A PROJECT
REPORT
ON
"GOOGLE PAGERANK ALGORITHM"

BY:

2015A7PS0121G BHAGWAT SWAROOP SANJAY

SUPERVISOR: Prof. BHARAT M. DESHPANDE,
Dept. of Computer Science & Information Systems

SEMESTER-I
2017-18
Study Oriented Project – CS F266

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I will fail in our duty if I do not thank all those who contributed directly or indirectly in making this project what it is.
I am deeply indebted to **Professor Bharat M. Deshpande** for his valuable guidance and encouragement. I am grateful for his advice and useful suggestions throughout the project
I would also like to express my heartfelt gratitude to my **family** and **friends** for their unceasing love, support and concern that kept us motivated.

ABSTRACT

Information retrieval is a very broad and major area of computer science. This project report details various aspects of web search engines and more specifically on Google PageRank algorithm which deals with ranking the pages as per their importance. This study mainly focuses on the mathematics and computing involved in it. Report gives a brief history about 'Information Retrieval', 'Web search engines' etc. It also discusses about the sensitivity of various parameters involved in pagerank calculation, some optimization techniques and finally it gives some idea about including personalization component in it.

1. INTRODUCTION

Internet is a vast source of information and recreation. Information retrieval is a major branch of computer science which deals with searching for particular data in a huge data collection. Internet is such a huge collection. We always wonder that when we give a query to a search engine like Google, it gives us the answer within a second. To achieve this speedup or optimization in the search process, there are many algorithms. This study is specifically about Google pagerank algorithm its mathematical and computational aspects and implementation.

This report gives a brief introduction about information retrieval, its history and how it has been evolved over a period of time. It talks about some models which were proposed for the web search engine. It goes on talking about the basic components of web search engines such as indexing, crawling etc. The project aims at studying various aspects of ranking web pages based on their importance or some priority so that getting appropriate information would become easier.

Mathematics, especially linear algebra is an integral part of Google pagerank algorithm. The project involves study of mathematics involved in the algorithm, its modeling, incorporating the mathematical tools such that graphs, probability models and some techniques of linear algebra. This also involves study of parameters involved in the pagerank calculation, their sensitivity and some kinds of optimizations in the pagerank calculation.

Finally, the report has some implementations of pagerank algorithm and some statistical details about its parameters. Some idea is discussed about how we can add some personalization components to it so that the search engine will become personalized.

2. INFORMATION RETRIEVAL

Information Retrieval is the process of searching for data within a document collection**.** It has its long and glorious tradition. This topic discusses about the history of information retrieval, web search engines, some models and terminologies related to them.

2.1. History of Information Retrieval:

Earliest documents were painted on walls of caves. Later people started using Papyrus rolls for documentation. The invention of paper accelerated the written record of information and collection of documents. In 1989, the storage, access and search of information

were revolutionized by the invention of World Wide Web (WWW). The description of computer searching for information was first given in 1948. In 1950, the idea of automated information retrieval systems was given. After that several small as well as large scale retrieval systems were evolved.

The introduction of web search engines has catalysed the process of information retrieval heavily. In 1998, link analysis was proposed. It improved the web search dramatically and also increased the quality of search. Nearly all major search engines use the link analysis concept. Google uses a very advanced link analysis algorithm namely 'Pagerank' algorithm. As of now, there are more than 3500 search engines.

In case of web based search engines, the pages are linked, so should have an efficient way to search for the query. For this, there are some basic search models such as Boolean model, vector space model and probabilistic model.

2.2. Search Engine Models

2.2.1. Boolean Model

It uses Boolean algebra and searches for exact match for the document as per the user query. There is no concept of partial match in this model. Thus, it leads to a poor performance. To overcome this, fuzzy Boolean logic is used. But this model also fails in case of common problems if information retrieval such that synonymy and polysemy.

2.2.2. Vector Space Models

The textual data is transformed into numeric data and matrices and matrix analysis techniques are employed to discover key features and connections in the document collection. Relevance feedback model allows user to select a subset of retrieved documents that are useful. The advantage of vector space model is relevant scoring and feedback. This model also allows partial match by assigning the value for each document between 0 and 1. But it has one drawback i.e. computational expense.

2.2.3 Probabilistic Models

As the name suggests, these model estimate the probability that a user will find the result of its query in a particular document. These models operate recursively and need an underlying algorithm to determine initial parameters. These models are very hard to build and program. Probabilistic frameworks can easily accommodate preferences. For the initial parameters, user's history can be incorporated which gives better results.

2.3 Elements of Web Search Process

Indexes: These hold the valuable information about each webpage. There are three types of indexes, structured, content and special purpose. These are contained in an index module.

Ranking: Ranking module takes specific set of pages and ranks them based on some criteria. The ranking depends upon content score and popularity score. Pagerank algorithm works on popularity score.

Page Repository: Crawling software returns with new webpages which are temporarily stored as full, complete webpages in the page repository.

Crawler: They are used for retrieving the webpages and web contents. Crawler module creates virtual robots called 'spiders' that constantly look for new information in the web structure and return it to the central repository.

Retrieval Engine: It performs lookups on index tables for the query.

