Markov Chains and Random Walks with Google PageRank

Written by Andrew Tang

Table Of Contents

Introduction

In graph and probability theory, a random walk is a process of traversing a graph, where at every node, the edge that is traversed next is chosen uniformly at random. A Markov chain on the other hand, is similar, but the outgoing edges at each node is chosen according to some fixed distribution and is represented by a transition matrix. An example of a random walk would be if a drunk man who is making his way home on a grid of streets, where he stumbles along each street, and at each intersection he picks a direction at random to travel towards. This is the most popular example of a random walk, and given a random walk or a Markov chain, we are generally interested in a few things. In this case it would be: How quickly will we be able to reach a particular node? Or for the drunken man, how quickly will he be able to reach home, or will he even be able to reach home?

In this report we will detail the mathematics and probability theory that surrounds random walks and Markov chains as well as the algorithmic applications of them, with the most notable one being Google's PageRank algorithm. The goal of this project and report is to present a deep dive into the intricacies of random walks and Markov chains and then relate it to Google PageRank in a format such that one will be able to understand the content with little to no prior knowledge about the topics discussed. It will also provide a comparison between the random surfer model of PageRank and the mathematical analysis version of PageRank involving matrix mathematics using a simulation programs.

Markov Chains

Markov Chains and random walks are related concepts, but they are distinct models and are also used in different contexts. A Markov Chain is a stochastic process which models the future state of a system depending on only its current state and not influenced by any state or process before it. It is defined by a state space and transition probabilities. A state space is the set of all possible events that the system can occupy. In the context of PageRank, the state space would be all the webpages. The set of transition probabilities is the probabilities that the state will transition from one state to another and is often presented as a transition probability matrix. Each state will have a transition probability for all other states including itself and the probability can be zero. A Markov Chain can be modelled as a graph where the nodes are the state spaces and the edges are the transition probabilities that are not zero and are also given probabilities at each connection. A visual representation of a Markov Chain is depicted in the Figure 1 below.

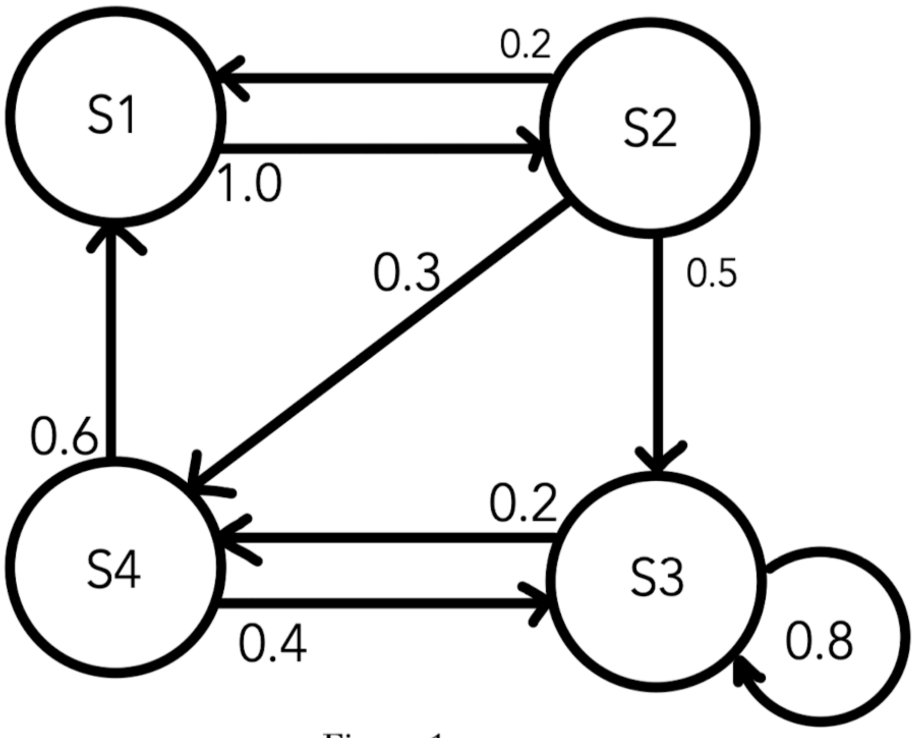


Figure 1

Observing this image, we can see there are 4 states in the state space, and there are probabilities associated with each edge. Also note that all outgoing edges add up to 1 for any particular node. This is because each state has options and those options have a particular probability of being chosen out of a 100% probability of 1.

We can also depict the probability matrix of Figure 1. As there are four states, the dimensions of the matrix will be 4x4 and the columns represent a state and each row represent the states that the state in the respective column is connected to. The transition probability matrix P is:

$$P = \begin{pmatrix} 0 & 1.0 & 0 & 0 \\ 0.2 & 0 & 0.5 & 0.3 \\ 0 & 0 & 0.8 & 0.2 \\ 0.6 & 0 & 0.4 & 0 \end{pmatrix}$$

Stationary Convergence

Stationary convergence is one of the most important aspects of Markov Chains. Essentially, stationary convergence refers to the probability distribution of the states in a Markov Chain whereby the probability distribution remains unchanged by the transition probabilities of the chain. This means as the of iteration process continues, the probability distribution over all the states converge to a stable value for each state. For a chain to be able to reach stationary convergence, it must be irreducible and aperiodic. Irreducible means that every state is reachable from every other state. Aperiodic means that starting at any state, it will be able to return to that state without follow fixed pattern or repeating cycle.

