MA 202 Project Report

8 May, 2021

Numerical Analysis of Google's PageRank Algorithm

AUTHORS:

Aadesh Desai 19110116

Eshan Gujarathi 19110082

Hiral Sharma 19110016

Sanjay Venkitesh 19110200

Under the guidance of:

Prof. Chetan Pahlajani

Prof. Dilip Sundaram

Prof. Satyajit Pramanik

INDIAN INSTITUTE OF TECHNOLOGY, GANDHINAGAR

Contents

1 Problem Statement

Webpage Searching algorithms have never ceased to evolve since as early as the 80s-90s. But, Google has been proved to be one of the most effective search engines and is the current state of the art, due to the PageRank algorithm developed by Larry Page and Sergey Brin, the founders of Google, while they were graduating from Stanford University in 1999.

For a given search on the engine, based on the keywords, the engine generates a list of webpages that need to be ranked according to their relevance. The PageRank algorithm judges the importance of a web page by looking at the pages that link to it. If a large number of pages hyperlink to it, regarding our search, then we could assert that this page is crucial and relevant. The algorithm's objective is to give all the pages a rank, which can be interpreted as the probability of a person arriving at a particular page by randomly clicking at links on each page.

In this project, we aim to understand the governing conditions that determine the page ranks for a given set of pages that are linked to each other and formulate the relevant equations. Further, we use a numerical method to implement the algorithm and analyse its performance.

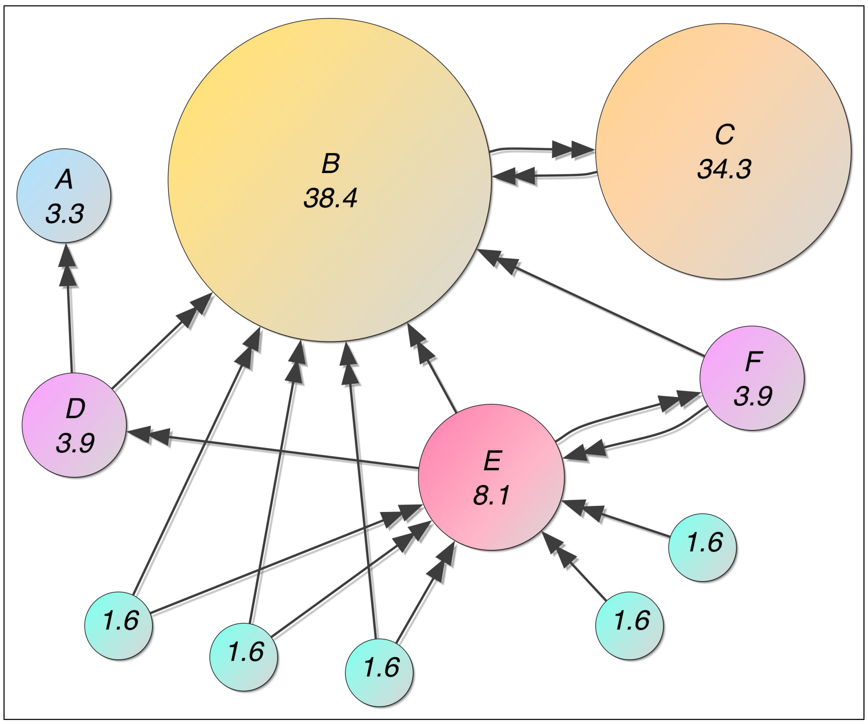


Figure 1: Graphical representation of Page Rank algorithm

2 Mathematical Model

2.1 Formulating Equations

Let's consider a small universe of four web pages linking each other in a random manner. We picture the web net with the help of this directed graph with nodes represented by web pages and edges represented by the links between them.