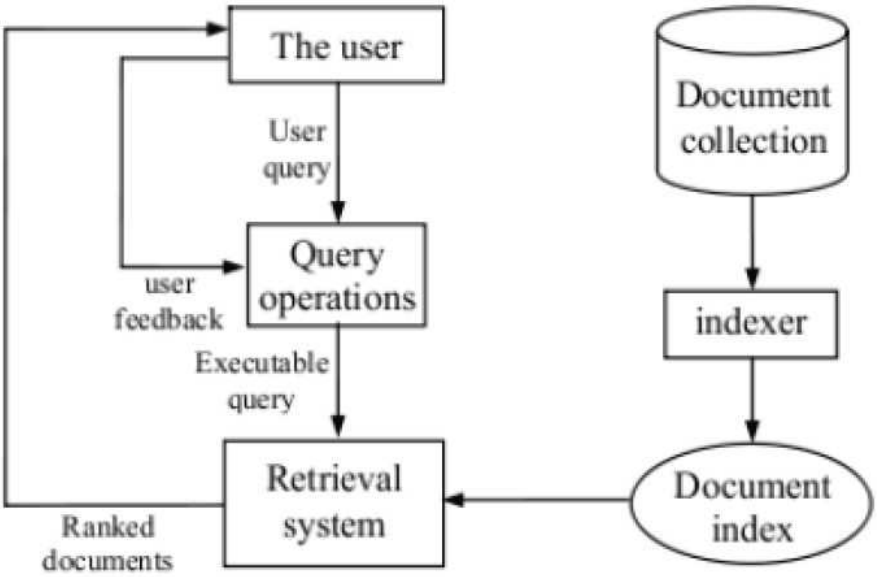


Fig.1: Search Engine System during search operation.

3.RANKING WEBPAGES

Now-a-days the information which is being added in the web network is tremendous. Information retrieval provides many tools for efficient web searching. Considering size of the web and requirement of the user, ranking webpages is the main operation that a search engine needs to perform. Search engine may return millions of webpages for a particular query but user cannot go through all of them to get the information. So ranking is helpful in web searching

There are two types of ranking mechanisms:

1. Content based rankers: They rank webpages on the basis of number of matched terms, frequency of terms, their locations etc. But these are very difficult to implement as they require rigorous text analysis of the pages.

2. **Connectivity based rankers:** They work on the principle of link analysis. Link is an edge that point from one page to another. The importance of a page is recursively defined as 'a page is important if it points to some important pages'. There are two widely used connectivity based rankers, HITS and Google pagerank algorithm

2.1. HITS

This method was given by Kleinberg. It uses in-links and out-links to determine two popularity scores for a page. This is query dependent algorithm. It thinks of webpages as authorities or hubs. Authorities are the webpages with many in-links and hubs are the webpages with many out-links. Good authorities are pointed by good authorities and good hubs point to good hubs. This method was used by IBM into their CLEVER project.

HITS provide innovative technology for web searching and topic distillation. In the CLEVER project, weight is assigned to each edge (link) based on the terms of queries and end points of the link. It also breaks large hubs into smaller hubs based on the topic.

There are some shortcomings of HITS algorithm. It assumes that all the links pointing to a particular page have same weight. But in reality, that is not necessarily true. Some in-links might have more importance than others. To overcome this issue, a new model called PHITS was proposed. It stands for Probabilistic analogue of HITS algorithm. It provides probabilistic interpretation of term-document relationship. The other issue with this model is mutual reinforcement relationship between the hosts. This occurs when a host points to a single document on another host. There is another problem called drift which arises because of interaction between local subgraph and surrounding links.

Computation of HITS score and algorithm in brief:

- Every page i has authority score x_i and hub score y_i. In each iteration these values are updated.

- e_{ij} denotes the edge from page i page j. Initial values of authority score $x_i^{(0)}$ and hub score $y_i^{(0)}$ are either computed based on user logs or assigned some value based on some probability distribution.
- In the successive iterations, the values are updated as follows.

$$x_i^{(k)} = \sum_{j:e_{ji}\in E} y_j^{(k-1)} \quad \text{and} \quad y_i^{(k)} = \sum_{j:e_{ij}\in E} x_j^{(k)} \quad \text{for } k = 1, 2, 3, \ldots$$

For computational purpose, we take adjacency matrix L of the link structure and x and y as column vectors holding the values for authority score and hub score of all pages respectively.
Now these scores can be updated as follows.

$$\mathbf{x}^{(k)} = \mathbf{L}^T\mathbf{y}^{(k-1)} \quad \text{and} \quad \mathbf{y}^{(k)} = \mathbf{L}\mathbf{x}^{(k)}$$

Here L^T denotes transpose of matrix L.
We have to perform these iterations till both x and y converge to a constant value.

By using linear algebra principles of diagonalization and eigenvalues, it can be proven that x and y converge in finite steps.

2.2. PageRank Algorithm

As per the pagerank thesis which was proposed in 1998, a page is important if it is pointed by some important pages. Pagerank value is assigned to pages rather than edges to distinguish highly relevant pages from less important ones. These values are derived based on hypothetical random web surfer. It is a model on graph with arbitrary set of nodes and transition probabilities. Teleporting step is designed to model an external influence on importance of a page. It mathematical model ensures that this importance parameter value exists and is unique.

There are many important characteristics or features of pagerank algorithm such as existence, uniqueness, generality and more importantly fast computation. There are several applications

of pagerank algorithm. It is used as a network centrality measure. It is also used to extract region of a large graph around a target set of interest. This is also used in Google search engine. This algorithm was given by Sergey Brin and Larry Page who are founders of Google.

There are many distinguishable features of this algorithm. It is content or query independent. It considers static state of the page. It does not rank a host or a web site as a whole, but it is determined for each page individually. The value of pagerank lies between 0 and 1 as it is more or less a probabilistic model.

The details, mathematics, properties and other features of this algorithm were studied as part of this project and they are discussed in the next few chapters of this report.

4. MATHEMATICS OF PAGERANK ALGORITHM

This chapter discusses the mathematical modelling of pagerank algorithm, its properties and some computational aspects.

4.1. Original Formulation

Let P_i be a page and $r(P_i)$ be sum of all pagerank values of pages pointing to P_i. Then the pagerank index of P_i is given by a classical formula,

$$r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|P_j|}$$

Where B_{Pi} denotes the set of all pages which are pointing to page P_i. But this is a recursive definition. So we need to give a base case. For this purpose, algorithm designers used an iterative procedure and assumed that pagerank of all pages is equal and has value 1/n, where n=number of pages in the cluster.