To calculate the stationary convergence π of a Markov Chain, we essentially want to find a vector π such that $\pi P = \pi$ where P is the transition matrix of our chain. Notice, this equation is much like finding the eigenvector of the transition matrix where the eigenvalue is 1. Similar conditions apply in this context and the main process of finding π is to seperate the matrix multiplication into its linear equations and solve for each variable. As this matrix is linearly dependent, we also need one more equation to be able to solve for the variables and the final vector. As we know this is a probability vector, we also know that the variables must add up to 1, meaning the values in π must add to 1 (row stochastic). Then with these equations, we will be able to solve the vectors and compute the final eigenvector π.

As an example, we will calculate the stationary distribution of our matrix given by the graph in Figure 1. First we have the probability matrix which is

$$P = \begin{pmatrix} 0 & 1.0 & 0 & 0 \\ 0.2 & 0 & 0.5 & 0.3 \\ 0 & 0 & 0.8 & 0.2 \\ 0.6 & 0 & 0.4 & 0 \end{pmatrix}$$

and we need to find π in the equation $\pi P = \pi$ which comes to

$$\pi \begin{pmatrix} 0 & 1.0 & 0 & 0 \\ 0.2 & 0 & 0.5 & 0.3 \\ 0 & 0 & 0.8 & 0.2 \\ 0.6 & 0 & 0.4 & 0 \end{pmatrix} = \pi.$$

We then write out our π values which we will write as a, b, c and d which becomes

$$(a, b, c, d) \begin{pmatrix} 0 & 1.0 & 0 & 0 \\ 0.2 & 0 & 0.5 & 0.3 \\ 0 & 0 & 0.8 & 0.2 \\ 0.6 & 0 & 0.4 & 0 \end{pmatrix} = (a, b, c, d).$$

Multiplying through to find our linear system of equations we get 4 equations:

(1) $1b = a$

(2) $0.2a + 0.5c + 0.3d = b$

(3) $0.8c + 0.2d = c$

(4) $0.6a + 0.4c = d$

then our final equation to help us solve for a, b, c and d:

(5) $a + b + c + d = 1$.

Solving this linear system we get $\pi = (0.118, 0.118, 0.608, 0.157)$ which is the stationary distribution of the Markov Chain shown in figure 1. An analogy for this in relation to the random walker model is that if we place our random walker on any of these states, if we let them walk through this graph infinitely and track the proportion of time spent at each state, π will be the proportion of time spent at each state after an infinite amount of steps taken. We can see from π that our random walker would spend the most time at state 3. This is due to the fact that the random walker has an 80% chance of staying in state 3 on each step once they arrive.

The stationary convergence of a Markov Chain is important for PageRank as it is one of the concepts used to determine rank. This is because the stable probabilities directly reflect the "importance" of its adjacent states/websites as well as the importance of those websites.

Mixing Time

The mixing time of a Markov Chain is the how quickly the chain approaches stationary convergence. This is heavily dependent on the overall features and structure of the graph. There are a few variations on how to calculate mixing time and it depends on the overall structure of graphs and is very complex for complex graphs. For the context of this report, we are interested in a rough estimate of the mixing time which can be approximated in a simulation algorithm where we count the number of steps it takes for a random walker to record an approximate stationary distribution.

Some of the main features that affect the mixing time is the graph diameter, probability distribution and general connectivity. Graph diameter means the longest shortest path between two nodes. A large diameter means it will ultimate take longer for the random walker to reach either sides of the diameter of the graph, thus a lower diameter may correlate to a lower mixing time. Probability distribution for each edge is how even the transition probabilities are from each node, and will affect the mixing time in that it may prevent the random walker from reaching certain areas in the graph, thus worsening mixing time. The overall connectivity of a graph is important to take into account when calculating mixing time as it means the random walker may be able to traverse the whole graph faster thus lowering the mixing time.

The mixing time for an algorithm using Markov Chains is very important as it is essentially the speed at which the algorithm will reach a stable state/ideal position. For PageRank, the mixing time is the time it takes for the random surfer to traverse the graph enough for the ranks to reach a stable state, meaning it is the time it takes for the naive algorithm to run.

For reference, the mathematic calculation for mixing time is:

$t_{mix}(\epsilon) \leq \frac{1}{1 - max(|\lambda_2|, |\lambda_{min}|)} log\left(\frac{1}{\epsilon \pi_{min}}\right)$ where ϵ is accuracy of the probability distribution.

Random Walks

To begin with, a random walk is a mathematical concept which describes a path or a sequence of steps where each step is taken randomly. It is a stochastic process, meaning it is defined by the series of random variables which happen over time. For the context of this report, the paths will be on a connected graph and at each intersection/node, a direction is randomly chosen, and the current state at which the random walk is at is not affected by any previous decisions. Other forms of the random walk include the lattice random walk and a one dimensional random walk which are points that move randomly on a 2-dimensional grid or a number line respectively.

Random walks can also be modelled as a Markov Chain, with the state space being each node in the graph and the edges being the transition possibilities, although the possibilities at a particular intersection is even.

One of the main things we find is that for a strongly connected graph, no matter where the random walk starts, the fraction of time the walk spends at each vertex of the graph converges to a certain number. The number of all of these fractions will also add up to one, and is also reliant on Markov Chain stationary distribution. Google PageRank uses this concept with their random surfer to model the importance of their webpages.

