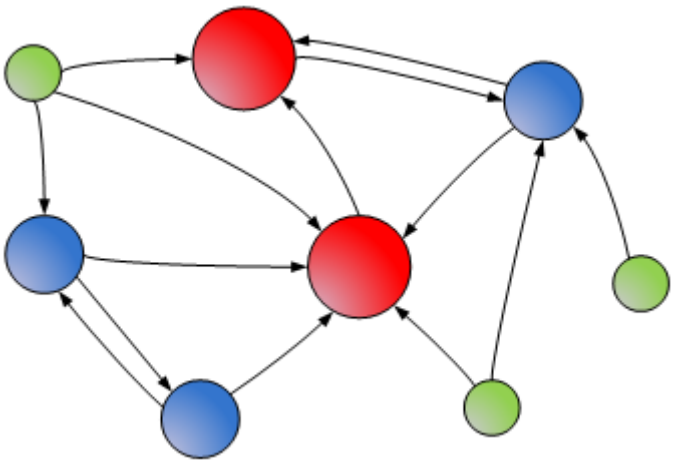
university of groningen

faculty of mathematics and natural sciences

PageRank algorithm, structure, dependency, improvements and beyond



A small Web graph

Bachelor Project Mathematics

Juli 2015

Student: D.A. Langbroek (S2226707)

First supervisor: Dr. B. Carpentieri

Second supervisor: Prof. dr. J. Top

Abstract

I present an explanation about the PageRank algorithm $\pi^T = \pi^T G$. Background of $G = \alpha H + (\alpha a + (1-\alpha)e)\frac{1}{n}e^t$ and construction of the matrix H and G while dealing with dangling-nodes. I cover the sensitivity of π^T to changes in the algorithm and structure of the web. I look at a method to improve upon the PageRank algorithm by changing v^T, and implementing a back button for dangling nodes. Moreover I cover the adaptive power method to decrease the number of computations needed per iteration and extrapolation methods to decreasse the number of iterrations required for convergence. While also looking at the storage of the massive matrices. Consider the accuracy of the ranking of the pages. Including proves of the spectrum of G, the power method iteration and the convergence of $\pi^T = \pi^T G$. Additionally a look at the HITS algorithm and other search engine algorithms. Included an example of the PageRank algorithm on a 15 page web graph and experiment with different α, $\pi^T(0)$, v^T and accuracy arguments.

Contents

1 Introduction to information retrieval

With the invention of the internet a new research area has been opened. For years there have been places for information retrieval, for example libraries. At a library there is usually a method to help you find the book you are looking for. It might be a person, it might be that the books have been sorted in alphabetic order or they have been categorized by topic. Usually there is in fact a combination of these tools to help you find the right book. Nowadays you might encounter a cataloge computer in a library where you can submit a query containing for example author, year, topic etc. and it will produce a list of books. Pages on the internet we would like to do the same. There are however a couple of huge differences between books in a library and pages on the web. First of all there are billions of pages, more then any human based indexing can be applied upon. Especially because the web is dynamic, the pages often change content. Lastly the pages are all self-organized. In a library someone indexes a book, or at least looks at the books to determine if it is even worthy for the collection, on the web anyone can post any page. A method to retrieve useful pages from the web, and a search engine algorithm. But before we look at this newer area of research we will take a look at more traditional methods of information retrieval that you could find in for example a library.

For the traditional information retrieval two basic models first one is the Boolean search engine. The engine is based on exact matching of the query with a document in an index using the Boolean operators: 'or', 'and' and 'not'. Any number of logical statements can be rewritten in these Boolean operators. This method judges a document as relevant when it satisfies the query and irrelevant when it does not. There is no ranking between documents. There is also no partial match. This has a couple of weaknesses, searching on the term 'car' this engine will mark a document about 'automobile' irrelevant, which obviously is not desired. The biggest weakness is synonyms, homonyms and polynyms. It however is also a very effective method on many data sets. Searching in a library's database on an author's name works perfect with this Boolean method (assuming you spell the author right). Boolean method often form the bases for search engines.

The second model is the vector space search engine. A vector space search engine transforms textual data to a matrix in one way or another and then matrix analysis to discover key features and connections between documents.The main goal of this model is to solve the synonym and polysym problem of the Boolean method. A document about a 'car' has probably a lot of similarity with a document about 'automobile' and more advanced vector space search engine should pick up on this fact and present the document about 'automobiles' even when searching on 'car'. Additionally this method derives a relevance ranking. More relevant ranked document are given as higher search results which is increasingly useful the larger the datasets become. However this method is in computational way more expensive then the Boolean and due to the computational expense it scales badly to larger datasets. A nice feature that can be implemented is relevance feedback. Where a user rates how usefull an given document was on his query after which a new ranking can be made according to the users relevance. So this method implements feedback, ranking and relevance for documents.

Of course we could also combine these two information retrieval models to create a search engine that contains the strengths and weaknesses of both, depending on precisely how it is implemented.
To compare different search engines and determine which one is better usually two criteria are used. Precision and recall. Precision is the percentage of retrieved relevant documents for the

query to the number of pages retrieved by the query. Recall is the percentage of relevant pages retrieved for the query to the number of relevant pages in the dataset for the query. Precision determines how many search results were good, and recall determines how good the search was. Additionally search time is a factor. Moreover computational expensiveness and memory usage is nowadays a huge concern as most datasets get bigger. A search engine such as Google has an additional criteria which is customer satisfaction. For the web the dataset is so massive that precision and recall can no longer be applied as we simply do not know how many pages are relevant. Additionally because of the enormous amount of pages and complexity of the web it is important that the first search results are relevant. The first twenty page that Google gives back are the most important ones as users usually give up after the first couple of results and either redefine their query or give up on the search. High precision on a thousand web pages is not relevant if the first 20 pages are wrong. In the end it is impossible to rate how good a search precisely is, but measuring whether the user is satisfied is more relevant and easier.

1.1 Crawling and indexing

A search engine needs a database with pages to give back as search result. Additionally it needs some system to decide which pages from its database to show and preferably to show pages with some form of ranking in such a manner that when it finds a lot of possible webpages the first results are most likely to match what the user wants. To do this search engines use a combination of two categorizing indexes. One is based on the content of the pages to check whether the page matches the search query. The other is some method to rank the pages relative to each other to show which one is more important when they are both found with some search objective. To rank the pages relative to each other Google uses an algorithm called PageRank. This paper will focus thoroughly on this subject. But first we need to take a quick look at how a search engine gets pages in its database and how it categorizes pages based on content. The picture below gives an overview of all the dierent parts of how a search engine works.

The crawler module is a rather short software program that instructs so called spiders to crawl the web. These spiders crawl over webpages and load all the information they can find about these pages in indexes. First we need a bit of definition. Assume we are on page A. A link to page A is called an inlink of page A. When a links from page A to another page it is called an outlink of page A. A spider starts at a random page and then follows outlinks to go to a next page. This is done by adding the pages that were linked towards in the so called crawling index. The crawling index consists of all pages the crawling module is aware of exist and determines which page to crawl next. Of course a spider can be programmed with all kinds of restriction to accommodate the type of search engine you want to make. For instance crawlers can be programmed to only visit .nl sites and add these to the crawling index. Because the web is dynamic, the content of pages is constantly changing, it is important to crawl pages regularly. This for example can be done by simply starting from the top of the crawling index every now and then. But also more sophisticated methods based on importance of site and time since last visited can be implemented. Each search engine can program its own specific crawler to suit their desires. A crawler module has a lot of spiders crawling at the same time to reach as many pages as possible. Besides using outlinks of crawled pages to find new pages to add in the crawling index pages can also manually be added. Google for example has a site where you can upload a url of a page which then will directly be added to the crawling index. Meaning that this page will be crawled even when it has no inlinks. This is often done by people that are afraid there pages may never be found by search engines.

A new page for example usually has no inlinks yet and if no can find the page with a search engine then it never will get one either. Google expressed their desire to index the whole web, or at least as much as possible.

Each page that is crawled by a spider is sent to the indexing module where software programs try to index the content of the page. All this information is stored in dierent indexes. Usually there is an index for the title of a page, one for description etc. The content index is one of the most important indexes that stores all the textual information of the page in compressed form. This is done by an inverted file that is the same as any index of a book. Basically all words that are mentioned in the pages of the database of the search engine are listed in alphabetic order. Then after each word the numbers of the pages in which that particular word occurs in. The number referring to the position of the page in the page index. This index is massive but we are not going to focus on the detail of it.

When a user types in a query the search engine starts to consult its indexes. For example when a user searches the query : Village People and it consults the content index all pages containing the word village or people are retrieved. Then there might be some operator imposed such as the Boolean and to minimize the list of pages to all pages containing both Village and People. Or it may look at pages with Village in the title or some other criteria. At this point we have created a list of pages that in some way (by title/ subject content etc.) are related to the given query. Now there might be a lot of pages that are listed so we impose some form of ranking criteria to give the most important page first. There are two types of ranking criteria. Based on the content of the website by weighting some indexes heavier than others. For example a page with the title Village People is probably more relevant then a page that has the word somewhere in the main text. The other form of ranking criteria is based on importance of sites. This ranking is achieved by the PageRank algorithm and is not based on content or indexes of the search engine. Rather it uses all the pages in the search engines database and their structure of in- and outlinks to determine which page is more important. This will be covered thoroughly later on. The final step is combining ranking scores based on the content score with the ranking scores given by PageRank. As we see there are a lot of stages where the designers of the search engine have to make ranking decisions. Exactly how important is a word in the titel of a page compared to a word in the body. This explains why two search engines with same method but differnt criteria can give the same pages in dierent order of importance to a user. One might value titles more than content, the other might value the web based structure more than content. However the basic structure of ranking a page is done the same for all search engines.

2 Constructing the PageRank algorithm

2.1 Using the structure of the web

The big idea, and breakthrough of Googles PageRank system originates from the question: What ranking is good and when are pages properly ranked?
Google wanted to only use information from the web to decide upon the rankings so no one has to read everything and judge it, is impossible for all the billions of web pages that exist. Instead they had the idea that a lot of information of what people think about pages is already stored in the structure of the web. When you make a link on your page, you are basically saying that you think that that is a useful page (and therefore a good page to find when searching on the topic). And in a similar way, when no one has a link towards a page, it is probably not considered very good. The same way that word of mouth advertising works. And because the structure of the web, i.e. which page links to which page can be found on the web we now have a decent sounding ranking criteria. Of course this method is not perfect and there is always an open discussion what the criteria should be for a page to receive a good/better rank.

The problem of defining good ranking is a subjective one. The one we are making here is still just an opinion or reasonable argument, but not necessarily the truth or just. This system base comes from the idea that a page that has a lot of inlinks is important, and therefore deserves a high PageRank. A page that has few inlinks is considered less important and deserves a lower PageRank. Moreover getting the endorsement of an important page weighs more than the endorsement of an unknown page. Just as the endorsement of the president helps you more than the endorsement of your local baker. This might sound like a circular reasoning, but when looked at it mathematically it turns out to be fine, something that will be discussed in more detail later on. Important pages either get a lot of inlinks, or inlinks from important pages. This is still not yet the final ranking argument for this basic method. When a page makes a lot of recommendations (outlinks), each one should probably be weighed less as this recommendation probably means less. Being complemented by that guy that never speaks is probably more meaningful than being complemented by the guy that squeaked the complement between the others for the people around you.

So now that we have our basis defined as to when we think a page is a good page and deserves a high ranking, let's put it into mathematics.

2.2 Translating the web into mathematics

2.2.1 Summation equation

Brin and Page, the founders of Google and the PageRank algorithm started with a simple summation equation. The PageRank of a page P_i is the sum of the PageRank scores of pages P_j that have a link toward P_i. With the addition that the score P_j is divided by the number of outlinks page P_j has. Let $r(P_i)$ denote the ranking PageRank score of page P_i and $o(P_i)$ denote the number of outlinks page P_i has and $B(P_i)$ be the set of all pages with an inlink to page P_i. Then the original summation equation is:

$$r(P_i) = \sum_{P_j \in B(P_i)} \frac{r(P_j)}{o(P_j)}$$

Of course for this definition to make any sense we need to assign some initial PageRank value to each page, this initial value has been set as $\frac{1}{n}$ where n is the number of pages in the system.

However we still have the problem that we do not know the true PageRank values of each page, and when we calculate a P_i we do not know the correct values of each P_j used in the calculation. We can start the process by updating every page his score by assumming the PageRank value $\frac{1}{n}$ for each page, but then we have not yet found the true value only a first indication/guess. To Solve this they created an iterative process updating every page from page number 1 to page number n. After page 1 was updated this updated value was not immediately implemented in the calculation to derive page 2 and higher numbers. Each page gets an update PageRank score based on the pagerank values of the inlinking page pagerank score previous for their update. After the first iteration, where each page got an updated PageRank score, the second iteration will start using the PageRank scores of iteration 1. This process will be continued until convergence, which will be the final PageRank scores. Let k be the index that denotes the number of iteration then this process can be denoted as:

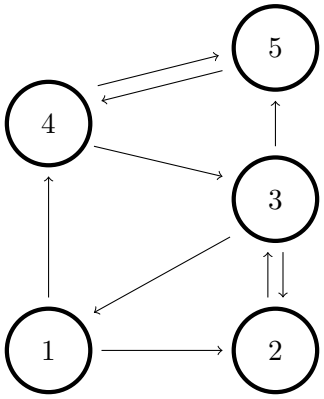
$$r_{k+1}(p_i) = \sum_{P_j \in B(P_i)} \frac{r_k(P_j)}{o(P_j)}$$

Remark: This iterative process was created with the hope that it would converge to some stable and unique scores. We will now rewrite this equation to matrix form, for which later on convergence will be shown. For now it is a system that represents the definiton of good page, a page with a lot of inlinks for which we assume convergence.

This summation equation is rather tedious, calculating the score of 1 page at a time. Using a matrix we can rewrite the system such that every page gets updated at the same time. Create a $1 \times n$ PageRank score vector called π^T that contains the PageRank scores of all the pages. Create a $n \times n$ matrix called H that will consist of the structure of the web. Then the calculation $\pi^T H$ creates an new $1 \times n$ vector called ϕ^T. The first entry of this vector ϕ^T is equal to the sum of all the PageRank scores in π^T multiplied with the first column of H. We could rewrite the values in ϕ^T to a summation formula:

$$\phi_1^T = \sum_{i=1}^{n} \pi_i^T \cdot H_{i,1}$$

The goal is to let ϕ^T be the next iteration of the PageRank score of π^T. Because we work with a set π^T we can only change H. Therefore the goal is to create H in such a way that is displays the structure of the web and we reproduce the summation equation. For this we need that the columns i of H mimic the inlink structure of page i. As a result each row of H represents the outlinks of page i. We had the additional criteria for the summation equation that each value should be divided by the amount of outlinks a page has. Because π contains the PageRank scores and can not be altered we need to implement this weighting of values in H. Creating H is actually relatively simple. First to copy the inlink and outlink structure of the web set the value of entry $H_{ij} = 1$ if page i has an outlink to page j and zero otherwise. After assigning each entry in H with a zero or a one we still need to weigh the value. To do so divide each entry by the number of nonzero entries in that row.
Below is a small example of a web graph, and the belonging H that depicts the structural notation.



$$H = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

We obtain the following general iteration equation with the use of H:

$$\pi_{k+1}^T = \pi_k^T \cdot H$$

This equation is comparable to the original summation formula.
Remark: As mentioned before for the summation formula we want the iterations to converge. To guarantee convergence for a system of the form $\pi_{k+1}^T = \pi_k^T \cdot H$ we need some conditions on H which are not yet qualified. The system denoted here still has several problems that will be handled step by step until we reach an H that will guarantee convergence.

2.2.2 Random Surfer

To reiterate the idea of good PageRank and why this equation and H are good implementations of this idea we will explain the random surfer model that Google used. Random surfer is a method to describe the behavior of a person surfing the web. This method does not accurately describe a person surfing the web but is a good first approximation step. As the word random suggest it is based on a lot of change where a real surfer probably does not let the next page he visits be determined by change. For the system that we are building up to we need some initial PageRank vector containing PageRank values. We want to update those PageRank values by following the matrix H using the idea of random surfer. In this model we start surfing at a page A and then jump to a new page chosen at random from the outlinks of page A (we are actively surfing the web). Choosing where to go completely random between the possible outlinks of each page (equal change to use each outlink). Proceed surfing the web by randomly following the outlink of the page that was visited after page A. Moreover we do this for all pages at the same time. The pages that have been visited more often, are more important (they have more inlinks, or inlinks from other pages that were often visited) and therefore should receive a higher rank. To keep account for the number of times a page has been visited in the system we can use a simple equivalent statement. Each page distributes its own PageRank over its outlinks each iteration. Meaning that after each step we get new PageRank values. We forgot the old PageRank values but only look at all the PageRank value each page received this iteration to determine the new PageRank values. Proceed generating new PageRank values until the

PageRank vector converges. There are some problems with this system which will be addressed to shortly, for now assume that this is a fair method to rank the pages. To apply this random surfing we want to work with changes, meaning that we want to work with probabilities so the values of each row should ad up to 1. Each outlink of a page i has the fair value $\frac{1}{d}$ where d is the number of outlinks that page i has.

2.2.3 Creating G

We have now created a method to produce the matrix H. H_{ij} is $\frac{1}{d}$ when page i has a link to page j and 0 otherwise. This matrix H is extremely sparse as a page has in average about 10 outlinks. Each row which is billions long only contains an order of 10 nonzero elements. As mentioned there are some problems with the random surfer model. The first one is easily seen from the structure of H. When a page has no outlinks there is a row full of zero's in H. Such a page is called a dangling node. Dangling nodes are commonly found on the web, as for example pdf files have this property. The problem here is that once a random surfer surfs towards a dangling node he never leaves. To solve this we behave like a completely random person and jump to a completely random page when we are stuck on a dangling node. So instead of a row full of zero's we fill these rows completely with the value $\frac{1}{n}$ where n is the number of pages in the system.

Similar problem to dangling nodes are sinks and cycles. A sink is a page or group of pages that just as a dangling node gets linked to but has no outlinks. We can get sent to these page but we never leave them. Meaning, we keep ranking these pages more important each step (they get visited) and all other pages less (they are never visited). As these sinks are gathering all the PageRank value that there is to divide we say that our PageRank gets sucked into a sink. A cycle is a group of pages that only links to each other. These cycles have the problem as sinks that once in a cycle we never leave those pages. But additionally we may never get a fair ranking within this cycle. For example a simple cycle of two pages. If one page starts with ranking value 1 and the other 0, then each time we surf these values change. Never getting to a state where the values converge. For a sink we at least reach the point that the ranking values converge. The sink contains a collective ranking of 1 and the other pages 0 but at least the state is stable. This also shows the problem of a page, or group of pages that is never linked towards (which we already fixed with our solution for dangling nodes, because this pages now get linked towards). Sure these pages may deserve low PageRank, but we cannot allow it that their values are always 0 because that would mean that the outlink structure of these pages are irrelevant. Imagine a page A that has a lot of inlinks from pages that have no inlinks themselves. This would than make the PageRank value of page A 0, although a lot of people judge it a good site.

The brilliance here is that we can fix all of the above by doing one simple adjustment. Using the same reasoning as how we fixed the dangling nodes. At each page give it the change to go a completely random page (each page equal change). Google ended up giving it a 15% chance to not use an outlink but go to a randomly new page. Meaning that the surfer can never get stuck at some page as it will eventually jump towards a completely random page. Having such a high percentage to randomly go to a whole new page even fixes the problem of these cycles, though not completely. For instance when a page that has no inlinks, has an outlink to page A and page A has no other inlinks. Then this page A will have a lower PageRank then a couple of Pages B and C, whose only out and inlinks are each other. This just shows that the system is not necessarily perfect, but it works, and give at least good approximation, which the success

of Google is an indication of. Notice as well that all sites with no inlinks will be rated the same, this might be something to improve upon later.

To summarize, we had a matrix H. This matrix had rows filled only with zero's (dangling nodes). We wanted to make the rows of this matrix represent probabilities of using a link. Making H into a stochastic matrix (each row adding up to 1, representing a probability) we replaced the dangling node rows with rows filled with the value $\frac{1}{n}$. The resulting matrix will be called S. $S = H + \frac{1}{n}ae^T$ Where a is a $1 \times n$ vector with 1's at each position associated with a dangling node, and 0's elsewhere and e^T represents the $n \times 1$ vector filled with ones. This matrix still did not suffice so we added a change to jump randomly to any page. This results in the matrix we call G, the Google matrix. $G = \alpha S + (1-\alpha)\frac{1}{n}ee^T$ where $0 \leq \alpha \leq 1$ is the factor that determines how often we randomly jump, or follow an outlink. And e is the vector of $n \times 1$ with all 1's.

2.2.4 Working with G

We are interested in the PageRank π^T whose value add up to 1. Following the iterative process $\pi_{k+1}^T = \pi_k^T G$ we would like to determine how this PageRank vector changes under the structure of the web. This is the general equation of the PageRank problem. This iterative method is the power method. We want this PageRank vector to converge, and to always converge to the same value given any starting values. A logical starting vector would be giving each page a value $\frac{1}{n}$. However we now run into a huge problem concerning our chosen structure. Our matrix G is massive, it has size $n \times n$ where n is the number of pages on the web. n Is at this moment several billions. Calculation of the product $\pi^T \cdot G$ requires n^2 computations (multiplications and additions). While not completely impossible, it is far from practical. Improving this PageRank iteration process is the goal. We can do this in two ways. Making it so that less computations are needed in each iteration, or that the vector converges more quickly so that less iterations are needed.