So as per the procedure we update the pagerank value of all pages in all the iterations. So the above formula is modified as follows.

$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|P_j|}$$

The process is instantiated with the base case, $r_0(P_i)=1/n$ for all i.

This formulation ensures that pagerank values converge at some constant values for a given link structure in finite number of steps.

We can give a following example to illustrate the above procedure.

Consider a directed graph of 6 pages and the pagerank values after some iterations are shown below in tubular format.

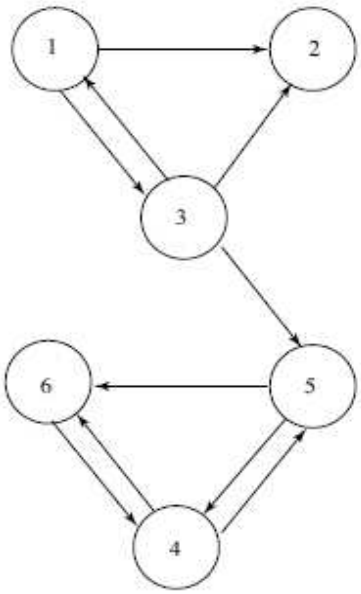


Fig.2: Graph with 6 nodes and directed links.

Now for the above graph, the pagerank values at each stage are calculated as follows,

Iteration 0	Iteration 1	Iteration 2	Rank at Iter. 2
$r_0(P_1) = 1/6$	$r_1(P_1) = 1/18$	$r_2(P_1) = 1/36$	5
$r_0(P_2) = 1/6$	$r_1(P_2) = 5/36$	$r_2(P_2) = 1/18$	4
$r_0(P_3) = 1/6$	$r_1(P_3) = 1/12$	$r_2(P_3) = 1/36$	5
$r_0(P_4) = 1/6$	$r_1(P_4) = 1/4$	$r_2(P_4) = 17/72$	1
$r_0(P_5) = 1/6$	$r_1(P_5) = 5/36$	$r_2(P_5) = 11/72$	3
$r_0(P_6) = 1/6$	$r_1(P_6) = 1/6$	$r_2(P_6) = 14/72$	2

Here, as we can see, the page with higher pagerank value is assigned higher rank. In this example, by observation also, it is very clear that page 4 should have rank 1 as it is pointed by page 5 and page 6 which have many in-links.

4.2. Computation and Matrix Representation

For fast computation, we consider a matrix H, called hyperlink matrix and a row matrix $\prod^T$. Initially, this matrix has all 1's in it. For hyperlink matrix, every cell entry is calculated as $H_{ij}=1/|P_i|$. This is a stochastic matrix and all the nonzero elements have probabilities associated with them.

Now in every iteration, we update $\prod^T$ as follows,

$$\pi^{(k+1)T} = \pi^{(k)T}\mathbf{H}.$$

We carry out this procedure till $\prod^T$ converges. Now the complexity of this procedure id $O(kn^2)$, where n is the size of row matrix and k is the number of iterations carried out. This is because in one iteration, matrix multiplication is performed which takes $O(n^2)$ time.

But as we know that H is very sparse matrix, so the above complexity can be revised as O(no. of non-zero elements in H).

In case of **dangling nodes**, i.e. nodes that do not point to any other node, one row in stochastic matrix will be 0 and that will create a trouble in calculation on pagerank values. Because of this case, H is called **sub stochastic** matrix.

There is another problem in case of **directed cycles or loops**. In that case, matrix $\prod^T$ doesn't converge; rather it oscillates between some

values, which is quite clear from the graph structure as it can't find a halting condition where the ranks can be determined.

4.3. Adjustments In The Model

Algorithm designers made some changes in the original model and incorporated idea of Markov-chains. For this, they used and idea called random surfer. Random surfer creates problems in case of dangling nodes. So to overcome this, the first adjustment was given called stochasticity adjustment. All zero rows in H matrix are replaced by $1/ne^T$, making H **stochastic**.

This adjustment guarantees that the matrix will become stochastic. But it does not guarantee that there will be finite bound on the number of iterations. For this purpose, we want to make our matrix primitive. Brin and Page achieved this by using random surfer concept. They introduced a matrix G called Google matrix. It is defined as follows,

$$\mathbf{G} = \alpha \mathbf{S} + (1-\alpha) 1/n \, \mathbf{e}\mathbf{e}^T$$

Here S is the stochastic matrix. α is a parameter having value between 0 and 1. S is defined as $S=H+a\,(1/ne^T)$, where a is dangling node vector. For this vector row $a_i=1$ if node i is dangling else it is assigned to 0.

α determines the contribution of random surfer and hyperlink matrix. If α tends to 0 then it is purely random surfer model. If it tends to 1 then it is purely based on the link structure. Generally its value is set around 0.6-0.8 for practical use.

Properties of Google Matrix

1. G is artificial in the sense that we do not use hyperlink matrix H directly, rather we use modified version of it and also we make modifications into it by introducing random surfer into it.
2. G is very dense because of introduction of $1/ne^T$ into it. This increases the computational complexity.
3. The most important characteristic is G is primitive. This ensures that a unique pagerank vector exists.

4. As it is clear from the definition of G that $G_{jj}>0$ for all j, G is not periodic.
5. By modifying hyperlink matrix, every page is directly connected to every other page with some probability. So G becomes irreducible as there are no transitive dependencies.
6. Use of matrix S and parameter α in the definition of G, makes G stochastic.

