation $x_0^{(i)}$ for all the s systems, we can write

$$x_k^{(i)} = \alpha_i^k \tilde{P}^k x_0 + (1 - \alpha_i^k) \sum_{j=0}^{k-1} \alpha_i^j \tilde{P}^j v \tag{10}$$

If the s systems in 4 are solved synchronously, this means that all the $x_k^{(i)}$ are computed only after all previous approximations $x_{k-1}^{(j)}$ are available. We can now rearrange the computation efficiently as reported in [1]:

- at the first iterations
  - compute and store $\mu_1 = \tilde{P} x_0$ and $\mu_2 = v$;
  - compute and store $x_1^{(i)} = \alpha_i \mu_1 + (1 - \alpha_i) \mu_2$;
- at any other subsequent iteration $k > 1$
  - compute and store $x_k^{(i)} := (1 - \alpha_i) \sum_{j=0}^{k-2} \alpha_i^j \tilde{P}^j v = x_{k-1}^{(i)} - \alpha_i^{k-1} \mu_1$;
  - compute and store $\mu_1 = \tilde{P} \mu_1$ and $\mu_2 = \tilde{P} \mu_2$;

[1] Note that, the option to choose any the same initial guess x_0 for the s systems is acceptable since the Power method converges for every positive initial probability vector [1]

– compute and store $x_k^{(i)} = \alpha_i \mu_1 + x_k^{(i)} + (1-\alpha_i)\alpha_i^{k-1}\mu_2$.

This implementation requires at most 2 matrix-vector products at each step, which is a significant gain compared to the s matrix-vector products required by the standard Power method to compute $x_{k+1}^{(i)}$, especially when $s \gg 2$. In particular, it can be found that if $x_0 = v$ then the vector $\tilde{P}^{k-1} v = \tilde{P}^{k-1} x_0$ is available from the $(k-1)-th$ iteration. Accordingly, all the approximations $x_k^{(i)}$ have the same computational cost, i.e 1 matrix-vector product, of solving one linear system in 4. The Power Iteration formulae 10 can be rewritten as

$$x_{(i)}^k = \alpha_i^k \tilde{P}^k v + (1-\alpha_i^k) \sum_{j=0}^{k-1} \alpha_i^j \tilde{P}^j v = \tag{11}$$

$$= \sum_{j=1}^{k} \alpha_i^j \tilde{P}^{j-1}(\tilde{P}v - v) + v$$

This was of course still a theoretical explanation. An efficient implementation can be written to compute and store $\mu = \tilde{P}v - v$ at the first iteration and then store

$$\mu = \tilde{P}^{k-1}(\tilde{P}v - v) = \tilde{P} \cdot (\tilde{P}^{k-2}(\tilde{P}v - v))$$

at each k-th iteration ($k > 1$), and then from each approximate solution as $x_k^{(i)} = \alpha_i^k \mu + x_{k-1}^{(i)}$. The residual vector $r_k^{(i)}$ associated with the approximate solution $x_k^{(i)}$ has the following expression

$$r_k^{(i)} = A x_k^{(i)} - x_k^{(i)} = x_{k+1}^{(i)} - x_k^{(i)} = \alpha_i^{k+1} \tilde{P}^k (\tilde{P}v - v) \tag{12}$$

Since in general each of the s linear systems may require a different number of Power iterations to converge, the s residual norms have to be monitored separately to test the convergence.

Now we can summarize the efficient implementation of the Power method presented in this section for solving problem 4 in Algorithm 1, as reported in [1]. From now on, we'll refer to this implementation as the *Shifted-Power method*.