Looking at the structure of G, we see that this matrix is completely dense. The matrix is completely filled with nonzero elements. Luckily the matrix we started with, H, consisted of a lot of zero's. On average a page does not have an outlink to all other sites, but about 10 outlinks. In addition H had dangling nodes, so on average less than $10n$ of the n^2 entries of H were filled with nonzero elements. Which is a huge difference for n of the size several billion. The matrix H is very sparse. Calculating with zero's is really easy, and saves computations. So rewriting G in terms of H : $G = \alpha(H + \frac{1}{n}ae^T) + (1-\alpha)\frac{1}{n}ee^t$. $G = \alpha H + (\alpha a + (1-\alpha)e)\frac{1}{n}e^t$. This is a so called rank one update. Multiplying by H is a lot easier then multiplying with G, and multiplication of a, e and e^T is computational way less then computing with a matrix. We are down from n^2 computation to a multiple of n computations.

We arrived at the iterative process: $\pi_{k+1}^T = \alpha\pi_k^T H + (\alpha\pi_k^T a + (1-\alpha))\frac{1}{n}e^t$. This equation which resembles the power method for iteration is our basis of operation. This systems works, converges, and is manageable. All of which I will explain in the upcoming paragraphs.

First of all This problem is very close to Markov chain theory. $\pi^T = \pi^T H$ in fact resembles a Markov chain power iteration for a transition probability matrix H. From this Markov chain theory, which is well developed, we know that any starting vector converged to the same positive and unique vector as long as the Markov matrix holds a couple of properties. The matrix should

be stochastic, irreducible, and aperiodic. And irreducible and aperiodic imply primitivity. As already mentioned S is a stochastic matrix, and our modified matrix G is also stochastic. The sums of the values in each row equal 1, all values are positive and it represents a probability. G is also aperiodic, because each page has an probability to link to itself. Because each page is linked to each other page, G is also irreducible. So we can immediately conclude that our iterative process does converge, to a unique positive vector. I have the formal proof written down in section 9.

So we are left with the question why is it manageable, which is split into two sub questions. First, how fast does it converge, are we not iterating for years? Secondly how can we save/ keep track of such a massive matrix, and each iteration.
The speed of convergence for the power method in a Markov chain of the matrix G depend on the two largest eigenvalues of G. Call them λ_1 and λ_2. Because G is stochastic, $\lambda_1 = 1$ and because G is primitive, $\mid \lambda_2 \mid < 1$. Infact $\lambda_2 \approx \alpha$, which i show in detail in section 8. The rate of convergence is $\mid \frac{\lambda_2}{\lambda_1} \mid \approx \alpha$. And α was chosen as 0.85. this means that after 50 iterations $\alpha^{50} = 0.85^{50} \approx 0.000296$. So after 50 iterations we have approximately 3 decimals accuracy. Google reports that this, at least for then was accurate. This might be improved later on, might even be necessary. But 50 iteration surely is manageable, a bit more should not be a problem. As already mentioned, G can be rewritten in terms of H and H is extremely sparse. Meaning that multiplication with H is fairly doable. Even better, this power method is a matrix-free iterative method. Meaning no manipulations on the matrix are done, only one matrix vector multiplication. Moreover The storage of this method is relatively small. Apart from H and a, only π_k^T must be stored, and π_k^T is the only one that changes over time. So we kept the amount of storing the data to a minimum. Combining all gives good argument why to use this equation, and the power method. We can however improve on this system, which we will try to do later. The power method is a relatively slow iterative method. And this system may not be the most accurate representation of "good" pages. This system is manageable, and therefore our basic starting point.

2.2.5 Teleportation matrix

I just described the phenomenon of random surfer, which led to the teleportation matrix : $E = \frac{1}{n} e e^T$. In the system this matrix E gets added to S with a factor α. This matrix says nothing more than that every page has a change to completely randomly go to any other page. The first improvement of our basic PageRank equation is improving this teleportation matrix. To a matrix $E = ev^T$. where v^T is an $1 \times n$ completely dense vector which instead of all its value being $\frac{1}{n}$ in the completely random case, hold weighted values. Some pages get a higher value in v^T, and others lower. The sum of all the elements of v^T should stay 1, as was the case for $\frac{1}{n} e^T$, and each element of this vector v^T should be a value larger than zero. This does not change the difficulty of the system computational wise. The only thing is that an additional n-sized vector has to be saved. So what are the advantages of such a weighted teleportation vector. The idea is that it describes an actual person surfing the web better. Making it into a slightly more intelligent surfer. This v^T is also called a personalized vector or teleportation vector. A person for example is more likely to use an outlink linking to a content filled page, or a well-known page such as Wikipedia or a page from its own country. It would make sense to weight these pages more. Another more extreme example is that such a vector could be used to classify different groups of people. A mathematics student that searches the term "square" might want other types of pages to show up then a tourist. This vector v^T can be used to mimic more the behavior of actual surfers, or groups of surfers. However calculating the according

PageRank vector is still a lot of work, so we can't just make one for each individual. But making some general adjustments based on whether or not the page has any content should improve the system. As well we might make a personalized vector based on languages or other large subgroups. Google recently reported having updated the search engine to show pages that have a special mobile phone page higher when using a mobile device. I do not know how they implemented this but this could very well be done by constructing v^T in such a manner. Another way could be based on the content ranking instead of PageRank of a webpage.

3 Sensitivity

With sensitivity we mean how much the final PageRank vector π^T changes when the Google matrix G changes. By construction $G = \alpha S + (1-\alpha)ev^T$ where α, S and v^T are components that can change or even be chosen. S itself depends on H which solely depends on the structure of the web, but this structure is constantly changing. α And v^T can more or less be freely chosen by the programmer of the algorithm. Therefore we will look at how π^T can change when these components change. What impact has an increased value of α on π^T and more interestingly does it change the ranking of π^T.

3.1 Sensitivity to α

We have seen the influence that α has on G. The closer that α is to 1, the more the structure of the web is used instead of the random teleportation matrix. Now we will look at how π^T changes as a result of changes in α. Moreover we are interested in the changes in value of entries of π^T but more importantly changes in ranking of pages. We will look at the derivative of π^T with respect to α which tell exactly how π^T reacts to changes in α. For small changes in α the derivative is rather precise, for larger changes however we will look at some other method. More precisely we will be looking at each element of the derivative of π^T with respect to α denoted $\frac{d\pi_i^T(\alpha)}{d\alpha}$ for the $i'th$ element. The larger the absolute value $\|\frac{d\pi_i^T(\alpha)}{d\alpha}\|$ is, the more sensitive page i is to changes in α. The derivative shows the direction the value is going in, and when the value is large it apparently changes a lot when α changes. Additionally the sign matters, if $\frac{d\pi_i^T(\alpha)}{d\alpha} > 0$ then the value of page i increases when α increases and decreases when α decreases. If $\frac{d\pi_i^T(\alpha)}{d\alpha} < 0$ then the value of page i decreases when α increases and increases when α decreases. When page i and j have the exact same derivative values then it is safe to assume that for changes of α their relative ranking stays the same. If page i was ranked higher, it stays higher ranked etc. More interesting is when pages i and j have different derivative values. Though this does not necessarily mean their relative positions change, it indicates that for different α it is possible that their relative positions change. But because their values can differ a lot, and we cannot say with certainty the derivative is accurate for large changes in α we do not have certainty. But what we do know for certain is that when pages have different differentiated values, then the choice of α has an effect on the PageRank values. You can make reasonable arguments which pages get higher ranked with larger α and vice versa. For simplicity lets assume that $v^T = (\frac{1}{n}, \frac{1}{n}, \cdots, \frac{1}{n})$. If $\alpha = 0$ then the structure of the web has no effect on the PageRank values and $\pi^T = v^T$. From this we can immediately conclude that if a page i for any $\alpha \neq 0$ has a higher PageRank value then $\frac{1}{n}$, then this value will increase if α increases or in other words $\frac{d\pi_i(\alpha)}{d\alpha} > 0$. However because the derivative is only an approximation we cannot immediately conclude that rankings never change.
Having spoken about what the derivate has as implications on π we actually need to prove that it exists and that the derivative values are of any relevant size. To show this we will use three different theorems that will be proven in detail later on in section 3.1.

Theorem 1: If the PageRank vector is given by: $\pi^T = \frac{1}{\sum_{i=1}^n D_i(\alpha)}(D_1(\alpha), D_2(\alpha), \cdots, D_n(\alpha))$

where $D_i(\alpha)$ is the i^th principal minor determinant of order $n-1$ in $I - G(\alpha)$. Because each $D_i(\alpha)$ is just a sum of products of numbers in $I - G(\alpha)$, it follows that each component in $\pi^T(\alpha)$ is a differentiable function of α on the interval (0,1).

having shown that the derivative exists, the next theorem provides an upper bound for each entry of $\frac{d\pi_i^T(\alpha)}{d\alpha}$ as well as an upper bound for $\|\frac{d\pi_i^T(\alpha)}{d\alpha}\|_1$.

Theorem 2: If $\pi^T(\alpha) = (\pi_1^T(\alpha), \pi_2^T(\alpha), \cdots, \pi_n^T(\alpha))$ is the PageRank vector, then $|\frac{d\pi_j^T(\alpha)}{d\alpha}| \leq \frac{1}{1-\alpha}$ for each $j \in (1, 2, \cdots, n))$. And $\|\frac{d\pi^T(\alpha)}{d\alpha}\|_1 \leq \frac{2}{1-\alpha}$

From theorem 2 we see that for larger α the derivative values can be higher and thus π^T is more sensitive to α. For smaller value of α the system is not that sensitive to α at all. When $\alpha \to 1$ however the upper bound goes to infinity and we cannot make any concrete statement on the sensitivity of π^T with respect to α, only that is can be extremely sensitive. Sadly larger α values are the most interesting ones as these imply a greater use ofthe structure of the web. As mentioned as well before by convergence, there will always be a balancing act for α. Larger α are desired but makes the algorithm converge slower and π^T more sensitive. Because the upper bound was not to useful for large α, and we work with $\alpha = 0..85$ which is relatively large we should look more closely into the sensitivity of π^T which the following theorem is a great asset to.

Theorem 3: If π^T is the PageRank vector associated with $G = \alpha S + (1-\alpha)ev^T$, then $\frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I-S)(I\alpha)S^{-2}$. Additionally the limiting values of this derivative are :
$\lim_{\alpha\to 0} \frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I-S)$ and $\lim_{\alpha\to 1} \frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I-S)^\star$ where $\star$ denotes the group inverse

Additionally in the proof in section 8 we have shown that $\lambda_2 = \alpha$ for G and that S has only one eigenvalue that has the value 1 which was also the largest eigenvalue. We do not know λ_2 for S precisely only that 1 is the upper bound. The Jordan form of the matrix S, $J = X^{-1}SX = \begin{pmatrix} I & 0 \\ 0 & C \end{pmatrix}$ where C is constructed of Jordan blocks corresponding to all eigenvalue of S excluding the eigenvalue 1. It follows that $I - S = X \begin{pmatrix} 0 & 0 \\ 0 & I-C \end{pmatrix} X^{-1}$ and $(I-S)^{-1} = X \begin{pmatrix} 0 & 0 \\ 0 & (I-C)^{-1} \end{pmatrix} X^{-1}$. Therefore we can conclude that the sensitivity of π^T as $\alpha \to 1$ is governed by the size of the entries of $(I-S)^{-1}$ which in turn are bounded by $\|(I-S)^{-1}\| \leq \kappa(X)\|(I-C)^{-1}\|$ where $\kappa(X)$ is the condition number of X. In turn $\|(I-C)^{-1}\|$ is driven by the size of $|1-\lambda_2\|^{-1}$ where λ_2 is the second largest eigenvalue of S. It is reasonable to assume λ_2 is close to α as proven for G however they are not necessarily exactly the same value. What we derive from this all is that the closer λ_2 is to 1, when α is close to 1, the more sensitive π^T is to α.
Concluding: π^T Is not sensitive for small perturbations for small α. For larger α π^T is more increasingly more sensitive to small changes. For α close to 1 π^T is really sensitive to changes in α governed by the size of λ_2 of S.

3.2 Proofs of theorems from section 3.1

In this section proofs of three theorems used in 3.1 are given. The theorems are used to define the derivative $\frac{d\pi_i^T(\alpha)}{d\alpha}$ which is used to determine the sensitivity of π^T.

Theorem 1: If the PageRank vector is given by: $\pi^T = \frac{1}{\sum_{i=1}^{n} D_i(\alpha)}(D_1(\alpha), D_2(\alpha), \cdots, D_n(\alpha))$ where $D_i(\alpha)$ is the i^th principal minor determinant of order $n-1$ in $I - G(\alpha)$. Because each $D_i(\alpha)$ is just a sum of products of numbers in $I - G(\alpha)$, it follows that each component in $\pi^T(\alpha)$ is a differentiable function of α on the interval (0,1).

Proof: For convenience we denote $\pi^T(\alpha)$ as π^T, and the same for $G(\alpha)$ and $D_i(\alpha)$. In section 8 I go more into detail about the spectrum and rank of G which also shows us that the rank of G is n. Therefore the rank of $A = I - G$ is $n-1$. The adjugate of A, $adj(A)$ is the transpose of the cofactor matrix of A. Implying that $A[adj(A)] = 0 = [adj(A)]A$. has then as an immediate result of the Perron-Frobenius theorem rank 1. Furthermore from the Perron-Frobenius theorem we know that each column of $adj(A)$ is an multiple of e, so $adj(A) = e \cdot w^T$ for some vector w^T. Additionally we have that $D_i = adj(A)_{ii}$ and thus we know that $w^T = D_1, D_2, \cdots, D_n$. With similar reasoning we know that each row of $adj(A)$ is a multiple of π^T, hence $w^T = \alpha\pi^T$ where α is some constant. If $\alpha = 0$ then each row of $adj(A)$ is a multiple of 0, and thus $adj(A) = 0$ which is impossible. Therefore we have that $w^T e = \alpha \neq 0$. and thus we have that $\frac{w^T}{w^T e} = \frac{w^T}{\alpha} = \pi^T$. $\square$

Theorem 2: If $\pi^T(\alpha) = (\pi_1^T(\alpha), \pi_2^T(\alpha), \cdots, \pi_n^T(\alpha))$ is the PageRank vector, then $|\frac{d\pi_j^T(\alpha)}{d\alpha}| \leq \frac{1}{1-\alpha}$ for each $j \in (1, 2, \cdots, n))$. And $\|\frac{d\pi^T(\alpha)}{d\alpha}\|_1 \leq \frac{2}{1-\alpha}$

Proof: We know that $\pi^T(\alpha)e = 1$, and the the derivative with respect to α is 0. Using this property we are taking the derivative of $\pi^T = \pi^T G$ where we write G is terms of S.

$$\begin{aligned}
\frac{d}{d\alpha}\pi^T(\alpha) &= \frac{d}{d\alpha}\pi^T(\alpha)(\alpha S + (1-\alpha)ev^T) \\
\frac{d\pi^T(\alpha)}{d\alpha} &= \frac{d\pi^T(\alpha)}{d\alpha}(\alpha S + (1-\alpha)ev^T) + \pi^T(S - ev^T) \\
\frac{d\pi^T(\alpha)}{d\alpha} &= \frac{d\pi^T(\alpha)}{d\alpha}\alpha S + \frac{d\pi^T(\alpha)}{d\alpha}ev^T - \frac{d\pi^T(\alpha)}{d\alpha}\alpha ev^T + \pi^T(S - ev^T) \\
\frac{d\pi^T(\alpha)}{d\alpha} &= \frac{d\pi^T(\alpha)}{d\alpha}\alpha S + \pi^T(S - ev^T) \\
\frac{d\pi^T(\alpha)}{d\alpha}(I - \alpha S) &= \pi^T(S - ev^T)
\end{aligned}$$

For $\alpha < 1$ we have that $I - \alpha S$ is nonsingular as a result from the fact that the characteristic polynomial $p(\alpha S(\alpha)$ will be smaller as one and thus $det(I - \alpha S \neq 0$ and thus the inverse exists. As a result we can rewrite the equation to:

$$\frac{d\pi^T(\alpha)}{d\alpha} = \pi^T(S - ev^T)(I - \alpha S)^{-1} \tag{1}$$

Let e_j be a $n \times 1$ vector whose elements are all 0 excepts the j^{th} elements which is 1. Then:

$$\frac{d\pi_j(\alpha)}{d\alpha} = \pi^T(S - ev^T)(I - \alpha S)^{-1}e_j$$

Additionally we have that $\pi^T(\alpha)(S - ev^T)e = 0$. For the next step we first need to define the following inequality.

Let x be an real vector and $x \in e^{\perp}$ where $e^{\perp}$ is the orthogonal complement of the *spane*. Let y be any real $n \times 1$ vector. Then the following equations hold:

$$x^T e = 0$$
$$|x^T y| = \|x^T(y - e\alpha)\| \leq \|x\|\|y - e\alpha\|$$

This holds for all norms and thus also when we chose specifically the 1 norm for x and the ∞ norm for $y - e\alpha$.

$$|x^T y| = \|x\|_1 \|y - e\alpha\|_\infty$$

We have that $\min_\alpha \|y - e\alpha\|_\infty = \frac{y_{max} - y_{min}}{2}$ where y_{max} is the largest entry in y and y_{min} is the minimal entry of y. This minimum is attained when $\alpha = \frac{y_{max} + y_{min}}{2}$. Let $y = (I - \alpha S)^{-1} e_j$, then we can combine these equations to obtain:

$$|\frac{d\pi^T(\alpha)}{d\alpha}| \leq \|\pi^T(\alpha)(S - ev^T)\|_1 \frac{y_{max} - y_{min}}{2}$$

But because $\|\pi^T(\alpha)(S - ev^T)\|_1 \leq \|\pi^T(\alpha)\|_1 \|(S - ev^T)\|_1$ and we know that $\|\pi^T(\alpha)\|_1 = 1$ and that $\|(S - ev^T)\|_1 \leq 2$ and thus $\|\pi^T(\alpha)(S - ev^T)\|_1 \leq 2$. So:

$$|\frac{d\pi^T(\alpha)}{d\alpha}| \leq (y_{max} - y_{min})$$

Now because S has only positive entries we know that $(I - \alpha S) - 1 \geq 0$ from which we can conclude that $y_{min} \geq 0$. Additionally we know that $(I - \alpha S)e = (1\alpha)e$, these combined give that $(I - \alpha S) - 1e = (1 - \alpha)^{-1} e$. Which lead to the following bound for y_{max}:

$$y_{max} \leq \max_{i,j} [(I - \alpha S)^{-1}]_{ij} \leq \|(I - \alpha S) - 1\|_\infty$$
$$\|(I - \alpha S) - 1\|_\infty = \|(I - \alpha S) - 1e\|_\infty = \frac{1}{1 - \alpha}$$

Which results in the final result we wanted to obtain:

$$|\frac{d\pi_j(\alpha)}{d\alpha}| \leq \frac{1}{1 - \alpha}$$

The second result of theorem 2 is a direct consequence of the same equations when applied on the 1-norm on equation 1.

$$\|\frac{d\pi^T(\alpha)}{d\alpha}\|_1 = \|\pi^T(S - ev^T)(I - \alpha S)^{-1}\|_1 \leq \frac{2}{1 - \alpha}$$

□

Theorem 3: If π^T is the PageRank vector associated with $G = \alpha S + (1 - \alpha)ev^T$, then $\frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I - S)(I\alpha)S^{-2}$. Additionally the limiting values of this derivative are : $\lim_{\alpha \to 0} \frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I - S)$ and $\lim_{\alpha \to 1} \frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I - S)^\star$ where $\star$ denotes the group inverse

Proof: In the proof for theorem 2 we had the following equation:

$$\pi^T(\alpha) = \pi^T(\alpha)(\alpha S + (1 - \alpha)ev^T)$$

Rewriting this equation such that o^T is on the left side, and the multiplying with $(I - \alpha S) - 1$ gives us:

$$\begin{aligned} 0^T &= \pi^T(\alpha)(I - \alpha S(1-\alpha)ev^T) \\ 0^T &= \pi^T(\alpha)(I - \alpha S(1-\alpha)ev^T)(I - \alpha S) - 1 \\ &= \pi^T(\alpha)(I - (1-\alpha)ev^T(I - \alpha S)^{-1} \\ \Rightarrow \pi^T &= (1-\alpha)v^T(I - \alpha S)^{-1} \end{aligned}$$

Taking the derivative with respect to α at both sides where we use the formula $\frac{dA^{-1}(\alpha)}{d\alpha} = -A^{-1}(\alpha)[\frac{dA(\alpha)}{d\alpha}]A^{-1}$:

$$\begin{aligned} \frac{d\pi^T}{d\alpha} &= \frac{d}{d\alpha}(1-\alpha)v^T(I - \alpha S)^{-1} \\ &= (1-\alpha)v^T(I - \alpha S)^{-1}S(I - \alpha S)^{-1} - v^T(I - \alpha S)^{-1} \\ &= -v^T(I - \alpha S)^{-1}[I - (1-\alpha)S(I - \alpha S) - 1] \\ &= -v^T(I - \alpha S)^{-1}(I - \alpha S - (1-\alpha)S)(I - \alpha S)^{-1} \\ &= -v^T(I - \alpha S)^{-1}(I - S)(I - \alpha S)^{-1} \\ &= -v^T(I - S)(I - \alpha S)^{-2} \end{aligned}$$

From this is follows that $\lim_{\alpha \to 0} \frac{d\pi^T}{d\alpha} = v^T(I - S)$.
Furthermore, by definition we know that matrices Y, Z are each others group inverse if and only if $YZY = Z$, $ZYZ = Y$ and $ZY = YZ$. Let $Y(\alpha)$ and $Z(\alpha)$ depend on each others such that:

$$Y(\alpha) = (I - S)(I - \alpha S)^{-2}Z(\alpha) = (I - S)^{\star}(I - \alpha S)^2$$

Then

$$\begin{aligned} Z^{\star}\alpha) &= Y(\alpha) \text{ for } \alpha < 1 \\ Z^{\star}(\alpha) &= I - S \text{ for } \alpha = 1 \end{aligned}$$

And thus it follows that

$$\lim_{\alpha \to 1} Y(\alpha) = \lim_{\alpha \to 1} Z^{\star}(\alpha) = [\lim_{\alpha \to 1} Z(\alpha)]^{\star} = (I - S)^{\star}$$

Which leads to the final equation that we wanted to show:

$$\lim_{\alpha \to 1} \frac{d\pi^T(\alpha)}{d\alpha} = -v^T(I - S)^{\star}$$

□

3.3 Sensitivity to H

Again we look at the derivative of π^T, this time with respect to H. $\frac{d\pi^T(Hij)}{dH_{ij}} = \alpha\pi_i^T(e_j^T - v^T)(I - \alpha S)^{-1}$. Of course we see that α again has a large effect as α portrays how much of an effect H has on π^T in the first place. Additionally we see the same connection with the derivative and $(I - S)^{-1}$ for $\alpha \to 1$. The other result is that π_i^T is part of the derivative from which we can make very logically conclusions. When an important page i, and thus π_i^T is larger, has its structure

changed in H the effect on π^T is larger than for non-important pages. Additionally what we do not see from the derivate but obviously is true. When pages are added or subtracted, and H changes because of that then π^T changes as well and thus is sensitive to it. Moreover the more important/impactful the pages are that are added or removed for the structure in H the larger the effect of changes in π^T.