4.4. Computational Aspects

The pagerank vector is calculated using G matrix as,

$$\prod^{(k+1)T} = \prod^{kT}.G$$

This method is known as power method to compute pagerank vector. We can also state the problem of calculating pagerank vector by using linear algebra. The problem can be stated as we have to solve eigenvalue problem for matrix $\prod^{T}$. The problem formulation can be done as follows,

$$\pi^T = \pi^T \mathbf{G},$$
$$\pi^T \mathbf{e} = 1.$$

Here equation 2 ensures that $\prod^{T}$ is a probability matrix (vector) and as per the first equation, we have to find an eigenvector of G which corresponds to eigenvalue (dominant) equal to 1.
For fast computation, we can use a slightly simplified formula,

$$\begin{aligned}\pi^{(k+1)T} &= \pi^{(k)T}\mathbf{G} \\ &= \alpha\,\pi^{(k)T}\mathbf{S} + \frac{1-\alpha}{n}\,\pi^{(k)T}\,\mathbf{e}\,\mathbf{e}^T \\ &= \alpha\,\pi^{(k)T}\mathbf{H} + (\alpha\,\pi^{(k)T}\mathbf{a} + 1 - \alpha)\,\mathbf{e}^T/n.\end{aligned}$$

Here the first term $\alpha\prod^{kT}H$ can be computed in approximately O(n) time because of a property of H that many entries of H are 0.
Now the no. of iterations that are required for convergence depend on many factors like structure of the graph, α factor etc.
We can also reduce number of iterations by introducing a factor ε for error. If we set an error limit, then experimentally we require very less iterations that actual ones.

4.5. Spectrum

It is experimentally found that around 50 to 100 iterations are sufficient for power method to get the converged pagerank vector with some error limit. The rate of convergence depends on absolute value of two largest eigenvalues of the matrix G. For G, as it is a stochastic matrix, largest eigenvalue is 1 so; essentially the rate depends on $|\lambda_2|$ where λ_2 is the second largest eigenvalue.

It is computationally difficult to find λ_2 but using one relation, we can compute it. The relation is if spectrum of stochastic matrix is $\sigma(S) = \{1, \lambda_1, \lambda_2, \lambda_3, \ldots\}$ then, eigenvalue spectrum of corresponding Google matrix G is given by $\sigma(G) = \{1, \alpha\lambda_1, \alpha\lambda_2, \alpha\lambda_3, \ldots\}$.

Using this result, we can compute ordered eigenvalues of G using those of S.

Proof of above theorem is given below,

Proof. Since S is stochastic, $(1, \mathbf{e})$ is an eigenpair of S. Let $\mathbf{Q} = (\mathbf{e} \quad \mathbf{X})$ be a non-singular matrix that has the eigenvector e as its first column. Let $\mathbf{Q}^{-1} = \begin{pmatrix} \mathbf{y}^T \\ \mathbf{Y}^T \end{pmatrix}$. Then $\mathbf{Q}^{-1}\mathbf{Q} = \begin{pmatrix} \mathbf{y}^T\mathbf{e} & \mathbf{y}^T\mathbf{X} \\ \mathbf{Y}^T\mathbf{e} & \mathbf{Y}^T\mathbf{X} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{I} \end{pmatrix}$, which gives two useful identities, $\mathbf{y}^T\mathbf{e} = 1$ and $\mathbf{Y}^T\mathbf{e} = 0$. As a result, the similarity transformation

$$\mathbf{Q}^{-1}\mathbf{S}\mathbf{Q} = \begin{pmatrix} \mathbf{y}^T\mathbf{e} & \mathbf{y}^T\mathbf{S}\mathbf{X} \\ \mathbf{Y}^T\mathbf{e} & \mathbf{Y}^T\mathbf{S}\mathbf{X} \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{y}^T\mathbf{S}\mathbf{X} \\ 0 & \mathbf{Y}^T\mathbf{S}\mathbf{X} \end{pmatrix}$$

reveals that $\mathbf{Y}^T\mathbf{S}\mathbf{X}$ contains the remaining eigenvalues of S, $\lambda_2, \ldots, \lambda_n$. Applying the similarity transformation to $\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T$ gives

$$\begin{aligned}
\mathbf{Q}^{-1}(\alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T)\mathbf{Q} &= \alpha\mathbf{Q}^{-1}\mathbf{S}\mathbf{Q} + (1-\alpha)\mathbf{Q}^{-1}\mathbf{e}\mathbf{v}^T\mathbf{Q} \\
&= \begin{pmatrix} \alpha & \alpha\mathbf{y}^T\mathbf{S}\mathbf{X} \\ 0 & \alpha\mathbf{Y}^T\mathbf{S}\mathbf{X} \end{pmatrix} + (1-\alpha)\begin{pmatrix} \mathbf{y}^T\mathbf{e} \\ \mathbf{Y}^T\mathbf{e} \end{pmatrix}(\mathbf{v}^T\mathbf{e} \quad \mathbf{v}^T\mathbf{X}) \\
&= \begin{pmatrix} \alpha & \alpha\mathbf{y}^T\mathbf{S}\mathbf{X} \\ 0 & \alpha\mathbf{Y}^T\mathbf{S}\mathbf{X} \end{pmatrix} + \begin{pmatrix} (1-\alpha) & (1-\alpha)\mathbf{v}^T\mathbf{X} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \\
&= \begin{pmatrix} 1 & \alpha\mathbf{y}^T\mathbf{S}\mathbf{X} + (1-\alpha)\mathbf{v}^T\mathbf{X} \\ \mathbf{0} & \alpha\mathbf{Y}^T\mathbf{S}\mathbf{X} \end{pmatrix}.
\end{aligned}$$

Therefore, the eigenvalues of $\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T$ are $\{1, \alpha\lambda_2, \alpha\lambda_3, \ldots, \alpha\lambda_n\}$.