If we refer back to the drunken man walking on a grid and trying to find his way home, the main question is whether the man will end up home or not, or whether he will end up at the bar. We find that with random walks on such a grid, this man will always be able to find his way home or back to the bar, as due to the nature of the walk and the grid, the man will theoretically cover every edge and every node on the grid given enough time. In the context of random walks and Markov Chains, this long term behaviour is called a recurrent state. A recurrent state in a Markov Chain or random walk is that if a process enters at state s_1, and continues, the process will have a one hundred percent chance of returning to s_1, in other words, the drunken man will always return to intersections he has been to before, and this will happen infinitely given he continues his random walk infinitely. More formally, a state i is recurrent if $P(T_i < \infty | X_0 = i) = 1$ where T_i is the first time the process returns to state i and X_0 is the initial state.

The opposite of this behaviour is called a transient state, meaning given a particular state and random walk, the walk is not guaranteed to return to that state, in essence, the walk may get "lost". More formally, a state i is transient if $P(T_i < \infty | X_0 = i) > 0$ where T_i is the first time the process returns to state i and X_0 is the initial state.

It is also important to note that walks on a 2-dimensional grid is recurrent and walks on a 3-dimensional grid is transient. Additionally, imagine the World Wide Web as a graph, where nodes are each webpage and directed edges are from websites that contain links to the websites that are being linked to. Processing a random walk in such a graph is a transient process, meaning if we do a random walk on the WWW, our random walk may not return to our current website, which causes issues as Google PageRank uses a random surfer to traverse the graph in order to rank pages. At the same time, PageRank also implements other solutions in order to ensure this does not happen, amongst other additions which make algorithm work.

Experimenting with Mixing Times

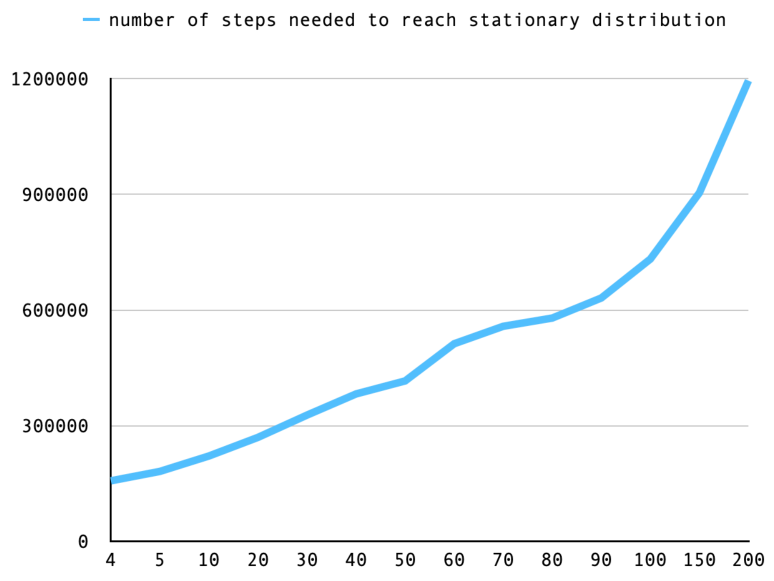
In Appendix A, there is code for a simple stationary distribution simulator used to approximate the mixing time as steps and the approximate stationary distribution. It works by essentially creating a graph from the input, and then simulating a random walk whilst recording the number of steps taken and the number of times it has seen any particular node. Then on each iteration it will calculate proportion of time it has spent at each node as $\frac{v}{T}$ where v is the number of times the random walker has visited the node and T is the number of steps the walker as taken Once the proportions for each of the nodes/states become stable, it will quit, allowing the user to observe the number of steps taken before the program reaches a stable distribution of 0.001 of the expected mathematical values. Due to the unstable nature of random walks and the ever changing values when calculating from the random walk, it is not possible for all the values to perfectly reach the stationary distribution.

In order to test the mixing time and estimate the stationary distribution of a random walker multiple runs of this simulation was used on different graphs. The graphs used were fully connected graphs, as other graphs either did not show signs of trends or were inconsistent due to the volatile nature of a random walker. Furthermore, fully connected graphs are scalable as it involved adding a new node and then adding edges from that node to every other node.

Then the process was done on graphs with number of nodes ranging from 5 to 200 and for each number of nodes the process was repeated 10 times. The results of the analysis is below:

steps required for sd

number of nodes	steps
4	157368
5	181795
10	221776
20	270058
30	327825
40	382668
50	416361
60	512785
70	558007
80	579402
90	631454
100	732421
150	904214
200	1194924



As we can see, the number of steps required to reach an accurate calculation of stationary distribution is quite large even for small numbers. This is generally ineffective for large numbers such as for the WWW which would contain around 1 billion websites as of February 2023 but serves as a proof of concept for smaller numbers. Furthermore, the WWW as a transition matrix is actually a fairly sparse matrix as websites are generally connected to around 30 to 50 out of the 1 billion other websites meaning the transition matrix would be zero in a majority of the cells. This would greatly affect the mixing time of the matrix and would most likely extend the mixing time.

Note that the mixing time is not the number of steps but the number of steps is a reflection of the mixing time over a range of nodes.

Google PageRank

Google PageRank is an algorithm used by Google Search and designed by Larry Page and Sergey Brin to essentially measure the relative importance of a particular webpage. According to Google: "PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites". By running the algorithm, we are able to rank each page according to a relative importance, and this can be used to decide the order in which webpages can be displayed on the search engine.

Intuition

Given all the webpages on the World Wide Web (WWW), rank all the pages according to their "importance".