Algorithm 1 Shifted-Power method for PageRank with multiple damping factors

Require: $\tilde{P}$, v, τ, $\max_{mv}$, α_i $(1 \le i \le s)$
Ensure: mv, $x^{(i)}$, $r^{(i)}$ $(1 \le i \le s)$
  Compute $\mu = \tilde{P}v - v$
  Set $mv = 1$
  for $i = 1 : s$ **do**
    Compute $r^{(i)} = \alpha_i \mu$
    Compute $Res(i) = \|r^{(i)}\|$
    if $Res(i) \ge \tau$ **then**
      Compute $x^{(i)} = r^{(i)} + v$
    end if
  end for
  while $\max(Res) \ge \tau$ and $mv \le \max_{mv}$ **do**
    compute $\mu = \tilde{P}\mu$
    $mv = mv + 1$
    for $i = 1 : s$ **do**
      if $Res(i) \ge \tau$ **then**
        Compute $r^{(i)} = \alpha_i^{k+1}\mu$
        Compute $Res(i) = \|r^{(i)}\|$
        if $Res(i) \ge \tau$ **then**
          Compute $x^{(i)} = r^{(i)} + x^{(i)}$
        end if
      end if
    end for
  end while

The algorithm stops when either all the residual norms (a measure of how close the current estimate is to the true solution) are smaller than a specified tolerance τ, or when the maximum number of matrix-vector products (multiplication of a matrix by a vector) has been reached. The integer mv counts the number of matrix-vector products performed by the algorithm. An implementation of this algorithm, written in Python, is available in the corresponding github repository for the project.

3 Shifted-GMRES method

This section discusses the approach used by the authors of [1] to combine the shifted power method with the fast shifted GMRES method to create a hybrid algorithm for solving complex PageRank problems with multiple damping factors. The goal of this combination is to create an efficient and reliable algorithm for solving these types of problems. The details of this approach and how it was implemented are described in the cited paper.

3.1 The Arnoldi decomposition

The GMRES method is a non-symmetric Krylov subspace solver based on the Arnoldi decomposition procedure, that the authors sketch in algorithm 2

Algorithm 2 Arnoldi

Require: A, v_0, m
Ensure: $V_m, H_m, v_{m+1}, h_{m+1,m}, \beta, j$
1: Compute $\beta = \|v_0\|$
2: $v_1 = v_0/\beta$
3: **for** $j = 1 : m$ **do**
4: Compute $w = Av_j$
5: **for** $i = 1 : j$ **do**
6: Compute $h_{i,j} = v_i^T w$
7: Compute $w = w - h_{i,j} v_i$
8: **end for**
9: $h_{j+1,j} = \|w_i\|$
10: **if** $h_{j+1,j} = 0$ **then**
11: $m = j$,
12: $v_{m+1} = 0$
13: **break**
14: **else**
15: $v_{j+1} = w/h_{j+1,j}$
16: **end if**
17: **end for**

The Arnoldi procedure, which is used as the basis for the GMRES method, involves iteratively constructing an orthogonal basis $V_m = [v_1, \ldots, v_m]$ and an upper Hessenberg matrix $H_m \in \mathbb{R}^{m \times m}$ from an initial vector $v_0 \in \mathbb{R}^{n \times 1}$ and a matrix $A \in \mathbb{R}^{n \times n}$. After m iterations, it also produces a residual vector $v_{m+1} \in \mathbb{R}^{n \times 1}$ and a residual norm $h_{m+1,m} \in \mathbb{R}$. The GMRES method then uses these to approximate the solution $\tilde{x}$ of the linear system $Ax = b$ that minimizes the residual norm $\|b - Ax\|$ in the Krylov subspace of dimension m. To do this, it starts with an initial guess x_0 and a residual vector $v_0 = b$, and runs m steps of the Arnoldi procedure outlined in algorithm 2.

3.2 The Shifted GMRES method

This modified GMRES algorithm represents an extension of the classical GMRES method, allowing for the solution of a series of shifted linear systems using the same Krylov subspace.

$$(A+\sigma_i I)x_i = b \qquad (\sigma_i \in \mathbb{R}, b \in \mathbb{R}^{n\times 1}, i = 1\dots s)$$

The central concept behind the shifted GMRES method is the utilization of the standard GMRES algorithm on a seed system, with the solutions to the other systems being approximated as byproducts. This approach is made possible through the shift-invariance property of the Krylov subspace, which allows for the utilization of a single Krylov subspace to solve the entire sequence of systems provided that the initial residual vectors of the shifted linear systems are collinear with one another. However, after a certain number of steps of the Arnoldi algorithm, the residual vectors of the additional systems and the residual vector of the seed system will typically lose collinearity, resulting in the loss of the shift-invariance property of the search space.

