21-241 Final Project

argao and evanz

December 3, 2021

1 Introduction

In the current technological era, the first thing most people think of when trying to find an answer to a question they might not know is to "google it." As such, the Google search engine's job is to return the most relevant information to the user after a search. Thus, in order to effectively conduct a search, it is essential that a search engine has a method to rank the importance of results (in this case webpages). Although this concept in theory is pretty simple, there are still a few defining parameters that aren't necessarily easy to rigorize when trying to tackle this issue, including those such as how do we actually assess the importance of a page? And how do we compare the importance of different pages (what makes a page more important than other pages)?

The answer to these questions for Stanford PhD students Sergey Brin and Larry Page was the PageRank algorithm they developed in 1996 (as part of a research project). Brin believed that web pages could be sorted based on something called "link popularity," the idea that a web page was more popular if it had more links to it. Thus, under this assumption, the internet can be visualized as a directed graph with web pages as the nodes and links between them as edges. The way that PageRank works is it calculates the ranking of a webpage based on the number of links coming to and from the graph as well as considering the possibility that the user goes to another page ("teleportation") or stops browsing (damping factor). In this paper, we will be explaining and analyzing each of these factors in terms of how they make up the PageRank algorithm and the linear algebra necessary for the whole process.

2 Mathematical Background

To understand how the PageRank algorithm works, it is important to understand the linear algebra behind it: specifically, Markov Matrices and Random Walks. Suppose that there exists a set S of n states such that $S = \{s_1, s_2, \cdots, s_n\}$. Now suppose that we start at one of these states s_i where $i \in [n]$ and at each step we are forced to move, which can mean either staying in the same state or transitioning to a different state. Let the transition probability be the probability that we move from another state s_j for $j \in [n]$ to our defined state s_i be given by p_{ij}.

This system of states is known as a Markov Chain, from which we can create the Markov Matrix M such that for each of i, j row column in M, $M_{ij} = p_{ij}$. Markov Matrices have the following key properties [4]

- Given n different states, M is an $n \times n$ square matrix
- For all $i, j \in [n]$, $M_{ij} \geq 0$ (no probabilities can be negative)
- $M_{*j} = 1, \ \forall j \in [n]$
- M has maximum eigenvalue $\lambda_{max} = 1$
- The eigenvector x_1 of M that corresponds to $\lambda = 1$ has the property that $Mx_1 = x_1$. x_1 is referred to as the steady state vector of M

Note on bullet point 3, when we consider the moves from state j in our previous example, we get that

$$\sum_{i\in[n]} p_{ij} = 1, \ \forall j \in [n]$$

So far, a lot of what we have discussed has been primarily so called "alphabet soup," so consider the following directed graph example: Suppose you are biking between 3 sites which we will call site A, site B, and site C ranked in order of elevation where A is at the highest elevation and C is the lowest and you can go from any site to any other site. From this scenario we get the following graph:

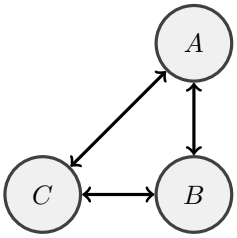


Figure 1: A graph of bike destination sites A, B, C

Consider the case where you start at site A i.e. the starting state is A. From here, there are two options: bike to site B or bike to site C. On each move following the first, we are again left with the choice of biking to the other two sites. This random change of state process is called a random walk. The following random walk as follows can be given by the following Markov matrix (assuming we are equally likely to bike to either site at each point):

$$M = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \end{pmatrix}$$

This matrix satisfies all above properties of a Markov matrix. To find the steady state of this matrix, we multiply by the eigenvector corresponding with $\lambda = 1$ (found as $(1, 1, 1)$ by taking the null space of $M - 1I$) to get

$$Mx_1 = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Thus, the attracting steady state of the matrix M is $(1, 1, 1)$ which we normalize to get the proportion of time the biker spends at each site or state is $(\mathbb{P}(A), \mathbb{P}(B), \mathbb{P}(C)) = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ which intuitively makes sense as a randomly travelling biker will spend roughly the same amount of time at each location.

However, since biking on a flat path is much easier than biking uphill or downhill, realistically we get that it is much easier to bike to and from site B to and from either A or C such that the paths to and from site B are weighted more. In this case of weighted edges, the edges with higher weights should be considered as more probable paths to take meaning the Markov matrix M must take into account this weight in order for the steady state vector to accurately represent the probabilities of being at each site or in each state. The next section will talk about how to implement these weighted edges in relation to the PageRank problem.

3 The PageRank Algorithm

PageRank is a specific application of Markov chains that is used to model a network by assigning a rank to each page in the network. Thus, we can let each page be a node such that the hyperlinks on one page are the edges which connect to other pages in the network [7]. Let matrix K represent one of these matrices such that the rows of K correspond to potential hyperlinked pages and the columns of K correspond to the specific pages themselves. An established link between a page j to a page i is represented by a 1 in the K_{ij} entry in the matrix and no link is represented by a 0 in the K_{ij} entry in the matrix. This however, does not have the properties of a Markov matrix as having a 1 would prevent there from being multiple hyperlinks from a single page. Thus, to consider the number of links n on a page j, H_{ij} is modified such that $H_{ij} = \frac{1}{n}$. This is how we will build our Markov matrix representation of the Internet.