Whilst there may be many factors that can differentiate pages, such as accuracy/importance of it contents or number of views, the most reliable is the amount of other webpages that link to it. Let us imagine the WWW as a directed graph whose nodes represent all the existing webpages and the edges from each node point towards webpages that it webpage itself contains a link to. The page with the most edges pointing towards it is generally deemed the most important. This however, comes with some concerns such as dangling webpages which will be further discussed in the following chapters of this report. Furthermore, it is also possible to artificially inflate a website's importance by creating many fake websites which simply link to the website. Thus, the PageRank algorithm also needs to take this into account, and the solution is that a webpage's importance is not only determined by how many links from other pages point towards it but also the importance of the other pages as well. Thus, the rank of the website will depend on the ranks of all the other websites as well, ensuring that ranks cannot be manipulated and so that even if someone did try to manipulate it they will only be able to alter a fraction of the graph and not the overall ranks.

There are multiple ways for which we can determine the importance of a website based on the number of links pointing to it. First, we will establish some notation and present some mathematics that will be important for future explanations.

Let the ranking of any page P be $\rho(P)$ and the number of outgoing edges of a website be $\#(P)$. We will also represent an outgoing link as $P_1 \rightarrow P_2$ meaning there is a link from P_1 to P_2.
Then, as we want the ranking of a page to be determined by all other pages we will have:

$$\rho(P) = \sum_{P_i \rightarrow P} \frac{\rho(P_i)}{\#(P_i)}$$

which means the rank of a certain page is equal to the sum of the ranks of all the pages connected to it divided by the number of outgoing edges those pages have. The division is necessary as it will mean a rank will only be high if a the connected webpages have a high rank or if the connecting webpages do not point to very many pages.

Notice that this equation cannot be directly computed as the ranking is based off of the ranking of other pages and those pages are based off the ranking of another set of rankings, meaning it is stuck in a circular loop. Thus we need to find an efficient and robust way to calculate these rankings where every connected edge and its own rankings will affect the ranking of the page we are looking at.

The main way we can overcome this looping reasoning is using the concept of Random Walks and Markov Chains, of which, the main way is to use stationary distribution and the concept of the Random Surfer which utilises the fact that if we allow the random surfer to traverse the graph enough times we fill find the stationary distribution. Alternatively we can use matrices to calculate the stationary distribution as described in Markov Chain chapter earlier in the report. Later in this report, we will discuss the time it takes for a random surfer to fully traverse a graph and compare it to regular matrix calculations for ranking.

Random Surfer

Now since we visualised the WWW as a graph, we are able to apply the theory of random walks to model our page ranks. As we have said that the stationary distribution of a random walker is the proportion of time the walker spends at each state, we can translate it to our random surfer and model the importance of a website as the probability the random surfer opens a random page and follows the link to any given website. Hence our stationary distribution is essentially our page rank. Unfortunately, there are a few structures in the WWW graph make finding the stationary distribution difficult.

In the context of this report, let $N(P)$ denote the number of times the surfer visits webpage P and T be the total number of times the surfer has taken a step, or made a click onto another website. This is the same as the process used for the random walker in the Random Walks chapter and it was said in that chapter that the proportion should converge to the stationary distribution. In this case it means that the ratio $\frac{N(P)}{T}$ will converge as the random surfer travels infinitely.

Damping Factor

In the Markov Chain chapter, we saw that if the chain is a irreducible, aperiodic and stochastic, then it has a stationary distribution. There are some graph structures which cause the graph to not have the correct properties. Structures such as dangling nodes or island structures in Figure 2 show situations where our random surfer will be "trapped" and unable to explore the rest of the graph.

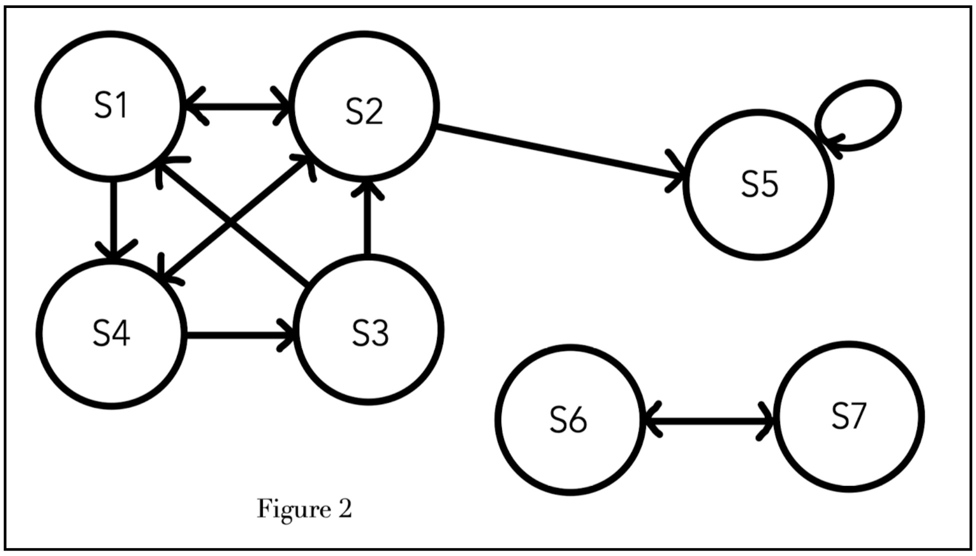


Figure 2

As we can see, if the surfer starts at state 1 and reaches state 5 they will be unable to leave due to the fact that there are no other options other than to loop back to itself. Furthermore state 6 and 7 are disconnected from the rest of states as an island so they will be unreachable.

To solve this, we can introduce a damping factor α. We cannot have the surfer backtrack as it will negate the randomness of our algorithm and cause the data to skew. The damping factor is essentially the probability that the surfer will continue travelling the edges as it usually does. If it does not, it will randomly select a node from the graph, and continue from there. The similar will happen when the surfer reaches a dangling node where there is no other option, the surfer will also pick a new random node and start from there. Generally, the damping factor is set to $\alpha = 0.85$, meaning it has an 85% chance to continue on its path and a 15% chance to jump to a random page.

