*PageRank Algorithm and MapReduce*

## 1 Introduction

PageRank is a widely used algorithm for ranking the importance of nodes in a network, initially developed by Google for ranking web pages in their search engine results. The algorithm assigns a numerical score to each node in the network based on the concept of a random surfer model, where a hypothetical surfer randomly clicks on links, with a certain probability of jumping to a random page. The PageRank score of a node represents the likelihood that the random surfer will visit that node, with higher scores indicating greater importance or centrality within the network [1].

This report presents an implementation of the PageRank algorithm using various techniques, including matrix operations, iterative methods, and the MapReduce programming model with PySpark. The report explores the impact of different parameters, such as the damping factor and the jump matrix, on the resulting PageRank scores and convergence behavior. Additionally, numerical examples and experiments are conducted on small and large-scale networks to illustrate the algorithm's performance and interpret the results.

## 2 Methodology

### *2.1 PageRank Implementation*

Our PageRank implementation involves several key steps:

1. Creating Directed Graph: The input graph is converted to a directed graph, relabeling nodes, and removing isolated nodes if any exist.
2. Building PageRank Matrix: The PageRank matrix for the given graph and damping factor is constructed. The damping factor represents the probability that a random surfer will continue surfing rather than jumping to a random page.
3. Running PageRank Algorithm: The PageRank algorithm is performed using the random walk method. It iteratively updates the PageRank scores until convergence or the maximum number of iterations is reached.
4. Plotting Graph and Convergence: The convergence plot is generated to visualize the convergence of the algorithm. Additionally, the graph can be plotted with optional node coloring based on PageRank scores.

### *2.2 Tuning Problem Parameters*

The PageRank algorithm has two main parameters that can be tuned:

1. Damping Factor: This parameter represents the probability of a random surfer following an outgoing link from the current page. A higher damping factor means the surfer is more likely to follow links, while a lower value means the surfer is more likely to jump to a random page. We perform experiments with damping factors ranging from 0.8 to 0.99.

2. Random Jump Matrix: This matrix represents the probability of transitioning from one page to another in the case of a random jump. We perform experiments with different random jump matrices: uniform distribution-based, in-degree-based and out-degree-based.

### *2.3 Numerical Examples and Experiments*

To illustrate the PageRank algorithm, numerical examples and experiments were conducted using Python and the NetworkX library. A small example of a four-webpages graph, from the Lecture Notes, was initially evaluated, and then larger graphs, such as the football network, were analyzed. The PageRank scores were computed for different damping factors ranging from 0.8 to 0.99, and the jump matrices were varied.

### *2.4 Interpretation of PageRank Results*

The PageRank scores represent the relative importance of each node in the network. In this study, we interpret each node as a website. Nodes with higher PageRank scores are considered more important or influential within the network. In the results obtained in the next section, the node with the highest PageRank score tends to receive more attention or traffic from other nodes in the network.

### *2.5 PageRank Equations*

$$\mathbf{P} = d\mathbf{S} + (1-d)\mathbf{R} \tag{1}$$

where:

- $\mathbf{P}$ is the PageRank matrix
- $d$ is the damping factor
- $\mathbf{S}$ is the stochastic matrix, which is calculated as:

$$\mathbf{S} = \mathbf{T} + \mathbf{A} \tag{2}$$

  - $\mathbf{T}$ is the transition matrix, obtained by normalizing the adjacency matrix by the sum of each row
  - $\mathbf{A}$ is the absorbing node matrix, which handles nodes with no outgoing edges (absorbing nodes) by distributing their probabilities equally among all nodes

- $\mathbf{R}$ is the random jump matrix, which is an $n \times n$ matrix with all elements equal to $\frac{1}{n}$, where $n$ is the number of nodes in the graph

The transition matrix $\mathbf{T}$ is calculated as:

$$\mathbf{T} = \frac{\mathbf{M}}{\sum_{j=1}^{n} \mathbf{M}_{ij}} \tag{3}$$

where $\mathbf{M}$ is the adjacency matrix of the graph.
The absorbing node matrix $\mathbf{A}$ is calculated as:

$$\mathbf{A}_{ij} = \begin{cases} \frac{1}{n} & \text{if } \sum_{k=1}^{n} \mathbf{M}_{kj} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The random jump matrix $\mathbf{R}$ is defined as:

$$\mathbf{R}_{ij} = \frac{1}{n} \quad \forall i, j \tag{5}$$

### *2.6 Validation of Convergence*

The code validates the convergence of the PageRank algorithm by checking if the L2 norm of the difference between successive iterations falls below a small threshold, $1 \times 10^{-8}$, the default set by `np.isclose()`. If the norm is sufficiently small, the algorithm is considered to have converged, and the final PageRank scores are returned.