Consider the following network, which instead of biking destinations now uses web pages as nodes and a random web surfer as the thing travelling between nodes (represented by Figure 2). From this we

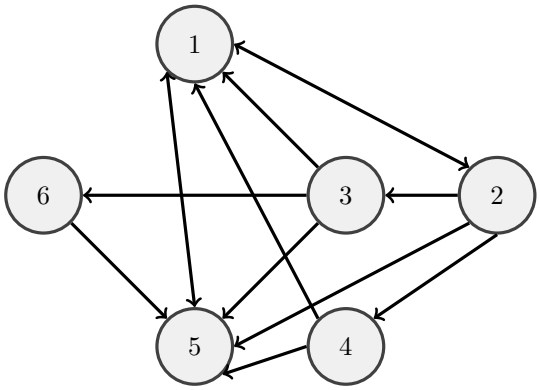


Figure 2: A given network with 6 web pages

construct the following weighted adjacency Markov matrix:

$$K = \begin{pmatrix} 0 & 1/4 & 1/3 & 1/2 & 1 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 & 0 & 0 \\ 1/2 & 1/4 & 1/3 & 1/2 & 0 & 1 \\ 0 & 0 & 1/3 & 0 & 0 & 0 \end{pmatrix}$$

Say for example the surfer is at page 2. From links on this page they have the 4 equally likely choices of going to pages 1, 3, 4, or 5. From here the surfer keeps surfing and the cycle continues such that after many cycles, the probability of the surfer being on any given page is given by the steady state vector. Usually this will happen after 50-100 cycles. Thus, we can calculate steady state vector x_{100} from

$$x_{100} = K^{100} x_0$$

where x_0 is the starting vector. Assuming there is an equally likely chance to start at any of the pages, we get that $x_0 = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$ which when plugged into the formula above yields (which is

later confirmed by our code)

$$x_{100} = (0.359, 0.177, 0.063, 0.063, 0.296, 0.043)$$

The higher the probability, the closer the rank is to the top giving us that the page rank of this network goes in this order: Page 1, Page 5, Page 2, Page 3, Page 4, Page 6.

Thus, we seem to have arrived at a working implementation of PageRank. However, this random walk implementation still has its flaws. Namely, consider the case where a matrix does not have a nice steady state vector. For example, a random walk implementation when coming to a page with no hyperlinks to take it to other pages, runs into an issue. Consider Figure 3 and corresponding matrix K which depict this issue.

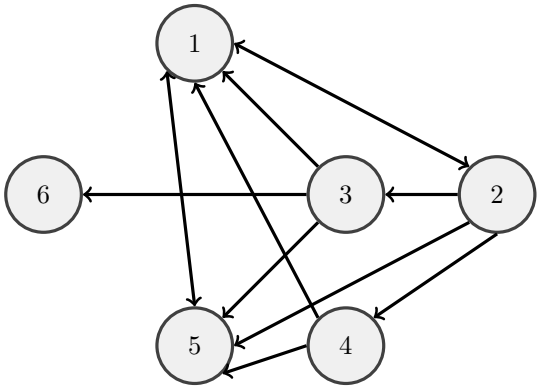


Figure 3: The given 6 page network with dangling node 6

$$K = \begin{pmatrix} 0 & 1/4 & 1/3 & 1/2 & 1 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 & 0 & 0 \\ 1/2 & 1/4 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 0 \end{pmatrix}$$

Terminology-wise, we call a page with no direct hyperlinks to other pages a "dangling node" meaning when we hit such a dangling node, there must be a way to reach another page [3]. One possible way to address this issue is by using a method called "teleportation" which gives every page in the network an equal likelihood of being selected given that the current node is a dangling node (a page with no outward hyperlinks). Applying this to the matrix K, we get a new matrix K_{new} such that

$$K_{new} = \begin{pmatrix} 0 & 1/4 & 1/3 & 1/2 & 1 & 1/6 \\ 1/2 & 0 & 0 & 0 & 0 & 1/6 \\ 0 & 1/4 & 0 & 0 & 0 & 1/6 \\ 0 & 1/4 & 0 & 0 & 0 & 1/6 \\ 1/2 & 1/4 & 1/3 & 1/2 & 0 & 1/6 \\ 0 & 0 & 1/3 & 0 & 0 & 1/6 \end{pmatrix}$$

Thus, to generalize this, we get rid of disconnects caused by dangling nodes by applying this teleportation strategy such that

$$K_{new} = K + v\left(\frac{1}{n}e_1^T\right)$$

where n is the number of nodes or pages in the network, e is a column vector of 1s of length n, and v is a column vector of length n such that $v_i = 1$ if the ith node is a dangling node and 0 if it isn't. From this, we have successfully created a Markov matrix representing the PageRank algorithm, but we are not yet finished.