This strategy fixes the obstacle of dangling nodes and disconnected islands whilst still maintaining the "randomness" of the algorithm. Interestingly, we find that given a graph with dangling nodes and disconnected islands, the ratio $\frac{N(P)}{T}$ will still converge, thus allowing our model with the random surfer to provide the "stationary distribution" and therefore our page ranks.

This was demonstrated using the code in Appendix B which is a modified version of Appendix A. It includes a damping factor and accounts for dangling nodes. When compared with results from Appendix A, the program showed a lower level of accuracy giving values for the stationary distribution within 0.005 of the expected values with a similar amount of steps.

Proper Computation

Putting this strategy to use with the random surfer algorithm in Appendix B, we theoretically will be able to calculate the page rank for any given graph. At the same time, the issue with this algorithm is that it is very inefficient. As we can see in the random walk chapter, getting an accurate reading from simulating the algorithm requires over 1 million steps for a complete graph with 200 nodes. The PageRank algorithm used currently (2023), uses matrix mathematics to calculate the stationary distribution for over 1 billion websites/nodes. Whilst there is much more depth to Google's modification of the matrix, the general idea is that the power method is used to complete iterations of the matrix until it reaches a stationary distribution. The power method is one of the simplest iterative methods of finding the stationary distribution in a matrix, although it is computationally expensive and slow, depending on the mixing time of the matrix. The random surfer model is not without merit, as it serves as a proof of concept and verification that the current PageRank algorithm works.

In order to translate the random surfer model with its damping factor to the current model, we essentially need a probability matrix with all the information required to work around the damping factor and dangling nodes, as well as the requirements for a matrix to have a unique stationary distribution. Requirements such as how every row needs to add up to one and on every node theres a possibility it jumps to a different node needs to be encapsulated in the matrix.

We are able to implement this using the idea of the damping factor. For this project we will set it to $\alpha = 0.85$. For any graph of websites, let M be the probability matrix for a given graph (or the WWW) with n nodes. We must modify the matrix such that for every row, the values add up to 1, and there must be no zero value, as at every node, there is a 15% chance that it switches to a different node, due to the damping factor.

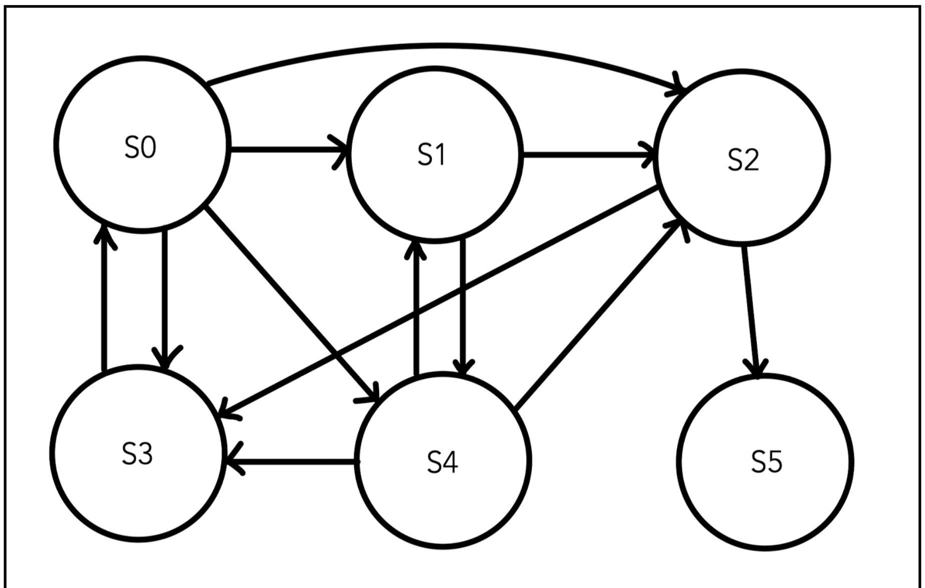


Figure 3

For the following example, we will demonstrate the mathematical method of computing page rank and also compare the results with the code in Appendix B. The graph of websites is in Figure 3. Translating this into a matrix M we get:

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

As this is not a valid matrix, we need to implement the damping factor. To do this, for every row which is a dangling node (a row of zeroes), the probabilities become $\frac{1}{n}$ for the whole row then for each cell c it becomes $c = 0.85c + 0.15 \times \frac{1}{n}$. This gives:

$$M = \begin{pmatrix} 0.0250 & 0.2375 & 0.2375 & 0.2375 & 0.2375 & 0.0250 \\ 0.0250 & 0.0250 & 0.4500 & 0.0250 & 0.4500 & 0.0250 \\ 0.0250 & 0.0250 & 0.0250 & 0.4500 & 0.0250 & 0.4500 \\ 0.8750 & 0.0250 & 0.0250 & 0.0250 & 0.0250 & 0.0250 \\ 0.0250 & 0.3083 & 0.3083 & 0.3083 & 0.0250 & 0.0250 \\ 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 & 0.1667 \end{pmatrix}.$$

Then using the power method we can find the stationary distribution of this matrix which becomes:

$\pi = (\ 0.218,\ 0.129,\ 0.184,\ 0.207,\ 0.143,\ 0.12\ )$

Then, inputing the graph into the simulation code we get the output:

$\pi = (\ 0.2229,\ 0.1282,\ 0.1825,\ 0.2097,\ 0.1393,\ 0.1174\ )$

As seen in the outputs, the code and the mathematical calculation are within 0.01 of each other for each value, meaning the random surfer model and the Markov Chain matrix model are very similar.

