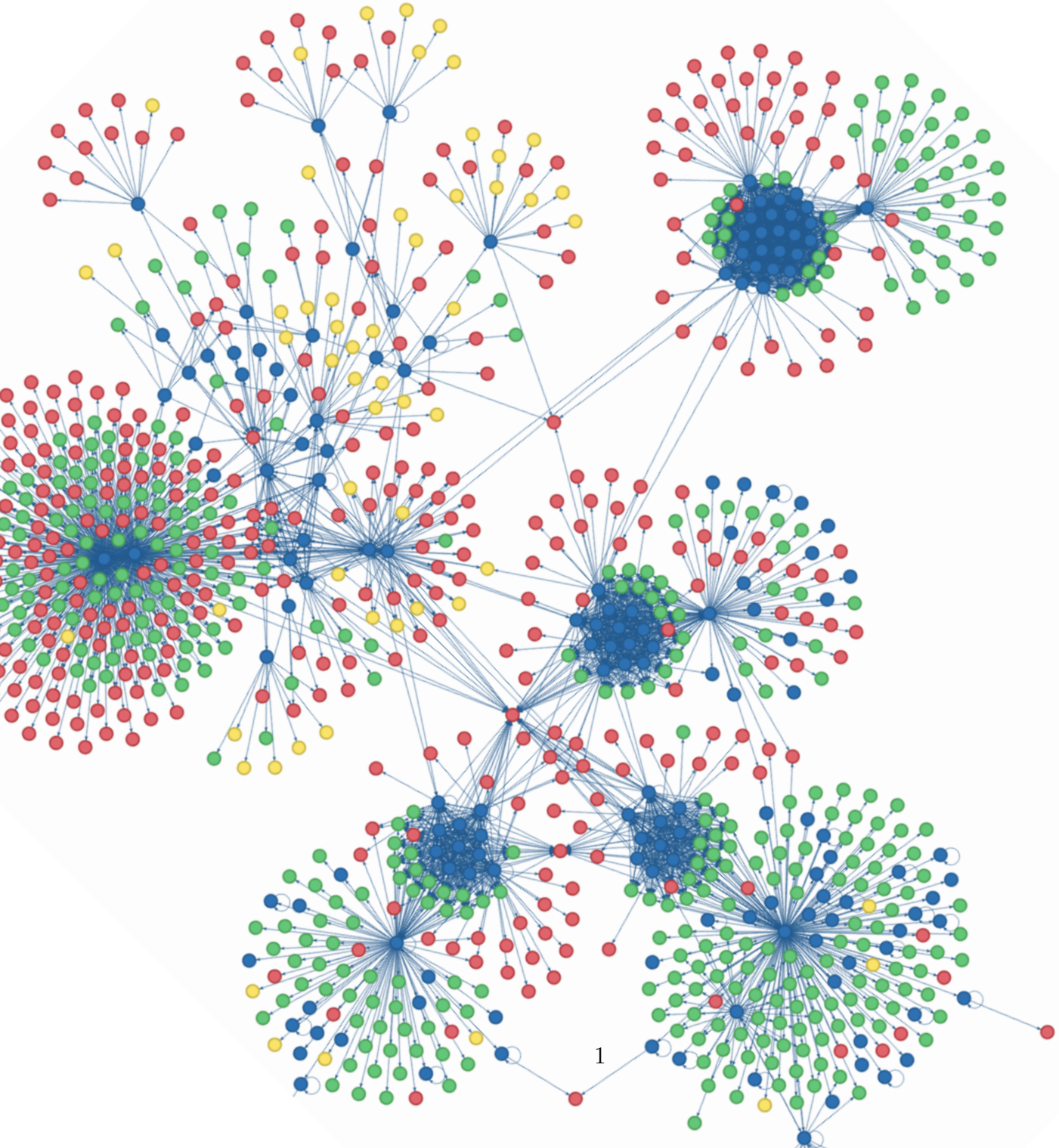
PageRank

Oliver Ricken
CSCI 148 - Graph Algorithms
Dr. Sarah Cannon
Claremont McKenna College

Wednesday, May 10th, 2023



Contents

1 Introduction

1.1 Problem Setting

Typing "Google PageRank" into Google generates 13,400,000 results in 0.36 seconds. That's a lot of information, and really fast. From a Computer Science perspective, a triumph. However, to the average person, this process is no longer a novel one. While modern search engines are unbelievably powerful, we rarely take notice. Who cares if 13 million sites are returned for a search? I certainly don't. I only want what's relevant and provides me with the information I'm looking for. *Relevance*—this is ultimately what matters, and what Google excels at.

Search existed before search was powerful. The World Wide Web started to become widely used in 1994, with search engines like Alta Vista, Lycos, and Excite available to users. Given the novelty of this technology, the Web started to explode in popularity. Brin and Page, in their original paper, stated that "there are over 150 million web pages with a doubling half life of less than one year" (Brin and Page, 1998). This was in 1998, and while some improvements had been made to the Web, it was still absurdly inefficient and not very practical for everyday use. This was due to the fact that the first search engines used text based ranking systems—which return the sites with the largest number of occurrences of key words from the query—to determine which websites would appear for a search. With these systems, a search for "Stanford University" would display a website with the name of the school printed thousands of times on the front page before the actual "stanford.edu" homepage. This is clearly not a functional model.