3.4 Sensitivity to v^T

Again we derive the derivate, this time of π^T with respect to v^T. $\frac{d\pi^T v^T}{dv^T} =$
$(1 - \alpha + \alpha \sum_{i \in D} \pi_i^T)(I\alpha S)^{-1}$ Where D is the set of dangling nodes. Again we have the same dependence on α and $(I - S)^{-1}$ as α determines how much of an impact v^T has on π^T. For $\alpha \to \pi^T$ is very sensitive to changes in v^T. Furthermore we see that the sensitivity depends on $\sum_{i \in D}$. This is very logical, when the dangling nodes combined have a larger portion of the PageRank value, then the dangling nodes are visited more often. Because a dangling node has the row structure of v^T, this means that the larger the PageRank value of the dangling nodes is the more important v^T is for π^T.

3.5 Updating π^T

An upper bound for changes of values in π^T. Let V be the set of all pages that are updated, and let $\tilde{\pi}^T$ be the updated PageRank vector then:

$$\|\pi^T - \tilde{\pi}^T\|_1 \leq \frac{2\alpha}{1 - \alpha} \sum_{i \in V} \pi_i^T$$

Here again we see that when α is small or the pages in V are not important then π^T is not sensitive to changes in the structure of those pages in V. However it fails again for large α or important pages. What we can carefully conclude from this bound is that one person or a small community cannot have a big impact on the total ranking of π^T. They might be able to cheat them selves to the top scores, but they do not have a great impact on the rankings of other pages.
A small caveat for everything discussed about the sensitivity is that we looked at the value of π^T and how those might change. More important is how the pages are ranked. Small changes are usually indicative that the rankings do not change severely though this is not a proven fact. Moreover we can take a simple example where we can change one outlink of one low ranked page which can turn the complete ranking upside down. Problem being that there is not yet a proof how sensitive the ranking of π^T is. What we can say for the normal conditions $\alpha = 0.85$ and $v^T = (\frac{1}{n}, \frac{1}{n}, \cdots, \frac{1}{n})$ is that the top ranked pages have a significant higher PageRank value then $\frac{1}{n}$. Additionally these top ranked pages are the most important ones to have ranked correctly as they are the ones that users rate the search engine on. But because the derivative values of these top pages are not to high with these variables these pages will stay highly ranked. Although π^T might be ranked sensitive to small changes, the top ranked pages are not in the standard PageRank algorithm.

3.6 Cheating rank scores

Above we described the PageRank problem and some of the weaknesses of the system which we even fixed. There is also a problem that does not come from the system itself but from users and webpage designers. When you write a page you would like to see your page get a high page ranking so that people that are searching on your subject actually will find your site. This is of even more importance for pages that try to sell products. Now that we know what information Google uses to produce its PageRank we can influence our page's PageRank value. We can, for instance, create a new page with an inlink to the page which we want to get a high PageRank value and instead of creating one page you could create thousands, all with links to your main site. This will lead to a very high ranking score of the site that will eventually be the number one ranked site on the topic. This artificial increasing of a page rank is a serious problem because that site does not actually deserve this high score. A salesman word is less reliable than the word of a critic. Imagine creating a page all with false content. No one will every recommend this site so it would normally get a low PageRank, which is fair. But if we were to artificially boost this fake site by creating all kinds of links, then all of a sudden it would show up highly in the search results, rendering google ever a completely useless program/site as it does not actually help you find good information on the web. Another method to artificially increase the likelihood of a page being found is hiding all kinds of data on the page. For instance having white text on a white background. No one notices nor knows that there is something written on the page but Googles crawlers will have indexed all content. Imagine a site about dogs that shows up when you search the word 'kitchen'. Because 'kitchen' was somewhere hidden on the site. Google only shows sites that contain in some way the content that has been searched on. Another cheating method would be to offer a page with high ranking score money to have a link to your site (they probably accept as it does not really matter for them. moreover they could hide it). Search engines are in a constant battle with these and other artificial methods of increasing a page PageRank. There are ways to prevent these cheating pages from showing up highly in the search results. Consider the problem that many pages are created to link to one site. Because these sites are solely created to add a link it is safe to assume they dont contain any real content or have a high rank themselves. We could, after calculating the PageRank vector, write some additional algorithms to check for odd events. Have it check for all 'high ranked pages' what the ranking score of its inlinks are and also how many inlinks it has. When the PageRank scores of all its inlinks are really low or it has way more inlinks than a normal site would have we could label this page a cheater and have the algorithm manually change the PageRank to zero instead of the carefully calculated value. But then the cheating site may change its approach and think of a way to bypass this checking algorithm. The search engine would have to updates its algorithm again and the cycle continues. There is no easy fix for this problem and it would probably have to be updated constantly.

4 Date storage

A whole new aspect of PageRank is storing all the data. We mentioned that the matrix G should be rewritten in terms of H due to sparsity. (Storing a zero is not actually necessary. We can store values on all other positions so having more zero's in the matrix is better for storage.) $G = \alpha H + (\alpha a + (1-\alpha)e)v^t$. But how much data is still left to store, and can this actually be done? H has about $10(n-d)$ nonzero entries (each page has on average 10 outlinks), where d stands for the number of dangling nodes in H. Each entry of H is a double in storage terms (representing a chance). The vector a is a sparse vector with d 1's, or integers in program storing. v^T is a completely dense vector containing n doubles and our PageRank vector π^T is a completely dense vector containing n doubles. With n being in the order of billions it should be quite obvious that storing all this data is not a trivial task. H contains by far the most nonzero elements, and therefore is the hardest to store of all these aspects. When trying to compute the PageRank vector on any type of computer at some point it should load the matrix H. Storing H is therefore the first bottleneck. For small portions of the web when H can actually be stored in the main memory of the computer it should be possible to compute the PageRank vector. As mentioned before, the only data that changes at each iteration is the PageRank vector. Assuming it could save everything the first iteration step, it should be able to calculate the convergence state. However even the biggest supercomputer can not quite handle the matrix H when we try to handle the entire web in its main memory. Therefore there are 2 options. Compiling the data in H or somehow calculating H section by section using an efficient input output method from the extern memory. The case that H could be stored in the main memory could also benefit from these operations.

4.1 $D^{-1}L$ decomposition

For this first simple decomposition the random surfer model is assumed. Meaning that in H each outlink of a page is weighed equally. We can then decompose H in the diagonal matrix D and a matrix L consisting of 0's and 1's as $H = D^{-1}L$. Each element of D^{-1} is zero except for all diagonal elements, $d_{ii}^{-1} = \frac{1}{q}$ where q is the number of outlinks of page i. L is a matrix where each column mimics a row of H. Column i of L has a 1 on the position where page i has an outlink and 0's everywhere else. Now both D^{-1} and L have integers as input and an integer is easier to store than a double. A double uses 8 bytes, and integer requires 4 bytes. Storing the data in integers instead of doubles saves half the space. So although we increased the number of elements that must be stored, we added an entire additional matrix D with n integers. It is easier to store computer memory wise. An additional profit comes from the computation of our equation $\pi_{k+1}^T = \alpha\pi_k^T H + (\alpha\pi_k^T a + (1-\alpha))v^T$. Computationally most expensive part is the multiplication between H and π_k^T, requiring $10(n-d)$ additions and $10(n-d)$ multiplications. Replacing H with $D^{-1}L$ we instead have the multiplication $\pi(k)D^{-1}L$. Where $\pi(k)D^{-1}$ requires n multiplications, as D^{-1} is diagonal. Considering the structure of L calculation the result of $\pi_k^T D^{-1}$ multiplied to L requires $10(n-d)$ additions. We saved $(10(n-d)-n)$ multiplications using this decomposition.

4.2 Clever storing

Each row in the matrix H, or L for that matter is extremely sparse. Only storing the position, and value of the nonzero elements is therefore already a huge storage saving concept. But we can improve even further. The first method is the gap method. The gap method uses the structure of inlinks of a page. Usually all inlinks of a page are rather close to each other. For instance a page of a larger site has inlinks from other subpages of this site, consider a site such a

www.rug.nl.. Additionally all pages of this site will be close to each other in the indexing vector and thus the PageRank vector. Of course this is not necessarily true for all pages and their inlinks. The gap method requires less storage for all page where the inlinks are close to each other, and otherwise does not require more storage. What the gap method does is storing the position and value of the first nonzero value of a column (an inlink from the page) and rather than the second nonzero entry position (which might well be a very high number) it stores the distance between these two entries (which should be relatively small following the just stated reasoning). Therefore saving the length of numbers. Meaning that it is very well possible to save the position of all nonzero entries in less than the standard amount of bytes. Illustrating the use of Gap storing a small example.
Imagine that page 1 has inlinks from pages $\{45632150, 45632155, 45632156, 45632161, 245632161\}$. Then instead of storing all the numbers we store the first number followed by a space and then the distance to the next, space again etc.: [45632150 5 0 5 200000000]. Storing a space and the numbers 5,0,6 can be done in one byte, where storing 45632161 alone requires more than a full byte. We see that for pages that have a large distance the method does not necessarily improve.

Reference coding is another method to cleverly store the data. Reference coding is based on the idea that a lot of pages have similar outlinks. For instance all sites on the topic of a specific law have most likely between the outlinks each other and the government site stating this particular law. The idea is that after having stored one page, P_i, its outlinks, defining the outlinks of P_j as P_i and then adding/subtracting the differences. Because the lists have shared sites and are rather similar this usually saves a lot of work. As it takes only 1-2 storage place to copy all the outlinks of P_i. An example to illustrate reference coding.
Imagine page 1 having inlinks from pages $\{5, 8, 9, 12, 15, 43, 100, 142\}$. Page 2 having inlinks from pages $\{1, 5, 8, 12, 15, 142\}$. Then storing all the same inlinks as page 1 has can be done using a singular byte. Having a 1 display that page 2 has the same inlinks as well and a zero indicating that page 2 does not have that inlink. Then this reference byte looks like: [11011001]. Additionally now has to be stored the page 1 which page 2 has but page 1 does not have. Saving a lot of bytes in the progress when the pages such as in the example share a lot of similarities. When they however have no similar pages reference coding does not help. These methods and other similar methods might be used to trim down the amount of storage that is needed.

5 Accuracy

I have mentioned the fact that the system converges. It even converged fairly quickly, about 50 iterations. But what convergence criteria have been used? We don't know how accurate the PageRank vector is that Google produces, as they haven't released this data. We also do not know which stopping criteria for convergence they used, not to mention how precise the criteria are. However we can make some statements about what kind of criteria should be used, and how precise it should be. The problem for this huge PageRank vector is that the important information is the ordering of the pages, not the actual values. Traditionally convergence criteria look at the change of the PageRank vector at each iteration step: $\|\pi_{k+1}^T - \pi_k^T\| < r$ where r is the convergence criteria. Choosing r sufficiently small usually leads to a good convergent criteria. But as mentioned we are now interested in the ordering not the actual values. A simple example why this might be a problem. Let $r = 10^{-10}, \pi_k^T = \{123 * 10^{-11}, 124 * 10^{-11}\}$ and $\pi_{k+1}^T = \{124 * 10^{-11}, 123 * 10^{-11}\}$. Then $\| \pi_{k+1}^T - \pi_k^T \| = 2 * 10^{-11} < r = 10^{-10}$. However, the ordering of the two elements flipped. With a couple of billion pages in the PageRank vector it is likely that very small changes in the values of pages change the orderings. I however think that we are only interested in the first couple of pages as discussed in section 1. All the other ones we do not care too much about. Only the first 100 search results matter (for example). Meaning, that once these orderings do not change it converged, or at least converged far enough to present π as the PageRank.

Luckily looking at the ordering of all the pages might also improve the convergence. It might very well be the case that the ordering does not change anymore, or at least not in any significant way (such that it falls in a convergence criteria). Even though the actual values may differ quite a bit (more than you would safely assume in your convergence criteria). So we are tasked with the new problem of storing the orderings of the PageRank vector, and testing convergence there. Instead of the original storing the actual values and checking convergence. This seems promising as Haveliwala has shown that he could come very close to the ranking of the traditional method with only 10 iterations [10].

However convergence criteria based on ranking has also some problems. Problems closely related to the adaptive power method in section 6.3 where we lock in individual PageRank values when they converge instead of the entire vector. It is hard to guarantee that the found π^T is the correct convergence state with convergence criteria based on ranking. Convergence criteria based on value have this problem as well but then at least we know the values are really close to the proper converged value, or really stable as they did not change a lot. For criteria based on rank these values can be way of. It is possible that a group of pages do not have their relative rankings changed even though their PageRank values change a lot. At this point we cannot speak of a converged state, even though for the convergence criteria it might be. Another problem is that two pages that have the exact same structure should get the exact same PageRank value and rank. Easiest example are two dangling nodes. This also means that the computer and software used to calculate the iterations should be really precise and is not tolerated to make rounding errors, or at least should do the same for the same pages. Otherwise the rankings of these two pages might differ a lot even though they are actually the same. This happend in my experiment at section 13.

Back to the original accuracy arguments based on values and not the rank. There are several billions of pages, so we are working with the order of 10^9. π^T is a probability vector and therefore contains values between 0 and 1. It has been shown that π^T follows a power law distribution

implying that a page with rank r has a PageRank value of approximately $\frac{1}{r}$. Therefore at least accuracy of 10^{-10} is needed to distinguish all elements. But because the values are approximate and some values may be very close to each other it is very possible that even more precise accuracy of 10^{12} is required. Because we know that Google reported convergence after 50 iterations, which implied roughly 3 decimals of accuracy implies that either the chosen α is small, λ_2 is far removed from 1 or the PageRank vector is not very accurate. We do not know how accurate Google is as they never published results but at least the success of their program shows that they give reasonable answers. Decreasing α or having an λ_2 farther from 1 means that the random teleportation is more important in the system. Meaning that the PageRank vector derives further from the structure of the web and becomes more arbitrary. However as only the first couple of pages that a search engine returns are very important to be ranked correctly. I think, but because Google does not release their results I cannot verify this, that 50 iterations is enough to sort out all the top ranked pages. That the order of pages ranked 114581 and page 114582 are reversed is actually not that relevant. When the first results are correct then there is no real need to conjure up to a more accurate π^T. I think this is the case but have no proof besides that 50 iterations would be too few to rank all the pages properly.

6 Improving the PageRank algorithm

6.1 Handling dangling nodes

There are a lot of dangling nodes in the web. Some pages simply do not have outlinks, others are pdf files or a picture or other such documents. There are also the pages that are fetched by a crawler(so it is added to the system) but is not yet visited by a crawler (and so its outlink structure is unknown and set as dangling node). Therefore it is important how these dangling nodes are managed. A very important philosophical question is held within. The way these dangling nodes are implemented can have huge effects on the rankings of the pages. As described with the making of S from H we solved dangling node by having the user teleport to any random page. If for instance we chose to solve the dangling node problem to always jump to the first page we will get a wildly different ranking of the pages as illustrated by effects of v^T discussed before. Originally the designers of PageRank wanted to not include dangling nodes for the problems these created. This however is simply a wrong approach, something the founders agreed on later. Although dangling nodes might not add new information to the system they can very well be important pages on a subject and should be listed as a search result. Then the creators thought about removing the dangling nodes and adding them back in for the last few iteration. With similar problems of fairness to the ranking of pages as result. So then they decided to instead replace a dangling node row of 0's with a completely dense teleportation row. Thereby increasing the density of the matrix and the complexity of the problem. Luckily we could rewrite it as a rank one update but still we made the problem more complex. A better but more complex method (not necessarily more complex to calculate for a computer) to manage these dangling nodes come from their structural similarity. Each dangling node is portrayed by exactly the same vector in H. So why not bundle them up, and replacing all the dangling node by one vector that has the same outlink structure and inlinks from all dangling nodes combined. This works wonderful and saves a lot of computations. However there are 2 downsides to this method. 1: We have no way to rank the dangling nodes (just as in the case where we removed them completely). We can only rank all the dangling nodes combined and this is basically just wrong. 2: the ranking obtained for the non-dangling nodes are biased.

Another interesting method that preserves a way to rank the dangling nodes is based on reordering H. By putting all the dangling nodes at the bottom of H we create a matrix that described the structure of the web just as H did. Let ND denote the set of non-dangling nodes and D the set of dangling nodes.

$$\text{Reordered } H = \begin{matrix} \\ ND \\ D \end{matrix} \begin{matrix} \begin{matrix} ND & D \end{matrix} \\ \begin{bmatrix} H_{11} & H_{12} \\ 0 & 0 \end{bmatrix} \end{matrix}$$

As seen before for computations to solve this sytem we need $(I-\alpha H)$ and its inverse which can be easely computed.

$$(I-\alpha H) = \begin{matrix} \\ ND \\ D \end{matrix} \begin{matrix} \begin{matrix} ND & D \end{matrix} \\ \begin{bmatrix} I-\alpha H_{11} & -\alpha H_{12} \\ 0 & I \end{bmatrix} \end{matrix}$$

$$(I-\alpha H)^{-1} = \begin{matrix} \\ ND \\ D \end{matrix} \begin{matrix} \begin{matrix} ND & D \end{matrix} \\ \begin{bmatrix} (I-\alpha H_{11})^{-1} & -\alpha(I-\alpha H_{11})^{-1}H_{12} \\ 0 & I \end{bmatrix} \end{matrix}$$

And therefore the normalized PageRank vector: $x^T = v^T(I-\alpha H)^{-1}$ can now be denoted as: $x^T = (v_1^T(I-\alpha H_{11})^{-1} \mid \alpha v_1^T(I-\alpha H_{11})^{-1}H_{12}+v_2^T)$ where v_1^T, v_2^T denotes the portioned portions

of v^T according to dangling and non-dangling nodes respectively. Because $(I - \alpha H_{11})$ has a lot of the same properties as the original $(I - \alpha H)$ we have a new algorithm to solve this system with an ordered H. A three step computation is needed:

1. compute x_1^T in $x_1^T(I - \alpha H_{11}) = v_1^T$
2. compute $x_2^T = \alpha x_1^T H_{11} + v_2^T$
3. normalize $\pi^T = (x_1^T, x_2^T)$

The nice thing is, this method actually reduced the amount of computations that are needed. As calculation x_1^T is done by operating on a smaller matrix this part saves calculations.
Using this same idea we can improve even further using sub-dangling nodes. A sub-dangling node is a node for which the first entries in its row are zero. Then ordering the matrix H in several layers of sub-dangling nodes, for example every 1-million pages. On top are the nodes that have an outlink to a page that has a number between 1-million in our matrix H. Then below all these are all nodes that have no outlink to a page number between 1-million but do have an outlink to 1-million-2million etc. until we have ordered the dangling nodes. Now we can write a similar step by step solution. Here $(I-\alpha H)$ has the following form where again H_{11} and others denotes a block matrix with size the number of sub-dangling nodes for that particular interval. b is the number of intervals of sub-dangling nodes. v_i^T and π^T are also reordered so that each value correspond to the same page as before the ordering of sub-dangling nodes.

$$(I - \alpha H) = \begin{pmatrix} I - \alpha H_{11} & -\alpha H_{12} & -\alpha H_{13} & \cdots & -\alpha H_{1b} \\ 0 & I & -\alpha H_{23} & \cdots & -\alpha H_{2b} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & I \end{pmatrix}$$

sub-dangling nodes algorithm:

step 1: reorder the matrix so that the above described structure is achieved.
step 2: Solve for x_1^T in $x_1^T(I - \alpha H_{11}) = v_1^T$
step 3: for $i = 2$ to b compute $x_i^T = \alpha \sum_{j=1}^{i-1} x_j^T H_{ji} + v_i^T$
step 4: Normalize $\pi^T = \dfrac{[x_1^T x_2^T \cdots x_b^T]}{\|[x_1^T x_2^T \cdots x_b^T]\|_1}$

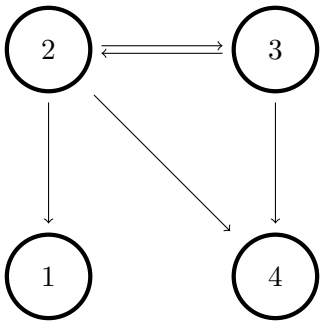
The problem with this method is that we need to find an efficient way to reorder the matrix H. But if we found such a reordering, then we are working on each iteration step with a smaller matrix and a lot of computations will be saved.

