Contents

1 Introduction to Networks

The idea of a network is fairly common in today's world, where the average person spends a decent bit of time on the computer and the Internet each day. The Internet is a commonly known giant network, but when one moves to define the word "network" it may get confusing. To clarify, a network is simply a collection of vertices joined by edges, if one were to look at it as a graph (Figure 1 and 2). The edges can either connect two different vertices, or connect one vertex to itself (a self-loop), and there can be more than one edge shared between any two given vertices. The basic properties of a network can be described by the number of vertices and the number of edges in the network, but we can go much deeper than this as we will soon see.

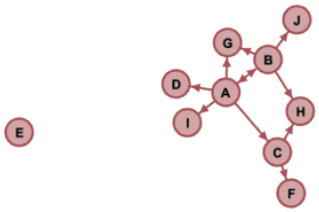


Figure 1: Directed Network

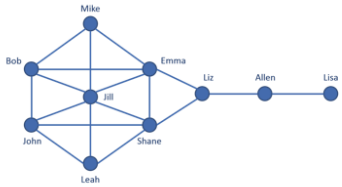


Figure 2: Undirected Network

Networks can vary drastically in complexity as well as what they represent. The Internet is about as complex of a network as there can be; while the network of relations to one's immediate family would be much simpler. Figures 1 and 2 are examples of basic networks. For now ignore the appearance of the arrows in the picture, they will be explained in the next section. Figure 1 is a simple example showing the vertices labeled as letters, which are connected to one another depending if they have a line between them. You can see all of the letters have some sort of line of connections in common except vertex E, we will learn about what the values of these connections are later on in the project. Figure 2 above attributes the vertices in the network to names of people; think of these as friends in a social circle. Bob has a connection to Emma who has a connection to Liz, so theoretically Bob could get a message through to Liz if he wanted to, he could even get it all the way to Lisa. These are the ways that more complex networks like the Internet, or even a biological network inside of an organism, work in a nutshell; there are many small connections between items that can be traced to send longer "messages".

To fully understand how beneficial different types of networks can be to us in the modern day, we must look at the history of computer networks and the Internet. In the 1940s, the first time a computing machine was used remotely when an electromechanical

typewriter was able to send commands to a computer. This was a huge technological leap, and showed there was great potential in remote connections between computers. This quickly evolved and in the early 1950s, the military implemented remote connections between computers in their radar system called SAGE, and airlines followed soon after so that they could get information to any airplane in 3 seconds. In 1969, there were a few Universities that were able to connect to one another through ARPANET, which was essentially a basic Internet, and a backbone to the modern Internet. In the early 1980s, access to ARPANET expanded heavily and then by 1990, ARPANET was retired, and what we now know as the modern Internet began to form. The technological changes brought upon by the advances in networks are some of the biggest leaps in mankind; the ability to transfer data between so many sources quickly and accurately allows for all the amazing technology we have today. After all, who can imagine a modern world without iPhones, Facebook, GPS, and just the Internet in general? Without the developments of computer networks we would have any of these things.

The first section of this project will explore the basics of the mathematical foundation of networks and the basics to network analysis, and how the simple methods lead to the more complex ones. The next part will describe in words how our selected network analysis algorithm – PageRank – operates, as well as how it is implemented to the real world. The following section will cover more in depth on the mathematics behind the PageRank algorithm as well some examples, and the following sections will cover both written examples and computer programmed examples of PageRank.

2 Introduction to Math in Networks

2.1 Mathematical Foundation of Networks

A network can be represented in many different ways, one of the most popular being the *adjacency matrix* (Figure 1 on the next page is a picture of a network and its adjacency matrix). This is where each edge between vertices i and j is marked with a "1" in both the (i,j) and the (j,i) positions of the matrix, indicating there is one connection (edge) between the vertices. If there is more than one connection between the vertices, then the number of connections that exist between the two is written in place of the 1. This matrix representation of the network is extremely useful in later computations to fully explore its information and trends. If we have a self-loop in the network, it counts as a "2" on that vector?s location (location: if vertex i, then position (i,i) in adjacency matrix) in the matrix, since the vertex technically has both ends of the edge touching it, otherwise the vectors location will contain a 0 in the matrix. Also, these adjacency matrices will always be symmetric, since if there is an edge between vertices i and j, then there will be a "1" in the (i,j) location of the adjacency matrix, as well as the (j,i) location of the adjacency matrix.