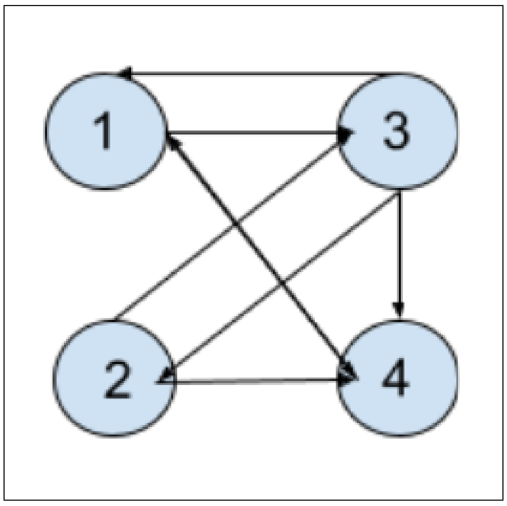


Figure 2: Graph showing web pages and links

Whenever, a website, say 1, hyperlinks another web page 2, we depict it by an arrow from 1 to 2. Here, page 1 has three outgoing links, page 2 has two outgoing links and so on. Let's define an Adjacency Matrix where each column depicts the outgoing links of each node. For this example the matrix will be,

$$\text{M} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Now, let's define a Degree Matrix which represents the number of outgoing links from each node.

$$D = diag(\{\sum M_{ij}\}) \mid \forall i \in V$$

where V is the set of nodes.

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Thus, we define a transition matrix A which is a normalised Adjacency matrix. We normalise the Adjacency matrix by the degree of each node.

$$A = M \cdot D^{-1}$$

In other words, considering that the example page 2 has two outgoing links thus it passes $\frac{1}{2}$ of its importance to 3 and 4. Therefore, we could generalise that if a node has n outgoing links, it will pass an equal fraction, that is, $\frac{1}{n}$ of its importance to all the other receiving nodes. In the above case, we obtain a transition matrix, A as follows:

$$A = \begin{pmatrix} 0 & 0 & \frac{1}{3} & 1 \\ 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{3} & 0 \end{pmatrix}$$

A is the Column stochastic matrix where all the columns of A satisfy the probability axioms i.e. for each column the entries are non-negative and the sum across a column is one.

Now, suppose that initially, all nodes hold the same importance or rank. Each node will thus get $\frac{1}{4}$ the of the rank. Let ‘r’ be the rank vector having all the entries $\frac{1}{4}$. When we consider linking between the web pages, the nodes possessing incoming links will have some elevation in their importance. We thus get the updated rank vector, r’= Ar. We can iterate the process, getting r’= A(Ar)= A^2r. Numerically computing as follow:

$$r = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \implies Ar = \begin{pmatrix} 0.32083 \\ 0.10833 \\ 0.25 \\ 0.32083 \end{pmatrix}$$

Please note that the above equation is an eigenvalue problem where we need to find the eigen matrix corresponding to the eigenvalue 1 for the matrix A. The numerical method discussed uses and iterative approach to determine the same.

Further, A^2r = A(Ar)= A $\cdot \begin{pmatrix} 0.38104 \\ 0.10833 \\ 0.2199 \\ 0.29073 \end{pmatrix}$

Computing so on :

$$A^3\text{r}=\begin{pmatrix} 0.34692 \\ 0.099804 \\ 0.24548 \\ 0.30779 \end{pmatrix}, A^4\text{r}=\begin{pmatrix} 0.34692 \\ 0.099804 \\ 0.24548 \\ 0.30779 \end{pmatrix}, A^5\text{r}=\begin{pmatrix} 0.36867 \\ 0.10705 \\ 0.22736 \\ 0.29691 \end{pmatrix},$$

$$A^6\text{r}=\begin{pmatrix} 0.35429 \\ 0.10192 \\ 0.23968 \\ 0.3041 \end{pmatrix}, A^7\text{r}=\begin{pmatrix} 0.3639 \\ 0.10541 \\ 0.23139 \\ 0.2993 \end{pmatrix}, A^8\text{r}=\begin{pmatrix} 0.3639 \\ 0.10541 \\ 0.23139 \\ 0.2993 \end{pmatrix}$$