Brin and Page made one insight that transformed search overnight. The Web is a collection of linked websites, and these links are endorsements to a website's importance. For a query, the most important sites are usually the most relevant, and should be displayed first. Therefore, a website's importance (and relevance) should be determined by the number of important websites that link to it. Consider how this model would handle the previous example: the site with "stanford university" printed thousands of times on the front page probably does not have many important sites linking to it. However, the "stanford.edu" site definitely has a lot of websites that point to it, and a lot of these websites probably also have a lot of backlinks as well. Therefore, "stanford.edu" would appear before the other irrelevant site, which is most likely what the user would want.

This insight led Brin and Page to the creation of the PageRank algorithm. The impact was incredibly profound, and revolutionized our collective access to information.

1.2 High Level Overview

Google's ability to quantify relevance can be directly attributed to the PageRank algorithm—the heart of the Google search engine. At an extremely high level, the algorithm assigns a PageRank for all websites fetched for a given query. The PageRank for each site is a numerical value between zero and one which represents

each site's relative importance, and the order in which the sites will be displayed to the user. PageRank relies on mathematical concepts from Graph Theory, Linear Algebra, Probability, and Stochastic Processes. At a basic level, PageRank creates a graph of the websites fetched for a given query, creates a Markov chain from this graph, forms a transition matrix for this chain, and performs iterations of the power method to approximate the stationary distribution of the transition matrix. The algorithm stops iterating after the accuracy of our PageRank vector drops below a certain threshold, and then returns that vector as the PageRank for all websites fetched for the given query. Formally, PageRank takes in a directed graph that represents the subset of websites fetched according to the users query, where nodes are websites and edges are the links that connect sites. PageRank then returns a probabilistic column vector of length equal to the number of nodes in our graph, where each element of the vector corresponds to a website, and has value equivalent to the PageRank for that website.

2 Algorithm

2.1 Notation

Symbol	Description
G	Directed graph representing the internet network for our query.
n	Number of nodes in G, where each node is a website.
m	Number of edges in G, where each edge is a link between two sites.
ϕ_i	Out-degree of a node i.
β	Probability the surfer follows an out-link at random.
d	Damping factor. Probability the surfer jumps to some random website. (note that $d = 1 - \beta$, and d is usually 0.15).
δ	Difference between current state and previous state, found by L_1 norm. Note: δ is initially set strictly $> \epsilon$.
ϵ	Predefined threshold for when we should stop iterating.
$M_{i,j}$	First iteration of our transition matrix, size $n \times n$. i, j values specified.
$M'_{i,j}$	Second iteration of our transition matrix.
T	Final iteration of our transition matrix. Note: T is column stochastic with all positive entries.
J_n	The $n \times n$ matrix of all ones.
S_0	Starting state. Column vector with n values equal to $\frac{1}{n}$.
S_k	Previous state, when we update. Note: Column vector with n probability values that sum up to 1.
S_{k+1}	Current state, when we update. Note: Column vector with n probability values that sum up to 1
$PageRank$	Final PageRank vector.

2.2 Pseudocode

Algorithm 1 PageRank(G)

Require: Directed graph G. G has n nodes, each representing a website, and m edges, each representing a link between two websites.

Ensure: *PageRank* column vector of length n. Each element of *PageRank* is the PageRank for that site—the probability that our random surfer visits each website. We return *PageRank* when S_{k+1} sufficiently approaches the steady-state for our Markov Chain.

$$M_{i,j} = \begin{cases} \frac{1}{\phi_j} & \text{if } j \text{ links to } i \\ 0 & \text{otherwise} \end{cases}$$

$$M'_{i,j} = \begin{cases} \frac{1}{n} & \forall i \text{ in } j, \text{ if column } j \text{ of } M \text{ has all zeros} \\ M_{i,j} & \text{otherwise} \end{cases}$$

$T = \beta M' + \frac{d}{n} J_n$
$k = 0$
while $\delta > \epsilon$ **do**
  $S_{k+1} = TS_k$
  $\delta = \|S_{k+1} - S_k\|_1$
  $k = k + 1$
end while
$PageRank = S_{k+1}$ {We have sufficiently approached our steady-state}
return *PageRank*

2.3 Explanation

2.3.1 Random Surfer Model

While a seemingly simple algorithm on the surface, PageRank employs a variety of mathematical concepts spanning across multiple disciplines. Thus, due to the interdisciplinary nature of PageRank, it would be a futile effort to explain the algorithm in a top-down nature; many fundamental concepts would quickly become lost. Therefore, in order to give due consideration to all the mathematical underpinnings of PageRank, an explanation will be conducted in a bottom-up manner. We will start by covering the idea of a random walk, Markov chains and their connection to transition matrices, and then a general approach to computing the PageRank for every website. Then, we will walk (but not randomly) through the algorithm, explaining the logic and purpose behind each step in the process.

As stated earlier, Brin and Page assume that the user is a random "surfer" who clicks through websites at random, never going back but always choosing new sites to visit. The idea of a random surfer clicking through the internet directly relates to the concept of a random walk in a graph. In this case, our graph is the collection of websites found by the surfer's search query, where nodes are websites and edges are the links between them. A random walk models the activity of the surfer on this graph. Specifically, the surfer can randomly move from a given node to any

of the node's neighbors, given there exists an outgoing edge from the current node to the next, at different time steps $t, t+1, t+2, ...$ etc.