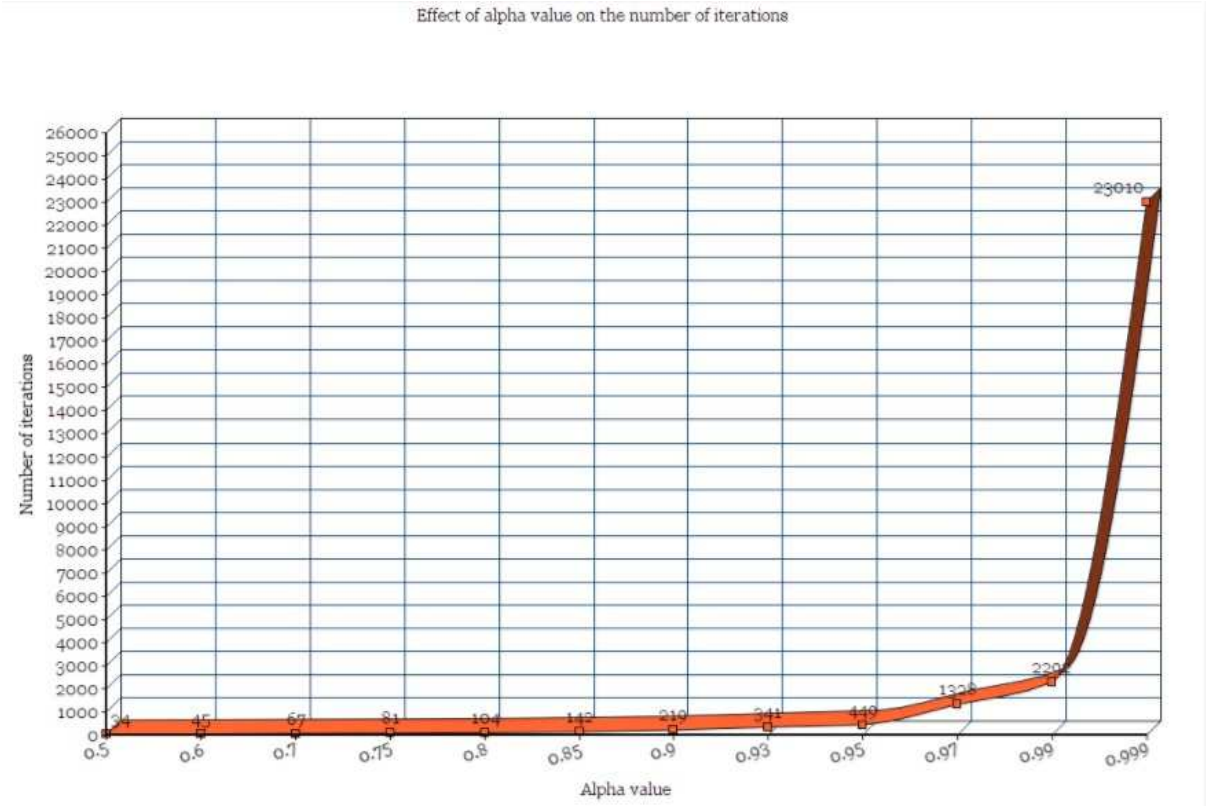
5.PARAMETERS IN PAGERANK MODEL

This chapter discusses the effect of parameters associated with the Google matrix on the number of iterations required for convergence and on overall structure and other properties.

5.1. α- Factor

Experimentally it is observed that when α tends to 1, the number of iterations required to converge the pagerank vector increases very rapidly. Generally α value is set to be .85 as per standards. This is because by using this value we can compromise between efficiency and accuracy (effectiveness).

The code is written to test the relationship between number of iterations and alpha factor. The output graph is shown below.



When α factor tends to 1, we essentially ignore the contribution from random surfer. As α tends to 1, G tends to S. As stated in the previous chapter of this report, S doesn't enforce any restriction on the convergence of pagerank vector, number of iterations increase rapidly.

5.2. Hyperlink Matrix

For a given link structure, hyperlink matrix is derived from its adjacency matrix. It is quite obvious that for a static link structure, adjacency matrix is fixed. So as we are using uniform distribution for calculating matrix H, it gets a fixed value automatically.

For observing the changes in the properties of G, we change the initial distribution of H. So instead of giving equal weightage for all pages (out-links), we give more weightage to some links over other. This also introduces personalization component to the search procedure and the effective pagerank calculation.

By using this updated H matrix, we can convert our random surfer model into an intelligent surfing model. This model jumps

between the pages and goes to content filled pages so these pages should be given more weights than other pages. So this involves text analysis techniques.

5.3. Teleportation Matrix

Algorithm designers suggested this modification to the basic pagerank matrix i.e. Google matrix construction that instead of using $1/ne^T$ i.e. uniform matrix, we can use ev^T where v^T is a probability vector or personalization vector. So by new definition, Google matrix G becomes,

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

As only $1/nee^T$ is replaced by ev^T, all other properties like rate of convergence etc. might change a bit but computational complexity, sparse vector – matrix multiplication and power method remains the same.

Personalization is introduced by incorporating this E vector. We can see this by one example. Let us say we want to surf pages about cricket then we make corresponding v_i entries tend to 1 and other entries tend to 0. By doing this with a very high probability our search query or auto suggestions will show results related to our personal topic (chosen topic).

In theory this personalization is possible, but in practice, computational cost will be very high. But several developers and researchers have created pseudo personalized search engines.

For this, we select finite number of topics and create $\prod^T (v_i^T)$ for all topics i and then finally take linear combination of all of them to get the actual pagerank vector. So for 16 topics it will be given by

$$\boldsymbol{\pi}^T = \beta_1 \boldsymbol{\pi}^T(\mathbf{v}_1^T) + \beta_2 \boldsymbol{\pi}^T(\mathbf{v}_2^T) + \cdots + \beta_{16} \boldsymbol{\pi}^T(\mathbf{v}_{16}^T)$$

Here β_i's are coefficients corresponding to each pagerank vector for each topic. Now based on our choice of these coefficients, we can give relative importance to the topics.