6.2 Back button

A method to make the dangling nodes in particular portrait the behavior of an actual person surfing the web more accurately then the random surfer model did is the implementation of the back button. With back button I mean a go to last visited page button. Often users use this button when they visited a dangling node and occasionally on other pages as well. Although this introduces a complicated issue. The method for calculating the PageRank vector is based on a Markov chain, and by definition a Markov chain is memoryless meaning that after each iteration there is no way to find how we got 'here'. And it should be obvious that implementing

a way to store these data is unwanted, as this drastically increases the amount of data that needs to be stored, and makes the whole problem more complex. The tradeoff is a more accurate search system which is worth trying to achieve. We will now look at a simpler method to implement this back button in the system so that we maintain the Markov chain structure by making it such that when a surfer comes to a dangling node he immediately uses the back button. This method requires adding additional rows to the system. The idea is that each dangling node gets a row specific for each inlink it has. Those rows will then be filled with all 0's except for one 1 on the position of the site that linked towards this row. In addition to adding these nodes order the matrix in the way we just discussed in the section dangling nodes where we created $\tilde{H}$. Now $\tilde{H}$ is already stochastic, though it still needs the irreducibility fix. Treating the system as $\tilde{G} = \alpha\tilde{H} + (1-\alpha)v^T$ we can now solve it as we solved G before. Then to obtain the PageRank vector add all PageRank values from nodes that originated from the same (dangling) node. This method will result in different rankings than the original system as desired. The question however is; is this a better ranking or not? This system values pages that are in some way (the closer the better) connected to dangling nodes slightly higher including the dangling nodes itself. But where the idea originated from implies this might not be bad.

Below is an example of implementing this back button idea on a simple web graph with dangling nodes 1 and 4.



The corresponding hyperlink matrix H is:

$$H = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Reordering H such that the dangling nodes are at the bottom and adding extra rows such that each inlink of a dangling node has its own row created $\tilde{H}$. Notice that adding these rows also requires adding the same amount of columns. As a result the rows corresponding to a non-dangling node might change as well to have each element correspond to the appropriate link, as well as the reordering might affect it. This is described more precisely during the reordering process in section 6.1.

$$\tilde{H} = \begin{array}{c} 2 \\ 3 \\ 1_2 \\ 4_2 \\ 4_3 \end{array} \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

As we can see, this $\tilde{H}$ is stochastic. Above is an example of a web graph, a link to the reordering process and a short description of the back button implementation. Next is a short general algorithm for this back button implementation. This algorithm is four steps long and along each step there is an overview of the structure.

1. Reorder H such that
$$H = \begin{matrix} \\ ND \\ D \end{matrix} \begin{matrix} \begin{matrix} ND & D \end{matrix} \\ \begin{pmatrix} H_{11} & H_{12} \\ 0 & 0 \end{pmatrix} \end{matrix}$$
, as done before in section 6.1. ND Is the set of non-dangling nodes, and D is the set of dangling nodes. Here H_{11} is constructed of all the inlinks from non-dangling nodes to non-dangling nodes and H_{12} is a precise buildup of all the links from non-dangling nodes to dangling nodes.

2. Create a bounce back node for each inlink to a dangling node. There will generally be more bounce back nodes then there were dangling nodes before as each dangling node had at least one inlink, this could lead to a drastic increase in size. Let *bb* denote the number of inlinks the dangling nodes combined had, and therefore also the amount of bounce back nodes. More precisely *bb* is equal to the amount of nonzero elements of H_{12}. Constructing the new hyperlink matrix
$$\tilde{H} = \begin{matrix} \\ ND \\ BB \end{matrix} \begin{matrix} \begin{matrix} ND & BB \end{matrix} \\ \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & 0 \end{pmatrix} \end{matrix}$$
Where H_{21} is structurally very similar to H_{12} and BB denotes the set of bounce back nodes. When H_{12} has a nonzero entry on position H_{ij}, then $H_{j,i} = 1$ which lies in H_{12}. And when $H_{ij} = 0$ with i, j such that it lies in H_{12} then so is H_{ji} which is located in H_{21}.

3. Run any algorithm to determine the PageRank values. Because $\tilde{H}$ is usually much larger then H there might be different and more effective algorithms than the normal PageRank algorithm. Notice as well that because $\tilde{H}$ is already stochastic, $\tilde{H}$ takes up the place of S in the Google matrix, thus $G = \alpha\tilde{H} + (1-\alpha)ev^T$ would be the Google matrix. Run the usual power method calculation to obtain $\tilde{\pi}^T$ from the newly obtained G.

4. Lastly the correct π^T should be constructed by collapsing the values from $\tilde{\pi}^T$. Collapsing simply means adding up the values that correspond to the same dangling node and portraying this added value as the PageRank value for the dangling node. Example using $\tilde{H}$ constructed in this paragraph, the following values are not accurate just for indication. Let $\tilde{\pi}^T = (\frac{1}{10}\ \frac{2}{10}\ \frac{3}{10}\ \frac{3}{10}\ \frac{1}{10})$ then $\pi^T = (\frac{1}{10}\ \frac{2}{10}\ \frac{3}{10}\ \frac{4}{10})$.

6.3 Adaptive power method

A method to improve the power method by reducing the amount of calculations that are needed is the adaptive power method. Assume that we somehow know π^T, that is the end result of convergence on the PageRank vector. And then use this to check how far each iteration is from this final PageRank vector. $\| \pi^T(k) - \pi^T \|_1$ is the distance of all the page combined between iteration k and the actual convergence result. Let's now take a look at the microscopic view of each element: $\| \pi^T(k)_i - \pi_i^T \|_1$. This shows how far page i is from the value it converges towards. The idea is that not all pages converge evenly fast (easy example, a dangling node already converged after 1 iteration assuming we applied $\pi^T(0) = \frac{1}{n}e$). This means that actually

unnecessary computations are made. Calculating every iteration of a page while it has already converged just because some other page has yet to converge is dumb labor. The idea is to set a converge criteria for individual pages. When a page falls within this convergence criteria simply lock its PageRank value and never compute it again. This means that the system might end up with a slightly different ranking as it otherwise would. But by choosing the convergence criteria sufficiently small the result should be minimal. However bigger problems result from more theoretical point of view. First of all we do not actually know the values of the converged PageRank vector. Instead we look at the differences between two iterations and apply a convergence criteria. Secondly it has not yet been proven to converge. Thirdly it is possible for a page to converge to a wildly different value. As it is possible for a page to have its PageRank value hardly changed for some iterations while actually not being within the convergence criteria of the actual convergence value. An example is a page that has one inlink from an important page. It might take a while for this important page to achieve its high rank value, meaning the other page is grossly undervalue and might actually seemed like it converged. Another example is a system of 5 pages such that page 1 has a link to page 2, page 2 has a link to page 3 etc.. If we then use the starting vector $\pi^T(0) = (1\ 0\ 0\ 0\ 0)$ we will find that the first 3 iterations the value of page 5 stays 0, as the fastest connection from page 1 to page 5 takes 4 links. Page 5 could easily get the value 0 locked in in such a system while it will not be the true convergence value.

Another idea to improve the power method is to reduce the number of iterations needed (instead of amount of calculation per iteration). This number of iterations needed was connected to λ_2. This method called extrapolation is based on the idea to remove λ_2 when it is too large, and therefore requires a lot of iterations. For simplicity assume G is diagonalizable and $1 >\mid \lambda_2 \mid > \cdots >\mid \lambda_n \mid$. Meaning that the power iteration has the form:

$$\begin{aligned}\pi^T(k) &= \pi^T(k-1)G = \pi^T(0)G^k \\ &= \pi^T(0)(e\pi^T + \lambda_2^k x_2 y_2 + \cdots + \lambda_n^k x_n y_n) \\ &= \pi^T + \lambda_2^k \gamma_2 y_2 + \cdots + \lambda_n^k \gamma_n y_n\end{aligned}$$

Where x_i and y_i are the right and left side eigenvectors of λ_i respectively. And $\gamma_i = \pi(0)x_i$. What can easelly be seen from this notation is that we need that $\lambda_i^k \to 0$. And because $1 >\mid \lambda_2 \mid >\mid \lambda_i \mid \forall i > 2, i \in N$ the bothering factor is indeed λ_2. And the larger λ_2 is the longer it takes to converge. Notice however that we can rewrite this final iterative equation a slight bit to:

$$\pi^T(k) - \lambda_2^k \gamma_2 y_2 = \pi^T + \lambda_3^k \gamma_3 y_3 + \cdots + \lambda_n^k \gamma_n y_n$$

From the assumption that $\mid \lambda_2 \mid >\mid \lambda_3 \mid$ follows that this $\pi^T(k) - \lambda_2^k \gamma_2 y_2$ is closer to the correct PageRank vector π^T. Implying that if we could subtract $\lambda_2^k \gamma_2 y_2$ from the current iterate we propel the system forward. Similarly when $\mid \lambda_i \mid$ is equal or close to $\mid \lambda_2 \mid$ we could also subtract these from the current iterate to really speed up the power method. The problem is calculation $\lambda_2^k \gamma_2 y_2$, or further terms. The first approximation comes from the classic Aitkin error. $\lambda_2^k \gamma_2 y_2 \approx \frac{(\pi^T(k+1) - \pi^T(k))^{\cdot 2}}{\pi^T(k+2) - 2\pi^T(k+1) - \pi^T(k)}$ Where the $^{\cdot 2}$ stand for component wise squaring of the elements. Clear disadvantage of this extrapolation method in this case is the need to store two additional PageRank vector (the last two). Besides this extra storage we also need an extra computations to calculate this extra operation. However if it saves iterations it could be worth it. Especially when it is only implemented every couple of iterations, say 10 iterations. This extrapolation method has an even bigger concern when $\mid \lambda_2 \mid = \mid \lambda_3 \mid$, which happens regularly when λ_2 and λ_3 are each others complex conjugates. In this case the Aitkins error performs poorly. A simple method to improve this extrapolation is to exclude λ_2 and λ_3. However,

this also enlarges the just mentioned problems and inaccuracy. Meaning that this method is expensive and can only be implemented occasionally. Yet it still showed great improvements of 50-300%. [10]

Another method called Blockrank is an aggregation method that lumps sections of the web by hosts. This method tries to reduce both the number of iterations and the number of computations per iteration. In the web there are so called host pages. For example our universities website www.rug.nl. . Rug.nl is a host site as it has many underlying page, for example my faculty page www.rug.nl/fwn. . In the web structure these pages under the same host often have a lot of links to each other and just a few to pages outside the host. The idea is to now for each host apply the PageRank algorithm separately to rank all the pages of the host. And then use the PageRank algorithm to rank all the hosts in the web. Both systems are drastically smaller than the original problem resulting is huge savings of calculations. Finally to compute the general PageRank vector multiply the PageRank value of a host to the PageRank values calculated for all pages within that host. Doing this for all the hosts, and then adding them all into 1 vector gives π^T. During this system all links from pages within the same host are ignored to compute the host PageRank vector. And all links that go to page from another host are ignored during the computation of the PageRank vector for 1 host. Meaning this is an approximation of the actual PageRank vector as not all links (and therefore information) is used.

6.4 Accelerating convergence

Having shown in section 9 that the power method converged we would like to take this method as basis and find some method to speed up the convergence process. The power method is known as a rather slow iteration process. It requires a multiple of n operations each iterations as mentioned in section 2.2.4. The first method to speed the power method up that we will be looking at is implementing Aitking extrapolation which was already hinted at in section 6.3. We were working with the power method that converged when $\pi^T G = \pi^T$ or equivalently $G^T\pi = \pi$. For convenience in the upcoming section I will work with $G^T\pi = \pi$ where I call G^T just G as I am discussing a general method of acceleration for the power method. To keep track of the iterations from now on we will name the converged vector π_∞ which has the property $G\pi_\infty = \pi_\infty$ exactly. π_k refers to the k^{th} iteration, belonging to the formula $\pi_k = G\pi_{k-1}$. The error is equal to $\|G\pi_k - \pi_k\|$ which is zero for π_∞. The Aitkin extrapolation is an approximation for π_∞ based on three subsequent iterations $\pi_{k-2}, \pi_{k-1}, \pi_k$. Based on the assumption that π_{k-2} can be written as a linear combination of the first two eigenvectors of G, remember the first eigenvector v_1 of G is equivalent to π_∞ but π_∞ is normalized. However we can actually only approximate π_{k-2} using the first two eigenvectors and not calculate the actual value. The assumption that we can write π_{k-2} in terms of the first two eigenvectors of G allows us to make an approximation of π_∞. Let us define the following vectors where we used that $G\pi_{k-1} = \pi_k$ and $\lambda_1 = 1$.

$$\begin{aligned}\pi_{k-2} &= v_1 + \alpha_2 v_2\\ \pi_{k-1} &= v_1 + \alpha_2\lambda_2 v_2\\ \pi_k &= v_1 + \alpha_2\lambda_2^2 v_2\end{aligned}$$

α_2 is a constant such that the first equation approximates π_{k-2}. λ_2 is the second largest eigenvalue of G and v_2 is the corresponding eigenvector.
Additionally we define g_i and h_i and f_i whose relation to the iterations will be clear later. In

these equation $\pi^{(i)}$ refers to the i^{th} component of π, similarly for $v^{(i)}$.

$$\begin{aligned} g_i &= (\pi_{k-1}^{(i)} - \pi_{k-2}^{(i)})^2 = \alpha_2^2(\lambda_2 - 1)^2(v_2^{(i)})^2 \\ h_i &= \pi_k^{(i)} - 2\pi_{k-1}^{(i)} + \pi_{k-2}^{(i)} = \alpha_2(\lambda_2 - 1)^2(v_2^{(i)}) \\ f^{(i)} &= \frac{g_i}{h_i} = \frac{\alpha_2^2(\lambda_2 - 1)^2(v_2^{(i)})^2}{\alpha_2(\lambda_2 - 1)^2(v_2^{(i)})} = \alpha_2 v_2^{(i)} \\ f &= \alpha_2 v_2 \end{aligned}$$

Combining these statements with our first assumption of how we have rewritten π_{k-2}

$$v_1 = \pi_{k-2} - \alpha_2 v_2 = \pi_{k-2} - f$$

Because f is defined using only $\pi_{k-2}, \pi_{k-1}, \pi_k$ we have now an approximation for v_1 after only 3 iterations. By normalizing v_1 we obtain π_∞ which we are looking for. Of course this is only a rough approximation but a better one than π_k. Therefore we can use this method to speed up the convergence method by replacing π_k with the newly calculated normalized v_1. The calculation of the Aitkin estimate takes an order of n operations. Because this is relatively few it is a useful method to speed up the convergence of the power method. Additionally it can be applied only periodically for example only every 10 iterations. [2] Shows that the Aitkin extrapolation can speed up the convergence process up to about 40%.

Another method that can be used is quadratic extrapolation. For the Aitkin extrapolation we assumed π_{k-2} could be expressed as a linear combination of v_1 and v_2. For quadratic extrapolation we make a similar assumption, this time π_{k-3} can be expressed as a linear combination of three eigenvectors of G. Additionally we assume G only has three eigenvectors. Of course G has more than three eigenvectors and π_{k-3} can only be approximated with three eigenvectors. Therefore the v_1 that we obtain using the quadratic extrapolation is only an approximation of the real v_1. Similar to the Aitkens extrapolation let us define π_{k-3} and the following iterations as follows:

$$\begin{aligned} \pi_{k-3} &= v_1 + \alpha_2 v_2 + \alpha_3 v_3 \\ \pi_{k-2} &= G\pi_{k-3} \\ \pi_{k-1} &= G\pi_{k-2} \\ \pi_k &= G\pi_{k-1} \end{aligned}$$

Because we assumed G has only three eigenvectors we know that the characteristic polynomial $p_G(\lambda)$ is of the form

$$p_G(\lambda) = \gamma_0 + \gamma_1\lambda + \gamma_2\lambda^2 + \gamma_3\lambda^3$$

Here each γ_j is a constant. Additonally from G we know that $\lambda_1 = 1$ and that each eigenvalue is a zero of the characteristic polynomial therefore:

$$p_G(\lambda_1) = \gamma_0 + \gamma_1 + \gamma_2 + \gamma_3 = 0$$

Or equivalently :

$$\gamma_0 = -\gamma_1 - \gamma_2 - \gamma_3$$

Moreover the Cayley Hamilton theorem states that each square matrix over a commutative ring satisfies its own characteristic polynomial [17]. Thus we know that G satisfies its own polynomial and the following equation holds:

$$p_G(G) = \gamma_0 I + \gamma_1 G + \gamma_2 G^2 + \gamma_3 G^3 = 0$$

Here I is the identity matrix. For any arbitrary vector $z \in \mathbb{R}^n$ it then holds that $p_G(G)z = 0$ as well. Let $z = \pi_{k-3}$ and combine the above equations to obtain:

$$\begin{aligned} p_G(G)\pi_{k-3} = (\gamma_0 I + \gamma_1 G + \gamma_2 G^2 + \gamma_3 G^3)\pi_{k-3} = 0 \\ \gamma_0\pi_{k-3} + \gamma_1\pi_{k-2} + \gamma_2\pi_{k-1} + \gamma_3\pi_k = 0 \\ (-\gamma_1 - \gamma_2 - \gamma_3)\pi_{k-3} + \gamma_1\pi_{k-2} + \gamma_2\pi_{k-1} + \gamma_3\pi_k = 0 \end{aligned}$$

Allowing to express this equation in three parts all dependant on π_{k-3} in the following way:

$$(\pi_{k-2} - \pi_{k-3})\gamma_1 + (\pi_{k-1} - \pi_{k-3})\gamma_2 + (\pi_k - \pi_{k-3})\gamma_3 = 0$$

For convenience we will name each part a bit and putting it in an vector equation in the following ways:

$$\begin{aligned} y_{k-2} &= \pi_{k-2} - \pi_{k-3} \\ y_{k-1} &= \pi_{k-1} - \pi_{k-3} \\ y_k &= \pi_k - \pi_{k-3} \\ (y_{k-2}\ y_{k-1}\ y_k)(\gamma_1\ \gamma_2\ \gamma_3)^T &= 0 \end{aligned}$$

Of course the trivial solution $\gamma = (\gamma_1\ \gamma_2\ \gamma_3)^T = 0$ is not of interest. We can actually constrain the leading term γ_3 without affecting the zero's as we are still left with three equations and three variables. We therefore choose the most convenient constrain which is $\gamma_3 = 1$. obtain the equations:

$$\begin{aligned} (y_{k-2}\ y_{k-1}\ y_k)(\gamma_1\ \gamma_2 1)^T &= 0 \\ (y_{k-2}\ y_{k-1})(\gamma_1\ \gamma_2)^T &= -y_k \end{aligned}$$

Because we know that the first eigenvalue of G is $\lambda_1 = 1$ we can eliminate this solution from the characteristic polynomial to find the polynomial $q_G(\lambda)$ whose zeros are λ_2 and λ_3. Because $\lambda_1 = 1$ is a solution of the characteristic polynomial, we know that $\lambda - 1$ is a factor of the equation. Therefore dividing $p_G(\lambda)$ by $\lambda - 1$ gives us $q_G(\lambda)$:

$$q_G(\lambda) = \frac{\gamma_0 + \gamma_1\lambda + \gamma_2\lambda^2 + \gamma_3\lambda^3}{\lambda - 1} = \beta_0 + \beta_1\lambda + \beta_2\lambda^2$$

Where β_i is a constant that can be rewritten in terms of our original constants γ_j.

$$\begin{aligned} \beta_0 &= -\gamma_0 = \gamma_1 + \gamma_2 + \gamma_3 \\ \beta_1 &= \gamma_2 + \gamma_3 \\ \beta_2 &= \gamma_3 \end{aligned}$$

By the Cayley-Hamilton theorem it follows that for any vector $r \in \mathbb{R}^n$, $q_G(G)r = v_1$. We have now arrived at an equation where we see the vector that we are trying to find, v_1. Let $r = \pi_{k-2}$ and substituting the equations gives a closed form solution for v_1:

$$\begin{aligned} v_1 = q_G(G)\pi_{k-2} = (\beta_0 I + \beta_1 G + \beta_2 G^2)\pi_{k-2} \\ v_1 = \beta_0\pi_{k-2} + \beta_1\pi_{k-1} + \beta_2\pi_k \end{aligned}$$

Quadratic extrapolation requires an order of n operations. By applying the quadratic extrapolation periodically it can help to speed up the convergence process of the power method. Remember the v_1 obtained from the quadratic extrapolation is only an approximation of the real v_1 of G. However the approximation is better than the iteration of any of the π_i used in the quadratic extrapolation algorithm. Combined with the fewer operations required then one iteration of the power method. [2] Obtains a 23% reduction of time required to reach the convergence state.

6.5 Web structure changes/Updating π^T

Google dance is the nickname to the monthly update of Googles PageRank vector. To show the importance of updating the PageRank vector regually I recall the dynamic property of thee web (it takes a lot of effort to calculate so you rather not). A study has shown that about 40% of the pages change within a week. About 25% of the ·com websites change daily. Larger sites change more often and more extensively [13]. Besides every month a lot of new pages are created, and others closed or change ownership. Additionally there are sites that change all the time, for example news sites. For now we do not take these sites in consideration and update the PageRank vector only periodically. Google stated not to use the PageRank vector of last month to determine the PageRank vector of this month as it did not yield any improvement. Last months PageRank value surely contains some useful information for the PageRank of this month and it would be a shame to not incorperate the work that has already been done. The goal here is to use last months PageRank vector to compute the new PageRank vector with a little less effort than starting from scratch. Usually G changes quite a bit each month. It grows or shrinks (grows normally speaking) as new pages are added or removed. And the rows change because content of page change and with that often their linking structure. Though for a lot of sites it is safe to assume that outlinks and inlinks have a large overlap between two months. We will look at separate cases. First link-updating problem where G stays the same size but some rows may change (only linking structure changes, no added or removed pages). After that we look at the page-updating problem where the size of G is no longer constant as pages get added or removed. This will basically always coincide with link-updates as well which we will treat as a special case. For convenience we call last months Google matrix Q.

There is a well know formula for exact link-updating problems based on updating one row at a time. The method works on the singular matrix $A = I - Q$. We then need to find $A^{\#}$ which is the group inverse of A. $A^{\#}$ is a unique matrix satisfying the following three equations: $AA^{\#}A = A, A^{\#}AA^{\#} = A^{\#}, AA^{\#} = A^{\#}A$.