2.3.2 Markov Chains

Now that we have modeled the movement of the surfer on the web as a random walk, we need a way to capture this process of movement. We can do this with a Markov chain. By definition, a Markov chain is a stochastic model that describes a sequence of events in which the probability for the next state only depends on the current state, and not on the rest of the process. Due to the fact that the random surfer advances to new websites without influence from their previous movement, Markov chains are ideal for capturing the process of the surfer's movement. Next, recall that the goal of PageRank is to predict which site the surfer will advance to next. In relation to Markov chains, the PageRank vector for any given query is simply the stationary distribution of our Markov chain. The stationary (or steady-state) distribution is, for each state of the chain, the distribution of the state that is unchanging from step to step. Therefore, the stationary distribution of our Markov chain gives us the PageRank for our collection of sites—the unchanging probability that the surfer will advance to each site next.

Given our graph of websites and links between them for a given query, we need a way to model the way in which different sites reference each other, and thus pass on their importance to other sites. We do this with a transition matrix, which is a component of a Markov chain. For any directed graph, we assume that the surfer departs to a neighboring node with equal probability for every outgoing edge. So, we define each element of an (i, j) transition matrix as having value $1/\phi_j$ if node j links to node i, and value 0 otherwise. Let's visualize an example below.

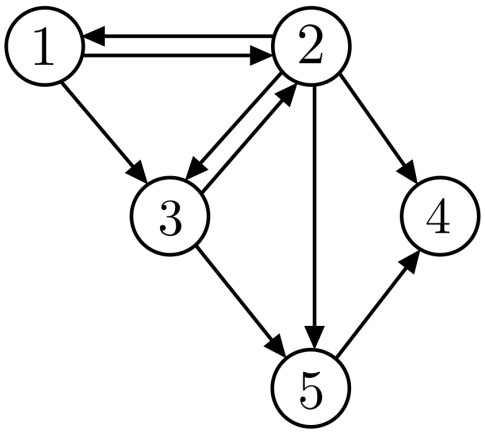


Figure 1: Simple directed graph, representing selected websites based on a query

$$M = \begin{bmatrix} 0 & 1/4 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1/2 & 1/4 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 & 1 \\ 0 & 1/4 & 1/2 & 0 & 0 \end{bmatrix}$$

Figure 2: Transition matrix M for the given graph

Next, we introduce some facts about Markov chains, relating to transition matrices (Crovella 2020): For a Markov chain with transition matrix M, the largest eigenvalue of M is 1. Additionally, if M is regular, then there is only one eigenvalue equal to 1, and the chain will converge to the corresponding eigenvector as its unique steady-state. Note that M is regular if some power of M only contains positive entries. Similarly, note that a Markov chain is considered regular if it has a regular transition matrix, which in this case is M. These facts about Markov chains and transition matrices will be extremely useful in future sections. With this in mind, we now are ready to move on to the PageRank algorithm itself.

2.3.3 Pre-Processing

Providing a general road map for the approach to the PageRank algorithm will be helpful here. As stated above, we encode the relative importance of websites in our transition matrix. As Brin and Page alluded to, our surfer should frequently visit important pages and infrequently visit unimportant pages. This motivation for PageRank informs our (very) general approach to the algorithm (where we assume that the directed graph for websites retrieved from our query is already formed as part of pre-processing, and passed into the PageRank algorithm as input):

General process for PageRank algorithm:

1. Construct the Markov Chain that corresponds to the surfer's movement on the graph

2. Rank-order the websites in our query according to the probabilities in the Markov Chain's stationary distribution.

How do we actually do this? Let's walk through PageRank step by step and see.

There is a considerable amount of pre-processing required in the PageRank algorithm before we can actually start updating our PageRank vector to approach the stationary distribution. We conduct this pre-processing, error handling along the way, so that our iterative process is more simple later on. First, we construct an $n \times n$ transition matrix, $M_{i,j}$ according to the definition of a transition matrix provided above. Immediately, we observe that M is not a regular transition matrix in all cases. For example, consider the case in which a node in our graph has no outgoing edges. Then, all elements in the column of our transition matrix corresponding to the j value of that node will be zero. It is crucial that our transition matrix is regular; without a regular transition matrix we are not guaranteed that there exists one eigenvalue equal to 1, and also that the chain will converge to the corresponding eigenvector as its unique steady-state. So, we need to modify M.

Consider the words of Brin and Page: "If a real web surfer ever gets into a small loop of web pages [(or finds a dead end)], it is unlikely that the surfer will continue in the loop forever. Instead, the surfer will jump to some other page [at random]"(Brin and Page, 1998).