As we iterate through the sequence, we notice that the value tends to achieve equilibrium at

$$\text{r}=\begin{pmatrix} 0.3639 \\ 0.10541 \\ 0.23139 \\ 0.2993 \end{pmatrix}$$

2.2 Probabilistic View of the Model

The importance of a page is directly related to its popularity. If a random surfer on the internet opens any random page following the hyperlinks, the probability that the person lands on page i is its importance. Now, if the random surfer is currently at page 2, the surfer has $\frac{1}{2}$ probability of going to pages 3 and 4. The probability of starting from any of the four pages would be equal for all the pages and thus, the initial probability of starting from any of the pages would be $\frac{1}{4}$ for all. Therefore, the initial probability distribution would be a column matrix with all the entries equal to $\frac{1}{4}$, that is, $r = [\frac{1}{4}\frac{1}{4}\frac{1}{4}\frac{1}{4}]^t$. Now after we take a step, the probability that the surfer ends up on i would be Ar. The probability of the random surfer visiting page i at the nth step would be $A^n r$. The sequence, $Ar, A^2r, A^3r. \dots\dots, A^n r, \dots$eventually converge into a unique probability distribution vector, that will be our page rank vector.

2.3 Dangling Nodes

While computing the Column stochastic matrix, we keep a track of the out-degree of all the nodes. And if we encounter a case where a node has zero out degrees i.e. a web page has no out links we add virtual edges directed from that node to all the other nodes. Such nodes are known as sink nodes and or dangling nodes. By doing this the sink nodes which absorbed the random surfer and made the probability of jumping to other pages 0, now can jump to other web pages with equal probability.

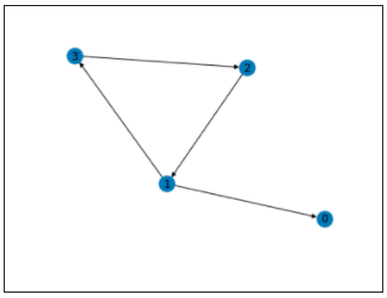
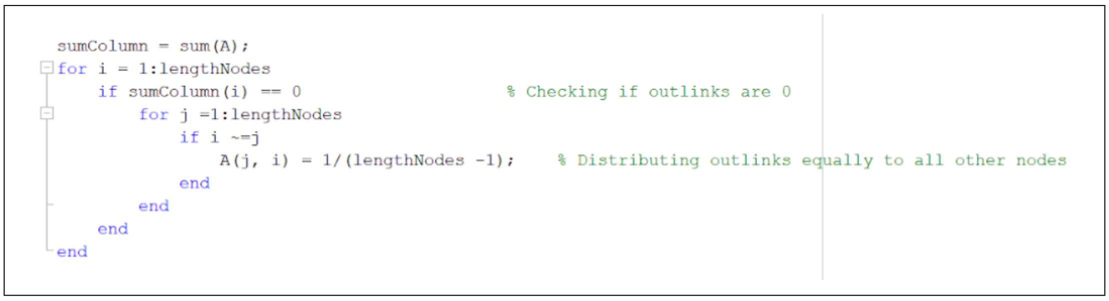


Figure 3: Graph with Dangling Node

In the above-directed graph, web page 0 is the dangling node. Here, the probability that a random surfer can jump from 0 to the other 3 nodes will be $\frac{1}{3}$. Hence the transition matrix M, which satisfies the probability axioms will be:

$$M = \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 1 & 0 \\ \frac{1}{3} & 0 & 0 & 1 \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

The code for handling the dangling nodes scenario is implemented as:



```
sumColumn = sum(A);
for i = 1:lengthNodes
    if sumColumn(i) == 0                    % Checking if outlinks are 0
        for j =1:lengthNodes
            if i ~=j
                A(j, i) = 1/(lengthNodes -1);    % Distributing outlinks equally to all other nodes
            end
        end
    end
end
```