6. SENSITIVITY CONSIDERATIONS

This chapter discusses about the sensitivity of pagerank vector with respect to the parameters involved in the pagerank calculations.

6.1. Sensitivity with respect to α

The classical tool or approach to calculate sensitivity of a function with respect to its arguments is the derivative. Here also we can use the same approach to find the sensitivity of pagerank vector towards its components.

If we consider i^{th} element in the pagerank vector as a function of α and take its partial derivative with respect to α then we can determine its rate of change. So $d\prod^{T}(\alpha)/d\alpha$ gives an approximation of how the pagerank vector reacts when α changes.

There are few results about the sensitivity of pagerank vector with respect to α. They are as follows.

1. If $D_i(\alpha)$ are principal minor determinant of order n-1 of the matrix I-G then,

The PageRank vector is given by

$$\pi^T(\alpha) = \frac{1}{\sum_{i=1}^{n} D_i(\alpha)} \left(D_1(\alpha), D_2(\alpha), \ldots, D_n(\alpha)\right)$$

This gives the upper bound on the each individual component of derivative vector.

2. This result gives the upper bound on absolute value of derivative of each individual element of $\prod^T$ and first norm of derivative vector. It is given by,

If $\pi^T(\alpha) = (\pi_1(\alpha), \pi_2(\alpha), \ldots \pi_n(\alpha))$ *is the PageRank vector , then*

$$\left|\frac{d\pi_j(\alpha)}{d\alpha}\right| \leq \frac{1}{1-\alpha} \qquad \text{for each } j = 1, 2, \ldots, n,$$

$$\left\|\frac{d\pi^T(\alpha)}{d\alpha}\right\|_1 \leq \frac{2}{1-\alpha}.$$

This theorem also tells us that when α tends to 1, pagerank vector is more sensitive towards α.

6.2. Sensitivity with respect to H

Here we need to consider derivative with respect to every entry in the H matrix to find the sensitivity. As G depends on α, H and personalization vector v^T, we need to isolate the dependency of G with respect to H_{ij}. The derivative of pagerank vector with respect to H_{ij} is given by,

$$\frac{d\pi^T(h_{ij})}{dh_{ij}} = \alpha\pi_i(\mathbf{e}_j^T - \mathbf{v}^T)(\mathbf{I} - \alpha\mathbf{S})^{-1}$$

But in practice, hyperlink matrix if often fixed because in pagerank calculation we don't consider any dynamism in the hyperlink structure.

6.3. Sensitivity With Respect to v^T

The derivative of $\prod^T(v^T)$ with respect to v^T is given by,

$$\frac{d\pi^T(\mathbf{v}^T)}{d\mathbf{v}^T} = (1 - \alpha + \alpha\sum_{i\in D}\pi_i)(\mathbf{I} - \alpha\mathbf{S})^{-1}$$

Here, D is the set of all dangling nodes. As α tends to 1, $(I-\alpha S)^{-1}$ approach infinity so we can see that pagerank vector becomes more sensitive when α approaches infinity.

If dangling nodes contribute to a very large proportion of pagerank then, pagerank vector is more sensitive towards v^T . We can also logically see this as if collection of dangling nodes is important then out random surfer visits them more often and follows teleportation probabilities more often.

Thus we can say that, random surfer actions and distribution of pagerank vector are more sensible towards any change in the personalization vector.

- There is one result regarding first norm of updated pagerank vector after varying matrix S that,

Suppose $G = \alpha S + (1-\alpha)ev^T$ *is the Google matrix with PageRank vector* π^T *and* $\tilde{G} = \alpha\tilde{S} + (1-\alpha)ev^T$ *is the updated Google matrix (of the same size) with corresponding PageRank vector* $\tilde{\pi}^T$*. Then*

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

Here in the above result, U is the set of all pages that are updated.

7. PAGERANK USING LINEAR ALGEBRA

This chapter gives idea about how the pagerank problem can be formulated as a linear algebra problem.

7.1. Problem Formulation

We know from the basic definition from the pagerank vector that for pagerank vector (terminating vector in the last iteration) ($\prod^T$), we can rewrite the original equation as

$$\prod^T G = \prod^T$$

Now by the definition of matrix G we can rewrite the above equation as

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

By rearranging the terms, we get

$$\prod^T(I- \alpha S) = (1- \alpha)v^T$$

This is the problem formulation in linear algebra. Now this also has one constraint that $\prod^T e=1$, as $\prod^T$ is a stochastic matrix.

Now to study this linear system, we need to study the properties of matrix I- αS. Here are some of the properties of I-αS.

- I-αS is non-singular matrix.
- Sum of all elements in each row of matrix I-αS is 1-α.
- Infinity norm i.e. maximum element of the matrix I-αS is 1+α.
- I-αS is an m-matrix i.e. real part of all of its eigenvalues are positive and its off-diagonal entries are less than or equal to zero.
- Row sums of inverse of I-αS is1/(1-α) and also infinity norm of inverse of I-αS is 1/(1-α).

7.2. Google Problem Formulation

Solving the linear algebra problem,

$$x^T(I-\alpha H)=x^T$$

and letting $\prod^T=x^T/x^Te$ gives the pagerank vector corresponding to google matrix.

Proof of the above theorem is given below,

Proof. π^T is the PageRank vector if it satisfies $\pi^T G = \pi^T$ and $\pi^T e = 1$. Clearly, $\pi^T e = 1$. Showing $\pi^T G = \pi^T$ is equivalent to showing $\pi^T(I - G) = 0^T$, which is equivalent to showing $x^T(I - G) = 0^T$.