Therefore, if the surfer reaches a small loop of websites or a site with no outgoing links, the surfer will randomly click to another page. So, we need to adjust M so that the column of zeros is replaced with another set of values that more accurately conveys the importance of the node with no outgoing links—if the node has incoming links, then it must be important to some degree! We conduct this modification by replacing every element in the column corresponding to our "dead-end" site with $\frac{1}{n}$. This way, the importance of our "dead-end" site is not overlooked, but redistributed equally among all other nodes in the graph.

$$M' = \begin{bmatrix} 0 & 1/4 & 0 & 1/5 & 0 \\ 1/2 & 0 & 1/2 & 1/5 & 0 \\ 1/2 & 1/4 & 0 & 1/5 & 0 \\ 0 & 1/4 & 0 & 1/5 & 1 \\ 0 & 1/4 & 1/2 & 1/5 & 0 \end{bmatrix}$$

Figure 3: Modified transition matrix, M', for the given graph

After this modification, we see that M' is still not regular—there exist some entries in the matrix that are zero. Let's revisit the words of Brin and Page and see how we can conduct another transformation of M' so that our transition matrix is regular. Brin and Page say, "the surfer periodically 'gets bored' and jumps to a random page" (Brin and Page, 1998). This means that, at every state, there is a small probability that the surfer will jump from a node in the graph to another random node that is not a neighbor of the previous node.

This small probability is called the damping factor, d, and is usually preset to a value around 0.15. So, we would like to incorporate the damping factor into our new transition matrix, T, when modifying M'. However, this modification is a bit tricky—we can not simply add the damping factor to every element in M', as some columns would then be greater than one, which does not make sense for a column of probabilities. Therefore, we normalize every element in M' by scaling every element by a factor of $(1 - d)$, and then adding $\frac{d}{n}$ to it. We call this new transition matrix T, and formally define it as follows:

$$T = (1 - d)M' + \frac{d}{n}J_n, \text{ where } J_n \text{ is the } n \times n \text{ matrix of ones}$$

After this modification, T now accounts for the case in which the surfer randomly jumps to another node that is not a neighbor of the previous node the surfer was on. T also accounts for the case in which the graph is disconnected. Because the web contains extremely diverse content, the case in which our graph is disconnected is a very realistic one. Incorporating the damping factor into T allows the

surfer to jump from one disconnected component to another, resolving this issue. Additionally, notice that every element in T is at least $\frac{d}{n}$ (greater than zero), so T is regular. Also, every column is a probabilistic vector that sums to one, so T is a column stochastic matrix. Here is a visualization of T:

$$T = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix}$$

Figure 4: Final transition matrix, T, for the given graph

Now that we have shown that T is a regular, column stochastic matrix, we introduce a theorem which will help us later on in finding the PageRank vector. Note that this theorem is the same as the facts from Crovella above; this version is simply formalized and involves the idea of a column stochastic matrix.

Theorem: Perron-Frobenius Theorem (Tanase and Radu, 2009): If M is a regular, column stochastic matrix, then:

1. There exists an eigenvalue of one, and this eigenvalue is unique. This is the largest eigenvalue: all other eigenvalues will have an absolute value less than one.

2. There exists a unique eigenvector for the eigenvalue one with the sum of its entries equal to one. This is the unique steady-state vector for the Markov Chain, and the PageRank vector we desire.

Therefore, by the Perron-Frobenius Theorem, because our transition matrix T is regular and column stochastic, then we know that T has a unique steady-state vector for the Markov chain, which is the PageRank vector we desire. At this point, all the pre-processing for the PageRank algorithm has been explained. Now, all that remains is to figure out how to find the PageRank vector for T.

2.3.4 Iterative Logic

Recall that the goal of the PageRank vector is to predict the probability that the surfer will advance to any given website at the next step. We know that the transition matrix T models the way in which different sites reference each other and pass on their importance to other sites. So, the logical relation between the PageRank vector and the transition matrix of our Markov chain is that we want the probabilities in our PageRank vector for any given page to reflect the importance of that page. Brin and Page said that the importance of any website should be determined by the importance of all the sites that link to it. For example, a site that is linked by only one really important site (say, Wikipedia) will have a greater importance ranking than a site that is linked by hundreds of sites with really low importance (say, a collection of amateur bird watchers' personal blogs). Therefore, we can define the importance of any given website as such:

$$\text{importance of site } i = \sum_j \text{importance of site } j \times \text{probability of going from } i \text{ to } j$$

This equation, with what we already have defined above, can simply be rewritten as $S = TS$, where S is a vector that contains the importance of all pages and P is a regular, column stochastic matrix that contains the probability of going from i to j. If satisfied, this equation contains all we desire out of PageRank—S is the stationary distribution for our Markov chain and also the final PageRank vector we want to return.

Actually finding the PageRank vector, assuming we have unlimited computational efficiency and space, is an extremely simple task. We can either solve the equation above, or find all the eigenvectors of T and return the unique eigenvector that corresponds to the eigenvalue, $\lambda = 1$. However, we do not have unlimited computational efficiency and space, so both of these tasks are extremely difficult. Search engines, especially, need to be really fast—neither of these options will work in practice. So, we need another method to find the steady-state of our chain.

We will now introduce two crucial facts that will form the basis for our upcoming approach. Refer to the proof section of this paper for proofs on why these facts are true. First, we claim that **the stationary distribution of our Markov chain is the principal eigenvector of the regular, column stochastic matrix** T. The principal eigenvector is the eigenvector that corresponds to the eigenvalue of largest magnitude, in this case, $\lambda = 1$. Next, we claim that **the Markov chain will eventually reach the steady state, independent of what the starting state is**. These two claims (which are proved later) inform a new approach for finding the stationary distribution of our chain.