4 Damping Factor

While the previous algorithm with the adjustment to account for dangling nodes ensures that the matrix is Markov, it is still necessary to ensure the matrix is primitive to converge, i.e. the matrix must be aperiodic and irreducible. Ensuring this allows the steady state of the matrix to be found through powers of the matrix which is what PageRank utilizes. To achieve this, we implement a damping factor, which simultaneously ensures the matrix is primitive as well as providing more accurate modeling of the surfer's behavior. Consider the behavior of the average person surfing the internet. Normally, people don't tend to keep clicking through links on the pages they find, but rather if they want to start a new search query they will redirect via a specific url to any other page in the network. This is where the damping factor comes into play. The damping factor can be expressed as a probability d such that $0 \leq d \leq 1$ such that d is the probability at a given state that the surfer will keep following hyperlinks.
If $d = 0$, then this means the surfer will never follow hyperlinks.
If $d = 1$, then this means the surfer will always follow hyperlinks.
When the surfer teleports, we know that they are equally likely from any node to arrive at any other node. This behavior can be represented by

$$\frac{1}{n}ee^T = \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{pmatrix}$$

Now, when we consider the damping factor, we utilize properties of probabilities to get a new matrix H that incorporates the damping factor such that

$$H = \frac{1-d}{n}ee^T + dK_{new}$$

which when used with damping factor $d = 0.85$ for the above network (Figure 3) gives us

$$H = 0.15\begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{pmatrix} + 0.85\begin{pmatrix} 0 & 1/4 & 1/3 & 1/2 & 1 & 1/6 \\ 1/2 & 0 & 0 & 0 & 0 & 1/6 \\ 0 & 1/4 & 0 & 0 & 0 & 1/6 \\ 0 & 1/4 & 0 & 0 & 0 & 1/6 \\ 1/2 & 1/4 & 1/3 & 1/2 & 0 & 1/6 \\ 0 & 0 & 1/3 & 0 & 0 & 1/6 \end{pmatrix}$$

$$\implies H = \begin{pmatrix} 0.025 & 0.2375 & 0.30833 & 0.45 & 0.875 & 0.16667 \\ 0.45 & 0.025 & 0.025 & 0.025 & 0.025 & 0.16667 \\ 0.025 & 0.2375 & 0.025 & 0.025 & 0.025 & 0.16667 \\ 0.025 & 0.2375 & 0.025 & 0.025 & 0.025 & 0.16667 \\ 0.45 & 0.2375 & 0.30833 & 0.45 & 0.025 & 0.16667 \\ 0.025 & 0.025 & 0.30833 & 0.025 & 0.025 & 0.16667 \end{pmatrix}$$

giving us a matrix H that is Markov, irreducible, aperiodic, and primitive. This will allow us to find the ranks for each page when the PageRank vector values converge (at k = 100 presumably). With an understanding of how the PageRank algorithm works we can now take a deeper look at the implementation and the results of application of the implementation.

5 Pseudocode

5.1 Parsing the File and Teleportation

Note that the custom input file is created as follows:

- Row 1: an integer which represents the number of nodes in the graph.
- Every subsequent row: a pair of integers, the first of which represents the "fromId" and the second of which represents the "toId" of an edge. There is a directed connection from node "fromId" to node "toId".

The file is represented by FILE in the pseudocode [5] below.

```
INIT DATA as lines from FILE
INIT numNodes as first row of DATA
INIT ADJ as adjacency matrix with dimension (numNodes x numNodes)
INIT hasOutNode as numNodes-element matrix
FOR each subsequent row in DATA
    INIT fromId to first number in DATA[i]
    INIT toId to second number in DATA[i]
    SET ADJ position (toId, fromId) to exists (1)
    IF hasOutNode position (fromId) is dangling (0) THEN
        SET hasOutNode position (fromId) is not dangling (1)
    END IF
END FOR
```

With the numNodes-element matrix hasOutNode, we can implement teleportation, i.e. dangling nodes link to every other node.

```
FOR every element with position i in hasOutNode
    IF the element is equal to 0 (dangling) THEN
        SET every element in the ith column of ADJ to 1
    END IF
END FOR
```

Finally, we want to return two important pieces of information.

```
RETURN ADJ, numNodes
```

Combining these three pieces of pseudocode gives us the pseudocode necessary for writing the **parse_file** function (see Appendix B Section 11.2).

5.2 Implementing the PageRank Algorithm

Define the following variables:

FILE: input file

ITER: number of iterations

DAMP: dampening factor

```
SET ADJ, numNodes to parse_file(FILE)
NORMALIZE each column in ADJ by dividing its sum
INIT V as numNodes-element matrix of random values
NORMALIZE V using the 1-norm
SET A_hat to PAGE_RANK_ALGORITHM
FOR ITER number of iterations
    SET V to A_hat times V
END FOR
```

PAGE_RANK_ALGORITHM is defined as the mathematical equation

$$\frac{1 - \text{DAMP}}{\text{numNodes}} + \text{numNodes} \cdot \text{ADJ}.$$

which corresponds to our algorithm from before with slightly different variable names in code

$$H = \frac{1-d}{n} ee^T + dK_{new}$$

See Appendix B Section 11.3 for the PageRank algorithm written in Julia.