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

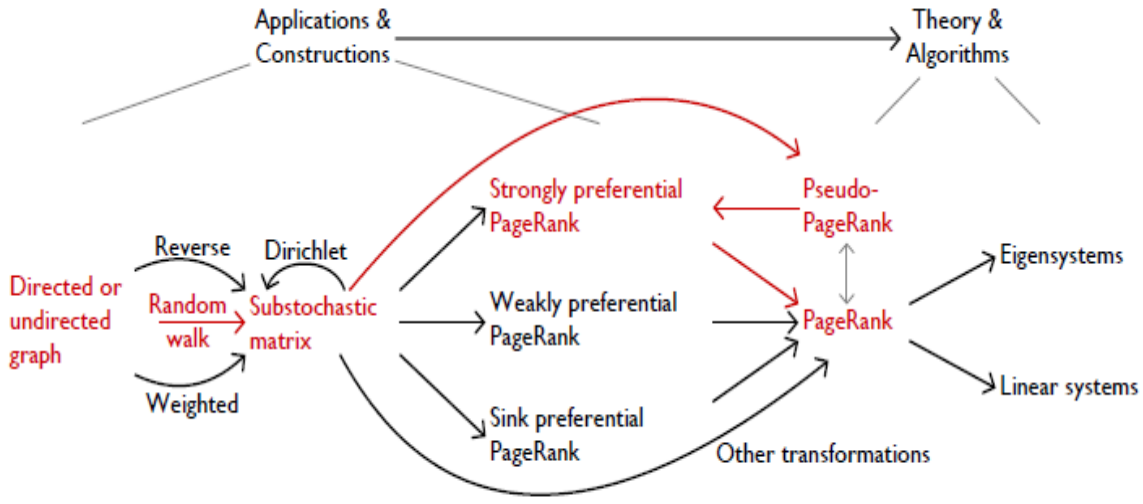
The above line results from the fact that $x^T(\alpha a + (1-\alpha)e)v^T = 1$ because

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

We can see that usually the problem is formulated using H rather that S for fast computation. We can use some techniques and precompute the results corresponding to the stationary matrix H to accelerate the computation.

7.3. Pagerank Formulations

This can be viewed as following figure,



The red coloured path shows vast variety of majority of pagerank formulations, variations and applications.

Following sections discuss the variations in the formulation

7.3.1. Reverse Pagerank

In reverse Pagerank, We compute the pagerank vector for matrix H^T i.e. we try to reverse the edges of the graph and formulate the problem. This technique is often used to determine why a particular page or node is more important than other nodes.

7.3.2. Dirichlet Pagerank

This problem is considered when we want to fix the importance of particular set of nodes. Let S be a set of nodes and for each node i in S has fixed boundary b_i. The goal is to find the solution x for following equation,

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1-\alpha)\mathbf{v} \qquad \text{where} \qquad x_i = b_i \text{ for } i \in S.$$

Here P is the block partitioning of set S.

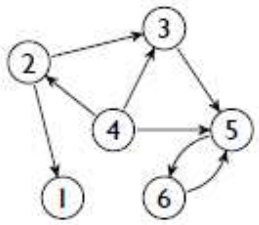
7.3.3. Weighted Pagerank

The underlying link structure that we consider in pagerank calculation is uniformly weighted graph. The weights are assigned to each edge indicating its importance. So this actually introduces the personalized web search that we talked about in the previous chapters.

7.3.4. Pagerank On Undirected Graph

A uniform random walk on connected, undirected graph has unique stationary distribution. In case of undirected graph, rows and columns of H and H^T are identical. So we can use reversible Markov chain models to formulate the problem.

The following figure shows pagerank problem's different variations and problem formulation.



A directed graph

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 3 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The adjacency matrix, degree vector, and correction vector

Random walk

$$\bar{\mathbf{P}} = \begin{bmatrix} 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 1/2 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1/3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\bar{\mathbf{P}} = \mathbf{A}^T \mathbf{D}^+$$

Strongly preferential

$$\mathbf{P} = \begin{bmatrix} 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/3 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 1 & 1/3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{P} = \bar{\mathbf{P}} + \mathbf{v}\mathbf{c}^T$$

Weakly preferential

$$\mathbf{P} = \begin{bmatrix} 1/6 & 1/2 & 0 & 0 & 0 & 0 \\ 1/6 & 0 & 0 & 1/3 & 0 & 0 \\ 1/6 & 1/2 & 0 & 1/3 & 0 & 0 \\ 1/6 & 0 & 0 & 0 & 0 & 0 \\ 1/6 & 0 & 1 & 1/3 & 0 & 1 \\ 1/6 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{P} = \bar{\mathbf{P}} + \mathbf{u}\mathbf{c}^T$$
$$\mathbf{u} \neq \mathbf{v}$$

Reverse

$$\bar{\mathbf{P}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 1 & 1/2 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1/3 & 0 \end{bmatrix}$$

$$\bar{\mathbf{P}} = \mathbf{A}\,\mathrm{diag}(\mathbf{A}^T\mathbf{e})^+$$

Dirichlet

$$\bar{\mathbf{P}} = \begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/3 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$S = \{2, 3, 4, 5, 6\}$

$$\bar{\mathbf{P}} = \bar{\mathbf{P}}_{\bar{S},\bar{S}}$$
$S \subset V$

Weighted

$$\bar{\mathbf{P}} = \begin{bmatrix} 0 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3/10 & 0 & 0 \\ 0 & 3/4 & 0 & 3/10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4/10 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\bar{\mathbf{P}} = (\mathbf{D}_W\mathbf{A}^T)\,\mathrm{diag}(\mathbf{A}\mathbf{D}_W\mathbf{e})^+$$
$\mathbf{D}_W$ is a diagonal weighting matrix, e.g. total degree here

IMPLEMENTATION

Here is the implementation of pagerank algorithm which is done as a part of this project. The implementation is done in python. There are two implementations. First one is the naïve implementation and second is based on the personalization vector concept.