2.3.5 Power Method

Because we know that the Markov chain will eventually reach the stationary distribution, we can compute the steady state in pieces, instead of directly as in the previous methods. The idea behind this process is that we initialize our starting state as a column vector of length n where every element of the vector has value $\frac{1}{n}$ (a better choice than starting with a random vector of probabilities; will lead to quicker convergence) and continually update that state at each time step k according to the equation defined above: $S_{k+1} = TS_k$. We then stop iterating once the difference between two states is sufficiently small, and has reached a predefined threshold. This is called the power method.

The beauty of the power method lies in its reliability, and efficiency. With respect to reliability, consider the following theorem:

Theorem: Power Method Convergence (Tanase and Radu, 2009): Let T be a regular, column stochastic $n \times n$ matrix. Denote the PageRank vector as the probabilistic eigenvector of T corresponding to the eigenvalue 1. Let s be the column vector with all entries equal to $\frac{1}{n}$. Then, the sequence $s, Ts, ..., (M^k)s$ converges to the PageRank vector.

Therefore, considering our pre-processing for transition matrix T, the power method will always converge to the steady state. However, this may take a considerable number of iterations the closer we get. This brings us to the predefined threshold mentioned above. At each iteration, we compare the difference in states, and see if we have achieved close enough convergence to the steady state. This threshold can be modified depending on the level of accuracy desired. We do not know the steady state distribution before starting iterations of the power method, so we measure the difference in states by taking the L_1 norm of the difference of the two state vectors we want to compare. Recall that the L_1 norm of a vector is the sum of absolute values over all elements. Once this difference falls strictly below our predefined threshold, we stop iterating and have found a PageRank vector sufficiently close to the stationary distribution of our Markov chain for the surfer's query, and we are done.

2.4 Proof

Consider the following multi step proof of correctness for the PageRank algorithm. First, we prove that the stationary distribution is the principal eigenvector of the regular, column stochastic transition matrix. Next, we prove that iterations of the power method will eventually converge to the principal eigenvector of the regular, column stochastic transition matrix. Because we use the power method in the PageRank algorithm, after proving both of these independently, we can conclude that we will find the principal eigenvector of our transition matrix, which is the stationary distribution of the transition matrix, and our desired PageRank vector.

Proof 1: For a regular, column stochastic transition matrix, the stationary distribution is the principal eigenvector of that matrix.

Let the stationary distribution for the transition matrix be denoted as S, where $S = TS$ by definition of the stationary distribution. We know that S must be an eigenvector of T, specifically, the eigenvector corresponding to the eigenvalue of 1. Our goal is to show that S is the principal eigenvector of T, that is, the eigenvector that corresponds to the eigenvalue of largest magnitude. So, by extension, all eigenvectors will have corresponding values strictly less than 1, because the stationary distribution for a regular, column stochastic matrix is unique. We proceed with some definitions.

First, recall the definition of the L_1 vector norm for a vector v:

$$\|v\|_1 = \sum_{k=1}^{n} |v_k|$$

Next, we consider the induced L_1 matrix norm, which is the maximum of the sums of the absolute values of the elements in each column of the matrix; in other words, the sum of the elements in the column with greatest magnitude. We can define the induced L_1 matrix norm for a matrix M using the above definition of the L_1 vector norm, as follows:

$$\|M\|_1 = \max_{v \neq 0; x \in \mathbb{R}^n} \frac{\|Mv\|_1}{\|v\|_1}$$

As stated above, an induced L_1 matrix norm is the sum of the elements in the column with greatest magnitude. So, we can evaluate M in the following manner:

$$\|M\|_1 = \max_j \sum_{k=1}^{n} |M_{i,j}|$$

By definition of an induced matrix, because M was induced by v, we consider M to be sub-multiplicative. As a result, we know that the following inequality holds:

$$\|Mv\|_1 \leq \|M\|_1 \|v\|_1$$

Now, let's incorporate eigenvalues into the proof. Let S be an eigenvector of regular, column stochastic matrix T. We then know that:

$$\|TS\|_1 \leq \|T\|_1 \|S\|_1$$

However, because T is regular and column stochastic matrix, all columns of T are probabilistic vectors that sum to 1. So, therefore, $\|T\|_1$ must be equal to 1. We can then modify our inequality above as follows:

$$\|TS\|_1 \leq \|S\|_1$$

Consider λ to be the associated eigenvalue of eigenvector S. By definition of eigenvalues and eigenvectors, we know that $TS = \lambda S$ must be true. We can then take the norm of both sides of this equation, and we have:

$$\|TS\|_1 \leq \lambda \|S\|_1$$

From above, we know that $\lambda \leq 1$ must hold true. So, any eigenvalue of matrix T must be less than or equal to 1. We also know that the stationary distribution of a regular, column stochastic matrix must exist, and will be an eigenvector of that matrix with corresponding eigenvalue 1. Because $\lambda \leq 1$ for regular, column stochastic matrix T, then the stationary distribution is the eigenvector that corresponds to the eigenvalue of largest magnitude in T. Therefore, S is the stationary distribution and the principal eigenvector of T.

In conclusion, we have proved that, for a regular, column stochastic transition matrix, the stationary distribution is the principal eigenvector of that matrix.

■

Proof 2: Iterations of the power method will eventually converge to the principal eigenvector of the regular, column stochastic transition matrix.