Figure 4: Snippet of the code

2.4 Disconnected Graphs

Sometimes, the web graph has disconnected components as shown in the figure:

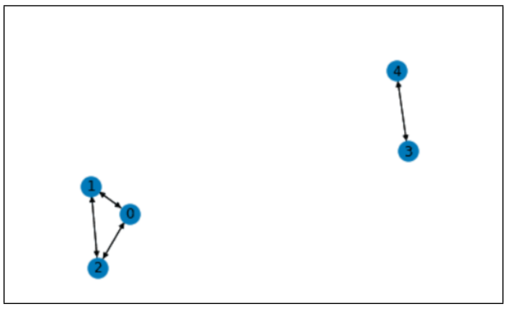


Figure 5: Disconnected graph

In such cases, the transition matrix becomes a block diagonal, and has multiple linearly independent eigenvectors corresponding to the eigenvalue 1.

$$M = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$v = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1/2 \\ 1/2 \end{pmatrix}$$

To address the issue of disconnected components and dangling nodes, Page and Brin introduced the concept of teleportation and damping factor.

2.5 Teleportation and Damping factor

In teleportation, we connect each node of the graph to all other nodes to make the graph complete. They used the idea that a person will continue surfing and go to a webpage according to the transition matrix with probability d, and jump to a random webpage with probability (1-d)/N. This makes our new transition matrix,

$$A = dM + \frac{1-d}{N}$$

Here, d is referred to as the damping factor. Teleportation and the use of damping factor solves both the problems, the problem of dangling nodes, and the problem of disconnected graphs. Since we connect each node with each other node, the entire graph gets connected and also, there is no dangling node left since we make outlinks from each of the nodes.
Let's understand the interpretation of the damping factor by having a view at its meaning at 0 and 1.
Damping factor = 1: It means that a person will keep clicking on links for infinite time and finally end up in a sink. This is the traditional method without the inclusion of damping factor.
Damping factor = 0: It means that all clicks are just random restarts, which are uniformly distributed. The pagerank of all the nodes in this case will be the same.

3 Numerical Solution

3.1 Power Iteration Method

As we saw above, we have the following equation:

$$r^{i+1} = Ar^i$$

where $A_{ij} = dM_{ij} + \frac{1-d}{N}$. This will require a lot of space and time. Thus, we will manipulate the equation to make our matrix sparse to reduce the space complexity and also the time complexity. Here, the rank of any webpage i will be equal to,

$$r_i = \sum_{j=1}^{N} A_{ij} \cdot r_j$$

Now, we will expand the A_{ij} written in the above equation,

$$r_i = \sum_{j=1}^{N} (dM_{ij} + \frac{1-d}{N}) \cdot r_j$$

Distributing the summation, we get,

$$r_i = \sum_{j=1}^{N} dM_{ij} \cdot r_j + \sum_{j=1}^{N} \frac{1-d}{N} \cdot r_j$$

$\frac{1-d}{N}$ is a constant, so we can take it out of the summation,

$$r_i = \sum_{j=1}^{N} dM_{ij} \cdot r_j + \frac{1-d}{N} \cdot \sum_{j=1}^{N} r_j$$

Now, we know that $\sum_{j=1}^{N} r_j$ is equal to 1. This is because the sum of page ranks of all the pages is always equal to 1. Therefore, we can write the equation as,

$$r_i = \sum_{j=1}^{N} dM_{ij} \cdot r_j + \frac{1-d}{N}$$

Therefore, the rank vector can be written as,

$$r^{i+1} = dM \cdot r^i + \left[\frac{1-d}{N}\right]_N$$

where $[x]_N$ denotes a vector of size N, with all values equal to x.
In the power iteration method, we will take an initial guess of the rank vector r, and keep iterating the above equation until convergence is reached.