## 3 Results and Discussion

### *3.1 Adjusting the Damping Factor*

| Damping Factor | No. of iterations until converged | Gini Index | PageRank scores (lowest node index to highest node index) |
|---|---|---|---|
| **0.80** | 21 | 0.4054 | [0.10135254 0.12838011 0.64188725 0.12838011] |
| **0.85** | 24 | 0.4674 | [0.0824943 0.10586789 0.70576993 0.10586789] |
| **0.90** | 28 | 0.5421 | [0.06024197 0.07831471 0.78312861 0.07831471] |
| **0.95** | 34 | 0.6340 | [0.033371 0.04393857 0.87875186 0.04393857] |
| **0.99** | 43 | 0.7244 | [0.00731758 0.0097324 0.97321763 0.0097324 ] |

Table 1: Numerical analysis for various damping factors based on 4 websites

| Damping Factor | No. of iterations until converged | Gini Index |
|---|---|---|
| **0.80** | 17 | 0.0317 |
| **0.85** | 20 | 0.0337 |
| **0.90** | 25 | 0.0358 |
| **0.95** | 32 | 0.0379 |
| **0.99** | 40 | 0.0395 |

Table 2: Numerical analysis for various damping factors based on 115 websites (PageRank scores omitted for brevity)

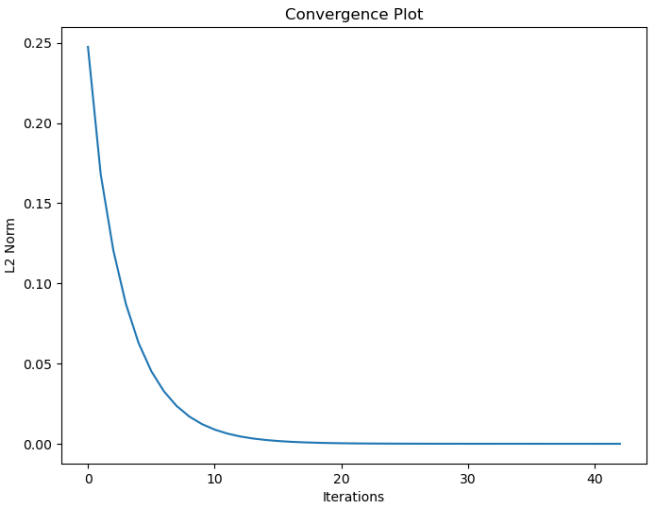


Fig 3.1: Convergence plot for 4-website PageRank with damping factor = 0.99

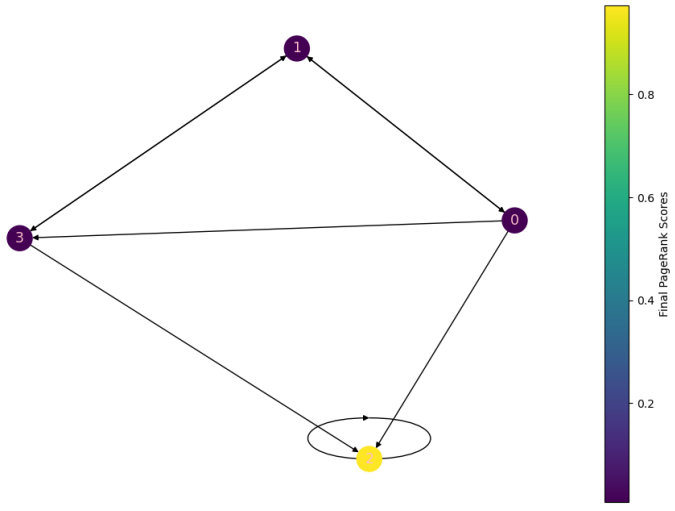


Fig 3.2: Graph representation for 4-website PageRank with damping factor = 0.99

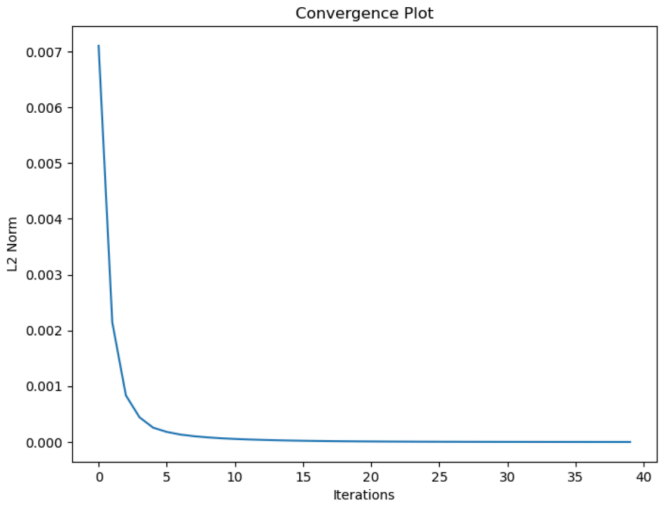


Fig 3.1: Convergence plot for 115-website PageRank with damping factor = 0.99

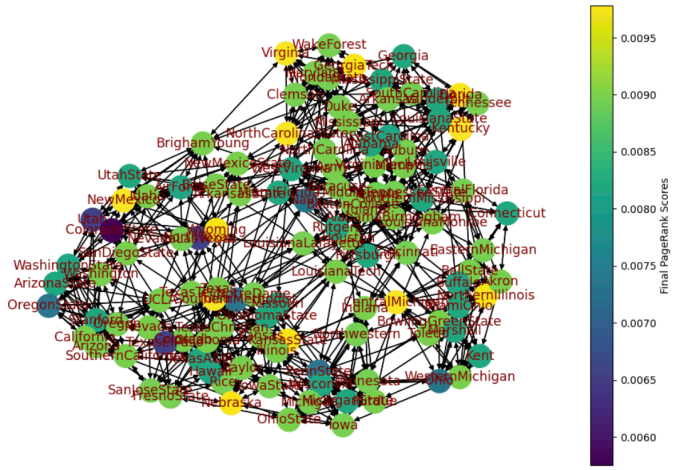


Fig 3.2: Graph representation for 115-website PageRank with damping factor = 0.99