If a network is weighted or directed then the adjacency matrix may not be symmetric, and it may not have a "2" in the vertex?s location if there is a self-loop. A *weighted network* is one in which each edge has a "weight" (or a level of strength) to it. If vertices i and j have an edge with a weight of 1.7 between them, then the adjacency matrix of the network will have "1.7" in the (i,j) and (j,i) positions. A *directed network* is one in which the edges only point in a single direction, rather than to both vertices. For example, there can be an edge pointing from vertex i to j but not from j to i. This means that there would be a "1" in the (j,i) position of the adjacency matrix but a "0" in the (i,j) position of the adjacency matrix. Now, look back at Figure 1 and you can see how it differs in its setup from Figure 2. These weighted and directed networks are very common; the majority of real-world examples involving networks tend to be weighted, directed, or both.

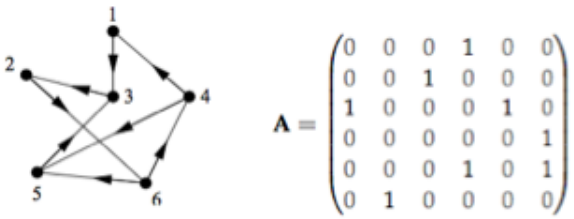


Figure 3: An Example of a Directed Network and its Adjacency Matrix

Two other key components of networks that can help see the connections between theory and the real world are the degrees of the vertices and the network's paths. The *degree* of a vertex in a graph is the number of edges connected to it, and in directed networks each vertex has an in-degree and an out-degree. The degree of a vertex can help to understand how important a vertex is in a given network, but as we will see later on there are many more ways to truly discover which vertex is of high importance in the network, this concept is called "centrality." A *path* in a network is any possible way through all of the edges and vertices of a network that you can connect any two given vertices in that network. The lengths and complexities of these paths often have a lot to do with the importance of certain vectors relative to others in the network; a vector with many short paths to the other vectors will often be considered to have relatively high importance in the network.

2.2 Intro to Network Analysis

Now that you are more familiar with the properties of networks, we can introduce the variety of network analysis methods utilized in industry today. In the past, network analysis methods were primarily used for the analysis of social networks, but today network analysis methods are also used for routers, modems, firewalls, and data transfer.

One method called *degree centrality* is the measure in a network where the degree of a vertex is focused. Depending whether the network is directed or undirected, the degree of the vertex may vary, which alters the result. The example Newman presents is the number of citations a scientific paper receives from other papers. In this example, the papers are vertices that connect each other in a directed network where the number of citations a paper receives is its in-degree, which displays how influential the paper is.

Next, an expansion of the degree centrality is *eigenvector centrality.* The issue with degree centrality is the number of citations a vertex had, but what about a paper with one citation, where millions cite this citation? Shouldn't we count this paper differently? Eigenvector centrality solves this issue, where a vertex is proportional to the centrality of its neighbors. Then in a social network, if Grace knows people in high positions, Grace would have a high eigenvector centrality. Now, we can solve whether a vertex has high degree or vertex has low degree with neighbors of high centrality equally.

However, there is a minor issue with eigenvector centrality, where a vectors centrality is forced to be 0 if it's only pointed to by other 0 centrality (degree 0) vectors. To fix this bug, we use a method called *Katz Centrality.* This method returns a centrality that is arbitrarily close to the original eigenvector centrality, but gives small non-zero values to vertices that have low degree centrality. This way, their small centralities amass over time, rather than remaining 0. We'll use a social network as an example again. An individual

might incorporate age or income as the constants (of the equation, which we will discuss later), where this returns a centrality close to the eigenvector centrality.

From here, you can understand there is a pattern. Katz Centrality has a problem, where if a vertex with high Katz Centrality points to many others then those others also get high-centrality. A vertex pointing to one hundred gives all neighbors degree one hundred. This does not seem right. An example is using Google search directory. Google links a Wikipedia page about dinosaurs. Wikipedia pages have hundreds of citations, but not all citations cite each other. Then, this logic does not suffice, since your webpage should not be rated high importance with hundreds of links cited. To fix this mistake, the *PageRank* algorithm retrieves the vertex pointed to centrality divided by its out-degree. Another problem arises, if the out-degree is zero. To fix this, we can artificially set the out-degree to 1 in this case and solve as necessary. This step will be justified in the math section of the project.