Using either/both the information given by the matrix calculation and simulation program, a valid ranking of this small network of websites is: [0, 3, 2, 4, 1, 5].

At the same time, the random surfer took approximately 18,300 steps to reach stationary convergence, whereas doing matrix calculations took around 15 iterations using the power method although it would take less steps if we desired a lower accuracy.

Additional Implementations

Whilst this is the main idea of Google's PageRank algorithm, there are also many other alterations and implementations which allow it to be reliable and accurate.

Link manipulation is a large issue as it greatly influences the reliability of PageRank. To remedy this, Google implements multiple strategies to ensure the page rank remains unchanged, such as a link quality metric. By using various web crawlers, Google is able to determine the relevance of the links to other websites and assign a quality to the links, which will impact the overall page rank calculation although the exact method of update is private to Google. Furthermore, links from reputable sources and authoritative websites will be provided more weight, which prevents a large number of low quality links from boosting the rank of a page. Google also considers a rate at which pages acquire/give out new links. Unnatural spikes in links from websites may be flagged as suspicious, upon which they will most likely be manually reviewed and evaluated.

Whilst this report covers the random surfer model of PageRank, it is said that Google has since changed their PageRank algorithm. According to Google engineer Jonathan Tang: " They replaced it in 2006 with an algorithm that gives approximately similar results but is significantly faster to compute. The replacement algorithm is the number reported in the toolbar and what Google claims as PageRank (it even has a similar name, and so Google's claim isn't technically incorrect). Both algorithms are O(N log N), but the replacement has a much smaller constant on the log N factor because it does away with the need to iterate until the algorithm converges. That's fairly important as the web grew from ~1-10M pages to 150B+". This new algorithm is described as a link analysis approach with seed sets and is described in Google's patent which does not refer to the algorithm as PageRank but much of the algorithm is still only known by Google.

Conclusion

This report has discussed concepts of Markov Chains and Random Walks in relation to Google PageRank. Stationary convergence is one of the most important concepts in reference to how the stationary probability is a direct reflection of a page rank. Mixing time on the other hand, is how fast the Markov Chain is able to reach stationary convergence which will directly reflect the time it takes for PageRank to run.

Random walk concepts were also discussed, and a simulation program was also developed alongside this report in order to explore the mixing time and stationary distribution of a random walker.

Then these concepts were related back to PageRank, where we discuss the random surfer/walker model and its issues with dangling nodes. To solve this, a damping factor is introduced, where there is a chance the surfer jumps to a random node instead of travelling along the links. A modification of the code in the Random Walks chapter was also introduced, which implemented the damping factor alongside the random walk. We then discussed the method in which modern PageRank algorithms can find the stationary distribution using eigenvalues and eigenvectors and found that the results from those calculations come close to the values given in the random surfer simulator, thus validating the process. At the same time, calculating the stationary distribution using the power method compared to the random surfer simulator shows that it would be much more sustainable to use the power method when the number of websites and edges reach large numbers even if the power method is computationally expensive.

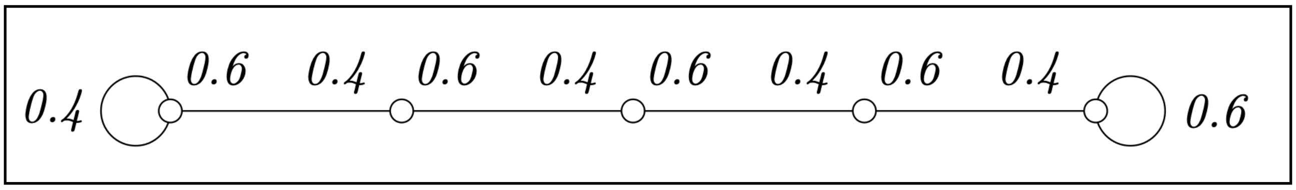
Exercises At The End of BHK

These are the solutions to the exercises at the end of the Random Walk and Markov Chain chapter of the textbook "Foundations of Data Science" by Blum, Hopcroft and Kannan. As there are over 50 exercises, only a few of them were selected to be in this report, namely, the ones that require the fundamentals of Random Walks and Markov Chains as well as the ones most closely related to Google PageRank and topics discussed in this report.

Exercise 4.2: Does $lim_{t\to\infty}a(t) - a(t+1) = 0$ imply that a(t) converges to some value? Hint: consider the average cumulative sum of the digits in the sequence $1^1 0^2 1^4 0^8 1^{16} \cdots$

No, it does not necessarily imply that a(t) converges to some value. Convergence means that the terms of a sequence approach a specific value. For the case in the hint, the limit to infinity does indeed become zero, but looking at the numbers directly, we find that a(t) actually oscillates between 1 and 0, meaning it does not converge.

Exercise 4.3: What is the stationary probability for the following networks?