The results in Table 1 and 2 show the PageRank scores obtained for a small graph consisting of four nodes (website) and eight edges, and a large graph consisting of 115 nodes and 1226 edges respectively (code). The number of iterations until convergence and the resulting PageRank scores are provided for different damping factors ranging from 0.8 to 0.99. A convergence plot and graph representation of the four websites, done with a damping factor of 0.99, are shown in Fig 3.1-3.2 respectively. Likewise, Fig 3.3 - 3.4 depict that for the 115 website graph.

As the damping factor increases, the PageRank scores tend to concentrate more on a smaller subset of nodes, leading to a higher Gini index due to greater inequality. This is evident from the fact that as the damping factor increases, the PageRank scores of some nodes decrease while others increase. Furthermore, with a higher damping factor, the convergence tends to occur slower, as seen from the increasing number of iterations required for convergence as the damping factor increases.

Lower damping factors (e.g., 0.8) result in more evenly distributed PageRank scores among nodes, whereas higher damping factors (e.g., 0.99) lead to highly concentrated PageRank scores, with one or a few nodes dominating the scores. This illustrates the importance of choosing an appropriate damping factor based on the specific characteristics of the network being analyzed.

These results demonstrate the sensitivity of PageRank scores to the damping factor and highlight the importance of parameter tuning for achieving meaningful results in PageRank analysis. Additionally, the convergence behavior of the PageRank algorithm provides insights into the stability and efficiency of the computation process.