3.2 Convergence Criteria

In the power iteration method, we keep iterating our equation until we converge to a particular vector r. Thus in our method, we stop iterating when the sum of the absolute difference between r_j^{i+1} and r_j^i becomes less than ϵ or the tolerance value.

Hence, the pseudo code for termination will be:

$$if(sum(abs(r_{new} - r_{old})) < epsilon)\ break$$

3.3 Performance Analysis

After we successfully implement the Power Iteration method and solve to get the rank vector r for a fixed graph, we can extend our analysis to determine the execution time and number of iterations required for convergence when the number of webpages (or nodes) in the graph is changed. To determine the complexity of the algorithm used, we compute the following:

1. Number of iterations needed for convergence vs Number of webpages to be ranked
2. Execution time needed for convergence vs Number of webpages to be ranked

The matrix multiplication takes $O(n^2)$ which is dominant in the algorithm so the time complexity of the algorithm will be $O(n^2)$ but we have manipulated the equations so as to use **sparse matrix** in the matrix multiplication so the actual time complexity of the algorithm is reduced to $O(n)$.

Methodology

- For a given number of webpages, the execution time and number of iterations may vary based on the case, ie, some cases may reach convergence faster than others.
- For a given number of nodes, we create different random graphs and perform the PageRank algorithm on each case.
- We can then use the average value of time and number of iterations for a given number of nodes.

4 Assumptions and Parameters

4.1 Variables and Parameters Used

1. ϵ = Minimum absolute error between successive rank vectors r^i and r^{i+1}: Set to a value of 10^{-12}.
2. **A** = Transition matrix (Column Stochastic Matrix)
3. **r** = Vector of a probability distribution over pages / Page Rank Vector
4. **d** = Damping factor: set to 0.85

4.2 Assumptions

1. **Value of Damping Factor:** The original paper assumes damping factor to be 0.85 as it gives the most optimized results.
2. **Value of** ϵ is taken to be 10^{-12}. This is the sum of the absolute error of all ranks between two successive rank vectors. As we perform the algorithm for a maximum number of pages of the order 10^5, this is a suitable value for absolute error for convergence.
3. **Dandling Nodes:** We need to account for the error caused due to dangling nodes, ie, nodes that do not have any out links. For these nodes, we modify the matrix giving one link from it to every other node.
4. **Performance:**

   (a) **Randomizing graphs for a fixed number of nodes**
   For a given number of nodes, we perform the PageRank algorithm 4 times to determine the average execution time and number of iterations. Each time, we choose $\binom{n}{2}$ links for n nodes and eliminate the repeating links. This is done because the maximum links possible for n nodes is $2\times\binom{n}{2}$.

   (b) **Determining average computation time and a number of iterations**: While analysing performance for a given number of nodes, we take 4 different, randomly chosen graphs and perform the algorithm. The average value is then considered for analysis

   (c) **Maximum number of pages** that can be ranked has been limited to 100 to limit computational overheads. This is a large enough dataset to understand the trend of the number of iterations required and the time complexity of the algorithm.

5 Algorithm Used

We have numerically computed the Page Rank of all the nodes by using MATLAB. The following steps demonstrate the code used to calculate the results:

1. Test the algorithm with an example case:
    (a) Take a base case having less number of nodes (NumberOfNodes $\leq$ 9) to verify the results of the Page Rank algorithm implemented.
    (b) Then find the Number Of Links in the graph using 'nchoosek' inbuilt function of MATLAB.
    (c) Create two arrays and create a graph that has randomly directed links between nodes.
    (d) As we create a data set by generating links randomly, use the 'Simplify' function to remove repeating links if any.
    (e) Compute the indegree and outdegree of each node using inbuilt MATLAB functions and plot the graph to visualize the dataset.
    (f) Initialize Transition matrix A with zeros. Fill the entries of matrix A using the data of links and handling the cases of zero outdegrees of a node.
    (g) Assign values to 'Epsilon' and 'd'.
    (h) Call the function 'pageRank' which outputs Page Rank of nodes, number of iterations and execution time.
2. Analyse the performance by varying the number of nodes:
    (a) Run the Page Rank algorithm 4 times on a network of fixed size.
    (b) Initialize an array with the different number of pages and two arrays empty.
    (c) Initialize two to zeros.
    (d) Repeat for 4 times:
        i. Assign the value of Number Of Nodes to a variable from the value stored in array.
        ii. Follow the steps b) - h) of 1).
        iii. Store the value of time and iterations in arrays assigned.
        iv. Compute the mean execution time and the number of iterations and store them in arrays assigned
    (e) Plot the bar graph comparing the number of iterations and execution time of the different number of pages.
3. 'PageRank' Implementation:
    (a) Start the time by 'tic' function and initialize vector 'r' with uniform distributed values. Initialize a variable to count the iterations.
    (b) Repeat:
        i. Assign the value of 'r' to an array(r_{prev}).
        ii. Update the value of 'r' using the equation $d * A * r_{prev} + (1 - d)/m$;
        iii. Verify the convergence criteria:
            A. Break if the summation of differences of individual elements of array 'r' and r_{prev} is less than Epsilon
        iv. Increment the parameter count.
        v. Stop the time computed using 'toc'.

6 Results and Discussions

6.1 Ranking 6 webpages

Too illustrate the steps involved in the function we have defined, we take an example of a network containing 6 nodes. The graph generated is as follows:

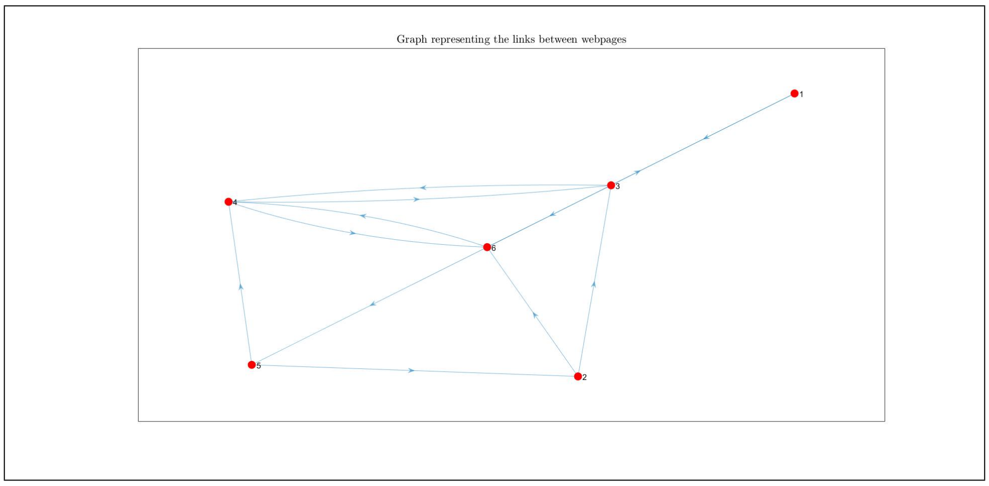
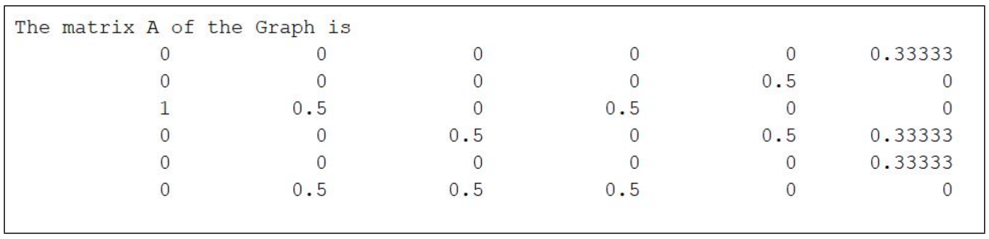