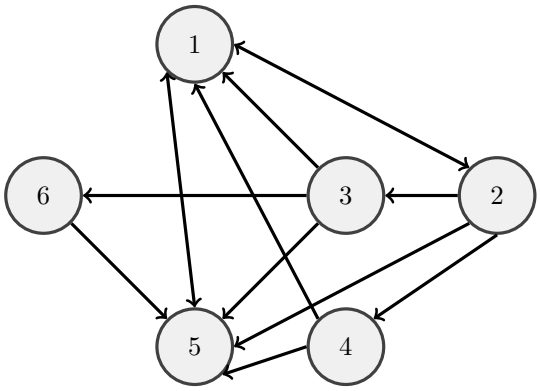
6 Extension - HITS

So far, our PageRank algorithm seems to be an effective standard method to sort the relative importance of pages based on the number of hyperlinks that point to it. This implementation however, is far from perfect as the idea of "importance" of a page is far more nuanced than the number of hyperlinks between pages. Thus, for our extension, we decided to learn about and implement the HITS (Hyperlink-Induced Topic Search) algorithm, a variation of the PageRank algorithm that stems from the idea that important pages can be categorized where certain important pages called "hubs" serve the purpose of pointing the user to other hyperlinks with authoritative information rather than directly providing it [6]. As such, a good hub is a webpage that has hyperlinks to many different (authoritative) pages and a good authority is a webpage that is linked to by many different hubs. By this method, each page has two attributes [1]:

- Authority Score: A node is high-quality if many high-quality nodes link to it.
- Hub Score: A node is high-quality if it links to many high-quality nodes.

Similar to PageRank, both the authority and hub values quickly converge when using an iterative process.

We can hypothesize that authority scores and PageRank scores are directly correlated, i.e. high authority scores in HITS will lead to higher PageRank scores since both authority and PageRank are based on how many other pages connect to them and the relative importance of the pages that connect to them. Consider, for example, our sample network from before (Figure 2):



As seen by the structure of the network, we can infer that the top hubs of this network are pages 2 and 3 because they have the most hyperlinks to other pages (specifically authoritative ones such as 1 and 5). Additionally, we can infer that the top authorities of this network are pages 1 and 5 because they have the most hyperlinks pointing to them (specifically hub links from pages 2 and 3). Page 4 should also be of relatively high authority as it is a non-hub page that is linked to by a hub. We will be comparing these inferences to the data we collect from running the HITS implementation.

6.1 HITS in Mathematical Notation

The mathematical background for understanding HITS is much less than one needs to understand PageRank; it only uses summations and norms. Let an individual page be P_i where $i \in \mathbb{N}$, let the authority score be P_{ia}, and let the hub scores be P_{ih}. Then for each iteration we compute

$$P_{ia} = \sum_{j \in P_{\text{to}}} P_{jh}$$

$$P_{ih} = \sum_{j \in P_{\text{from}}} P_{ja}$$

where P_{to} represents a list of all pages which link *to* page P and P_{from} represents a list of all pages which have a link *from* page P.

6.2 Pseudocode

Define the following variables:

FILE: input file

ITER: number of iterations

```
SET ADJ, numNodes to parse_file(FILE)
INIT scoresData to matrix of 1s with dimension (numNodes x 2)
FOR ITER number of iterations
    INIT NORM as 0
    FOR every node i
        INIT AUTH as 0
        FOR every node j
            IF there exists a connection from j to i (using ADJ) THEN
                SET AUTH to AUTH plus hub score of jth node
            END IF
        END FOR
        SET NORM to NORM plus AUTH squared
        SET auth score of ith node to AUTH
    END FOR
    SET NORM to square root of NORM
    NORMALIZE first column of scoresData using NORM

    SET NORM to 0
    FOR every node i
        INIT HUB as 0
        FOR every node j
            IF there exists a connection from i to j (using ADJ) THEN
                SET HUB to HUB plus auth score of jth node
            END IF
        END FOR
        SET NORM to NORM plus HUB squared
        SET hub score of ith node to HUB
    END FOR
    SET NORM to square root of NORM
    NORMALIZE second column of scoresData using NORM
END FOR
```

See Appendix B Section 11.4 for the HITS algorithm written in Julia.

7 Results and Data

When analyzing our data, we want to focus on a couple of key points:

- Which nodes have the largest PageRank, authority, and hub scores? Why?
- How fast do PageRank, authority, and hub scores converge?
- Is there correlation between PageRank, authority, and hub scores? If so, in what networks is the correlation most evident?

By default, we initialize the number of iterations for the PageRank and HITS algorithm to be 10, and we initialize the dampening factor to be 0.85.

7.1 Our Sample Network

Our sample network is used throughout our paper and consists of 6 pages (nodes) and 13 edges (with 2 bidirectional edges).

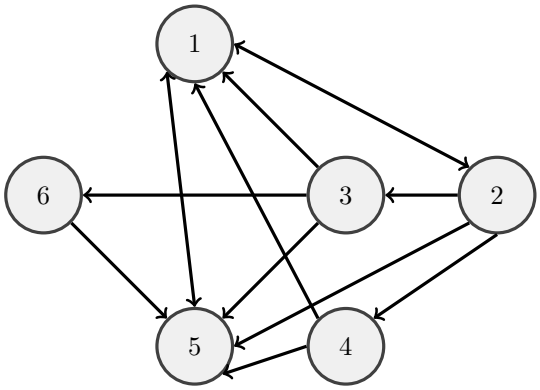


Figure 4: Our Sample Network

Using our PageRank algorithm, we were able to find the following PageRank scores for each page.

Page #	PageRank Score
1	0.359
2	0.177
3	0.063
4	0.063
5	0.296
6	0.043

Figure 5: Table of PageRank scores for our sample network

Page #	PageRank	Authority Rank (HITS)	Hub Rank (HITS)
1	1	2	4
2	3	6	1
3	4	3	2
4	5	4	3
5	2	1	6
6	6	5	5

Figure 6: Table of scores rank for our sample network

These results make sense; many nodes link to Page #1 and Page #5, hence they have high PageRank scores, whereas there is only one page which links to Page #6, hence it has a low PageRank score. We were also able to find the following rankings for the PageRank, authority, and hub scores of each node.