Recall the general approach to the power method: we start at an initial state, S_0, which is a length n vector with all elements equal to $\frac{1}{n}$. Then, we compute the next state by multiplying the old state and the transition matrix, as such: $S_{k+1} = TS_k$. We then repeatedly iterate until the difference between two consecutive states is sufficiently small, and has reached our threshold.

Our goal is to show that the sequence of states, $S_0, S_1, S_2, ..., S_k$ converges to the principal eigenvector of our regular, column stochastic transition matrix T.

By definition of a Markov Chain, we know that the state of the chain at any step k is given by the following:

$$S_k = c_1 v_1 \lambda_1^k + c_2 v_2 \lambda_2^k + ... + c_n v_n \lambda_n^k$$

where S_k is the state of the chain at time step k, c_n are constants, v_n are the eigenvectors of the transition matrix, and λ_n are the eigenvalues associated with the eigenvectors. Note that the principal eigenvector is v_1, which is also the stationary distribution of the Markov Chain, as proved in the last proof. By extension, we also know that $\lambda_1 = 1$. Additionally, because our transition matrix is regular and column stochastic, we know that the Markov Chain will eventually converge to the steady state. Therefore, $\forall k$ excluding $k = 1$, $\lambda_k < 1$.

With these facts in mind, we then take the limit of the state of our Markov Chain at any step k. Because all eigenvalues other than λ_1 are less than 1 in magnitude, we notice the following:

$$\lim_{k \to \infty} S_k = c_1 v_1$$

Note that c_1 is merely a constant and does not have an affect on S_k as k gets infinitely large.

We have made a crucial observation. As the number of iterations increases and k goes to infinity, all other elements of the state of the Markov Chain go to zero, and we are left with $c_1 v_1$. As stated above, the principal eigenvector of our transition matrix is v_1. So, for any Markov Chain, taking infinite (but usually significantly less) iterations of the power method will give us the principal eigenvector of the transition matrix. This is our goal, and we are done.

Conclusively, we have proved that iterations of the power method will eventually converge to the principal eigenvector of the regular, column stochastic transition matrix, independent of the starting state.

■

Final Proof: The PageRank algorithm computes the correct PageRank for every website in a given query.

Given the thoroughness of the proofs above, this proof is quite simple. First, recall that the pre-processing step of our PageRank algorithm finds the regular, column stochastic stationary distribution of the Markov chain for the given query. We then employ the power method, repeatedly iterating and finding the state of the chain at increasing time steps. By Proof 2 above, we will eventually find the principle eigenvector of the regular, column stochastic transition matrix using the power method. By Proof 1 above, the stationary distribution is the principal eigenvector of the transition matrix found by the power method. We know that the PageRank vector is the stationary distribution of the Markov chain for a given query. Therefore, the PageRank algorithm correctly finds the PageRank vector, which gives the correct PageRank for every website in the query.

■

2.5 Running Time

There is not one standard running time for the PageRank algorithm—the running time varies greatly depending on the implementation and data structures used. For example, there are a considerable amount of studies that propose new implementations of PageRank to achieve sub-linear time. Explaining these methods could be the topic of an entirely different paper, and will not be discussed here. With this in mind, consider the running time for the PageRank algorithm given in this paper:

For pre-processing, we have to:

- Compute M, M', and T. Each of these steps requires us to look through every element of the $n \times n$ matrix at most, and takes $\mathcal{O}(n^2)$ time.

For one iteration of our while loop, we have to:

- Compute the next state using matrix multiplication of T and S_k. This calculation requires us to compute n multiplications and n-1 additions for each row of the matrix (by the definition of the Dot Product), and we have n rows in the matrix. This is $\mathcal{O}(n(n + (n - 1))$, which simplifies to $\mathcal{O}(n^2)$.
- Compute the L_1 norm of the difference between S_{k+1} and S_k. This requires us to take the absolute value of n elements, which is $\mathcal{O}(n)$ time.

Therefore, our total running time for pre-processing and one iteration of our while loop simplifies to $\mathcal{O}(n^2)$.

The number of iterations needed to find a PageRank vector that is sufficiently close to the steady state of our Markov chain depends greatly on the size of the graph for the query. Determining the average number of iterations needed for convergence is an entire field of research in iteslf and out of the scope of this paper. So, for our purposes, we can assume that we will find the PageRank vector after a constant number of iterations. Therefore, this brings the overall running time of our PageRank algorithm to $\mathcal{O}(n^2)$.

3 Sample Case

We will now run a sample case of the PageRank algorithm based on the example graph provided earlier, and the corresponding transition matrix T:

$$T = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix}$$

For the sake of comparison, consider the stationary distribution of T, called $PageRank$. This is the steady state we desire for this chain, and was computed by finding the principal eigenvector of T. Presenting $PageRank$ before we start iterating with the power method provides us with a reference point as to when our states start to converge to the PageRank vector. (Refer to the appendix for the Python code used for this computation)

$$PageRank = \begin{bmatrix} 0.12391346 \\ 0.2075231 \\ 0.17657668 \\ 0.29302822 \\ 0.19895854 \end{bmatrix}$$

Now that we have our regular, column stochastic transition matrix and the stationary distribution for that matrix (for comparison sake), we will begin updating the state vector at each time step, just like in the PageRank algorithm. Additionally, we can define a threshold for our example. Let's use $\epsilon = .01$. First, we start at S_0, the column vector with all elements equal to $\frac{1}{n}$.