### *3.2 Adjusting the Jump Matrix*

| Jump Matrix Type | No. of iterations until converged | Gini Index | PageRank scores (lowest node index to highest node index) |
|---|---|---|---|
| **Traditional (Uniform distribution)** | 24 | 0.4674 | [0.0824943 0.10586789 0.70576993 0.10586789] |

| | | | |
|---|---|---|---|
| **Weighted distribution based on in-degree** | 20 | 0.2555 | [0.12822964 0.20141637 0.46893762 0.20141637] |
| **Weighted distribution based on out-degree** | 20 | 0.2555 | [0.46893762 0.20141637 0.12822964 0.20141637] |

Table 3: Numerical analysis for various jump matrices based on 4 websites

| Jump Matrix Type | No. of iterations until converged | Gini Index |
|---|---|---|
| **Traditional (Uniform distribution)** | 20 | 0.0337 |
| **Weighted distribution based on in-degree** | 9 | 0.0019 |
| **Weighted distribution based on out-degree** | 9 | 0.0019 |

Table 4: Numerical analysis for various jump matrices based on 115 websites (PageRank scores omitted for brevity)

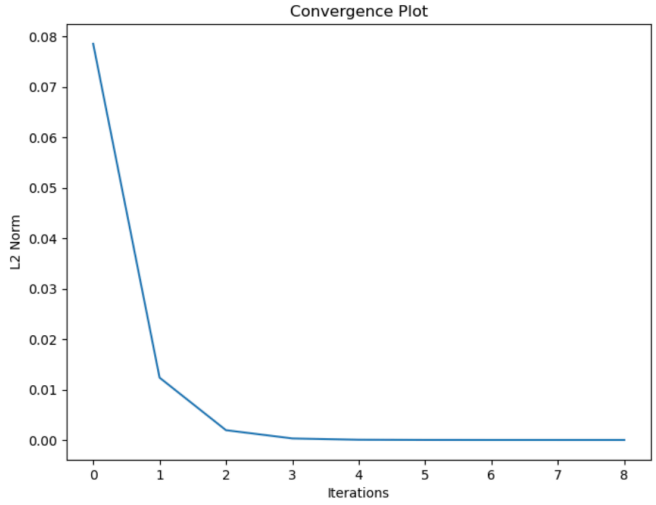


Fig 3.1: Convergence plot for 115-website PageRank with jump matrix based on weighted in-degree

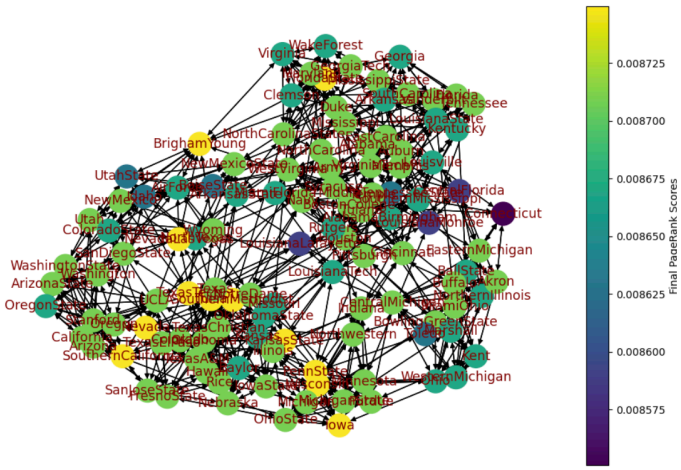


Fig 3.2: Graph representation for 115-website PageRank with jump matrix based on weighted in-degree

In this numerical experiment, we conducted experiments on the same two distinct graphs, with a damping factor of 0.85 (code). Across both experiments, we investigated the influence of different jump matrix types, namely traditional (uniform distribution), in-degree-based, and out-degree-based, on the final PageRank scores and network centrality metrics. The convergence results and Gini coefficients are reported in Tables 3-4.

In the case of the smaller graph, all three jump matrix types led to convergence, albeit with varying numbers of iterations. With the traditional jump matrix, which assigns equal probabilities for transitioning between nodes, the resulting PageRank scores exhibited a balanced distribution. However, nodes with higher in-degree or out-degree centrality received proportionately higher scores when in-degree or out-degree-based jump matrices were employed, respectively.

Moving to the larger-scale graph, a similar pattern emerged in terms of convergence and distribution of PageRank scores. Despite the increased complexity of the network, convergence was achieved within a reasonable number of iterations for all jump matrix types. Notably, the Gini coefficient, a measure of inequality

in the distribution of PageRank scores, was considerably lower in both the in-degree-based and out-degree-based scenarios compared to the traditional approach. This suggests a more equitable allocation of centrality across nodes when considering their respective degrees.