$$\mathscr{K}_m(A+\sigma_i I, r_m^i) \neq \mathscr{K}_m(A+\sigma_1 I, r_m^1) \qquad i = 2,3,\dots,s$$

This implies that the mutual collinearity of all the residual vectors must be enforced explicitly upon restarting the GMRES algorithm. Specifically, if we assume that the initial residual vectors are collinear, meaning $r_0^i = \gamma_0^i r_0^1$ for $i = 2,3,\dots,s$ where r_0^i is the initial residual vector corresponding to the i-th system and $\gamma_0^i \in \mathbb{R}$, then the shift-invariance property of the Krylov subspace can be maintained.

Algorithm 3 Shifted GMRES

Require: $\tilde{P}, v, m, \alpha_i, maxit, x_0^i \ \ (1 \le i \le s)$
Ensure: $x^i, res_i \ \ (1 \le i \le s), mv$
1: Set $r_0^i = \frac{1-\alpha_i}{\alpha_i} v - \left(\frac{1}{\alpha_i} I - \tilde{P}\right) x_0^i$
2: Set $res_i = \alpha_i \| r_0^i \| \ \ (1 \le i \le s)$
3: Set $mv, iter = 0, 1$
4: **while** $\max(res_i) \ge \tau$ && $iter \le maxit$ **do**
5: Find k that satisfies $res_k = \max(res_i)$
6: Compute $\gamma^i = \frac{res_i \alpha_k}{res_k \alpha_i}$ for all $i \ne k$
7: Run Arnoldi by $[V_m, \bar{H}_m^k, v_{m+1}, \bar{h}_{m+1,m}, \beta, j] = Arnoldi(\frac{1}{\alpha_k} I - \tilde{P}, r_0^k, m)$
8: Set $mv = mv + j$
9: Compute y_k, the minimizer of $\| \beta e_1 - \bar{H}_m^k y_k \|_2$
10: Compute $x^k = x_0^k + V_m y_k$
11: Compute $res_k = \alpha_k \| \beta e_1 - \bar{H}_m^k y^k \|$
12: **for** i = 1, 2, . . . , k-1, k+1, . . . , s **do**
13: **if** $res_i \ge \tau$ **then**
14: Set $\bar{H}_m^i = \bar{H}_m^k + \left(\frac{1-\alpha_i}{\alpha_i} - \frac{1-\alpha_k}{\alpha_k}\right) I_m$
15: Solve y_i and γ_i from $\begin{bmatrix} \bar{H}_m^i & z \end{bmatrix} \begin{bmatrix} y^i \\ \gamma^i \end{bmatrix} = \gamma^i \beta e_1$
16: Set $x^i = x_0^i + V_m y^i$
17: Set $res_i = \frac{\alpha_i}{\alpha_k} \gamma_k^i res_k$
18: **end if**
19: **end for**
20: Set $iter = iter + 1$
21: Set $x_0^i = x^i$
22: **end while**

Where $z = \beta e_1 - H_m^1 y_m^1$. In line 15, by solving this small size system, we can obtain the vector y_m^i and scalar γ_m^i that ensures the collinearity of the shifted results.

Problems: There have been significant issues with the implementation of the shifted GMRES algorithm. A key component of the algorithm is the use of the seed choosing strategy outlined in [1]. However, during testing, it was observed that after the second iteration, the value of k remained constant and the res vector did not change, resulting in a stall situation where the program runs without updating values until the maximum number of iterations is reached. The cause of this problem is currently under investigation. Although a notebook containing the code for the algorithm has been included in the GitHub repository for completeness, it is not functional at this time. It is believed that the issue may be related to a misunderstanding of the algorithm as presented in the pseudo-code, but this has not yet been determined. As a result, there will be no test results for this algorithm in the following section.