Proof. Suppose the i^{th} row of Q changes to obtain the i^{th} row of G. Then $g_i^T = q_i^T - \delta^T$ where δ^T is a vector that views the changes. If π^T and ϕ^T denote the PageRank vectors of G and Q respectively, and $A = I - Q$. Then $\pi^T = \Phi^T - \epsilon$ where $\epsilon = \left(\frac{\phi_i^T}{1+\delta^T A_i^{\#}}\right) \delta^T A^{\#}$ and i indicates the i'th column. To handle multiple row changes on Q apply this formula sequentially one row at the time. Updating the group inverse sequentially. Meaning that updateing $(I - Q)^{\#}$ to $(I - G)^{\#}$ is as follows:

$$(I - G)^{\#} = A^{\#} + e\epsilon^T + [A^{\#} - \gamma I] - \frac{A_i^{\#}\epsilon^T}{\phi_i}$$

where $\gamma = \frac{\epsilon^T A_i^{\#}}{\phi_i}$ and e is a column of ones. □

Although this works theoretically, it is far from optimal. When each row would change it requires an order of n^3 floating point operations. Other updating formulas exist but they are all based on the same general idea, with the same problem that when all rows are changed a number of operations of order n^3 are needed. Besides, they are only useful for the simpler link-updating problem. And therefore we can conclude that up till now there is no improvement to be made by implementing last month matrix in the page-updating problem.

Because the page-updating problem coincides with the link-updating problem it appears to be better to start from scratch. Continue concentrating on the link-updating problem. We might

still be able to speed up the process. For instance use the old PageRank vector Φ^T as starting vector for the iteration process. If G and Q are close(few changes), we can expect π^T to be close to ϕ^T and therefore require less iteration than starting with the conventional starting vector $\frac{1}{n}e$. However this sounds way more attractive than that it in reality is. Recall, the rate of convergence is $R = -log_10 \mid \lambda_2 \mid$. R portrays how many digits of accuracy can be gained each iteration. Suppose that component wise entries $\mid G - Q \mid$ are small enough so that π^T agrees with ϕ^T in the first significant digit. And suppose we want to calculate ϕ^T for twelve significant digits. Assume $\mid \lambda_2 \mid \approx 0.85$. Then $R \approx 0.07$, and because about $\frac{1}{R}$ iteration are required to gain each additional significant digit we save about 14 iterations. Which is great, but for twelve significant digits we expect to need $\frac{12}{R} \approx 172$ iterations. So in reality saveing about 8%. Of course each improvement is great, it just sounded a lot better than it actually proves to be. However because this does not in any way help for the page-updating problem which is the problem that in reality occurs and is hard to implement it seems that we are yet to find some value from last month PageRank vector.

If instead of aiming for the exact (up to the accuracy criteria) PageRank values we look at an approximation then there are still chances for last month PageRank vector. Using aggregation we have a method that can be implemented for both link and page-update problems. The idea of approximate aggregation is to use the known distribution of $\Phi^T = (\phi_1\ \phi_2\ \cdots\ \phi_m)$ together with the updated transition probabilities in G to build an aggregated Markov chain. Having a transition probability matrix C that is smaller in size than G. The stationary distribution ξ^T of C is used to generate an estimate of the true updated distribution π^T as outlined below. The state space S of the updated Markov chain is first partitioned into two groups as $S = L \cup \bar{L}$, where L is the subset of states whose stationary probabilities are likely to be most affected by the updates (newly added states are automatically included in L, and deleted states are accounted for by changing affected transition probabilities to zero). The complement $\bar{L}$ naturally contains all other states. The intuition is that the effect on the stationary vector of perturbations involving only a few states in large sparse chains (such as those in Google's PageRank application) is primarily local, and as a result, most stationary probabilities are not significantly affected. Deriving good methods for determining L is a pivotal issue.

7 PageRank as a linear system

We have found solutions for the PageRank problem and methods to derive π^T. This was based on the eigenvector problem, finding the dominant eigenvector for the matrix G. This followed from the normalized PageRank equation: $\pi^T(\alpha S + (1+\alpha)ev^T) = \pi^T$. This equation can be rewritten to the following linear system : $\pi^T(I-\alpha S) = (1-\alpha)v^T$ where the fact that $\pi^T e = 1$ is used (π^T is the vector that contains all the PageRank values and all these values combined should add up to 1 as we have normalized this vector). Instead of solving the eigenvector problem with the power method we could also look at solving it as a linear system with direct algorithms. To do this we would need some properties of $(I - \alpha S)$. But because S can be rather dense, depending on the amount of dangling nodes we rather work with H which is guaranteed very sparse. Filling in the definition of H in the linear system we get: $\pi^T(I-\alpha H-\alpha a v^T) = (1-\alpha)v^T$. We now have the additional term $\pi^T(-\alpha a v^T)$ on the left side of the equation. But because α is only a factor this can be rewritten to $-\alpha\pi^T a v^T$. Here $\pi^T a$ is a constant value that we will call γ, which denotes the combined values of the PageRank scores of all the dangling nodes. We would like to choose this value as convenient as possible which would mean $\gamma = 1$, which can be done without consequence for the system if we replace π^T with some arbitrary vector x^T that holds all the PageRank values but is not normalized. In fact x^T is multiplied by a factor that portraits the same difference between 1 and the actual value of $\pi^T a$. After solving the system for x^T, use $\pi^T = \frac{x^T}{x^T e}$ to find the normalized PageRank vector. Now we can rewrite the linear system to: $x^T(I - \alpha H) - \alpha v^T = (1-\alpha)v^T$, or in other words $x^T(I - \alpha H) = v^T$ where we let $\pi^T = \frac{x^T}{x^T e}$ produce the PageRank value. Additonally $(I - \alpha H)$ has some rather nice properties.

1. $(I - \alpha H)$ is an M-matrix, a matrix whose eigenvalues all have positive real part, and all its offdiagonal entries have values ≤ 0. $(I-\alpha H)$ is nonsingular(inverteble) and its inverse is nonnegative.

2. $(I - \alpha H)$ has rank n, where n is the size of H.

3. The row sums of $(I - \alpha H)$ are 1 for dangling nodes and $(1 - \alpha)$ for nondangling nodes. And $\|(I - \alpha H)\|_\infty = 1 + \alpha$.

4. The row sums for $(I-\alpha H)^{-1}$ are 1 for dangling nodes and $\leq (1-\alpha)^{-1}$ for the nondangling nodes. And therefore $\|(I - \alpha H)^{-1}\|_\infty \leq (1 - \alpha)^{-1}$.

5. the condition number $k_\infty(I - \alpha H) \leq \frac{1+\alpha}{1-\alpha}$.

6. The row of $(I - \alpha H)^{-1}$ corresponding to the dangling node i is q_i^T, where q_i is the i^{th} column of the identity matrix.

As mentioned, both the eigenvector method and this linear method result in the same π^T. So what makes either system a better choice? When solving a small system direct methods for solving the linear system are much faster than solving the power method. But most importantly the rate of convergence for the power method is dependent on α. And in fact we actually want α as large as possible because then the system is more dependent on the actual structure of the web. The solution time of the direct method is independent of α so can work with larger α then with the power method. Sadly however the sensitivity issue's remain for large α also in the linear system.

Now a short and quick proof which shows that solving the PageRank problem and the linear system are equivalent. And that working with x^T instead of π^T does not influence the outcome.

Proof. π^T is the PageRank vector if it satisfies $\pi^T G = \pi^T$ and $\pi^T e = 1$. Because of the way π_0^T and G are constructed, $\pi^T e = 1$ is always true. Showing that $\pi^T G = \pi^T$ is equivalent to $\pi^T(I - G) = 0^T$ where 0^T is the $1 \times n$ vector full of zero's. Because $\pi^T = \frac{x^T}{x^T e}$ the difference between π^T and x^T with some factor, all the entries have the same relative values to each other in both vectors. Thus solving $\pi^T(I - G) = 0^T$ is the same as solving $x^T(I - G) = 0^T$.

$$\begin{aligned} x^T(I - G) &= x^T(I - \alpha H - \alpha a v^T - (1 - \alpha) e v^T) \\ &= x^T(I - \alpha H) - x^T(\alpha a + (1 - \alpha) e) v^T \\ &= v^T - v^T = 0^T \end{aligned}$$

where $x^T(I - \alpha H) = v^T$ comes from rewriting in the linear system.
And $x^T(\alpha a + (1 - \alpha)e)v^T = v^T$ because

$$\begin{aligned} x^T(\alpha a + (1 - \alpha)e) &= (1 - \alpha)x^T e + \alpha x^T a) = x^T e - \alpha x^T(e - a) \\ &= x^T e - \alpha x^T H e \quad \text{Because } He \text{ is a } n \times 1 \text{ vector with values 1 for} \\ &\quad \text{non-dangling nodes and 0 for dangling nodes, or in other words } (e - a). \\ &= x^T(I - \alpha H)e = v^T e \quad \text{Again by how } x^T(I - \alpha H) \text{ has been defined} \\ &\quad \text{in the linear system.} \\ &= 1 \quad \text{Because } v^T \text{ is constructed so that all the sum entries of } v^T \text{ are 1.} \end{aligned}$$

□

8 Spectrum of G

At several occasions we mentioned the fact that the matrix G has a lot of eigenvalues with the following property: $\lambda_1 \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$ and additionally $\lambda_1 = 1$. I will now show why this is the case, and prove it. First I will show that a $\lambda_j = 1$ for G exists, and follow it up by showing that this is the largest eigenvalue of G.
G is a row stochastic matrix and as a result the sum of all elements in each row is 1. By definition λ is an eigenvalue of G and v is an eigenvector corresponding to λ of G if $Gv = \lambda v$. When we chose $v = \{1, 1, \cdots, 1\}^T$ then $Gv = v$, which is equal to $Gv = \lambda v$ for $\lambda = 1$. Therefore we know that there is an eigenvalue for G whose value is 1. By using the norms of matrices we can bound the values of the eigenvalues. $Gv = \lambda v$, therefore $\parallel Gv \parallel = \parallel \lambda v \parallel$ for all matrix norms. $\parallel Gv \parallel \leq \parallel G \parallel \parallel v \parallel$ and $\parallel \lambda v \parallel = \|\lambda\| \parallel v \parallel$. Thus $\|\lambda\| \parallel v \parallel \leq \parallel G \parallel \parallel v \parallel$, and therefore we have that $\|\lambda\| \leq \parallel G \parallel$ for all matrix norms. One matrix norm that we can apply is the infinity norm $\parallel G \parallel_\infty = \max_{1 \leq i \leq n} (\sum_{j=1}^{m} \|g_{ij}\|) = 1$. Because we found an eigenvalue whose value is 1, we know that all other eigenvalues in absolute form are bounded by this eigenvalue. Moreover I used $\lambda_1 = 1$ was the dominant eigenvalue and that $|\lambda_2| \leq \lambda_1$ where λ_2 in absolute sense is the second largest eigenvalue of G. That $\lambda_1 = 1$ is a dominant eigenvalue of G follows directly from *Perron's Theorem* which states:
Perron's theorem: If A is a positive $n \times n$ matrix, then A has a positive real eigenvalue r with the following properties:

1. r is a simple root of the characteristic equation.
2. r has a positive eigenvector x.
3. If λ is any other eigenvalue of A, then $|\lambda| < r$.

We happend to have found this r for G, $r = \lambda_1 = 1$. The corresponding eigenvector $v = \{1, 1, \cdots, 1\}^T$ is indeed positive and we had already shown that every other eigenvalue was smaller or equal to λ_1.
Lastly I mentioned that $\lambda_2 = \alpha$ which is important for the rate of convergence. The proof for this fairly long. First I will prove that α is the upper bound for $|\lambda_2|$. Further I will prove that if S has at least two irreducible closed subsets then $\lambda_2 = \alpha$ and end with showing that S indeed has two irreducible closed subsets. The following proof is inspired by [1]. I included this proof especially because it shows the structure of G and the importance of the special features such as irreducibility nicely. The following theorems will be used:

1. *Ergodic Theorem:* If A is the transition matrix for a finite Markov chain, then the multiplicity of the eigenvalue 1 is equal to the number of irreducible closed subsets of the chain.
2. *Theorem 4:* If x_i is an eigenvalue of A corresponding to the eigenvalue λ_i, and y_j is an eigenvector of A^T corresponding to λ_j, then $x_i^T y_j = 0$ as long as $\lambda_i \neq \lambda_j$.
3. *Theorem 5:* Two states corresponding to the same irreducible closed subset (class) have the same period. In other words, the property of having period d is a class property.

Additionally we need the formal definition for a closed subset and irreducible closed subset of a Markov chain. A set of states C is a closed subset of a Markov matrix M if and only if $i \in C$ and $j \notin C$ implies that $M_{ij} = 0$. A set of states C is an irreducible closed subset of M if and

only if C is a closed subset, and no proper subset of C is a closed subset.
To recall $G = \alpha S + (1-\alpha)ev^T$. Now follows a short list of all the involved matrices to name the appropriate eigenvalues and eigenvectors, as well as some important earlier discussed properties.
Remark: In the proof $G^T = (\alpha S + (1-\alpha)ev^T)^T$ will be used instead of G, because the characteristic polynomials of G and G^T are the same, the eigenvalues are also the same and the proof also applies to G. Additionally transposes of the other matrices will be used.
Remark: Every eigenvector we come upon is normalised, i.e. $\| x_i \| = 1$. The derivation of the eigenvalues listed below follows directly from the proof before in the begining of section 8 for stochastic matrices.
G, S and E are all row stochastic matrices, and all row stochastic matrices M have the property that $Me = e$.

1. Let λ_i denote the i^{th} eigenvalue of G^T and let x_i be the corresponding eigenvector. Then $\lambda_1 = 1$ and $|\lambda_n| \leq \cdots \leq |\lambda_2| \leq 1$.

2. Let γ_i denote the i^{th} eigenvalue of S^T and let y_i be the corresponding eigenvector. Then $\gamma_1 = 1$ and $|\gamma_n| \leq \cdots \leq |\gamma_2| \leq 1$.

3. Let E denote ev^T. Let μ_i denote the i^{th} eigenvalue of E^T and let z_i be the corresponding eigenvector. Because E is stochastic and rank one (every row is the exact same) $\mu_1 = 1$ and $|\mu_2| = \cdots = |\mu_n| = 0$.

Proof that $\lambda_2 \leq \alpha$: First the two special cases, $\alpha = 1$ and $\alpha = 0$.
If $\alpha = 0 \Rightarrow G^T = E^T$, and thus $\lambda_2 = 0$ as listed above.
If $\alpha = 1 \Rightarrow G^T = S^T$, and thus $\lambda_2 \leq 1$ as listed above.
Now the more general case that $0 < \alpha < 1$ will be done in several steps. First showing that $|\lambda_2| < 1$. By the Ergodic Theorem it follows that if we can show that G has only one irreducible aperiodic sub chain C, then the multiplicity of the eigenvalue 1 is 1, directly implying that $|\lambda_2| < 1$. We know that G is completely dense and $G_{ij} \neq 0$ by construction. If U is a set of states from G but not all the states, then there are i, j such that $i \in U$ and $j \notin U$ with $G_{ij} \neq 0$, and therefore U is not closed. Therefore the only possible closed subset C of states in G is G itself. As proven before G is irreducible and aperiodic. Additionally G itself is closed by definition and unique.
Next are some properties about x_2, the second normalized eigenvector of G corresponding to λ_2. A direct implication to the fact that the multiplicity of the eigenvalue $\lambda_1 = 1$ is 1, is that x_2 is orthogonal to x_1. Because x_1 is the normalized vector e, and therefore in some sense is equivalent to e, it follows that $e^T x_2 = 0$. Therefore:

$$E^T x_2 = ve^T x_2 = 0$$

λ_2 is an eigenvalue of G^T and the following equations hold, implemented with $E^T x_2 = 0$

$$\begin{aligned} G^T x_2 &= \lambda_2 x_2 \\ \alpha S^T x_2 + (1-\alpha) E^T x_2 &= \lambda_2 x_2 \\ \alpha S^T x_2 &= \lambda_2 x_2 \\ S^T x_2 &= \frac{\lambda_2}{\alpha} x_2 \end{aligned}$$

We can conclude that x_2 is also an eigenvector of S^T corresponding to $\gamma_i = \frac{\lambda_2}{\alpha}$. This implies that $\lambda_2 = \alpha\gamma_i$ for a certain i. S is stochastic, $|\gamma_i| \leq 1$. Therefore $|\lambda_2| \leq \alpha \square$

Now the proof that $|\lambda_2| = \alpha$ when S has at least two irreducible closed subsets. If $\alpha = 0$, then it is trivially true as $0 \leq \lambda_2 \leq \alpha$. When $\alpha = 1$, it immediately follows from the Ergodic Theorem that the eigenvalue 1 has multiplicity 2, and therefore that $\lambda_2 = 1$. The case that $0 < \lambda_2 < 1$. For the proof we assume that S has at least two irreducible closed subsets.
we start the proof with similar reasoning as how last proof ended. Let y_i be an eigenvector of S^T that is orthogonal to e, i.e. $e^T y_i = 0$. Therefore $E^T y_i = v e^T y_i = 0$. By definition, y_i is an eigenvector of S^T corresponding to the eigenvalue γ_i and thus:

$$S^T y_i = \gamma_i y_i$$

Implementing this in the equation of G gives:

$$G^T y_i = \alpha S^T y_i + (1 - \alpha) E^T y_i$$
$$G^T y_i = \alpha S^T y_i = \alpha \gamma_i y_i$$

Thus y_i is an eigenvector of G^T corresponding to the eigenvalue $\alpha\gamma_i$.
Because S^T has the eigenvalue 1 with at least multiplicity 2, we know that there exist $\gamma_1 = 1, \gamma_2 = 1$ with corresponding eigenvectors y_1 and y_2 that are linearly independent. Let $k_1 = y_1^T e$ and $k_2 = y_2^T e$. The goal is to show that either k_1 or k_2 is zero, implying y_1 or y_2 is orthogonal to e, and therefore that $\lambda_i = \alpha \cdot 1$ with corresponding eigenvector x_i exists.
If $k_1 = 0$, let $x_i = y_1$. If $k_2 = 0$, let $x_i = y_2$. If $k_1, k_2 > 0$, then let $x_i = \frac{y_1}{k_1} - \frac{y_2}{k_2}$. Then x_i is an eigenvector of S^T with the eigenvalue 1 and x_i is orthogonal to e. The first two cases should be obvious, the last one however deserves a closer look. Because $k_1 = y_1^T e$ and $k_2 = y_2^T e$ we have that $1 = \frac{y_1^T}{k_1} e = \frac{y_2^T}{k_2} e$. Therefore $x_i e = (\frac{y_1}{k_1} - \frac{y_2}{k_2}) e = \frac{y_1}{k_1} e - \frac{y_2}{k_2} e = 0$, and thus x_i is orthogonal to e. And because x_i is constructed from multiples of eigenvectors of S^T corresponding to the eigenvector 1, x_i is an eigenvector of S^T corresponding to the eigenvalue 1.
We can now conclude that $|\lambda_2| \geq \alpha$ as there is an $\lambda_i = \alpha$. Moreover we have shown that $|\lambda_2| \leq \alpha$ for G. Thus $\lambda_2 = \alpha$ is the only possible second largest eigenvalue of the matrix G if S has at least two irreducible closed subsets. [1] Indicates that the web graph S contains many irreducible closed subsets.
To conclude: G has a dominant eigenvalue $\lambda_1 = 1$ and all other eigenvalue satisfies $|\lambda| < 1$. Additionally G's second largest eigenvalue $\lambda_2 = \alpha$.

9 Convergence of $\pi^T(k+1) = \pi^T(k)G$

As stated before this system converges to a unique π^T, but why? For convergence we needed three properties of G. G should be nonnegative, primitive and stochastic. I listed irreducibility and aperiodic before, those two combined induce primitivity. Here I will list and prove why these properties are sufficient to conclude convergence of the system.
First of all I will show that G indeed meets the required properties, but to do so we need to define the properties.

Positive or nonnegative matrix: A squared $n \times n$ matrix A is called nonnegative if all the entries are real and $a_{ij} \geq 0$ for each $i, j \in \{1, 2, ..., n\}$ and positive if $a_{ij} > 0$. Clearly by construction of G the matrix is positive.

Irreducibility: A Markov chain is called irreducible if there is a nonzero probability to transition from any state to any other state. This transition does not necessarily need to be done in one step. In our Markov or transition matrix G every page represents a state. A Markov chain is also called irreducible if the transition matrix is irreducible. I will use a definition of irreducibility for nonnegative matrices as this applies to our G. A nonnegative matrix A is said to be reducible if there exists a partition of the index set $\{1, 2, ..., n\}$ into nonempty disjoint sets I_1 and I_2 such that $a_{ij} = 0$ whenever $i \in I_1$ and $j \in I_2$. Otherwise, A is said to be irreducible. This is best shown with an example.

$$\text{Let } A \text{ be the following matrix } \begin{pmatrix} \star & \star & 0 & 0 & \star \\ \star & \star & 0 & 0 & \star \\ \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star \\ \star & \star & 0 & 0 & \star \end{pmatrix}$$

Let $I_1 = \{1, 2, 5\}$ and let $I_2 = \{3, 4\}$. Then $I_1 \cup I_2 = \{1, 2, 3, 4, 5\}$ and $a_{ij} = 0$ whenever $i \in I_1$ and $j \in I_2$. Therefore A in this example is reducible.
Taking a look at our matrix G we notice that every single entry is nonzero as it is a positive matrix. As an immediate result it is impossible to find a partition of indexes I_1 and I_2 such that $g_{ij} = 0$ when $i \in I_1$ and $j \in I_2$. Therefore G is irreducible.

Stochastic matrix, probability matrix, transition matrix, Markov matrix: The terms stochastic matrix, probability matrix, transition matrix and Markov matrix are all equivalent and describe the transition of a Markov chain. The word stochastic is used to describe three different types of matrices. All three definitions describe a square matrix with nonnegative entries. A right stochastic matrix A has the additional property that each row sums to one, i.e. $\sum_j A_{ij} = 1$ for $i, j \in \{1, 2, ..., n\}$. A left stochastic matrix has the additional property that each column sums to one, i.e. $\sum_i A_{ij} = 1$ for $i, j \in \{1, 2, ..., n\}$. A double stochastic matrix is both left stochastic and right stochastic. The term stochastic usually refers to matrix of either form. In this paper the stochastic refers to the definition of right stochastic matrix. G is by construction right stochastic. G is not necessarily left or double stochastic.