PageRank algorithm is most applicable to modern day usage of the Internet. According to Internet Live Stats, there are over 3.4 billion Google searches a day. This is an incredible amount data processed through our computers, which is why it's important to understand how our every day information is loaded on our screens. What type of mathematics, formulas, and algorithms do Yahoo, Bing, Baidu, and etc. use? In this project, we will analyze and prove mathematics behind the algorithm as well as provide examples through illustrations and models about the PageRank method.

3 PageRank

3.1 How PageRank Algorithm Works

Here I will show the PageRank algorithm, and then describe in words its methods of finding the centralities, x_i for each vector. The math behind discovering the method will be heavily expanded on in the math section of the project to follow.

$$x_i = \alpha \sum_j A_{ij} \frac{x_j}{k_j^{out}} + \beta$$

The PageRank method is an iterative method, meaning it will rerun itself many times (starting with an original guess of centralities of the vectors) until it can find centralities that change very little within a certain amount of steps, to ensure accuracy of the solution. Here we can see there are 5 parameters required to calculate x_i, which represents the centrality of the vector i. These parameters are the following:

- α represents a constant scalar which will multiply the following summation, with $0 < \alpha \leq {k_1}^{-1}$, where k_1 is the maximum eigenvalue of the adjacency matrix A of the network.
- A_{ij} represents the (i,j) position of the adjacency matrix A of the network.
- x_j represents the previous guess of centrality of the given vector (that is being used in order to calculate the next guess of centrality of the vector).
- k_j^{out} represents the out degree of the vector j.
- β is another constant added to the computation to ensure that the lowest possible centrality of a vertex in the network will be β rather than 0, to help with calculations.

After using enough iterations of this algorithm each vector will amass its own centrality, which is uniquely dependent on their location in the overall network, rather than just considering the nearby vertices or number of vertices connected. One vector can potentially have a single connection in a giant network, but if that vector is connected to a second vector that itself is very well connecter (or "central"), then the first vector will amass some centrality from it's connection and remain relatively central in the network.

This method was developed by Larry Page, cofounder of Google, and is one of Google?s top tools to this day when it comes to their web searches. The complexity of this method allows web search giants like Google to maintain order through millions of web pages, rank each one accordingly and sort them based on these rankings. PageRank is often thought to simulate a human's perception of what would be important in a network, which is demonstrated by Google?s global success.

3.2 How PageRank is Implemented

1. First, the user enters text into the search query in the desired search engine. A giant list of webpages matching the text is then created by "crawling."

> To understand this, crawling must be defined. Crawling is a verb described to find webpages and record its contents, then creating an annotated index of those contents. PageRank will first search the index for pages matching a given query by the following: The search is done by using many computers at distributed locations to search for web pages from different places. They check specific pages more often if it has been recently edited or popular. (706) They also check if there is altered behavior from the owner of the specific sites, who allow certain crawlers on their pages or allow crawling on certain pages.

2. Text of those webpages will be ranked based on (19.1) pages which the words of the query that appear or pages which the words don?t appear but link to a site that does. The text of those webpages created as an annotated index, and the link structure of the hyperlinks between them is used to calculate a centrality score.

> For Google, PageRank accords pages a high score if they receive hyperlinks from other pages. PageRank ranks pages that have the searched query directly in the page higher than if the searched query was linked by a different site. (Google uses PageRank as one of algorithms to find websites, while others may use frequency of occurrence of words in page text and position of occurrence such as the title, heading, text, or side margins.)

3. The scores of PageRank and other pre-computed scores such as the degree centrality are then combined to give an overall score. Then the hyperlinks with most relevance appear based on the overall score from highest on top to the least on the bottom.

4 Mathematics Behind PageRank

4.1 Degree Centrality

We start with degree centrality, where the centrality is equal to the in-degree of the vertex. I.e. if vertex i has in-degree a and out-degree b, then

$$x_i = a$$

Where x_i = centrality of vertex i. Or we can look at it in matrix form rather than component form, then

$$x = A1$$

Where x = vector of centralities of all vertices, and A = the adjacency matrix of the network, and $1 = \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix}$.