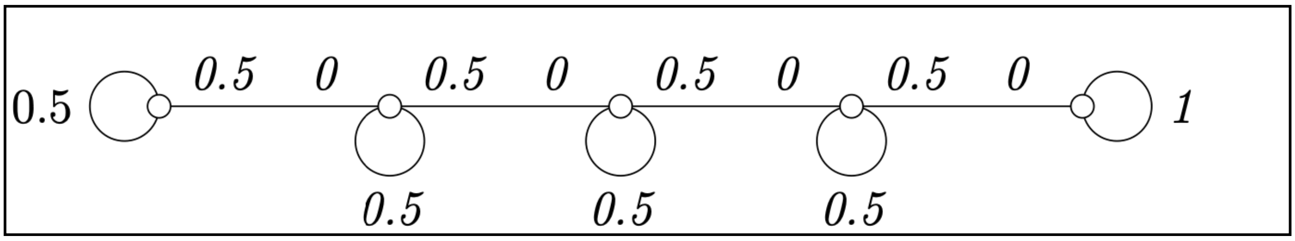


The transition probability matrix for this network is as follows with the ith column and row correspond to the ith node in the above network:

$$P = \begin{pmatrix} 0.4 & 0.6 & 0 & 0 & 0 \\ 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

Then to calculate the stationary distribution, we must find a probability vector such that $\pi P = \pi$. Then we need to solve the linear equations given by $\pi P = \pi$ and the constraint that the elements of π sum to 1. By doing so we are able to find $\pi(x)$ where x is the probability for the given node in the network.

Solving this, we get: $\pi = (0.076, 0.114, 0.170, 0.256, 0.384)$ which is the stationary probability distribution for the above network. For verification we can see that the numbers in π add up to 1 and also that the larger numbers appear towards the right side. This matches with the given network as for each node, the higher probability edge is on the right side. This means if we start our walk at any node, there is a higher chance we move towards the right, and on average, we spend more time on the right side nodes compared to the left during out walk, as represented by the stationary distribution π.



The method for finding the stationary distribution for this network is the same. The transition matrix is:

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

and thus the stationary probabilities are:

$\pi = (0, 0, 0, 0, 1)$

At first glance, it may seem strange that all the probabilities are zero except for the last state, because if we place a random walker at the first state, there is the possibility that the walker never reaches the last state. At the same time, due to the probability, there is a 100% chance the walker reaches the last state, and if the walker reaches the last state, the walker is never able to escape the last state. As the stationary distribution is the proportion over an infinite time, when T steps approaches infinity (and once the walker gets "trapped" in the last state" the proportion of time the walker spends in the last state is also 1.

Exercise 4.4: A Markov Chain is said to be symmetric if for all i and j, $p_{ij} = p_{ji}$. What is the stationary distribution of a connected symmetric chain?

First, we can observe that for a connected symmetric Markov Chain, all the nodes in the given network must have the same degree and every edge must have the same probability, equal to 1 divided by the degree for the node the edges is connected to. If this was not the case, it would not be possible for the probabilities for each node and edge add up to one. This means that for the probability matrix, all columns and rows have the same amount of zeroes and every other number is equal.

Given this, we can deduce that the probability in the stationary distribution must be equal across all states, meaning each state/node has a probability of 1 divided by the number of states/nodes.

Mathematically speaking, this is due to how the calculations for stationary distribution work. Because the solutions are derived from simultaneous equations, made from each row and column of the matrix, we can see for example, for a symmetric chain with n nodes and k edges for each node will be as follows:

$\pi_1 = 1/k\pi_1 + 1/k\pi_2 + \ldots 1/k\pi_n$

$\pi_2 = 1/k\pi_1 + 1/k\pi_2 + \ldots 1/k\pi_n$

$\ldots$

$\pi_n = 1/k\pi_1 + 1/k\pi_2 + \ldots 1/k\pi_n$

$\pi_1 + \pi_2 + \ldots \pi_n = 1$

Solving equations such as this, will always result in $\pi_{1\ldots n}$ being equal. Because it also needs to add up to 1, it means that for a symmetric Markov Chain, the stationary distribution for each node must be 1 divided by the number of nodes, so a chain with 4 nodes will have a stationary distribution of $\pi = (0.25, 0.25, 0.25, 0.25)$.

Exercise 4.21: Find the ε-mixing time for a 2-dimensional lattice with n vertices in each coordinate direction with a uniform probability distribution. To do this solve the following problems.

1. The minimum number of edges leaving a set S of size greater than or equal to $\dfrac{n^2}{4}$ is n.
2. The minimum number of edges leaving a set S of size less than or equal to $\dfrac{n^2}{4}$ is $\lfloor\sqrt{S}\rfloor$

3. Compute $\Phi(S)$.

The conductance is defined as the ratio of the number of edges leaving set S to the minimum size of S and the size of its complement.

Thus:

$$\Phi(S) = \frac{n}{min(|S|, |\overline{S}|)}$$

4. Compute Φ.

To compute this, we need to find the normalised/minimum conductance in each step giving:

$\Phi = min_S \Phi(S)$.

5. Compute the ϵ-mixing time.

In either case, after $O(n^2 ln(\frac{n}{\epsilon^3}))$ steps, $|a(t) - \pi|_1 \leq \epsilon$.

Exercise 4.45: Prove that two independent random walks starting at the origin on a two dimensional lattice will eventually meet with probability one.

First, we need to prove that a random walk on a 2D lattice will return to its origin. The extended proof of this has been done many a times. An extended version of the proof can be found at Durett's textbook Probability: Theory and Examples. For now, let us assume that a random walk on a 2D lattice will always return to its origin, much like how the drunken man will always return home. As we are now interested in two independent random walks, we can focus on the relative distance between the two random walks. By doing so, we can simplify the situation by fixing the position of one random walk and treating it as the origin, and letting the moving random walk to take two steps at a time. As we know that the random walk will always return to the origin, we can say that there is a probability of one that the moving random walk will return to the origin. Therefore, with both the independent random walks moving, the walks will both eventually meet with a probability of one.