Aperiodic: The period of a state i of a transition matrix is the number k such that any return to state i path length is a multiple of k. k is the greatest common divider of all the return to state paths lengths that are possible for state i. It is therefore not necessarily true that there is a return to state in k steps. A state is called aperiodic when $k = 1$. A Markov matrix is called

aperiodic if all the states are aperiodic. G is clearly aperiodic as by construction every state has a link towards itself and thus a return to state path of length 1.

Primitive: A matrix A is primitive if it is nonnegative and A^m is positive for some $m \in \mathbb{N}$. G satisfies this requirement for $m = 1$.

Knowing that our matrix is positive there is a very nice theorem that we can use called the Perron Frobenius theorem, an extention of the Perron theorem in section 8. This theorem consists of several statements that hold for squared positive matrices, I list the relevant ones.
The Perron Frobenius theorem.
Let A be an $n \times n$ positive matrix. I.e., $0 < (a_{ij}) \, \forall \, 1 \leq \{i, j\} \leq n$. Then the following statements hold:

1. There is a positive real number r called the Perron-Frobenius eigenvalue, also named dominant eigenvalue, such that r is an eigenvalue of A and any other eigenvalue λ of A is strictly smaller than r in positive value. $|\lambda| < r$, even if λ is a complex value.

2. There exists an eigenvector v of A corresponding to the eigenvalue r. This eigenvector v consists only of positive elements. I.e., $v = (v_1, v_2, \cdots, v_n), Av = rv, v_i > 0 \, \forall \, 1 \leq i \leq n$.

3. There are no other positive eigenvectors corresponding to A except for (non-negative) multiples of v. I.e., all other eigenvectors must have at least one negative element.

4. The value of r is bounded by the minimum and maximum sum of the rows of A. I.e., $\min\limits_i \sum\limits_{j=1}^{n} a_{ij} \leq r \leq \max\limits_i \sum\limits_{j=1}^{n} a_{ij}, \forall \, 1 \leq i \leq n$

Combining the fact that G is stochastic and the fourth property of the Perron-Frobenius theorem we can immediately deduce that $r = 1$. This immediately implies with the second statement of the Perron-Frobenius theorem that the PageRank vector π^T consists solely of positive elements as we wanted.

The power method is an algorithm to find an eigenvalue r and corresponding eigenvector π^T for a system $\pi^T(k+1)r = \pi^T(k)G$. However it only finds one eigenvalue of the many possible eigenvalues of G. The important thing is that it actually converges to the dominant eigenvalue and corresponding eigenvector. Which in our system is $r = 1$ and the PageRank vector π^T we are trying to find. So all that is left to show is that this power method converges. We have shown that the system converges to our desired π^T and towards the proper eigenvalue, that the resulting π^T has the required properties that all elements are larger than zero and π^T is unique excluding multiples of π^T. But as we can normalize π^T, which is done by each iteration step of the power method, we immediately obtain the right one.

The power method is described by $\pi^T(k+1) = \frac{\pi^T(k)A}{\|\pi^T(k)A\|}$, which is equivalent to $\pi^T(k)$ multiplying with A and then normalizing the result. The power method requires that the matrix G has a dominant eigenvalue, and that the initial vector $\pi^T(0)$ consists of positive elements.

Decompose the matrix G into its Jordan Canonical form: $G = VJV^{-1}$, the first column of V is an eigenvector corresponding to the eigenvalue 1 (as it is not necessarily normalized). Because the dominant eigenvalue is unique the first Jordan block J_1 is an 1×1 matrix consisting only of the dominant eigenvalue. Then the starting vector $\pi(0)$ can be denoted as a linear combination of the columns of V. I.e., $\pi(0) = c_1v_1 + c_2v_2 + \cdots + c_nv_n$ where $c_1 \neq 0$. Reformulating the

power method to our equation:

$$\pi(k+1) = \frac{G\pi(k)}{\|G\pi(k)\|}$$

Because $\pi(k) = \frac{G\pi(k-1)}{\|G\pi(k-1)\|}$ by the same equation we have that:

$$\pi(k+1) = \frac{G\pi(k)}{\|G\pi(k)\|} = \frac{G \cdot G\pi(k-1)}{\|G \cdot G\pi(k-1)\|}$$

Or in genera,l we can denote $\pi(k+1)$ in terms of G and $\pi(0)$ as:

$$\pi(k+1) = \frac{G^{k+1}\pi(0)}{\|G^{k+1}\pi(0)\|}$$

Using the fact that $V^{-1} \cdot V = I$, and therefore $(VJV^{-1})^2 = VJV^{-1}VJV^{-1} = VJ^2V^{-1}$, in addition to the fact that $V^{-1} \cdot v_i = e_i$ where e_i is $1 \times n$ column vector filled with zeros except for entry i which is 1, and $\lambda_1 = 1$ we can rewrite it to the following equation.

$$\begin{aligned}
\pi(k) &= \frac{G^k\pi(0)}{\|G^k\pi(0)\|} = \frac{(VJV^{-1})^k\pi(0)}{\|(VJV^{-1})^k\pi(0)\|} = \frac{VJ^kV^{-1}\pi(0)}{\|VJ^kV^{-1})\pi(0)\|} \\
&= \frac{VJ^kV^{-1}(c_1v_1 + c_2v_2 + \cdots + c_nv_n)}{\|VJ^kV^{-1}(c_1v_1 + c_2v_2 + \cdots + c_nv_n)\|} = \frac{VJ^k(c_1e_1 + c_2e_2 + \cdots + c_ne_n)}{\|VJ^k(c_1e_1 + c_2e_2 + \cdots + c_ne_n)\|} \\
&= (\frac{\lambda_1}{\|\lambda_1\|})(\frac{c_1}{\|c_1\|})(\frac{v_1 + \frac{1}{c_1}V(\frac{1}{\lambda_1}J)^k(c_2e_2 + \cdots + c_ne_n)}{\|v_1 + \frac{1}{c_1}V(\frac{1}{\lambda_1}J)^k(c_2e_2 + \cdots + c_ne_n)\|}) \\
&= (\frac{c_1}{\|c_1\|})(\frac{v_1 + \frac{1}{c_1}V(J)^k(c_2e_2 + \cdots + c_ne_n)}{\|v_1 + \frac{1}{c_1}V(J)^k(c_2e_2 + \cdots + c_ne_n)\|})
\end{aligned}$$

Now zoom in on the part J^k, which multiplies the largest part of the equation. Let J_i denote a Jordan block.

$$J^k = \begin{pmatrix} 1 & & & \\ & J_2^k & & \\ & & \ddots & \\ & & & J_m^K \end{pmatrix} \xrightarrow[k \to \infty]{} \begin{pmatrix} 1 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix}$$

This is a direct consequence of the fact that $\lambda_1 = 1$ and $|\lambda_i| < \lambda_1$ for all $1 < i \leq n$. For example J_2 might be the following matrix:

$$J_2 = \begin{pmatrix} \lambda_2 & 1 \\ 0 & \lambda_2 \end{pmatrix} \to J_2^2 = \begin{pmatrix} \lambda_2^2 & 2 \cdot \lambda_2 \\ 0 & \lambda_2^2 \end{pmatrix} \to J_2^k \begin{pmatrix} \lambda_2^k & \gamma \\ 0 & \lambda_2^k \end{pmatrix}$$

The value of γ might not be trivial but is not relevant. The limit $k \to \infty$ for λ_2^k is 0 since $|\lambda_2| < 1$. As a result J_2^k will have zeros on the diagonal for $k \to \infty$ and γ will be zero as well. The structure shown in this example holds for Jordan blocks.
We can then simplify the above equation quite a bit. Let δ denote the part that in the limit $k \to \infty$ is 0.

$$\pi(k) = \frac{c_1}{\|c_1\|}\frac{v_1}{\|v_1\|} + \delta$$

So for $k \to \infty$ we know that $\pi(k)$ is a constant value which means that $\pi(k)$ converged. The rate of convergence of π is the same as the rate of δ converging to 0, which comes from $\frac{1}{\lambda_1}J^k)i$. The rate of convergence of J_i^k depends of the i^{th} eigenvalue, with a rate of convergence of $|\lambda_i|^k$. $|\lambda_2|$ is the largest eigenvalue and therefore the slowest converging towards 0. Thus the rate of convergence is $|\lambda_2|^k$ for both δ and π.

9.1 Prove convergence power method

Let A be an $n \times n$ matrix with n eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ and the associated linearly independent eigenvectors $\{v_1, v_2, \cdots, v_n\}$. Thus $Av_j = \lambda_j v_j$ for all $j \in (1, 2, \cdots, n)$. Moreover λ_1 is the dominant eigenvalue and the other eigenvalue are ordered such that $|\lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \geq 0$. Let x be any vector in $\mathbb{R}^n$. linear independence of $\{v_1, v_2, \cdots, v_n\}$ implies that constants $\{\beta_1, \beta_2 \cdots \beta_n\}$ exists such that $x = \sum_{j=1}^{n} \beta_j v^j$.
Multiplying by A gives

$$Ax = \sum_{j=1}^{n} \beta_j A v_j = \sum_{j=1}^{n} \beta_j \lambda_j v_j. \Rightarrow$$

$$A^2 x = \sum_{j=1}^{n} \beta_j \lambda_j A v_j = \sum^{n} \beta_j \lambda_j^2 v_j.$$

$$\text{And Generally } A^k x = \sum_{j=1}^{N} \beta_j \lambda_j^k v_j.$$

By factoring out λ_1^k on the right sides we get :

$$A^k x = \lambda_1^k \sum_{j=1}^{N} \beta_j (\frac{\lambda_j}{\lambda_1}^k) v_j.$$

Since $\lambda_1 > \lambda_j$ for all $j \in (2, 3, \cdots, n)$ we have that $|\frac{\lambda_j}{\lambda_1}| < 1$ and thus $\lim_{k\to\infty} \frac{\lambda_j}{\lambda_1}^k = 0$. Therefore we see:

$$\lim_{k\to\infty} A^k x = \lim_{k\to\infty} \lambda_1^k \beta_1 v_1$$

For G, $\lambda_1 = 1$ and we see that the limit converges as long as $\beta_1 \neq 0$. β_1 however is not zero for all π_0 as constructed in section 2. We constructed π_0 as a dense positive vector. The n linear independent vectors v_i span $\mathbb{R}^n$. The n elementary vectors $\{e_1, e_2, \cdots, e_n\}$ also span $\mathbb{R}^n$. $\pi(0)$ can be rewritten as $\sum_{i=1}^{n} \gamma_i e_i$ where γ_i is some positive (nonzero) constant. Thus each e_i is required to construct $\pi(0)$. From the same reasoning we require each e_i to construct v_1, as this is a positive vector aswell. Thus $\beta_1 \neq 0$, $beta_1$ is atleast as large as the smallest γ_i.
However the final statement looks rather different from the equation we have been working with for the Google matrix: $\pi^T G = \pi^T$. It is however the same which we will now show. First of all $\pi^T G = \pi^T$ is equivalent to $G^T \pi = \pi$. Additionally G and G^T have all the same properties, except that G is row stochatic, and G^T is column stochastic. The proof was in general for a $n \times n$ matrix, which G and G^T both are. For $G^T, \lambda_1 = 1$ and thus we have $\lim_{k\to\infty} G^{T^k} x = \lim_{k\to\infty} \beta_1 v_1$. v_1 is the eigenvector corresponding to λ_1 and β_1 is a positive constant. π is the normalisation of v_1, which is the same as the normalisation of $\beta_1 v_1$. x is a positive vector in $\mathbb{R}^n$ and in the equation reflects $\pi(0)$, a "randomly" chosen starting vector. Thus we have that $A^k \pi_0 = \pi$. Let $A\pi_0 = \pi_1$ then $A\pi_1 = A \cdot A \cdot \pi_0 = A^2 \pi_0$ aswell as $A\pi_1 = \pi_2$ and generally $A^k \pi_0 = \pi_k$. For convergence we were looking for a vector π that for further iterations of the algorithm does not change anymore, and thus is the convergence state, in that case $A\pi = \pi$.

10 HITS algorithm

Besides Googles PageRank algorithm there are other algorithms to rank webpages. I will now discuss the HITS method. This method has a lot of similarities to the PageRank algorithm. It also introduces a very interesting new idea, making the ranking query dependent. This could theoretically also be implemented into Googles PageRank method. I will discuss the advantages and disadvantages of this new method compared to the PageRank algorithm after I have explained how it works. The main difference between the two methods, except for the query dependence, is that the HITS algorithm actually produces two ranking vectors. One ranking vector to display the hub scores of a page, and one ranking vector to display the authority score of a page. The hub score is largely determined by the outlink structure of the pages. A page deserves a "good" hub score if it has many outlinks and these outlink link to "good" authorities. Good between quotation marks because it is an open and ethical discussion of what "good" ranking of a page should be. This method is however based on these definitions of good. The authority vector is largely based on the inlink structure of the pages. A good authority is a page with many inlinks and inlinks from important hubs. This may sound like a circular reasoning but just as with the PageRank problem in mathematics it produces a proper convergent system. Every page has both an authority score and a hub score. We can then display the sites that scored either a high authority score, when the user wants to do a depth search on the subject. Or a page with a high hub score, when the user wants a broad search on the topic. Or just display both lists. We do not combine these scores to come to a final ranking. The authority vector will be denoted as x and the hub vector will be denoted by y. We also need a matrix that represents the web structure just as for the PageRank problem. However this matrix L will have to be constructed slightly different. For all entries of L holds $L_{ij} = 1$ if there is a link from page i to page j. This means that L is fully filled with one's and zero's. Furthermore row i denotes all outlinks from page i and column i denotes all inlinks for page i, resulting in the equation for the ranking vector :

$$x(k) = L^T y(k-1), y(k) = L^T x(k)$$

where $x(k)$ and $y(k)$ are the ranking vectors for iteration k, both $n \times 1$ vector where n is the number of pages in the system. Additionally we need some starting vectors $x(0), y(0)$. But because we can calculate $x(0)$ using $y(0)$ we will work with only one starting vector which is usually set as $y(0) = \frac{1}{n}e$ where e in an $n \times 1$ vector of ones. Other starting vectors may be used.

The HITS algorithm: create $y(0)$, then compute until convergence:

$$\begin{aligned} &x(k) = L^T y(k-1) \\ &y(k) = Lx(k) \\ &k = k+1 \text{ (the repeating step to proceed to the next iteration)} \\ &\text{Normalize } x(k) \text{ and } y(k) \text{ (which i will come back to)} \end{aligned} \tag{2}$$

It is also possible to rewrite the equations for $x(k)$ and $y(k)$ with substitution to get the simplified equations:

$$\begin{aligned} x(k) &= L^T L x(k-1) \\ y(k) &= L L^T y(k-1) \end{aligned} \tag{3}$$

Here $x(k)$ and $y(k)$ are not directly related to each other anymore.
These last two equations determine the iterative power method for computing the dominant

eigenvector for the matrices LL^T, L^TL. From this we can conclude that we only need to do one iteration process, because if we calculate $x(k)$ using the power method on $x(k) = L^TLx(k-1)$ we can compute $y(k)$ with the simple computation $y(k) = Lx(k)$. This is very similar to the iterative power method in the PageRank problem. G Is now replaced by LL^T or L^TL. LL^T Is related to the hub scores and is therefore called the hub matrix, L^TL is related to the authority scores and is therefore called the authorities matrix. Both are symmetric positive semi-definite matrices due to the structure of L. So far this method looks a lot like the PageRank vector only creating two separate values for hubs and authorities. However for the PageRank method we needed to construct G such that a couple of properties hold, such as irreducibility and that no rows consist of all zeros. Both LL^T and L^TL do not fulfill these properties. I will cover this when I discuss the convergence of the system. For now it should be noted that we need to make some adjustments to L just as we did to H to construct G. But first I want to discuss the other difference with the PageRank method, query dependence.

10.1 Query dependense

The Google PageRank method constructs a ranking for all web pages, independent of the query. All pages are ranked (query independence). HITS only ranks the pages that are related to the subject that is being searching for (query dependence). Some method to filter out all page that are related to this subject and make the matrix L out of those webpages is required. The neighborhood graph N is created. To do this we use the inverted file index method which I mentioned in section 1.1. Assuming we have some method to find all pages related to the query we list these pages in N. Additionally we load all pages directly connected to these pages. The idea is that when searching for the word "car" we would initially load all pages that mention "car" somewhere. Then because a page about "cars" usually has inlinks or outlinks to similar pages such as pages about "automobiles or "garage" we want to also load these neighboring pages. Whether this is correct is hard to say. In this example "automobile" is a must load to give any reasonable result. However you can also get the problem that some huge site such as Ebay.com is being highly ranked for a search on cars while it probably does not give the user the information they are looking for. Having loaded all the pages on the topic or the neighborhood in the graph N we then build L as described. This should result in an L that is hugely smaller than the whole web.

10.2 Convergence

The rate of convergence is $(\frac{|\lambda_2(B)|}{|\lambda_1(B)|})^k$ where B is either LL^T or L^TL and $\mid \lambda_1 \mid \geq \mid \lambda_2 \mid$ are the two (in absolute terms) largest eigenvectors of B. And because both LL^T, L^TL are symmetric, positive semi-definite we know that all distinct eigenvalues are real. This means that we could possibly have repeated eigenvalues. In the case that we have no repeated eigenvalues the HITS algorithm converges nicely using the same proof as for the PageRank algorithm in section 9. Even though we do not have a good general approximation of λ_1 and λ_2. Many research results show that the difference between λ_1, λ_2 is rather large so only 10-15 iterations are needed for convergence [10]. But when $\lambda_1 = \lambda_2$, there is a problem. Even though the problem converges in this case it does not necessarily converge to a unique vector. Meaning that different starting vector could lead to different ranking vectors, which of course is a huge problem as this would render all arguments of a "good" ranking of this system useless. This problem comes forth of reducibility in the matrices LL^T, L^TL. The Google matrix with the PageRank algorithm came across the same problem which we then fixed by adding the random teleportation inducing irreducibility on the matrix. Making LL^t, L^TL irreducible with the same fix would guarantee

convergence towards a unique ranking vector. Change the matrix LL^T to $\eta LL^T + (1-\eta)\frac{1}{n}ee^T$ where $0 < \eta < 1$ and similar for L^TL. An additional problem for the HITS algorithm is that it is not guaranteed that the randomly chosen starting vector $y(0)$ has a component in the direction of the eigenvector corresponding to λ_1. Again using the same fix as the PageRank algorithm used works, require that each entry of $y(0)$ is a positive value.

10.3 Advantages and disadvantages

One simple advantage of the HITS method is that it produces two rankings for each page. Giving more information to a user than the single ranking of PageRank. The biggest advantage however comes from the query dependence step. HITS uses relatively small matrices. In fact small enough that it should bypass all computer storage issues. In general it should be easy and quick to compute because of this small portion of the web that is used. However this also has a lot of disadvantages. First of all HITS requires fast calculation of the ranking vectors. When a user submits a query, N must be constructed and the ranking vectors should be calculated in an acceptable timespan or the search engine would not be used. Additionally this must be done every time a user submits a query. After some querys more computations are required than with the PageRank method. Also the procedure to find N has some disadvantages. Most likely it does not find all sites that are actually related to the subject. Luckily these sites would usually receive a low ranking anyway, no big downside. More problematic is that some sites such as Ebay.com might always dominate the rankings. This could be fixed to weight sites that are on topic more than the neighborhood sites. Also we could make the HITS procedure query independent just as the PageRank method. Simply drop the step of creating N and construct L for the entire web. Though the advantage of working with one small sized matrices is then lost. The last disadvantage is that HITS is relatively sensitive. The matrices are smaller thus changing a page influences the rankings more. Implying that users could easily crank up the ranking values of their pages by adding additional outlinks, or creating more pages to increase the number of inlinks that their main site has. The HITS method is rather sensitive to "cheating" the rank of a site.

The nice thing about HITS is that no matter what kind of changes take place to the structure of the web LL^T and L^TL will always be symmetric positive semi-definite matrices. Because we always construct a N it is obvious that adding pages to the system does not influence the procudure. By the same reasoning, adding or subtracting links does not affect the properties. When the Eigengap, $\delta = \lambda_1 - \lambda_2$ is large then the ranking vectors are insensitive to small changes in L. When the Eigengap is small the ranking vector might be very sensitive, similar to the case of the PageRank algorithm.

During the disadvantages we mentioned that we could make this HITS method query independent. The following algorithm uses the modified HITS method to guarantee convergence and is query independent. Here N is no longer formed and L represents the entire web.

$$
\begin{aligned}
&\text{use starting vector } x(0) = \frac{1}{n}e, \text{ then compute until convergence:} \\
x(k) &= \eta L^T L x(k-1) + (1-\eta)\frac{1}{n}e \\
x(k) &= \frac{x(k)}{\|x(k)\|_1} \\
y(k) &= \eta L L^T y(k-1) + (1-\eta)\frac{1}{n}e \\
y(k) &= \frac{y(k)}{\|y(k)\|_1} \\
k &= k+1 \\
&\text{set the authority vector } x = x(k) \text{ and the hub vector } y = y(k).
\end{aligned}
\tag{4}
$$

Comparing the work between this HITS algorithm and the PageRank method, most of the work is done by computing the step $L^T L x(k-1)$ during each iteration. For query independent modified HITS we require $4\text{nnz}(L) + 2n$ additions per iteration, where n is the number of pages in the web and $\text{nnz}(L)$ denotes the number of nonzero entries in L. The intelligent PageRank method required $\text{nnz}(L)$+n additions and $\text{nnz}(L)$ multiplications per iteration ($\text{nnz}(H)$= $\text{nnz}(L)$ is this case). So we see that we actually require a sizeable higher amount of computations using the query independent modified HITS method each iteration. However we still need to look at the number of iterations needed. HITS has a convergence rate of approximately 0.5 whereas PageRank had a rate of approximately 0.85. So HITS requires much fewer iterations, and is a bit lighter to compute. And you get an extra ranking vector on top!