4.2 Eigenvector Centrality

We then take into account the properties of neighboring vertices, resulting in eigenvector centrality. We start with an initial guess of centrality x_0 and then repeatedly multiple our adjacency matrix A by this initial guess.

$$\begin{gathered} x = Ax_0 \\ \text{i.e. } x(1) = Ax_0 \\ x(2) = Ax(1) = AAx_0 = A^2x_0 \\ ... \\ x(t) = A^tx_0 \end{gathered}$$

So we can see that $x(t) = A^tx_0$ after t steps of A. Now choose x_0 as $x_0 = \sum_i c_iv_i$ for some constants (c_i) and then eigenvectors of A (v_i). Then,

$$\begin{gathered} x(t) = A^t \sum_i c_iv_i \\ x(t) = \sum_i c_iA^tv_i \\ (*1)\ x(t) = \sum_i c_ik_i^tv_i \\ (*2)\ x(t) = k_1^t \sum_i c_i(\frac{k_i}{k_1})^tv_i \end{gathered}$$

Notes:
($*1$): k_i are the eigenvalues of A corresponding to eigenvectors v_i of A.

(∗2): $k_1 = max(k_i)$

And since $k_i \leq k_1$ for any i, then $\frac{k_i}{k_1} \leq 1$ (further, $\frac{k_i}{k_1} < 1$ for all i except $i = 1$). Therefore, if we expand $x(t)$ and then take it's limit:

$$x(t) = k_1^t \sum_i c_i (\frac{k_i}{k_1})^t v_i$$
$$x(t) = k_1^t [c_1(1)^t v_1 + c_2(\frac{k_2}{k_1})^t v_2 + c_3(\frac{k_3}{k_1})^t v_3 + \ ... \ + c_n(\frac{k_n}{k_1})^t v_n]$$
$$\lim_{t\to\infty} x(t) = c_1 k_1^t v_1$$

Now we can see that the limiting vector of centralities is proportional to the leading eigenvector of the adjacency matrix. Equivalently we can say that the centralities, x, satisfy

$$Ax = k_1 x$$

We can then multiply both sides by k_1^-1 on the left side, and we then get:

$$x = k_1^{-1} Ax$$

Or in component form:

$$x_i = k_1^{-1} \sum_j A_{ij} x_j$$

This is the algorithm known as eigenvector centrality.

4.3 Katz Centrality

Eigenvector centrality has the fault that a vertex's centrality will be 0 if it is pointed to by only vectors with no in-degree. As described before, this is unrealistic and must be modified, which results in Katz Centrality. With Katz Centrality, we will give each vertex a certain small amount of "free centrality," so that if one vertex is pointed to by many others with no in-degree, the one vertex will at least sum up some of the "free centrality".

From before, we had:

$$x = k_1^{-1} Ax$$

Now, substitute α for k_1^{-1} and introduce the constant β. Then we have

$$x = \alpha Ax + \beta 1$$

with 1 vector defined as before.

The minimum centrality is now β for any vertices with no in-degree.
Now, isolate x.

$$x - \alpha Ax = \beta 1$$
$$(I - \alpha A)x = \beta 1$$
$$x = (I - \alpha A)^{-1}\beta 1 = \beta(I - \alpha A)^{-1}$$

Usually for convenience, $\beta = 1$

$$x = (I - \alpha A)^{-1}$$

This is Katz Centrality.

Or instead of isolating x, we can use it as an iterative method, where x' is the new centrality and x is the old centrality (stemming from initial estimate).

$$x' = \alpha Ax + \beta 1$$

This iterative method converges to a value close to correct centrality.

Note: To have the method converge to the correct centralities, it is required that $0 < \alpha \leq k_1^{-1}$. Choosing $\alpha \approx k_1^{-1}$ leads to similar centralities as before, except now the "zero centrality" vertices now have non-zero low centrality.

4.4 PageRank

The problem with Katz Centrality is that high centrality is transferred too easily from one vertex to another. The solution now is to divide the centrality of each vertex by its out-degree. What we are doing is essentially "normalizing" the centrality scores in math terminology, which results in the PageRank algorithm.

$$x_i = \alpha \sum_j A_{ij} \frac{x_j}{k_j^{out}} + \beta$$

Where k_j^{out} = out degree of vertex j.
There is a problem is $k_j^{out} = 0$ because we would essentially be dividing 0 by 0, since A_{ij} also equals 0 if $k_j^{out} = 0$.
The solution is to artificially let $k_j^{out} = 1$ in this case. Now, $\frac{A_{ij}}{k_j^{out}} \to \frac{0}{1} = 0$, so these vertices are still contributing zero additional centrality (aside from β). Now,

$$x = \alpha A D^{-1} x + \beta 1$$

With D being a diagonal matrix with $D_{ii} = max(k_i^{out}, 1)$ (i.e. $D_{ii} = k_i^{out}$ if $k_i^{out} \geq 1$, otherwise $D_{ii} = 1$). Now, solve for x.