In this sample network, PageRank and Authority Rank seem to have positive correlation with each other, likely due to the fact both scores are related to the quality of the pages which link to it. Furthermore, nodes with many outgoing connections, like Page #2 and Page #3, have a high hub score, which matches the definition of a hub score.

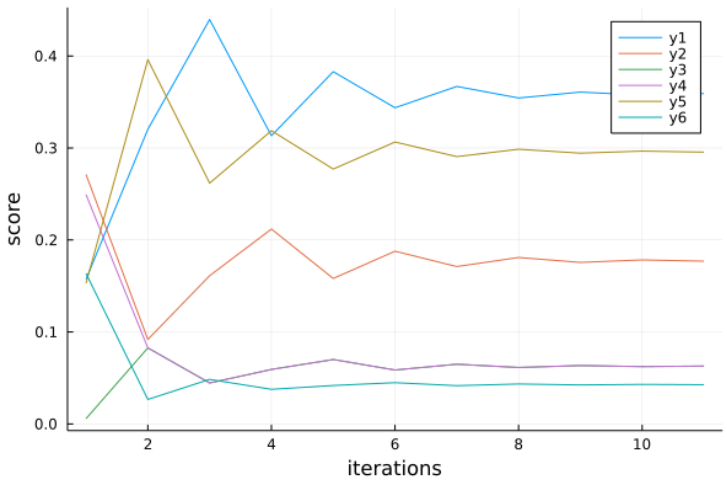


Figure 7: PageRank Convergence

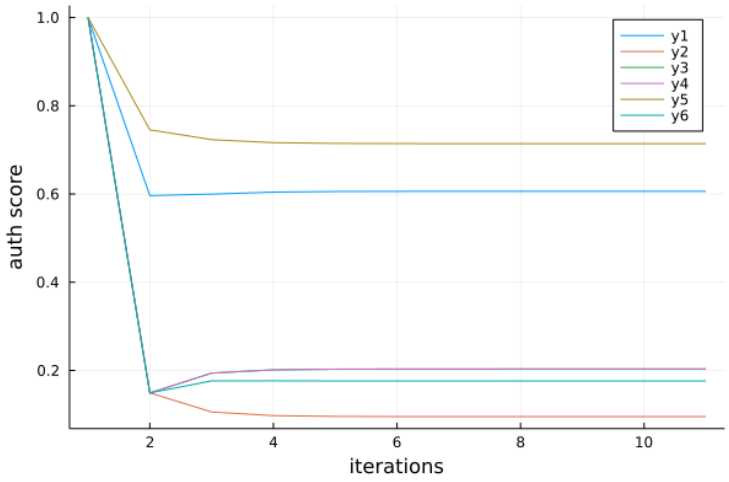


Figure 8: Authority Scores Convergence

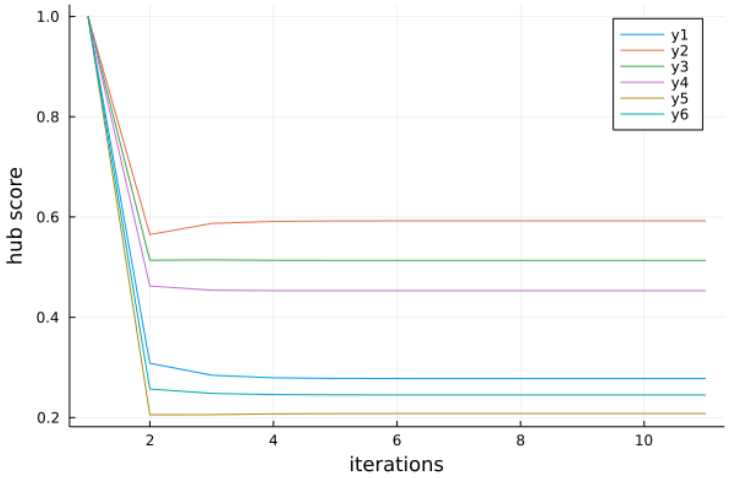


Figure 9: Hub Scores Convergence

PageRank seems to converge in around 8 iterations, whereas the HITS algorithm converges almost instantly in around 3 iterations. This fast convergence is mostly due to how small our network is.

7.2 Wikipedia Sample Network

Our "wiki" sample network consists of 11 nodes and 17 edges. The image below is taken directly from the PageRank Wikipedia page and denotes pages alphabetically. Let Page A be Page #1, Page B be Page #2, and so on.

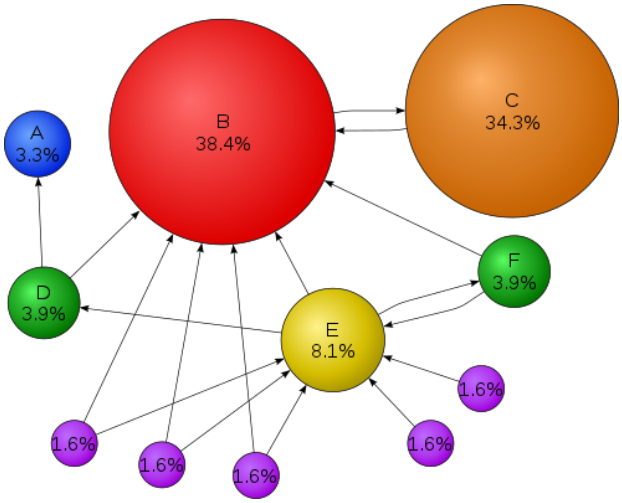


Figure 10: Wiki Sample Network [2]

Once again, using our PageRank algorithm, we were able to find the following PageRank scores for each page.