$$S_0 = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$$

Now, we begin iterating according to the power method, and noting the difference between states $k+1$ and k, $\delta_{k+1,k}$, along the way:

$$S_1 = \begin{bmatrix} 0.1065 \\ 0.234 \\ 0.1915 \\ 0.2765 \\ 0.1915 \end{bmatrix} = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix} \cdot \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$$

$$\delta_{1,0} = 0.22100000000000003$$

$$S_2 = \begin{bmatrix} 0.12673 \\ 0.203655 \\ 0.1719925 \\ 0.289505 \\ 0.2081175 \end{bmatrix} = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix} \cdot \begin{bmatrix} 0.1065 \\ 0.234 \\ 0.1915 \\ 0.2765 \\ 0.1915 \end{bmatrix}$$

$$\delta_{2,1} = 0.09970500000000002$$

$$S_3 = \begin{bmatrix} 0.12249254 \\ 0.20617291 \\ 0.17635279 \\ 0.29939241 \\ 0.19558935 \end{bmatrix} = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix} \cdot \begin{bmatrix} 0.12673 \\ 0.203655 \\ 0.1719925 \\ 0.289505 \\ 0.2081175 \end{bmatrix}$$

$$\delta_{3,2} = 0.033531225$$

$$S_4 = \begin{bmatrix} 0.12470845 \\ 0.20790597 \\ 0.17676778 \\ 0.2909594 \\ 0.19965839 \end{bmatrix} = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix} \cdot \begin{bmatrix} 0.12249254 \\ 0.20617291 \\ 0.17635279 \\ 0.29939241 \\ 0.19558935 \end{bmatrix}$$

$$\delta_{4,3} = 0.01686602193749996$$

$$S_5 = \begin{bmatrix} 0.12364312 \\ 0.2075905 \\ 0.17664421 \\ 0.29335275 \\ 0.19876943 \end{bmatrix} = \begin{bmatrix} 0.03 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.455 & 0.03 & 0.455 & 0.2 & 0.03 \\ 0.455 & 0.2425 & 0.03 & 0.2 & 0.03 \\ 0.03 & 0.2425 & 0.03 & 0.2 & 0.88 \\ 0.03 & 0.2425 & 0.455 & 0.2 & 0.03 \end{bmatrix} \cdot \begin{bmatrix} 0.12470845 \\ 0.20790597 \\ 0.17676778 \\ 0.2909594 \\ 0.19965839 \end{bmatrix}$$

$$\delta_{5,4} = 0.004786692911249987$$

Above, we see that $\delta_{5,4}$ is less than our threshold of 0.01, and sufficiently close to the stationary distribution of T. Therefore, our algorithm will stop iterating and return S_5 as our PageRank vector.

$$\text{PageRank vector returned for sample case} = \boxed{\begin{bmatrix} 0.12364312 \\ 0.2075905 \\ 0.17664421 \\ 0.29335275 \\ 0.19876943 \end{bmatrix}}$$

This PageRank determines which websites in our example are most important, and should be displayed first. Given the query that generated the Markov chain used in our example, the websites will be displayed in the following order:

Website 4
Website 2
Website 5
Website 3
Website 1

4 Applications

4.1 Applications to Other Problems

PageRank is perhaps the most influential algorithm ever created, and its real-world applications require little elaboration. Access to information defines our modern world, and PageRank made this possible. However, PageRank also has significant applications in various fields beyond search engines, which are discussed in this section. Note that the purpose of this section is to provide the reader with an idea as to how PageRank can be used to solve other problems, not how these other problems are actually solved. To this end, brevity will be employed. Consider the following applications below:

- **Collaborative Filtering and Recommender Systems**: PageRank can be used to recommend items to users based on their preferences and the preferences of other users with similar interests. By constructing a bipartite graph with users and items as nodes, and user-item interactions as edges, the PageRank algorithm can identify items with high importance scores that are likely to be relevant to a user. This method has gained popularity in recent years, as it can be effectively applied in various domains such as online shopping, and movie streaming services, to name a few.

- **Text Summarization**: By considering sentences as nodes and their similarities as edges, PageRank can be used to identify the most important sentences in a document, helping to create a summary of the text. The algorithm distributes scores iteratively through the graph based on the connectivity and edge weights, resulting in the convergence of scores that signify the importance of each sentence. Although text summarization has seen the development of other effective techniques (such as extractive, abstractive, and keyword-based methods), the PageRank-based approach has demonstrated an effectiveness in generating concise and relevant summaries in some scenarios.

- **Image Recognition and Segmentation**: PageRank has been used in computer vision to rank image regions based on their importance, which can help in tasks such as object recognition or image segmentation. In this application, the algorithm operates on a graph where nodes represent image regions and edges are weighted based on visual similarity or spatial proximity, allowing PageRank to identify and prioritize important regions. Although

other methods—such as convolutional neural networks (CNNs) and deep learning techniques—dominate the field of computer vision, PageRank-based approaches have demonstrated effectiveness in certain situations and can provide complementary insights.