$$x - \alpha A D^{-1} x = \beta 1$$
$$(I - \alpha A D^{-1}) x = \beta 1$$
$$(D - \alpha A) D^{-1} x = \beta 1$$
$$x = [(D - \alpha A) D^{-1}]^{-1} \beta 1$$
$$x = D(D - \alpha A)^{-1} \beta 1$$
$$x = \beta D(D - \alpha A)^{-1}$$

Again, β is essentially an overall multiplier that can be set to 1 for more simplicity. This results in what we know as the PageRank algorithm.

$$x = D(D - \alpha A)^{-1}$$

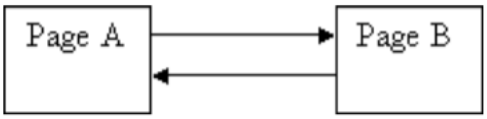
4.5 Summary

With Constant Term	Without Constant Term	
Divide by Out-Degree	$x = D(D - \alpha A)^{-1} 1 \rightarrow$ PageRank	$x = AD^{-1}x \rightarrow$ Degree Centrality
No Division	$x = (I - \alpha A)^{-1} 1 \rightarrow$ Katz Centrality	$x = k_1^{-1} Ax \rightarrow$ Eigenvector Centrality

5 PageRank Examples By Hand

5.1 Example 1

The following is a simple example to understand the general concept of PageRank.



In the figure above, Page A and Page B are pointing outward to each other. Thus, they both have an out degree of 1. According to *The Anatomy of a Large-Scale Hypertexual Web Search Engine*, we calculate the page rank of A with PageRank algorithm:

$$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)})$$

Or in the figure above,

$$PR(A) = (1-0.85) + (0.85)(\frac{PR(B)}{C(B)})$$
$$PR(B) = (1-0.85) + (0.85)(\frac{PR(A)}{C(A)})$$

where PR is the PageRank of a given vertex, d is the damping factor, C is the out degree of a vertex, and $T_1, ..., T_n$ are the citations from Page A, since that is the Page Rank we are solving for. In the following example we will use a damping factor of $d = 0.85$. Take note that since our damping factor is 0.85, the PageRank of the vertices can range from $(1 - 0.85) = 0.15$ to possibly billions. The damping factor is chosen to lessen influence on other pages. $\frac{PR(T_1)}{C(T_1)}$ is to show that Page A linked to from another page (in this case named T_1), and the PageRank of A will get a portion of the PageRank of T_1.
Continuing the example of the figure above:

$$PR(A) = (1-0.85) + (0.85)(\frac{PR(B)}{C(B)})$$
$$PR(B) = (1-0.85) + (0.85)(\frac{PR(A)}{C(A)})$$

Example 1a

But since Page A and Page B each only have one connection (to each other), then $C(A) = C(B) = 1$. Let's use an initial guess of $PR(A)$ and $PR(B)$ to be 1. This guess

can be anything, but running the PageRank algorithm many times should eventually lead you towards the actual PageRank centrality; the equilibrium point in the algorithm.

$$PR(A) = (1 - 0.85) + (0.85)(\frac{1}{1}) = 1$$

$$PR(B) = (1 - 0.85) + (0.85)(\frac{1}{1}) = 1$$

Therefore we have hit an equilibrium point, and thus the PageRank of vertices A and B are both 1. This example seems too simple; lets see one that must go through more iterations.

Example 1b

Suppose $PR(A) = PR(B) = 40$

First Iteration:

$$PR(A) = (1 - 0.85) + (0.85)(\frac{40}{1}) = 34.15$$

$$PR(B) = (1 - 0.85) + (0.85)(\frac{34.15}{1}) = 29.1775$$

Second Iteration:

$$PR(A) = (1 - 0.85) + (0.85)(\frac{29.1775}{1}) = 24.950875$$

$$PR(B) = (1 - 0.85) + (0.85)(\frac{24.050875}{1}) = 21.358244$$

Notice that for every iteration, the values of the PageRank score are getting lower and lower until they converge to 1.