Figure 6: Graph comprising of 6 nodes

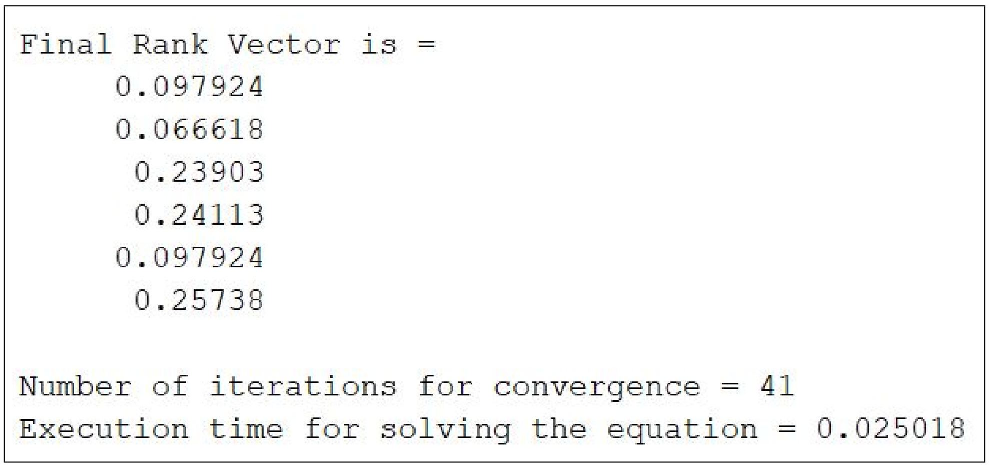
Corresponding to this graph, the code generates the following transition matrix A:



```
The matrix A of the Graph is
            0           0           0           0           0     0.33333
            0           0           0           0         0.5           0
            1         0.5           0         0.5           0           0
            0           0         0.5           0         0.5     0.33333
            0           0           0           0           0     0.33333
            0         0.5         0.5         0.5           0           0
```

Figure 7: Matrix A corresponding to the graph

The final result that we obtain is as follows:



```
Final Rank Vector is =
     0.097924
     0.066618
      0.23903
      0.24113
     0.097924
      0.25738

Number of iterations for convergence = 41
Execution time for solving the equation = 0.025018
```

Figure 8: Rank Vector, Number of Iterations and Execution Time

6.2 Performance Analysis

Upon determining the average execution time and number of iterations needed for convergence on changing the size of the graphs, we get the following result:

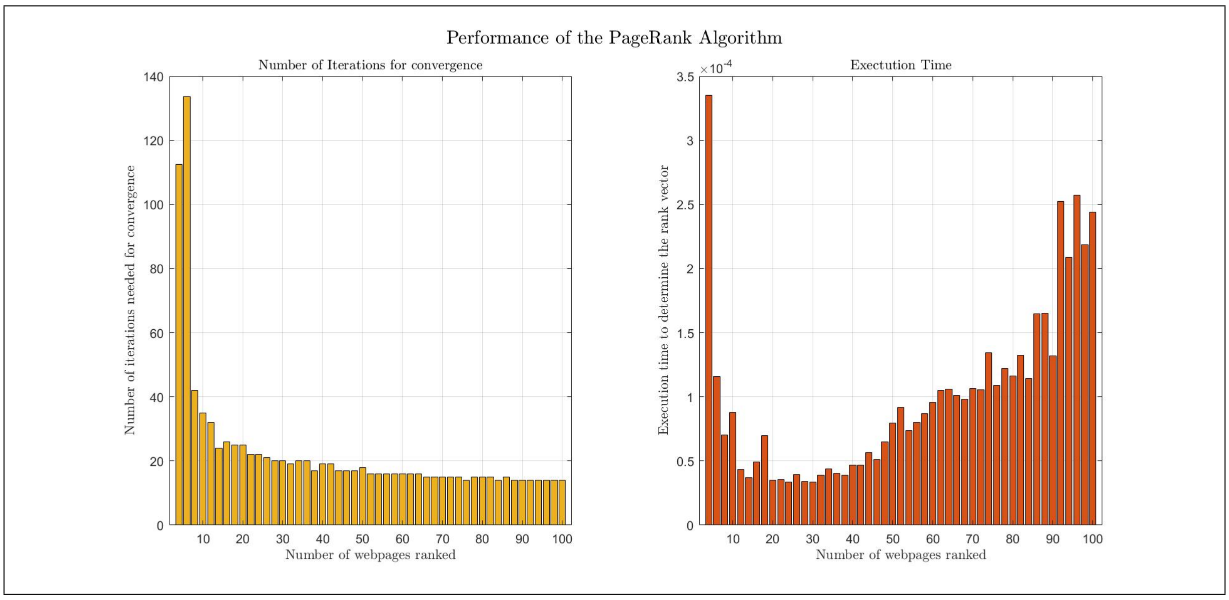


Figure 9: Rank Vector, Number of Iterations and Execution Time

- As we increase the number of pages, the number of iterations needed for convergence reduce but the execution time increases as the time required to perform the matrix multiplication increases.
- The execution time for smaller pages is higher as these pages ranked first. This is an **anomaly** that is observed due to the **inline** property of MATLAB. The execution time of a function that is called multiple times is much higher during the first few calls.
- From the trend of the execution time, one can verify that the time complexity of the algorithm is $O(n)$.
- We observe that the time required to rank 100 pages is in the order of 10^{-4} s. It can also be shown that the execution time is of the order 10^{-3} s when the number of pages is of the order 10^3.
- Thus, we can conclude that the power iteration method to determine the ranks of pages is highly effective to solve the equation obtained by the Google PageRank algorithm due to its linear time complexity.

7 References

[1] Larry Page, Sergey Brin, ed. 1998, ”The PageRank Citation Ranking: Bringing Order to the Web”.
http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf.
[2] Raluca Tanase, Remus Radu. “PageRank Algorithm - the Mathematics of Google Search.” Cornell.edu. N.p., n.d. Web. 8 May 2021
http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html
[3] Tiefengeist. 2019. “A Simplified Implementation of PageRank - Tiefengeist - Medium.” Medium. July 9, 2019.
https://medium.com/@sahirnambiar/a-simplified-implementation-of-pagerank-b8b5d282dc42
[4] “Use PageRank Algorithm to Rank Websites.” MathWorks,
https://in.mathworks.com/help/matlab/math/use-page-rank-algorithm-to-rank-websites.html
[5] “How Does Pageranking Algorithm Deal with Webpage without Outbound Links?” n.d. Stackoverflow.Com. Accessed May 8, 2021. https://stackoverflow.com/questions/21507375/how-does-pageranking-algorithm-deal-with-webpage-without-outbound-links
[5] Honda, Shion. 2021. “PageRank Explained: Theory, Algorithm, and Some Experiments.” Hippocampus-Garden.Com(blog).ShionHonda.May8,2021.https://hippocampus-garden.com/pagerank/

[7] Moler, Cleave.2011. “Google PageRank”, Mathworks.Com. Accessed May 8, 2021.
https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/moler/exm/chapters/pagerank.pdf
[8] Amine, Amrani. 2020. “PageRank Algorithm, Fully Explained.” Towards Data Science. December 19, 2020.
https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af
[9] Artificial Intelligence-All in one, “Mining Massive Datasets - Stanford University”, YouTube Playlist, June 17, 2018.
https://www.youtube.com/playlist?list=PLLssT5z_DsK9JDLcT8T62VtzwyW9LNepV