It is also noteworthy that the Gini coefficient for out-degree-based PageRank scores mirrored that of the in-degree-based approach, indicating a similar level of centrality inequality within the network.

Overall, these experiments underscore the sensitivity of the PageRank algorithm to the choice of jump matrix type. While the traditional approach yields a balanced distribution of scores, in-degree-based and out-degree-based formulations offer insights into the influence of specific node characteristics on network centrality. By tailoring the jump matrix according to the underlying network structure and analytical objectives, practitioners can adjust this jump matrix to suit the desired network dynamics.

### *3.3 PySpark MapReduce Implementation*

The PageRank algorithm is also implemented using the MapReduce programming model with PySpark (code). We run this on the football network because it makes sense to parallelize the computation a large number of nodes [2].

Similar to the earlier method, we preprocess the input graph. Thereafter, we set the initial PageRank scores of all nodes to 1/N, where N is the total number of nodes in the graph. These initial scores are distributed across the cluster using PySpark's parallelization feature. The algorithm then enters an iterative process, where it repeatedly updates the PageRank scores until convergence is reached or a maximum number of iterations is exceeded. Each iteration consists of two phases: the mapper phase and the reducer phase.

In the mapper phase, each node in the graph is processed independently and in parallel. For each node, the mapper function calculates the contribution of the node to its outgoing neighbors. The contribution is determined based on the damping factor and the number of outgoing edges from the node. The mapper emits key-value pairs, where the key is the destination node and the value is a tuple containing the source node and its contribution.

After the mapper phase, the reducer phase begins. In this phase, each node receives the contributions from its incoming neighbors. The reducer function collects these contributions and calculates the new PageRank score for each node by summing the contributions and applying the damping factor. The reducer emits key-value pairs, where the key is the node and the value is the updated PageRank score.

The algorithm compares the new PageRank scores with the previous scores to check for convergence. If the absolute difference between the new and old scores falls below a specified convergence threshold, the algorithm terminates, indicating that the scores have stabilized.

Once the iterations are complete or convergence is reached, the final PageRank scores are collected from the distributed cluster using PySpark's `collectAsMap` action. The integer node IDs are then mapped back to their original labels using the `node_int_dict` dictionary, and the final PageRank scores are returned as a dictionary, where the keys are the node labels and the values are the corresponding PageRank scores.

To visualize the convergence of the algorithm, we also plot the absolute differences between iterations using Matplotlib. This convergence plot helps to analyze how quickly the PageRank scores stabilize over the iterations.

The MapReduce implementation of PageRank allows for efficient and distributed computation of the algorithm. By leveraging the power of PySpark, the PageRank algorithm can be applied to large-scale graphs efficiently.

## 4 Conclusion

The results have shown that the choice of parameters, such as the damping factor and the jump matrix, can significantly influence the resulting PageRank scores and their distribution across the network. Higher damping factors lead to more concentrated PageRank scores, with a few nodes dominating the rankings, while lower damping factors result in a more even distribution of scores. Additionally, the choice of jump matrix, whether uniform, in-degree-based, or out-degree-based, can impact the allocation of centrality within the network, allowing for tailored analyses based on specific network characteristics.

The convergence behavior of the PageRank algorithm has also been explored, with the provided code validating the convergence by monitoring the L2 norm of the difference between successive iterations. The convergence plots generated for different scenarios provide insights into the stability and efficiency of the computation process.

Furthermore, the implementation of the PageRank algorithm using the MapReduce programming model with PySpark demonstrates the potential for efficient and distributed computation, enabling the analysis of large-scale networks with numerous nodes and edges.

Overall, this report has provided a comprehensive exploration of the PageRank algorithm, its implementation, parameter tuning, and interpretation of results. The insights gained from this study can be valuable for researchers, data scientists, and practitioners working with network analysis and ranking problems across various domains.

## 5 References

[1] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," Computer networks and ISDN systems, vol. 30, no. 1-7, pp. 107-117, 1998.

[2] H. Eedi and M. K. Kokkula, "Parallel PageRank Algorithm Using MapReduce," Singapore, 2020: Springer Singapore, in ICDSMLA 2019, pp. 787-795.

## 6 Appendix

All code can be found on: https://github.com/googlercolin/pagerank