Example 1c

Now, suppose $PR(A) = PR(B) = 0$

First Iteration:

$$PR(A) = (1 - 0.85) + (0.85)(\frac{0}{1}) = 0.15$$

$$PR(B) = (1 - 0.85) + (0.85)(\frac{0.15}{1}) = 0.2775$$

Second Iteration:

$$PR(A) = (1 - 0.85) + (0.85)(\frac{0.2775}{1}) = 0.385875$$

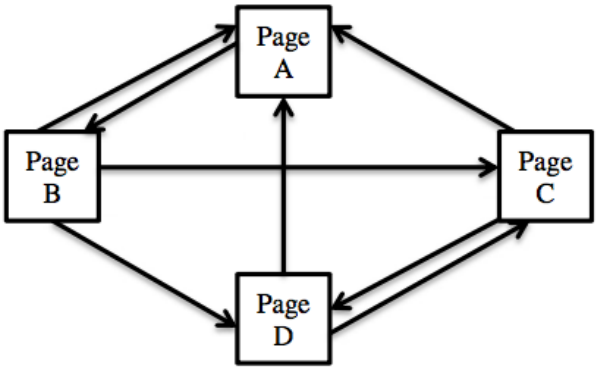
$$PR(B) = (1 - 0.85) + (0.85)(\frac{0.385875}{1}) = 0.477994$$

In this case, the values of PageRank are increasing until they converge to 1. Based on these two conclusions, there is a Lemma for this example: No matter what PageRank value is the initial guess, the final iteration will always converge to the *normalized probability distribution* 1.

Now that we have a general idea of the PageRank algorithm, the following will be a more comprehensive and complicated example.

5.2 Example 2

Suppose we have a small network of paper citations:



In this scenario: Page A links to Page B; Page B links to Page A, C, and D; Page C links to A and D; and Page D links to Page A and C. Notice the out-degree of each vertex in this network:

Vertex	Out Degree
A	1
B	3
C	2
D	2

Then we can create a matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Where $A_{ij} = 1$ if there exists an edge from j to i, 0 otherwise; and

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

Where D is diagonal and $D_{ii} = max(k_i^{out}, 1)$, where k_i^{out} is the degree of the vertex. From the text, we know that

$$x = \alpha A D^{-1} x + \beta 1$$

$$x = \beta(I - \alpha A D^{-1})^{-1} 1, \text{ but } \beta\text{=1 by convention}$$

$$x = (I - \alpha A D^{-1})^{-1} 1$$

$$x = D(I - 0.85A)^{-1} \text{ by Google search engine}$$

Then when we plug our known matrices into this last equation, we get

$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - 0.85 \left(\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x = \begin{bmatrix} 2.48045... & 1.74171... & 1.8338... & 1.8338... \\ 2.10839... & 2.48045... & 1.55837... & 1.55837... \\ 1.03891... & 1.22225... & 1.98834... & 1.28658... \\ 1.03891... & 1.2225... & 1.28658... & 1.98834... \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x = \begin{bmatrix} 7.88892... \\ 7.70558... \\ 5.53608... \\ 5.53608... \end{bmatrix}$$

Where x_i is the PageRank centrality for vertex i, for i = A, B, C, or D.

6 PageRank Examples by Computer Algorithm

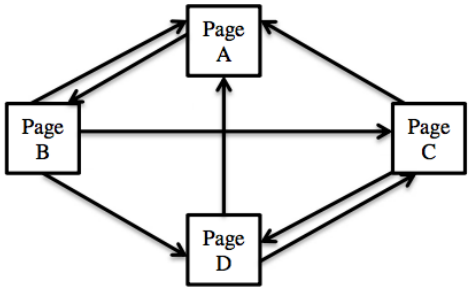
Now we will continue with examples of PageRank, but now the PageRank scores will be repeatedly iterated by a computer until the scores hit an equilibrium point.

The "Graph" portion of the code designates which vertices are pointing towards each other, while the "outdegree" and "indegree" show the given degrees of each respective vertex in the order listed in the "Graph" section, and then the adjacency matrix A and diagonal matrix D are created based on these values. *Note: Although indegree is shown here, it is not required for the PageRank calculation, but it can be used if you were to tweak the program to calculate centralities by a different method.*

To be continued on next page.

6.1 Example 1

We will continue with the last example from the previous section of this report.