Lastly it should be mentioned that the power method is a rather slow iteration method. PageRank uses it because it is memoryless. The query dependent HITS method works with a small matrix L. Therefore some computationally or storage costly iteration method can be applied. Thus we can speed up the HITS method by using a more efficient iteration method. The query dependent HITS method possibly can additionally be accelerated by applying similar techniques as done for the PageRank vector such as extrapolation.

11 Other search engine algorithms

Another search engine algorithm is Salsa. Salsa uses just as Hits authority and hub scores to rank pages. Similarto the PageRank algorithm it uses Markov chains. Salsa is an acronym for Stochastic approach to link structure analysis. This sounds great as it combines the best components of PageRank and HITS in one algorithm. The Salsa algorithm starts by creating the neighborhood graph N, which is query dependent, just as HITS did. But instead of L this algorithm creates a 'bipartite undirected graph' which I will call B. B is built of three sets, V_h, V_a, E. V_h is the set of all hub pages, that are pages with outlinks. V_a is the set of all authoritie pages, pages with inlinks. A site may be both in V_a and V_h. E is the set that stores all links including the direction. Then two Markov chain matrices are formed, the authorities matrix A and The hub matrix H, from B. I demonstrate the construction with an example. Let L be the matrix constructed in the same manner as for the HITS algorithm. Then we are going to create L_c, L_r which are column and row stochastic respectively. By dividing each column/row respectively by its sum.

$$L = \begin{array}{c} \textit{page} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \rightarrow ,$$

$$L_c = \begin{array}{c} \textit{page} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 0 & \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \end{pmatrix} \end{array}, L_r = \begin{array}{c} \textit{page} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 0 & \frac{1}{2} & 0 & \frac{1}{1} & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Then we create A from $L_c^T L_r$ with the addition that we subtract both the zero rows and columns. Similarly we create H from $L_r L_c^T$ and removing the zero rows and columns.

$$L_c^T L_r = \begin{array}{c} \textit{page} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{6} & 0 & \frac{5}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \rightarrow ,$$

$$A = \begin{array}{c} page \\ 1 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{6} & 0 & \frac{5}{6} \end{pmatrix} \end{array}, H = \begin{array}{c} page \\ 1 \\ 2 \\ 3 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 5 & 6 \end{array} \\ \begin{pmatrix} \frac{5}{12} & 0 & \frac{2}{12} & \frac{3}{12} & \frac{2}{12} \\ 0 & 1 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{4} & 0 & 0 & \frac{3}{4} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \end{pmatrix} \end{array}$$

Notice that both A and H are now row stochastic and similar structure as S for the PageRank algorithm. To guarantee convergence to a unique stationary solution vector we need irreducibility of the matrices. In the usual case where these matrices are not irreducible we look at the connectedness of the pages. Irreducibility implies that from every page there is a linking structure to any other page. But our example A is not irreducible, from page 1 we cannot reach page 3,4 or 5. From page 3 however we can reach pages 4,5 but not 1. Moreover when we look at the structure of both A and H we see that they are symmetric. If A_{13} is nonzero then A_{31} is also nonzero. If page 1 is connected to page 3, then page 3 is connected to page 1. Because of this structure we can now split the matrices in connected subsets. For A these connected component subsets are $\{1\}$ and $\{3\ 4\ 5\}$. Then we create matrices belonging to each subset. These matrices are smaller than the original matrix A and therefore easier to calculate with. Then we derive an π_p for each submatrix similar as we derived π for the PageRank algorithm and p is an index to keep track of all subset and their corresponding matrices. We end up with several π_p that consists PageRank values corresponding to the connected subsets and each π_p adds up to 1. in this case $\pi_1 = \{1\}$ and $\pi_2 = \{\frac{1}{3}\ \frac{1}{6}\ \frac{1}{2}\}$. To achieve the final ranking vector we multiply each value in this vector π_p by $\frac{i}{n}$ where i is the number of pages in π_p and n the total number of pages in the set V_a. In this case the final PageRank vector for A will be: $\pi = \{\frac{1}{4}\ \frac{1}{4}\ \frac{1}{8}\ \frac{3}{8}\}$. Calculate the PageRank values belonging to H with a similar process. Of course we could use other weighting methods to add the different π_p together.

Salsa combines the best features of HITS and PageRank. It creates two ranking scores, and it work with relatively small matrices considering the size of the web. In fact we work with even smaller matrices then the HITS algorithm. In practice it turns out that A, H both consists of several connected components which is computational really beneficial. However it also inherits the weaknesses. Salsa is also query dependent just as HITS and it does not force irreducibility on the matrix, not guarenteeing convergence. We could again implement the irreducibility fix of a random teleporter to guarantee convergence to a unique vector.

Additional to these search models there are several others, most of which are in bases a combination of HITS, Salsa or PageRank. However I want to cover one other interesting idea, called traffic rank. Traffic rank tries to rank pages on popularity by measuring the amount of traffic a page has. A page that is visited frequently is deemed better, and therefore a good search result. However, we do not look at the amount of visitors of the page, but the amount of surfers each link has. Sadly this is impossible but we can approximate it. A simpler way would be the amount of visitors the page has, which we can track. We want to use the amount of traffic on a link because the number of links a page has and the traffic per link is also telling us more about the popularity of the page. Imagine a road network and trying to determine how popular a city is by measuring the traffic coming in by different roads. The amount of visitors divided by the amount of links gives a relative popularity score.
Let p_{ij} denote the proportion of all web traffic on the link from page i to page j. i and j are indexing value from 1 to the number of pages in the web. So $p_{ij} = 0$ when there is no link, and a very small number otherwise. Then $\sum_i p_{ij}$ denotes the total proportion of traffic entering

page j. We define the proportions in such a way that $\sum_{i,j} p_{ij} = 1$. Secondly we assume that traffic to a page is always equal to the traffic out of the page, $\sum_i p_{ij} - \sum_j p_{ji} = 0$. Now minimalizing $-\sum_{i,j} p_{ij} \log p_{ij}$ with these constraints but free p_{ij} gives the best unbiased probability assignments to p_{ij}. Although the system is huge, even larger than the number of webpages it has been shown that it takes approximately 2.5 times as long to calculate these probabilities than to solve the PageRank algorithm. [10]

Traffic flow gives pages a high score if it has a lot of traffic. Interestingly these pages seem to coincide with the HITS algorithm's result for good hub pages. Logical in a sense, as a page with a lot of traffic requires a lot of inlinks and outlinks to handle just as the example of the city. The real interest for the traffic method comes from simple extensions that can be made on the basis model. When more knowledge about the links actual traffic comes available this can be implemented as an additional constraint in the algorithm. Setting $a \leq p_{ij} \leq b$ where a and b denote numbers to bound the value of p_{ij} around the known value. Similar boundaries can be made for the total traffic of a page. Secondly to derive the traffic scores from the algorithm we applied Lagrange multiplier. Each link has such a Lagrange multiplier. We can also invert these multipliers which gives us the temperature of a page instead of the traffic. The temperature mark similarities with the authorities scores of HITS.

12 Future of PageRank

I have spoken about cheating of PageRank scores before. The biggest threat for the future of PageRank is closely related to the idea of cheating. The PageRank algorithm was founded on the idea that each link from a page to a page is an endorsement. However when people know how a search engine ranks pages, the assumption that all links are made as endorsements and no ulterior motives is no longer true. The whole fundamental of the PageRank algorithm and brings the question, does it still give good ranking? Corresponding to these links with ulterior motives is the never ending cycle of updating algorithm as described by cheating rank scores. Additionally there is the problem of spam. Optimization of algorithms to both rank pages and include methods to identify cheaters is an ongoing research area. Ideally an algorithm that is spam proof can be constructed (this would mean it is not based on the linking structure), but as long as we have not found such a system search engines try to identify cheating pages with algorithms. An example algorithm to find cheating pages is bases on back links. When a page has over 80% of the pages link back to it, it is considered a cheater. Resulting in the removal of that page from the index and the matrix G. Another idea is the so called 'Badrank'. Badrank is close to PageRank, except that it is based on outlinks instead of the inlinks of a page. A page receives a high badrank score if it points to other bad pages. The most reliable method however might be to offer a service to boost the PageRank score of a page, usually for payment. Instead of going through the trouble to cheat someone would then pay for this sponsored link. Of course all mentioned methods are not fool proof, and that's why it is an ongoing area of development.

Another area of ongoing improvement for search engines is query refining. An average user is rather lazy and only looks at the first couple of results. Therefore it is extremely important to have the first results to be correctly ranked. Moreover a user often does not quite know what he is searching for (hence the use of a search engine). So an area of improvement findind what users are looking for, instead of finding the page that corresponds the best to a given query. For instance when I use the search query 'coffee' in Google now at 20-05-2015 I find several companies that sell coffee, an informational site about coffee, a review site and a couple of instances that are a combination of those. Most likely when a user searches the term coffee he is not interested in all these categories of pages but only in one category, lets say buying coffee. It would be helpful to produce a list of all pages that sell coffee and give these pages as search result. An idea is to categorise search results and give these categories back to the user, who then quickly can find the top pages in the category he is looking for. Continuing on this line of thought would be giving a user search results based on what you think he wants to see. The Users browser history, or previous searched for querys could be used to help predict what the user wants to find this time. A simple example would be that if a user first search the term 'coffee', and proceed to go to pages that sells coffee, listing pages that sell tea is then probably a better search result for the query 'tea' than a list of informational pages.

Another interesting newer area are pages with different linking structures as average webpages. An example of such a subgroup are blogs. Blogs usually are relatively content empty but have a lot of links to other blogs. Additionally blogs are usually dated as they refer to news articles or other dated events. PageRank takes a long time to update, and therefore would not be useful to rank blogs. Besides having different ranking criteria for different types of pages also improves search results. For example the algorithm iRank is made to rank blogs. The algorithm looks a lot like PageRank with two major differences. It is time dependent, a blog that cites other recent blogs get rewarded. And implicit links, are not actual links, when two pages have high

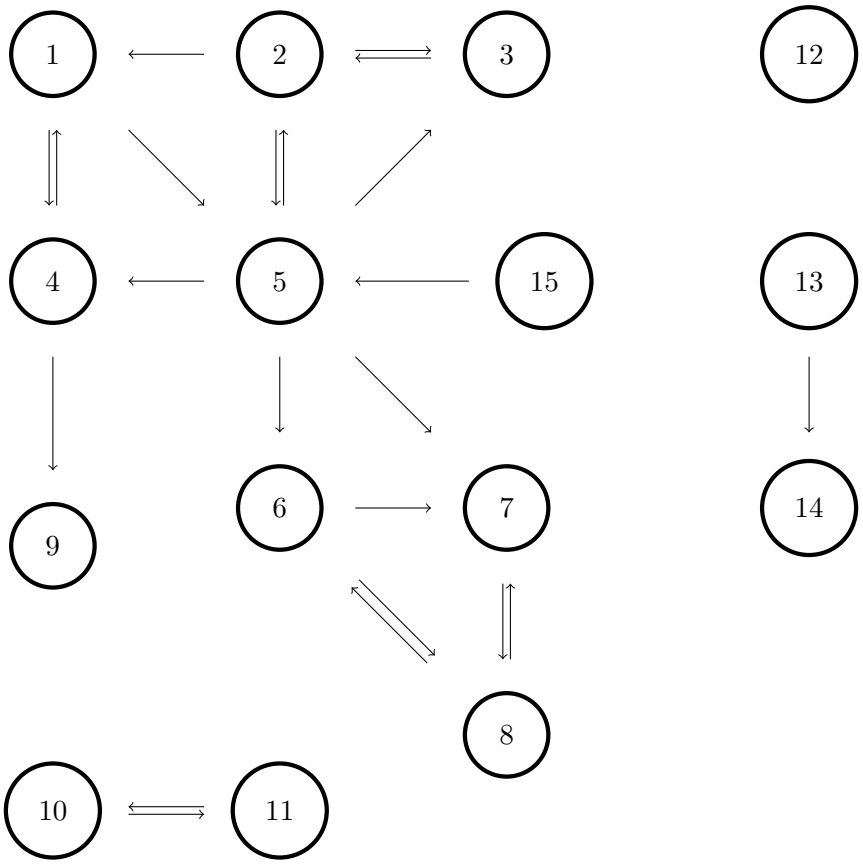
similarity (same new event for example) the algorithm adds a link to each other as people that read the first most likely read the second. IRank and PageRank give wildly different results. PageRank is more likely to give an older authorities blog that started the subject, where iRank gives pages that either help find the most recent information or give the most recent information. The use of time in weighting the pages is rather interesting. For example let there be a disaster in the world. Every page written at that day about this will most likely highly differ from a page written three months later about the same disaster.

A small step aside, censorship. Additionally to all debates of good ranking search engines are also constantly debating about censoring pages. Because everyone can access a search engine easily you might want to prevent the possibility to give children access to porn pages or other pages deemed wrong. Construct a crawler that looks for certain words and does not index pages that contain words such as 'porn'. However once again this is an though subject to make the right decisions. For example this artical has the word 'porn' in it, but surely it should be found when using Google and searching on 'PageRank' (assuming this artical gets uploaded to the internet). More problematic is the idea that search engines could censor political views or ideas. I only mention this to show how many complex ethical and liberal subject arise by creating a search engine.

The last possible feature improvement for search engines that we want to mention is fusion of information. For example use a map of a city, and use a search engine to show pages that correspond to places on that map. Or even information only, such as opening hours of the restaurants. Implementing a search engine with other layers such as maps.

13 Example of PageRank algorithm for a web graph

We would like to discuss an example of a very small web graph consisting of $n = 15$ pages as shown below.



For this web graph we will use the discussed power method to solve corresponding to the equation $\pi^T(k+1) = \pi^T(k)G$. I will show the first two iterations and the final result. Then I will discuss why the system would not work when working with H instead of G by choosing different $\pi^T(0)$. I will also deploy two different accuracy arguments. One is based on the values of the vector, the other one is based on the ranking of the elements. Lasstly I will aplly differnt $\pi^T(0)$, α and v^T to and look at the influence on the ranking of the pages.

But first I discuss why I chose web graph and all the types of pages that it contains. Dangling nodes, pages 9, 12 and 14. Moreover page 12 has no inlinks, a very special case, and if the system is anything accurate this page should receive a low PageRank value. A sink in the form of the combined pages 6, 7, 8. A well connected page 5, and a couple of pages whose relative importance would be very interesting to calculate as they are hard to compare with each other without calculation, pages 1, 2, 3, 4 and these scores relative to pages 7, 8, 9. Page 15 has no inlinks but is feeding the system. And the odd disconnected pages 10, 11, 12, 13 and 14. I added 10, 11 and 13, 14 especially to show the scores of these pages and the entire system as it will be interesting to see how important pages are ranked.

13.1 Constructing the matrices

To find the G corresponding to the web graph we first need to construct H as described in section 2 where I also denoted the equation for G and S in terms of H. Below is denoted the matrix H corresponding to the web graph. Important to note the structure of H, such as zeros on the diagonal, the extreme sparsity, the rows of zeros corresponding to the dangling nodes and that the rank$(H) \leq 15$.
Remark: the number next to the matrix H denotes the page. Each row reflects the outlinks from the page. Each column reflects the inlinks to the page.

$$H = \begin{array}{c} \textit{page} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Now that we have created H we can simply construct S and G. To create S fill all the dangling rows of H with the value the vector v^T, here $v^T = \frac{1}{15}e^T$. This is done by adding $a \cdot (\frac{1}{n} \cdot e^T)$, a is the column vector shown below and e^T is the row vector 1×15 of ones. Below S I have denoted $a \cdot e^T$.

$$S = \begin{array}{c} \textit{page} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$a = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \; e^T = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T, \; v^T = \begin{pmatrix} \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \\ \frac{1}{15} \end{pmatrix}^T$$

$$a \cdot e^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$G = \begin{array}{c} page \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} \begin{array}{c} \begin{array}{ccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array} \\ \begin{pmatrix} \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{31}{75} & \frac{31}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{24}{75} & \frac{1}{75} & \frac{24}{75} & \frac{1}{75} & \frac{24}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{75} & \frac{61}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{31}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{31}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{75} & \frac{13}{75} & \frac{13}{75} & \frac{13}{75} & \frac{1}{75} & \frac{13}{75} & \frac{13}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{31}{75} & \frac{31}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{61}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{31}{75} & \frac{31}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{61}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{61}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \\ \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{61}{75} & \frac{1}{75} \\ \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{61}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} & \frac{1}{75} \end{pmatrix} \end{array}$$

For G I chose $\alpha = 0.8$, $G = 0.8S + 0.2ev^T$ which is shown above.
Remark: Each zero element of S is replaced by the value $\frac{1}{75}$ in G. Because $0.2 \cdot \frac{1}{15} = \frac{1}{75}$. This shows that at each iteration a change of $\frac{1}{75}$ to teleport (not following outlinks) to page 1, or any other page.
Remark: The dangling nodes are still filled with the values $\frac{1}{15}$ as $v^T = \frac{1}{15}e^T$. For different v^T the dangling nodes will have different values in G compared to S.

Lastly before we can start the iteration algorithm we need chose a $\pi^T(0)$, a vector containing initial rankings. I chose the starting vector with equal value $\frac{1}{15}$ for each page.

13.2 Convergence bases system

The first iteration, calculated using a matlab program listed in section 15, is shown below. Then I let the program run until the vector π^T converged with an accuracy argument based on values such that the sum difference between two consecutive PageRank vectors is less than 10^{-9}, resulting vector π^T is also shown below. It required 50 iteration to converge to π^T. The first iteration had an accuracy of 0.0959. Below are results rounded on 4 digits accuracy. It should be noted that Matlab works with high accuracy, but makes accuracy errors.

$$\pi^T(1) = \begin{pmatrix} 0.0684 \\ 0.0880 \\ 0.0524 \\ 0.0613 \\ 0.1218 \\ 0.0613 \\ 0.0880 \\ 0.1040 \\ 0.0507 \\ 0.0773 \\ 0.0773 \\ 0.0240 \\ 0.0240 \\ 0.0773 \\ 0.0240 \end{pmatrix}, \pi^T(2) = \begin{pmatrix} 0.0694 \\ 0.0829 \\ 0.0644 \\ 0.0683 \\ 0.0915 \\ 0.0825 \\ 0.1071 \\ 0.1164 \\ 0.0460 \\ 0.0833 \\ 0.0833 \\ 0.0214 \\ 0.0214 \\ 0.0406 \\ 0.0214 \end{pmatrix}, \pi^T = \begin{pmatrix} 0.0577 \\ 0.0686 \\ 0.0483 \\ 0.0530 \\ 0.0740 \\ 0.0950 \\ 0.1330 \\ 0.1625 \\ 0.0399 \\ 0.0907 \\ 0.0907 \\ 0.0181 \\ 0.0181 \\ 0.0327 \\ 0.0181 \end{pmatrix}$$

The first iteration is rather traceable. A page with inlinks has its value increased. In the second iteration we see that most pages follow the trend they had in the first iteration. If they increased first iteration, they increase again and vice versa. However some pages do not, such as page 14. This page still got an endorsement from page 13 the first iteration. The second iteration page 13 lost most of its own value and thus page 14 loses its endorsemen. The resulting converged vector π^T shows some interesting results. Some pages such as 8 and 12 were already on the right track the first couple of iterations. But page 5 in particular seems to be rated higher at the first iterations than in the converged state. There are also some pages such as pages 1 and 4 that reverse in direction somewhere during the iteration process. Page 4 in particular is interesting as this page starts decreasing ($\frac{1}{5} \approx 0.6666$), then increases next iteration but ultimately ends up with a lower score. Clearly it requires some iterations before we can even predict where some PageRank scores are going to end up. Notice that the difference between lowest and highest page value are larger in the converged vector then the starting iterations. As we see some page have the

exact same structure, such as the dangling nodes, have the exact same value each iteration. Below are the relative rankings computed with matlab of the pages corresponding to these vectors.

$$\text{rank } \pi^T(1) = \begin{pmatrix} 7 \\ 4 \\ 10 \\ 8 \\ 1 \\ 9 \\ 3 \\ 2 \\ 11 \\ 6 \\ 5 \\ 12 \\ 12 \\ 6 \\ 12 \end{pmatrix}, \text{rank } \pi^T(2) = \begin{pmatrix} 8 \\ 6 \\ 10 \\ 9 \\ 3 \\ 7 \\ 2 \\ 1 \\ 11 \\ 4 \\ 5 \\ 13 \\ 13 \\ 12 \\ 13 \end{pmatrix}, \text{rank } \pi^T = \begin{pmatrix} 7 \\ 6 \\ 9 \\ 8 \\ 5 \\ 3 \\ 2 \\ 1 \\ 10 \\ 4 \\ 4 \\ 12 \\ 12 \\ 11 \\ 12 \end{pmatrix}$$

I immediately notice a surprising result. Page 10 and 11 are not rated the same position every iteration even though they have the exact same structure and therefore should have the exact same PageRank value. One of the problems that can arise with the PageRank agorithm. It originates from a roundoff accuracy error in the computation (software/computer related). Another interesting point that we can now easily see is that some page really quickly get their proper relative rank, which is promising for accuracy based on rank. Overall I showed these relative ranks because they are easier to compare.