4 Numerical experiments

This experiment aims to compare the performance of the shifted Power method to the traditional Power method in solving PageRank problems involving multiple damping factors, specifically $\alpha_1 = 0.85, \alpha_2 = 0.86, ..., \alpha_{15} = 0.99$, on the `web-stanford` and `web-BerkStan` datasets. The `web-stanford` dataset consists of a directed graph with $|V| = 281,903$ nodes and $|E| = 1,810,314$ edges, while the `web-BerkStan` dataset is a directed graph with $|V| = 1,013,320$ nodes and $|E| = 5,308,054$ edges. These datasets can be found at `http://snap.stanford.edu/data/web-Stanford.html` and `http://snap.stanford.edu/data/web-BerkStan.html` respectively and are stored in the `.txt` edge-list format. A summary of the characteristics of the datasets is provided in Table 1.

Dataset	Nodes	Edges	Density
`web-Stanford`	281,903	2,312,497	2.9099×10^{-5}
`web-BerkStan`	685,230	7,600,595	1.6187×10^{-5}

Table 1: Summary of the datasets used in the experiments.

In this study, the personalization vector v was set to $v = [1, 1, ..., 1]^T / n$. All experiments were conducted using Python 3.10 on a 64-bit Arch Linux machine equipped with an AMD Ryzen™ 5 2600 Processor and 16 GB of RAM.

4.1 Technical details

GitHub repository of this project

`https://github.com/lukefleed/ShfitedPowGMRES`

In the GitHub repository for this project, there is an `algo.py` file which contains the implementation of all the functions used in the experiments. The `algo.py` file includes the following functions:

load_data This function loads datasets from the `.txt` edge-list format and returns a networkx graph object. It takes a string as input, with the options being `web-stanford` and `web-BerkStan`.

pagerank Returns the PageRank of the nodes in the graph. It takes as input the following parameters:

- `G`: a networkx graph object.
- `alpha`: Damping parameter for PageRank, default=0.85.
- `personalization`: The "personalization vector" consisting of a dictionary with a key some subset of graph nodes and personalization value each of those. At least one personalization value must be non-zero. If not specified, a nodes personalization value will $1/N$ where N is the number of nodes in `G`.

- `max_iter`: The maximum number of iterations in power method eigenvalue solver. Default is 200.
- `nstart`: Starting value of PageRank iteration for each node. Default is *None*.
- `tol`: Error tolerance used to check convergence in power method solver. Default is 10^{-6}.
- `weight`: Edge data key corresponding to the edge weight. If None, then uniform weights are assumed. Default is *None*.
- `dangling`: The outedges to be assigned to any "dangling" nodes, i.e., nodes without any outedges. The dict key is the node the outedge points to and the dict value is the weight of that outedge. By default, dangling nodes are given outedges according to the personalization vector (uniform if not specified).

This function is strongly based on the `pagerank_scipy` function of the networkx library.

shifted_pow_pagerank : This is the implementation of algorithm 1

There is also another function called `pagerank_numpy` which utilizes NumPy's interface to the `LAPACK` eigenvalue solvers for the calculation of the eigenvector. This method is the fastest and most accurate for small graphs. However, the eigenvector calculation is not stable for large graphs, so the `pagerank_numpy` function is not used in the experiments.

4.2 Convergence results for the Shifted Power method

In the PageRank formulation involving multiple damping factors, the iterative solution of each i-th linear system is initialized with the initial guess $x_0^{(i)} = v$ and is terminated when the solution $x_k^{(i)}$ meets the following criteria:

$$\frac{\|(1-\alpha_i)v - (I - \alpha_i \tilde{P} x_k^{(i)})\|_2}{\|x_k^{(i)}\|_2} < 10^{-8}$$

or the number of matrix-vector products exceeds 1000.

In this experiment, the performance of the shifted Power method is compared to that of the traditional Power method in solving PageRank problems with multiple damping factors.

Dataset	Method	CPU Time (s)	mv
web-Stanford	Power	74.5	738
web-Stanford	Shifted Power	2320.5	276
web-BerkStan	Power	194.1	555
web-BerkStan	Shifted Power	5740.7	356

Table 2: Summary of the experiments.

The results presented in Table 2 differ somewhat from those reported in the study by Shen et al. [1], where the CPU time of the shifted Power method was found to be lower than that of the standard Power method. In contrast, our experiments showed that the CPU time of the shifted Power

method was significantly higher than that of the standard Power method. On the other hand, as predicted by theory, the number of matrix-vector products is lower for the shifted Power method.