5 Appendix

Python code used in the sample case section:

```
"""
Although this code has been modified,
credit for the initial version goes to:
Mark Crovella at Boston University
https://www.cs.bu.edu/fac/crovella/cs132-book/L19PageRank.html
"""

import numpy as np

# T is our regular, column stochastic transition matrix
T = np.array([[0.03, 0.2425, 0.03, 0.2, 0.03],
    [0.455, 0.03, 0.455, 0.2, 0.03],
    [0.455, 0.2425, 0.03, 0.2, 0.03],
    [0.03, 0.2425, 0.03, 0.2, 0.88],
    [0.03, 0.2425, 0.455, 0.2, 0.03]])

# find eigenvalues and eigenvectors using NumPy
eigenvalues, eigenvectors = np.linalg.eig(T)

# next we sort eigenvalues from smallest to largest,
# and find the principal eigenvector (with eigenvalue = 1)
# first we find index of the eigenvalue 1
eigen_indices = np.argsort(eigenvalues)
# and take the index of the largest eigenvalue
principal_eigval = eigen_indices[-1]

# using the index of eigenvalue 1, we find the
# corresponding eigenvector (the steady state vector)
steadyState = np.real(eigenvectors[:,principal_eigval])
steadyState = steadyState/np.sum(steadyState)

# next we find the order of sites in the steady state vector
# (sorts in reverse order)
reverseOrder = np.argsort(steadyState)

# reverse the (reversed) order so most important page is first
# add one to convert from zero indexing to indexing of example
# (example starts indexing at 1)
order = 1 + reverseOrder[::-1]
print('steady state = ', steadyState)
print('final order = {}'.format(order))
print('importance = {}'.format(steadyState[order-1]))

# this is how we perform iterations of the power method
# start at initial state, every element has value 1/n
S_0 = np.array([0.2, 0.2, 0.2, 0.2, 0.2])
```

```
S_1 = np.matmul(T, S_0)
S_2 = np.matmul(T, S_1)
S_3 = np.matmul(T, S_2)
S_4 = np.matmul(T, S_3)
S_5 = np.matmul(T, S_4)

# can modify sub_states to find difference between
# any two consecutive states
sub_states = S_5 - S_4
# we find difference using L1 norm
difference = np.linalg.norm(sub_states, ord=1)
#print(difference)
```

6 Works Cited

1. Crovella, Mark. "PageRank — Linear Algebra, Geometry, and Computation." Linear Algebra, Geometry, and Computation. Mark Crovella, 2020. https://www.cs.bu.edu/fac/crovella/cs132-book/L19PageRank.html

2. Harel, Jonathan, Christof Koch, and Pietro Perona. "Graph-Based Visual Saliency." In Advances in Neural Information Processing Systems, edited by B. Schölkopf, J. Platt, and T. Hoffman, Vol. 19. MIT Press, 2006. https://rb.gy/w4oxq

3. Huber, Mark. Stochastic Processes. Claremont McKenna College: Mark Huber, 2017. https://www.markhuberdatascience.org/_files/ugd/c2b9b6_98b69ae5caa54f68aeaabf198918e918.pdf

4. Jamali, Mohsen, and Martin Ester. "A Matrix Factorization Technique with Trust Propagation for Recommendation in Social Networks." In Proceedings of the Fourth ACM Conference on Recommender Systems, 135–42. RecSys '10. New York, NY, USA: Association for Computing Machinery, 2010. https://doi.org/10.1145/1864708.1864736

5. Jauregui, Jeff. "Markov Chains, Google's PageRank Algorithm." Lecture, University of Pennsylvania, October 25, 2012. https://www2.math.upenn.edu/~kazdan/312F12/JJ/MarkovChains/markov_google.pdf

6. Leskovec, Jure. "Analysis of Large Graphs: Link Analysis, PageRank." Stanford University, February 3, 2014. http://snap.stanford.edu/class/cs246-2014/slides/09-pagerank.pdf

7. McGregor, Andrew. "Reasoning about Uncertainty." Lecture, University of Massachusetts, March 27, 2017. https://people.cs.umass.edu/~mcgregor/240S17/lec15.pdf

8. Mihalcea, Rada, and Paul Tarau. "TextRank: Bringing Order into Text." In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, 404–11. Barcelona, Spain: Association for Computational Linguistics, 2004. https://aclanthology.org/W04-3252

9. Page, Lawrence, and Sergey Brin. "The PageRank Citation Ranking: Bringing Order to the Web," 1998. https://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife/pagerank.pdf

10. "PageRank." In Wikipedia, April 25, 2023. https://en.wikipedia.org/w/index.php?title=PageRank&oldid=1151731265

11. Roberts, Eric. "The Google PageRank Algorithm." Stanford University, November 9, 2016. https://web.stanford.edu/class/cs54n/handouts/24-GooglePageRankAlgorithm.pdf

12. Su, Jessica. "CS 224W – PageRank." Stanford University, n.d. http://snap.stanford.edu/class/cs224w-2017/slides/handout-page_rank.pdf

13. Tanase, Raluca, and Remus Radu. "PageRank Algorithm - The Mathematics of Google Search." Accessed May 7, 2023. http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

14. Tomlinson, Kiran. "Site Graph." Kiran Tomlinson, March 15, 2020. https://www.cs.cornell.edu/~kt/post/site-graph/

15. Weisstein, Eric W. "L^1-Norm." Text. Accessed May 7, 2023. https://mathworld.wolfram.com/