Page #	PageRank Score
1	0.033
2	0.384
3	0.342
4	0.039
5	0.081
6	0.039
7	0.016
8	0.016
9	0.016
10	0.016
11	0.016

Figure 11: Table of PageRank scores for wiki sample network

This table of data matches the PageRank scores seen in the image from Wikipedia.

It's obvious that Page #2 should have the highest PageRank score; 7 different pages link to it. Interestingly, however, Page #3 has the second highest PageRank score even though it only has one connection. This seems to be due to the fact that the one page which links to it, Page #2, has a very high PageRank score, and additionally, Page #2 only links to Page #3. As a result, if random surfers of the web click on Page #2 (which is very likely), then they must click on Page #3. We were also able to find the following rankings for the PageRank, authority, and hub scores of each node.

Page #	PageRank	Authority Rank (HITS)	Hub Rank (HITS)
1	6	5	1
2	1	1	11
3	2	6	8
4	4	3	7
5	3	2	6
6	5	4	2
7	7	7	9
8	8	8	10
9	9	9	3
10	10	10	4
11	11	11	5

Figure 12: Table of scores rank for wiki sample network

Due to *teleportation*, Page #1 is obviously going to have a high hub score since Page #1 connects to every other page. Futhermore, once again, pages with high PageRank scores have high authority scores.

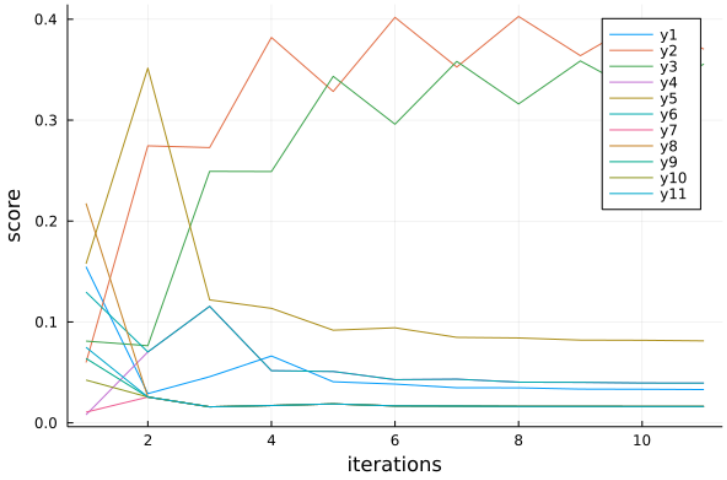


Figure 13: PageRank Convergence

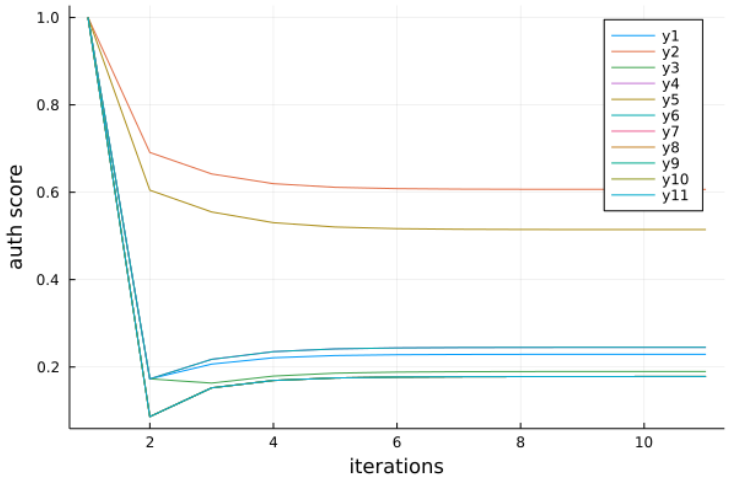


Figure 14: Auth Scores Convergence

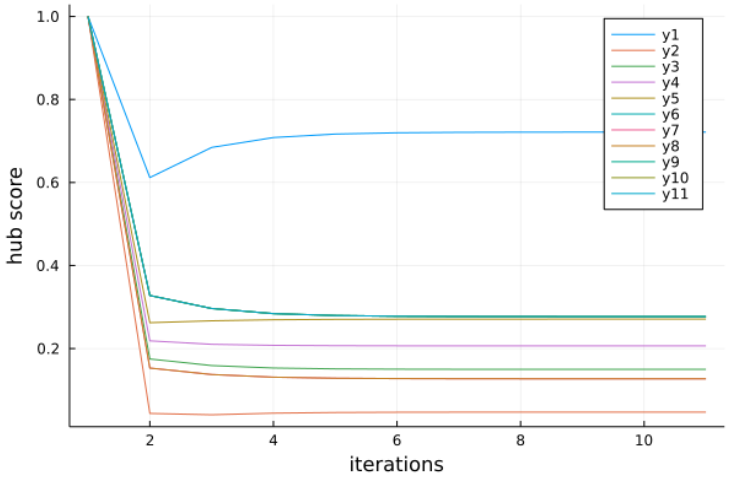


Figure 15: Hub Scores Convergence

Similar to our sample network, PageRank seems to converge in around 9 iterations, whereas the HITS algorithm converges almost instantly in around 3 iterations. This is also likely because this wiki sample network is quite small.