There could be various reasons for the discrepancies in the results. One potential explanation is the difference in programming language and implementation, as well as the possibility of a misunderstanding of the pseudo-code provided in [1]. It is also possible that the standard PageRank function, which is a slightly modified version of the network library function `pagerank_scipy`, is better optimized compared to the implementation of the shifted Power method written for this study. Additionally, the `Web-BerkStan` network is quite different from the `web-stanford` network, with the adjacency matrix for the former containing many rows with a large number of zeros compared to the latter (4744 vs 172). This could potentially have a negative impact on the performance of the shifted Power method for networks with a significant number of dangling nodes.

4.2.1 Matrix-vector products for the standard pagerank

In this study, we compared the number of matrix-vector products required to solve the PageRank problem using the shifted Power method and the standard Power method. Results showed that the number of matrix-vector products required for the shifted Power method was significantly lower than that of the standard Power method. Figure 1 demonstrates that the number of matrix-vector products required for the standard power method increases exponentially as the value of α increases. The number of matrix-vector products required for the standard power method to converge for various values of α is presented in Table 3.

	Web-Stanford	**Web-BerkStan**
α	*matrix-vector products*	
0.85	19	15
0.86	20	16
0.87	21	17
0.88	23	18
0.89	24	19
0.90	26	21
0.91	29	22
0.92	32	25
0.93	35	27
0.94	40	31
0.95	47	36
0.96	57	43
0.97	73	54
0.98	103	76
0.99	189	135

Table 3: Table of the results

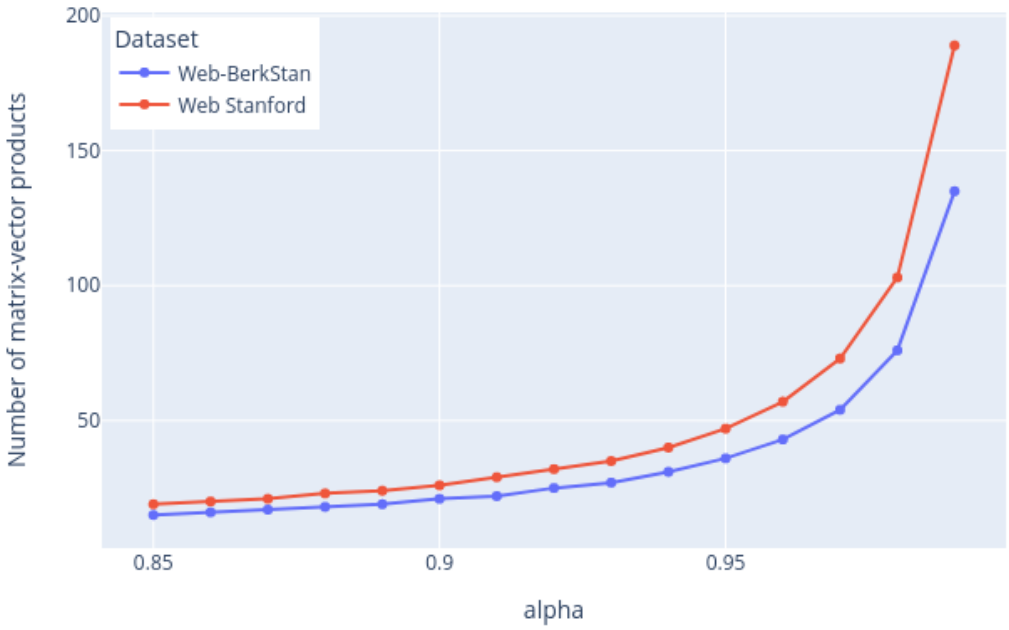

Figure 1: Number of matrix-vector products required for the standard Power method for different values of α.

References

[1] Zhao-Li Shen, Meng Su, Bruno Carpentieri, and Chun Wen. Shifted power-gmres method accelerated by extrapolation for solving pagerank with multiple damping factors. *Applied Mathematics and Computation*, 420:126799, 2022.

[2] Paul G. Constantine and David F. Gleich. Random alpha pagerank. *Internet Mathematics*, 6(2), 1 2009.