References

Boldi, P, Santini, M & Vigna, S n.d., *PageRank as a Function of the Damping Factor* *, viewed 23 November 2023, <https://users.mai.liu.se/larel04/matrix-methods/computer-assignments/PageRankAsFunction.pdf>. (Accessed: 22 November 2023).

Avrim Blum, Hopcroft, J. and Ravindran Kannan (2020). *Foundations of data science*. Cambridge: Cambridge University Press. (Accessed: 22 November 2023).

Aleks Ignjatovic. COMP4121 Lecture Notes. *The PageRank, Markov chains and the Perron Frobenius theory*. (Accessed: 22 November 2023).

Collins, K. (2018). *Google's PageRank algorithm, explained*. [online] Search Engine Watch. Available at: https://www.searchenginewatch.com/2018/10/25/googles-pagerank-algorithm-explained/.
Mathsisfun.com. (2019). *Eigenvector and Eigenvalue*. [online] Available at: https://www.mathsisfun.com/algebra/eigenvalue.html. (Accessed: 22 November 2023).

David Asher Levin, Y Peres, Wilmer, E.L., Propp, J. and Wilson, D.B. (2017). *Markov chains and mixing times*. Providence, Rhode Island: American Mathematical Society. (Accessed: 22 November 2023).

CS787: Advanced Algorithms. (n.d.). Available at: https://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec15.pdf. (Accessed: 22 November 2023).

Slawski, B. *et al.* (2021) *PageRank Update*, *SEO by the Sea* ⚓. Available at: https://www.seobythesea.com/2018/04/pagerank-updated/. (Accessed: 22 November 2023).

Appendix

Appendix A: Code for a random walk simulator, terminates when the proportion of visiting time for each node becomes stable.

```
double round_to(double value, double precision = 0.0001) {
    return std::round(value / precision) * precision;
}

int main(void) {
    size_t n, m, undir = 0;
    vector<vector<size_t>> adj;
    double total = 0;
    vector<double> visits;

    cin >> n >> m >> undir;

    size_t curr = static_cast<size_t>(rand() % n);
    adj = vector<vector<size_t>>(n);
    visits = vector<double>(n);
    size_t from, to;
    for (int i = 0; i < m; i++) {
        cin >> from >> to;
        adj[from].push_back(to);
        if (undir == 1) {
            adj[to].push_back(from);
        }
    }
    cout.precision(3);
    vector<double> previous(n);
    bool check = true;
    int num = 0;
    while(1) {
        total++;
        visits[curr]++;
        cout << static_cast<int>(total) << "  --  " << "visiting
node: " << curr << "   ";
        size_t next = static_cast<size_t>(rand() %
(adj[curr].size()));
        curr = adj[curr][next];
        for (int i = 0; i < n; i++) {
            double t = round_to(visits[i]/total);
            cout << i << ": " << t << "  ";
            if (t != previous[i]) check = false;
            previous[i] = t;
        }
        cout << endl;

        if (check) {
            num++;
        } else {
            num = 0;
        }
        check = true;
        if (num == 100) break;
    }
}
```

Appendix B: Code for a random surfer simulator, terminates when the proportion of visiting time for each node becomes stable and implements a damping factor. Modified version of Appendix A.

```
using namespace std;

// round numbers to particular precision
double round_to(double value, double precision = 0.0001) {
    return round(value / precision) * precision;
}

int main(void) {
    // take input
    // n = number of nodes
    // m = number of edges
    // undir = 0 for directed graphs
    // undir = 1 for undirected graphs
    // undir = 3 for complete graph, ignores m
    int n, m, undir = 0;
    cin >> n >> m >> undir;

    vector<vector<size_t>> adj;
    double total = 0;
    vector<double> visits;
    srand((unsigned)time(NULL));
    size_t curr = static_cast<size_t>(rand() % n);
    adj = vector<vector<size_t>>(n);
    visits = vector<double>(n);
    cout.precision(4);
    vector<double> previous(n);
    bool check = true;
    int num = 0;

    // take inputs
    if (undir != 3 && m != 0) {
        size_t from, to;
        for (int i = 0; i < m; i++) {
            cin >> from >> to;
            adj[from].push_back(to);
            if (undir == 1) {
                adj[to].push_back(from);
            }
        }
    } else {
        for (size_t i = 0; i < n; i++) {
            for (size_t j = 0; j < n; j++) {
                adj[i].push_back(j);
            }
        }
    }

    // send random surfer
    while(1) {
        total++;
        visits[curr]++;

        cout << static_cast<int>(total) << "  --  " << "visiting
node: " << curr << "   ";

        // compute damping factor
        size_t damping = static_cast<size_t>(rand() % 100);
        if (adj[curr].size() == 0 || damping < 15) {
            curr = static_cast<size_t>(rand() % n);
        } else {
            size_t next = static_cast<size_t>(rand() %
(adj[curr].size()));
            curr = adj[curr][next];
        }

        // check previous iteration for similarity
        for (int i = 0; i < n; i++) {
            double t = round_to(visits[i]/total);
            cout << i << ": " << t << "  ";
            if (t != previous[i]) check = false;
            previous[i] = t;
        }
        cout << endl;

        if (check) {
            num++;
        } else {
            num = 0;
        }
        check = true;
        if (num == 10) break;
    }

    // print final values
    for (int i = 0; i < n; i++) {
        double t = round_to(visits[i]/total);
        cout << i << ": " << t << endl;
    }
}
```

The input for the graph in figure 3 is:

```
6 12 0
0 2
0 1
1 2
3 0
0 3
0 4
4 1
1 4
2 3
4 2
2 5
4 3
```