7.3 Random Networks

Define the sparse factor δ as follows: for an arbitrary δ and nodes n, there is on average $\frac{n^2}{\delta}$ edges. Here, we generate two different datasets: the first is a dataset of 100 random networks of 10 pages with sparse

factor $\delta = 1, 2, 4, 9$. The second is a dataset of 100 random networks of 100 pages with sparse factor $\delta = 1, 3, 15, 99$. Then, we extrapolate the PageRank, authority, and hub scores of each network to find the correlation between authority and PageRank scores (ap_c), hub and PageRank scores (hp_c), and authority and hub scores (ah_c).

Sparsity Factor (δ)	ap_c	hp_c	ah_c
1	0.785	-0.070	-0.056
2	0.622	-0.059	0.055
4	0.638	0.060	-0.035
9	0.731	0.090	0.051

Figure 16: Correlation of PageRank, authority, and hub scores for 10-page network

Sparsity Factor (δ)	ap_c	hp_c	ah_c
1	0.973	0.007	0.002
3	0.915	0.035	0.041
15	0.685	-0.004	0.002
99	0.757	0.030	0.026

Figure 17: Correlation of PageRank, authority, and hub scores for 100-page network

It is evident that the correlation between hub and PageRank scores and the authority and hub scores are not significant as the correlation is approximately 0. On the other hand, there is strong correlation between authority and PageRank scores. This correlation seems to be highest when $\delta = 1$ or, in other words, when sparsity is the highest. The correlation also seems to increase as the size of the network increases. We can conclude that our hypothesis that there should be strong correlation between authority and PageRank scores (stated in the HITS section) is correct and is most prevalent in larger, dense graphs.

8 Conclusion

In this paper, we discussed the PageRank algorithm and a variation to the PageRank algorithm, HITS. We then used both algorithms to generate data on specially created and randomly created networks and then analyzed the scores, speed of convergence, and correlation of scores. There are many things we could do in the future:

- We could change some the parameters used in this project (increase number of iterations, choose different dampening factor, etc.).
- We could try to be more space efficient by implementing our code via an adjacency list instead of an adjacency matrix. We would probably have to use the power method of computing PageRank if we were to represent the data using an adjacency list instead.
- We could analyze the efficiency / speed of our code with larger datasets and perform rigorous time complexity analysis.

The world of linear algebra is vast, and with it comes endless possibilities. We just explored one (maybe two) possibilities, but there are so many more that this class has opened our eyes to. Thank you for a great year :)

9 Appendix A

We provide the input files we used below:

9.1 Our Sample Network

```
6
1 2
1 5
2 1
2 5
2 3
2 4
3 1
3 5
3 6
4 1
4 5
5 1
6 5
```

9.2 Wiki Sample Network

```
11
2 3
3 2
4 1
4 2
5 4
5 2
5 6
6 2
6 5
7 5
8 5
9 5
9 2
10 5
10 2
11 5
11 2
```

10 Appendix B

All code is written under one Julia file, **pagerank.jl**. The code below includes the code for plotting and saving figures. Randomly generated data can be found on the Github repository for this project found here: `https://github.com/IronicNinja/21241-f21`.

10.1 Importing the Packages

```
using LinearAlgebra
using Random
using StatsBase
using Statistics
using Plots
```

10.2 Parsing the File and Teleportation

```
function parse_file(file)
    """
    INPUT: file name; the file is initialized
        such that the 1st row is the number of
        nodes and each subsequent node
        has a fromId and toId
    OUTPUT: adjacency matrix, number of nodes
    - Includes teleportation
    """
    data = readlines(file) # read all lines from file
    numNodes = parse(Int32, data[1]) # extract number of nodes
    A = fill(float(0), (numNodes, numNodes)) # initialize adjacency matrix
    hasOutNode = fill(0, numNodes) # initialize "visited" matrix
    for i in 2:(size(data)[1])
        edge = split(data[i])
        fromId = parse(Int32, edge[1])
        toId = parse(Int32, edge[2])
        A[toId, fromId] = 1 # fromId is the column, toId is the row
        if hasOutNode[fromId] == 0
            hasOutNode[fromId] = 1 # update "visited" matrix
        end
    end

    # Teleportation Implementation
    for i in 1:numNodes
        if hasOutNode[i] == 0
            A[:, i] .= 1 # visits all other nodes
        end
    end

    return A, numNodes
end
```

10.3 Implementing the PageRank Algorithm

```
function pagerank(file, iter, damp, plotting=false)
    """
```

```
    INPUT: file name (STR), number of iterations (INT), damping factor (FLOAT)
    OUTPUT: An array A where the ith index in the array is A[i]th most important
    - Uses the iterative method
    """
    A, n = parse_file(file)
    A  = A ./ sum(A, dims=1) # normalize each column in the matrix
    v = rand!(zeros(n)) # create a n-element matrix of zeros
    v /= norm(v, 1) # use 1-norm
    A_hat = (damp * A .+ (1 - damp) / n) # pagerank algorithm
    plotArray = fill(float(0), (iter+1, n)) # initialize plot array
    plotArray[1, :] = v # start with initial
    for x in 1:iter
        v = A_hat * v
        plotArray[x+1, :] = v
    end

    #print("Page rank scores are: ", v, "\n")
    if plotting
        savefig(plot(1:iter+1, plotArray, xlabel="iterations", ylabel="score"), replace(file, ".
    txt" => "-") * string(iter) * "-plot.png")
    end

    return ordinalrank(v, rev=true)
end
```