I would like to discuss some noticeable pages and their ranks. Page 12 is a nice comparison point. This page has no in or outlink and therefore cannot be important and indeed it is rated the lowest page. However we see that all pages without inlinks are rated the exact same, pages 12, 13, 15. The dangling nodes however have all different ranks. Page 14 only has one inlink from a low ranked page and not surprisingly is rated second lowest. Page 10 and 11 are however ranked fourth. Even though they are not important for the overall structure of the web graph, they are a standalone cycle. This shows the weakness of PageRank for cycles. Once you realize that PageRank actually favors cycles and sinks then the ranks can be explained. We 'fixed' the algorithm such that cycles and sinks were no problem anymore. However this 'fix' was to guarantee convergence, not to fix fairness. Recall the idea of the random surfer, where we followed one page, and a page where we spent more time was more important. With $\alpha = 0.8$ this still means that once we visit page 10, on average we visit it again following the links (to page 11 and back) before we teleport away. Not surprising that these pages are still favored. Although its better than before with an α of 1 as they we would never leave the pages. And a lower α was also undesirable as then the structure of the web is increasingly less important. For the extreme $\alpha = 0$ the web graph is literally irrelevant. Seeing how PageRank favors cycles and sinks, its not surprising anymore that pages 6, 7, 8 are the top ranking pages followed by page 10 and 11. After that the pages 1, 2, 3, 4, 5, 9 and lastly ranked the pages without any relevant inlinks. How PageRank ranked pages 1, 2, 3, 4, 5, 9 is really interesting as this importance is fully based on their underlyng links, no sink or cycle structure to help any page. The well connected page 5 seems to be the winner, and the poorly connected page 9 the loser. For the whole web, the graph is very large and very complicated. There will (most likely) not be any relevant sized sinks/-

cycles and there will be some pages that are so well connected that they arise at the top. And only the most important pages matter. This example shows the importance of the sink/cycle structure simply indicates a weakness of PageRank and why there is always a discussion of what a good ranking criteria is. It is an indication of the complexity of correctly ranking a web graph.

Next I tried to implement an accuracy argument based on the relative rank of each page. The results that I got were really interesting but also may be flawed due to the problems of accuracy. As mentioned page 10 and 11 should always have the same relative rank. They do always have the same value with accuracy of 4 digits. However when computing the vector with relative ranks page 10 and 11 do not always have the same. When I used the accuracy argument that the vector converged if it had 3 consecutive iterations the same ranking vector, then it converged after 13 iterations. The 14th iteration however gave the same rankings except for page 10 and 11. For some reason page 10 was suddenly rated higher then page 11. When looking back at the ranking vector of the first two iterations we see that in the first page 10 is rated lower and the second iteration is rated higher than page 11. This is a problem for my accuracy argument. If instead of requireing 3 consecutive ranking vectors requiring 5, then it takes 103 iterations. And when I changed it to 15 it required 113 iterations. Indicating that iteration 98 is the first true converged vector. However thee resulting ranking vector was identical to the ranking vector after 13 iterations. Therefore I assumed that the system indeed converged after 13 iterations using accuracy on rank, even though it might converge faster or slower.

13.3 Changing $\pi^T(0)$

We could also use a different starting vector $\pi^T(0)$. This time the vector whose first element is one and all others zero will be used. The accuracy argument is based on the values and then the system takes 74 iterations to converge. The starting accuracy in this case was 0.9423 which partly explains the required amount of iterations.

$$\pi^T(1) = \begin{pmatrix} \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{31}{75} \\ \frac{31}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \\ \frac{1}{75} \end{pmatrix}, \pi^T(2) = \begin{pmatrix} 0.1844 \\ 0.0923 \\ 0.0852 \\ 0.0869 \\ 0.0350 \\ 0.0869 \\ 0.0923 \\ 0.0315 \\ 0.1808 \\ 0.0261 \\ 0.0261 \\ 0.0155 \\ 0.0155 \\ 0.0261 \\ 0.0155 \end{pmatrix}, \pi^T = \begin{pmatrix} 0.0577 \\ 0.0686 \\ 0.0483 \\ 0.0530 \\ 0.0740 \\ 0.0950 \\ 0.1330 \\ 0.1625 \\ 0.0394 \\ 0.0907 \\ 0.0907 \\ 0.0181 \\ 0.0181 \\ 0.0327 \\ 0.0181 \end{pmatrix}$$

$$\text{rank } \pi^T(2) = \begin{pmatrix} 1 \\ 3 \\ 6 \\ 5 \\ 7 \\ 4 \\ 3 \\ 8 \\ 2 \\ 10 \\ 9 \\ 11 \\ 11 \\ 10 \\ 11 \end{pmatrix}, \text{rank } \pi^T = \begin{pmatrix} 7 \\ 6 \\ 9 \\ 8 \\ 5 \\ 3 \\ 2 \\ 1 \\ 10 \\ 4 \\ 4 \\ 12 \\ 12 \\ 11 \\ 12 \end{pmatrix}$$

As we can see the first two iteration show different results as expected with the two starting vectors differ. However they converged to exactly the same vector as the theorems indicated. Even the number of iterations was not too much affected, as expected. Only a couple more iterations are needed, due to a larger accuracy error of the first iteration and a couple more iterations before the PageRank value has reached all the pages. In the second iterations is for instance page 8 still to receive a PageRank value boost, as page 8 is reached from page 1 using 3 links.
Remark: $\pi^T(1)$ is the first row of G with this specific $\pi^T(0)$.

13.4 Changing α

Next a look at the systems for different α. Using $\pi(0) = \frac{1}{15}e$, $\alpha = 0.5$, $v^T = \frac{1}{15}e$ and accuracy argument based on value, it required 22 iterations to converge to the vector and rank below.

$$\pi^T = \begin{pmatrix} 0.0671 \\ 0.0770 \\ 0.0599 \\ 0.0638 \\ 0.0871 \\ 0.0725 \\ 0.0906 \\ 0.1018 \\ 0.0543 \\ 0.0767 \\ 0.0767 \\ 0.0383 \\ 0.0383 \\ 0.0575 \\ 0.0383 \end{pmatrix}, \text{rank } \pi^T = \begin{pmatrix} 7 \\ 4 \\ 9 \\ 8 \\ 3 \\ 6 \\ 2 \\ 1 \\ 11 \\ 5 \\ 5 \\ 12 \\ 12 \\ 10 \\ 12 \end{pmatrix}$$

Different values from the basic system, reasonably expected, but finding that the pages are differently ranked is surprising. Notice as well that the difference between highest and lowest PageRank value is less than for $\alpha = 0.8$. Because of the decrease of α the importance of the structure of the web is lower than before, and the influence of the teleportation vector v^T is higher. As a result some pages lose a bit of PageRank value and even ranks. Pages 9, 10, 11 all lose one rank, and page 6 loses three ranks. Page 2 and 5 both gain two ranks and page 14 also gains a rank. We see that page 10 and 11 actually score a rather high PageRank even though they only link to each other. Their interlinking is apparently important for their high rank, and indeed they lost ranks when links became less important. Page 2 and 5 are rather connected, however, gain ranks due to the lower importance of links. This can be explained by the reduction of value in the sink and cycles. What also explains why page 6 decreased in rank. Because there is more teleportation, less value is stored in the sinks/cycles. As a result those pages lose a lot of value (page 8 for example lost the most value, however is still ranked first). This lost value then spreads to all the other pages. And page 2, 5 are well connected and therefore receive a lot of additional PageRank. They receive more PageRank due to random teleportation, but also because all their inlinks add more PageRank value because they are from page with higer PageRank value. So even though a lower α means that the teleportation is more important and the structure of the web less, well connected pages see their value increased. Of course worth mentioning as well, the system converged faster.

For $\alpha = 0.95$ we see that it took a couple more iterations before convergence, but converged to the exact same rankings as for $\alpha = 0.8$. The required amount of iterations was 97. Surprisingly when I used the accuracy argument that looked at consecutive rankings I found that it took 18 iterations to converge. The difference between highest and lowest values are larger in these vectors. All these vectors with their ranks are shown below.

$$\pi^T = \begin{pmatrix} 0.0282 \\ 0.0336 \\ 0.0227 \\ 0.0255 \\ 0.0347 \\ 0.1348 \\ 0.1988 \\ 0.2583 \\ 0.0176 \\ 0.1093 \\ 0.1093 \\ 0.0055 \\ 0.0055 \\ 0.0107 \\ 0.0055 \end{pmatrix}, \text{rank } \pi^T = \begin{pmatrix} 7 \\ 6 \\ 9 \\ 8 \\ 5 \\ 3 \\ 2 \\ 1 \\ 10 \\ 4 \\ 4 \\ 12 \\ 12 \\ 11 \\ 12 \end{pmatrix}, \text{after 18 iterations } \pi^T = \begin{pmatrix} 0.0311 \\ 0.0370 \\ 0.0249 \\ 0.0280 \\ 0.0380 \\ 0.1318 \\ 0.1942 \\ 0.2516 \\ 0.0192 \\ 0.1082 \\ 0.1082 \\ 0.0056 \\ 0.0056 \\ 0.0110 \\ 0.0056 \end{pmatrix}$$

13.5 Changing v^T

Then I used a different teleportation vector v^T to see the effects on the system. The chosen v^T is shown below along with the results. The standard starting vector $\pi(0) = \frac{1}{15}e$, $\alpha = 0.8$ and accuracy argument were applied.The system converged in 76 iterations.

$$v^T = \begin{pmatrix} \frac{1}{100} \\ \frac{20}{100} \\ \frac{1}{100} \\ \frac{1}{100} \\ \frac{40}{100} \\ \frac{1}{100} \\ \frac{1}{100} \\ \frac{10}{100} \\ \frac{10}{100} \\ \frac{10}{100} \\ \frac{1}{100} \\ \frac{1}{100} \\ \frac{1}{100} \\ \frac{1}{100} \\ \frac{1}{100} \end{pmatrix}, \pi^T = \begin{pmatrix} 0.0539 \\ 0.1103 \\ 0.0565 \\ 0.0486 \\ 0.1380 \\ 0.0926 \\ 0.1296 \\ 0.1638 \\ 0.0425 \\ 0.0751 \\ 0.0651 \\ 0.0050 \\ 0.0050 \\ 0.0090 \\ 0.0050 \end{pmatrix}, \text{rank } \pi^T = \begin{pmatrix} 9 \\ 4 \\ 8 \\ 10 \\ 2 \\ 5 \\ 3 \\ 1 \\ 11 \\ 6 \\ 7 \\ 13 \\ 13 \\ 12 \\ 13 \end{pmatrix}$$

The chosen v^T is designed to hugely boost page 5, give a boost to page 2 and page 8, 9, 10, while the other pages are hardly teleported towards. The resulting rankings are different from the rankings we found before. The dangling nodes are hardly affected. Page 10 suddenly is rated a fair bit higher then page 11. Of course page 5 and 2 are rated higher, but the effect for example for page 1 are though to predict without the simulation. The important lesson here is that v^T can hugely influence the ranking of the vectors and also pages whose linking value in v^T are not influenced can all of a sudden change rank. For example page 1 and 3 get the same relative value from v^T in both cases, but for the original v^T page 1 was rated higher then page 3. with this new v^T it is reversed.

13.6 Using H and S

But what would have gone wrong if we used H or S instead of G? I use the equation $\pi^T(k+1) = \pi^T(k)H$ as the only dfference between H and S ares the dangling nodes, which I discuss first. When a convenient $\pi^T(0)$ is used a lot of problems immediately become apparent. For the web we do not know which $\pi^T(0)$ lead to problems, thus we needd convergence for all $\pi^T(0)$. Looking at H we see that row 9 belongs to a dangling node, meaning that when $\pi^T(0)$ is chosen as the vector of zero's except for the $nine^{th}$ entry which we set ,1 we get a rather particular result for $\pi^T(1)$. Namely that $\pi^T(1)$ is a vector of zero's. This obviously is a problem as we still have no ranking.
Remark: S does not have dangling nodes so does not have this problem. all problems that will be shown hold for both.

Another interesting choice would be $\pi^T(0)$ with entries zero except for the entry corresponding to page 10 which will have the value 1. This will lead to the following PageRank vectors for the first 2 iterations.

$$\pi^T(0) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \pi^T(1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \pi^T(2) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Now that $\pi^T(2) = \pi^T(0)$ it should be apparent that this will lead to an infinitely long sequence of vectors where position 10 and 11 switch value every step. Bottom line, this vector never converges. So we know that for H we cannot use any arbitrary starting vector. But you might think that a starting vector with the same values for page 10 and 11 would work. This is true but then there is the problem of finding the specific starting vector that would lead to convergence. As seen a cycle of two requires the same value for both. So a logical starting value would be to start each page with the same value. However consider the very simple three page system consisting of pages A, B and C, where A links to page B, page B links to page A and page C links to page A. Then the starting vector would contain values $\frac{1}{3}$. After the first iteration page A would have received PageRank value $\frac{1}{3}$ from both B and C and therefore has PageRank value $\frac{2}{3}$. While B only received from A and has PageRank value $\frac{1}{3}$. Page C has no inlinks and therefore receives no PageRank value and becomes zero. Then for the second iteration we see that page C has value zero and because page C has no inlinks will never receive value anymore, meaning that page C has no influence on the system anymore. Essentially we have created a cycle of two pages whose values are not equivalent. Of course we could find a starting vector that would work. But for a system of billions of pages it is probably faster to compute the PageRank using G than H.

Lastly I show the calculated PageRank vector corresponding to the starting vector which was also used for G, $\pi^T(0)$ with entries $\frac{1}{15}$. This vector does not lead to any immediate problems for this web structure. The problem of page 10 and 11 exchanging values is solved because they have the same value $\frac{1}{15}$ each step. In fact the vector nicely converges. Below are the first two iterations.

$$\pi^T(1) = \begin{pmatrix} 0.0556 \\ 0.0800 \\ 0.0356 \\ 0.0407 \\ 0.1222 \\ 0.0407 \\ 0.0800 \\ 0.1000 \\ 0.0333 \\ 0.0667 \\ 0.0667 \\ 0.0000 \\ 0.0000 \\ 0.0667 \\ 0.0000 \end{pmatrix}, \pi^T(2) = \begin{pmatrix} 0.0500 \\ 0.0600 \\ 0.0511 \\ 0.0522 \\ 0.0544 \\ 0.0744 \\ 0.0978 \\ 0.1033 \\ 0.0233 \\ 0.0677 \\ 0.0677 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{pmatrix}$$

Notice that $\|\pi^T(1)\|_\infty < 1$, in fact $\|\pi^T(1)\|_\infty = \frac{12}{15}$. This is because there are three dangling nodes and each one lost the value $\frac{1}{15}$. Now this is not a problem as we can normalize $\pi^T(1)$ to give PageRank scores, but it is an indicator that something is going wrong. Moreover notice that the PageRank value belonging to page 9, which is a dangling node, is not 0 but $\frac{1}{2} \cdot \frac{1}{15}$. I worte these fractions explicitly to show where the value comes from. Column 9 of H has the value $\frac{1}{2}$ which means that page 9 in the first iteration will get that value multiplied by $\frac{1}{15}$. Thus the next iteration even more PageRank score is lost out of the system as the dangling nodes do not have the PageRank zero. Dangling node 12 already has no more score, but dangling node 14 has score $\frac{1}{15}$ which will also be lost. This bleeding might stop at some moment as for instance page 13 is feeding page 14, which is leaking the score. But after the first iteration page 13 has value zero, meaning that at the second iteration page 14 will also have value zero as we see in the vector. But not all the PageRank value is lost, as we can see that the sink of page 6, 7, 8 actually increased value. The combined values of page 6, 7, 8 increased by $\frac{2}{5} \cdot \frac{1}{15}$, collective inlink value for this sink. Because the sink has no outlinks no value is sent away. Other pages such as 1, 2, 3, 4 may fluctuate in value a bit, but eventually they will lose all their value to the dangling nodes or sinks. Therefore the coverged PageRank has a lot of elements zero and only the pages in the sink have nonzero value. Meaning that we found ranking for these sink pages relative to each other but no ranking for the entire system. The converged vector to is shown below.

$$\pi^T = \begin{pmatrix} 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0905 \\ 0.1358 \\ 0.1811 \\ 0.0000 \\ 0.0667 \\ 0.0667 \\ 0.0000 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{pmatrix}, \text{rank } \pi^T = \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 3 \\ 2 \\ 1 \\ 5 \\ 4 \\ 4 \\ 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}$$

As we can see a lot of pages ended up with a PageRank value of zero, all ranked number 5. The only pages that still had some PageRank score are the sink pages and cycle pages. These pages however have the same relative rank as they did for G. The PageRank value left in the vector is approxiamtly 0.6408.

Remark When S is used instead of H then the dangling nodes do not leak value. However their value spreads over the entire system. Therefore they eventually send their value towards the sink or cycle, resulting in a similar convergent scenario as for H. The only big difference is that $\|\pi^T(k)\|_\infty = 1$ for each iteration. Noteworthy is that it takes a fair bit longer to converge.

14 Matlab coding

Below is the Matlab code I used for the experiment. Notice that it begins with some variables that can be chosen according to the experiment. To create H the matlab command sparse is very usefull, construct a F consisting of allnonzero elements in H and then run the program. The final ranking vectors this program calculates are different from the ranking vectors in the section experiment. I personally liked these vectors more, but for clarity for the reader I chose to reorder them in my paper.

This coding uses an accuracy argument based on the values inside π^T.

```
F=sparse([2 4 3 5 2 5 1 5 1 2 15 5 8 5 6 8 6 7 4 11 10 13],
[1 1 2 2 3 3 4 4 5 5 5 6 6 7 7 7 8 8 9 10 11 14],
[1/3 1/2 1 1/5 1/3 1/5 1/2 1/5 1/2 1/3 1 1/5 1/2 1/5 1/2 1/2 1/2 1 1/2 1 1 1],15,15)
H=full(F);
alpha=4/5;
delta=1/15;
a=[0 0 0 0 0 0 0 0 1 0 0 1 0 1 0]';
et=ones(1,15);
e=ones(15,1);
vt=delta*et;
vt=[1/100 20/100 1/100 1/100 40/100 1/100 1/100
 10/100 10/100 10/100 1/100 1/100 1/100 1/100 1/100];
P=a*et;
S=H+delta*P;
G=alpha*S+(1-alpha)*(e*vt);
pi=delta*et;
i=0;
argu=1/1000000000;
pi1=pi*G;
pi2=pi1*G;
for t= 1:1000
    check=pi;
    pi=pi*G;
    acc=check-pi;
    nacc=norm(acc,inf);
    if nacc > argu
        i=i+1;
    else
        i=i+1;
       break
    end
end
rank=floor(tiedrank(pi));
```

This coding describes the exact same algorithm as the first one does but uses an accuracy argument based on the rank of pages in π^T.

```
F=sparse([2 4 3 5 2 5 1 5 1 2 15 5 8 5 6 8 6 7 4 11 10 13],
[1 1 2 2 3 3 4 4 5 5 5 6 6 7 7 7 8 8 9 10 11 14],
[1/3 1/2 1 1/5 1/3 1/5 1/2 1/5 1/2 1/3 1 1/5 1/2 1/5 1/2 1/2 1/2 1 1/2 1 1 1],15,15)
H=full(F);
alpha=4/5;
delta=1/15;
a=[0 0 0 0 0 0 0 0 1 0 0 1 0 1 0]';
et=ones(1,15);
e=ones(15,1);
vt=delta*et;
vt=[1/100 20/100 1/100 1/100 40/100 1/100 1/100
10/100 10/100 10/100 1/100 1/100 1/100 1/100 1/100];
P=a*et;
S=H+delta*P;
G=alpha*S+(1-alpha)*(e*vt);
pi=delta*et;
i=0;
j=0;
pi1=pi*G;
pi2=pi1*G;
for t= 1:10000;
    granko=tiedrank(pi);
    ranko=floor(granko);
    pi=pi*G;
    grank=tiedrank(pi);
    rank=floor(grank)
    racc=rank-ranko;
    if racc == zeros(1,15);
        j=j+1;
        i=i+1;
    else
        j=1;
        i=i+1;
    end
    if j==15;
        break;
    end
end
```

15 References

1. Taher H. Haveliwala and Sepandar D. Kamvar , **The Second Eigenvalue of the Google Matrix**, http://nlp.stanford.edu/pubs/secondeigenvalue.pdf

2. Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, Gene H. Golub. **Extrapolation Methods for Accelerating PageRank Computations.** http://ilpubs.stanford.edu:8090/865/1/2003-16.pdf

3. Lada A. Adamic, **Zipf, Power-laws, and Pareto - a ranking tutorial** , vhttp://www.hpl.hp.com/research/idl/papers/ranking/ranking.html

4. **The Perron-Frobenius Theorem**, http://www.prenhall.com/divisions/esm/app/ph-linear/leon/html/perron.html

5. Ilse C.F. Ipsen and Steve Kirkland, **Convergence analysis of a PageRank updating algorithm by Langville and Meyer** ,http://www4.ncsu.edu/~ipsen/ps/simax43980.pdf

6. L. Olson, **Google and Markov Chains, Power Method, SVD** , (2009) https://www.fer.unizg.hr/_download/repository/GoogleSVD.pdf

7. L. Page, **The PageRank Citation Ranking: Bringing Order to the Web**, (2009) http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

8. Richard L. Burden, J. Douglas Faires, **Numerical Analysis**, Cengage learning Brooks/-Cole, canada , ninth edition (2011)

9. Steven J. Leon, **Linear Algebra with apllications**, Pearson education international, US, eighth edition (2010)

10. Amy N. Langville and Carl D. Meyer, **Google's Pagerank and Beyond: The Science of Search Engine Rankings** , Princeton university press (2006)

11. Kurt Bryan, Tanya Leise, **The $25,000,000,000 Eigenvector:The Linear Algebra behind Google** (2006)

12. David F. Gleich , **PageRank Beyond the web** (2014)

13. Amy Langville, **Google's PageRank and Beyond: The science of search Engine Rankings**, Slideshow (2007)

14. Rebecca S. Wils, **Google's PageRank: The Math behind The Search Engine**, (2006)

15. Ilse Ipsen and Rebecca Wills, **The Mathematics Behind Google's PageRank**

16. Cleve moler, **Experiments with Matlab**, Chapter 7, texbfGoogle Pagerank (2011)

17. Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, **Numerical Mathematics**, Spring (1991)

18. Wikipedia contributors, **Cayley-Hamilton Theorem** , *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Cayley%E2%80%93Hamilton_theorem