It is now easy to understand the following output of the PageRank program:

```
Graph:
A: B
B: A, C, D
C: A, D
D: A, C

Outdegree: [1, 3, 2, 2]

Indegree: [3, 1, 2, 2]

A
[[ 0.  1.  1.  1.]
 [ 1.  0.  0.  0.]
 [ 0.  1.  0.  1.]
 [ 0.  1.  1.  0.]]
D
[[ 1.  0.  0.  0.]
 [ 0.  3.  0.  0.]
 [ 0.  0.  2.  0.]
 [ 0.  0.  0.  2.]]
1
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]]

Pagerank is D*(D-0.85*A)^-1 * 1
[[ 7.88891885]
 [ 7.70558102]
 [ 5.5360834 ]
 [ 5.5360834 ]]
```

We can see that the PageRank centrality of Vertex A is 7.889, Vertex B is 7.706, Vertex C is 5.536, and Vertex D is also 5.536.

6.2 Example 2

Here we will look at an example of a simple social network. Suppose Mark added Kevin on Facebook. This means Mark has a direct connection with Kevin, where he can see Kevin's profile and information, but Kevin can also see Mark's information. Therefore we are working with an undirected network as you can see by the setup of the Graph:

```
Graph:
Kevin: Mark, Katie
Katie: Kevin, Mark, Jieun
Jay: Jieun, Alex
Mark: Kevin, Katie
Jieun: Katie, Mark, Jay
Alex: Jay

Outdegree: [2, 3, 2, 2, 3, 1]

Indegree: [2, 3, 2, 3, 2, 1]

A
[[ 0.  1.  0.  1.  0.  0.]
 [ 1.  0.  0.  1.  1.  0.]
 [ 0.  0.  0.  0.  1.  1.]
 [ 1.  1.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.]]
D
[[ 2.  0.  0.  0.  0.  0.]
 [ 0.  3.  0.  0.  0.  0.]
 [ 0.  0.  2.  0.  0.  0.]
 [ 0.  0.  0.  2.  0.  0.]
 [ 0.  0.  0.  0.  3.  0.]
 [ 0.  0.  0.  0.  0.  1.]]
1
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]]

Pagerank is D*(D-0.85*A)^-1 * 1
[[ 7.24047385]
 [ 9.36892914]
 [ 5.56678484]
 [ 8.43751513]
 [ 6.02041348]
 [ 3.36588356]]
```

6.3 Example 3

```
Graph:
1: 2
2: 1, 3, 4
3: 2, 1
4: 3, 4

Outdegree: [1, 3, 2, 2]

Indegree: [2, 2, 2, 2]

A
[[ 0.  1.  1.  0.]
 [ 1.  0.  1.  0.]
 [ 0.  1.  0.  1.]
 [ 0.  1.  0.  1.]]
D
[[ 1.  0.  0.  0.]
 [ 0.  3.  0.  0.]
 [ 0.  0.  2.  0.]
 [ 0.  0.  0.  2.]]
1
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]]

Pagerank is D*(D-0.85*A)^-1 * 1
[[ 6.00389864]
 [ 8.65497076]
 [ 6.00389864]
 [ 6.00389864]]
```

6.4 Example 4

```
Graph:
A: B, C
B: C, D
C: D
D: C
E: F
F: C

Outdegree: [2, 2, 1, 1, 1, 1]

Indegree: [0, 1, 4, 2, 0, 1]

A
[[ 0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.]
 [ 1.  1.  0.  1.  0.  1.]
 [ 0.  1.  1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.]]
D
[[ 2.  0.  0.  0.  0.  0.]
 [ 0.  2.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  1.]]
1
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]]

Pagerank is D*(D-0.85*A)^-1 * 1
[[  1.        ]
 [  1.425     ]
 [ 17.90236486]
 [ 16.82263514]
 [  1.        ]
 [  1.85      ]]
```

7 Bibliography

Newman, Mark E. J. Networks: An Introduction. Oxford: Oxford U, 2015. Print.

"Pagerank Explained Correctly with Examples." Princeton University. The Trustees of Princeton University, n.d. Web. 12 June 2017.

Radu, Raluca Tanase Remus. "Lecture 3." PageRank Algorithm - The Mathematics of Google Search. N.p., n.d. Web. 12 June 2017.

8 Appendix

To view our code for the program used to compute the PageRank of the networks in section 6, please visit:

https://github.com/slin18/Google-Pagerank