10.4 Implementing the HITS Algorithm

```
function hits(file, iter, plotting=false)
    """
    INPUT: file name (STR), number of iterations (INT)
    OUTPUT: An array A with the authority scores in the 1st column and the
        hub scores in the 2nd column
    """
    A, n = parse_file(file)
    scoresData = fill(float(1), (n, 2)) # auth, hub; initialized with 1
    authPlotArray = fill(float(1), (iter+1, n)) # initialize auth plot array
    hubPlotArray = fill(float(1), (iter+1, n)) # initialize hub plot array
    for x in 1:iter
        # Update authority values first
        norm = 0
        for i in 1:n
            auth = 0
            for j in 1:n
                if A[i, j] == 1
                    auth += scoresData[j, 2]
                end
            end
            norm += (auth * auth)
            scoresData[i, 1] = auth
        end
        norm = sqrt(norm)
        scoresData[:, 1] = scoresData[:, 1] ./ norm # update the auth scores
        norm = 0
        for i in 1:n
            # Then update hub values
            hub = 0
            for j in 1:n
                if A[j, i] == 1
```

```
                    hub += scoresData[j, 1]
                end
            end
            norm += (hub * hub)
            scoresData[i, 2] = hub
        end
        norm = sqrt(norm)
        scoresData[:, 2] = scoresData[:, 2] ./ norm # update the hub scores

        authPlotArray[x+1, :] = scoresData[:, 1]
        hubPlotArray[x+1, :] = scoresData[:, 2]
    end

    if plotting
        savefig(plot(1:iter+1, authPlotArray, xlabel="iterations", ylabel="auth score"), replace(file, ".txt" => "-") * string(iter) * "-auth-plot.png")
        savefig(plot(1:iter+1, hubPlotArray, xlabel="iterations", ylabel="hub score"), replace(file, ".txt" => "-") * string(iter) * "-hub-plot.png")
    end

    #display(scoresData)
    #print("\n")
    return ordinalrank(scoresData[:, 1], rev=true), ordinalrank(scoresData[:, 2], rev=true)
end
```

10.5 Generating Random Data

```
function generate_random(file_loc, numNodes, iter, fac=1)
    """
    INPUT: file location (STR), number of nodes (INT), number of iterations (INT), sparsity factor (INT)
    OUTPUT: Writes file which consists of randomly generated adjacency matrix
    """
    for x in 1:iter
        A = rand(append!(zeros(fac), 1), numNodes, numNodes) # random adjacency matrix
        io = open(file_loc*"/i"*string(x)*"n"*string(numNodes)*"f"*string(fac)*".txt", "w") # Create file
        println(io, string(numNodes)) # write number of nodes to file
        for i in 1:numNodes
            for j in 1:numNodes
                if A[i, j] == 1
                    println(io, string(j)*" "*string(i)) # write the adjacency pairing to file
                end
            end
        end
        close(io) # close file
    end
end
```

10.6 Extrapolating Correlations Between Scores

```
function comparison(file)
    """
    INPUT: file directory
    OUTPUT: Correlation scores ap_c, hp_c, and ah_c
    """
    dir = readdir(file)
```

```
    authPlotArray = fill(float(0), size(dir))
    hubPlotArray = fill(float(0), size(dir))
    hitsPlotArray = fill(float(0), size(dir))
    c = 1
    for i in dir
        fileName = file*i # concatenate for file name
        ranks = pagerank(fileName, 10, 0.85)
        authScores, hubScores = hits(fileName, 10)
        authPlotArray[c] = cor(ranks, authScores)
        hubPlotArray[c] = cor(ranks, hubScores)
        hitsPlotArray[c] = cor(authScores, hubScores)
        c += 1
    end

    return authPlotArray, hubPlotArray, hitsPlotArray
end
```

References

[1] Chonyy. HITS Algorithm: Link Analysis Explanation and Python Implementation from Scratch. Jan. 2021. URL: https://towardsdatascience.com/hits-algorithm-link-analysis-explanation-and-python-implementation-61f0762fd7cf.

[2] HITS algorithm. Feb. 2021. URL: https://en.wikipedia.org/wiki/HITS_algorithm.

[3] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. URL: http://www.cs.cornell.edu/home/kleinber/auth.pdf.

[4] Oliver Knill. Markov Matrices. URL: https://people.math.harvard.edu/~knill/teaching/math19b%5C_2011/handouts/lecture33.pdf.

[5] Sara A. Metwalli. Pseudocode 101: An Introduction to Writing Good Pseudocode. June 2021. URL: https://towardsdatascience.com/pseudocode-101-an-introduction-to-writing-good-pseudocode-1331cb855be7.

[6] Remus Radu Raluca Tanase. URL: http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/lecture4.html.

[7] Christiane Rousseau. "How Google Works". In: (Aug. 2010). URL: http://dmuw.zum.de/images/f/f8/Google%5C_klein%5C_2.pdf.