Code :

```
import numpy as np
from scipy.sparse import csc_matrix

def pageRank(G, s = .85, maxerr = .0001):

    n = G.shape[0]

    # transform G into markov matrix A
    A = csc_matrix(G,dtype=np.float)
    rsums = np.array(A.sum(1))[:,0]
    ri, ci = A.nonzero()
    A.data /= rsums[ri]
```

```
    # bool array of sink states
    sink = rsums==0

    # Compute pagerank r until we converge
    ro, r = np.zeros(n), np.ones(n)
    while np.sum(np.abs(r-ro)) > maxerr:
        ro = r.copy()
        # calculate each pagerank at a time
        for i in xrange(0,n):
            # inlinks of state i
            Ai = np.array(A[:,i].todense())[:,0]
            # account for sink states
            Di = sink / float(n)
            # account for teleportation to state i
            Ei = np.ones(n) / float(n)

            r[i] = ro.dot( Ai*s + Di*s + Ei*(1-s) )

    # return normalized pagerank
    return r/float(sum(r))



if __name__=='__main__':

    G = np.array([[0,0,1,0,0,0,0],
                  [0,1,1,0,0,0,0],
                  [1,0,1,1,0,0,0],
                  [0,0,0,1,1,0,0],
                  [0,0,0,0,0,0,1],
                  [0,0,0,0,0,1,1],
                  [0,0,0,1,1,0,1]])
print pageRank(G,s=.86)
```

Observed Output:

```
/usr/bin/python2.7/home/swaroop/basic.py
[0.12727557   0.03616954   0.12221594   0.22608452   0.28934412   0.03616954
0.16274076]

Process finished with exit code 0
```

This array printed as an output is the pagerank vector of the graph specified in the code.

Code:

```
import numpy as np
import networkx as nx

def pagerank(G, alpha=0.85, personalization=None,
             max_iter=100, tol=1.0e-6, nstart=None,
weight='weight',
             dangling=None):

    if len(G) == 0:
        return {}

    if not G.is_directed():
        D = G.to_directed()
    else:
        D = G

    # Create a copy in (right) stochastic form
    W = nx.stochastic_graph(D, weight=weight)
    N = W.number_of_nodes()

    # Choose fixed starting vector if not given
    if nstart is None:
        x = dict.fromkeys(W, 1.0 / N)
    else:
        # Normalized nstart vector
        s = float(sum(nstart.values()))
        x = dict((k, v / s) for k, v in nstart.items())

    if personalization is None:

        # Assign uniform personalization vector if not given
        p = dict.fromkeys(W, 1.0 / N)
    else:
        s = float(sum(personalization.values()))
        p = dict((k, v / s) for k, v in personalization.items())

    if dangling is None:
        # Use personalization vector if dangling vector not
specified
        dangling_weights = p
    else:
        missing = set(G) - set(dangling)
        s = float(sum(dangling.values()))
        dangling_weights = dict((k, v / s) for k, v in
dangling.items())
    dangling_nodes = [n for n in W if W.out_degree(n,
weight=weight) == 0.0]

    # power iteration: make up to max_iter iterations
    for _ in range(max_iter):
        xlast = x
        x = dict.fromkeys(xlast.keys(), 0)
        danglesum = alpha * sum(xlast[n] for n in dangling_nodes)
```

```
        for n in x:

           for nbr in W[n]:
                x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
            x[n] += danglesum * dangling_weights[n] + (1.0 -
alpha) * p[n]

        err = sum([abs(x[n] - xlast[n]) for n in x])
        if err < N * tol:
            return x
    raise NetworkXError('pagerank: power iteration failed to
converge '
                        'in %d iterations.' % max_iter)

if __name__=='__main__':

    G = barabasi_albert_graph(60,41)
    pp=nx.pagerank(G)
    print(pp)
```

Observed output:

/usr/bin/python2.7/home/swaroop/PageRank.py

```
[0: 0.012774147598875784, 1: 0.013359655345577266, 2: 0.01315735573137792
4,
3: 0.012142198569313045, 4: 0.013160014506830858, 5: 0.012973342862730735
,
 6: 0.012166706783753325, 7: 0.011985935451513014, 8: 0.01297350269606171
8,
9: 0.013374146193499381, 10: 0.01296354505412387, 11: 0.01316322032606333
2,
 12: 0.013368514624403237, 13: 0.013169335617283102, 14: 0.01275207180052
0563,
15: 0.012951601882210992, 16: 0.013776032065400283, 17: 0.012356820581336
275,
18: 0.013151652554311779, 19: 0.012551059531065245, 20: 0.012583415756427
995,
 21: 0.013574117265891684, 22: 0.013167552803671937, 23: 0.01316552858340
0423,
 24: 0.012584981049854336, 25: 0.013372989228254582, 26: 0.01256941607684
8989,
 27: 0.013165322299539031, 28: 0.012954300960607157, 29: 0.01277609197339
7076,
 30: 0.012771016515779594, 31: 0.012953404860268598, 32: 0.01336494785400
5844,
33: 0.012370004022947507, 34: 0.012977539153099526, 35: 0.013170376268827
118,
 36: 0.012959579020039328, 37: 0.013155319659777197, 38: 0.01356714713313
7161,
 39: 0.012171548109779459, 40: 0.01296692767996657, 41: 0.028089802328702
826,
 42: 0.027646981396639115, 43: 0.027300188191869485, 44: 0.02689771667021
551,
 45: 0.02650459107960327, 46: 0.025971186884778535, 47: 0.025852625713319
37,
```

```
48: 0.02565482923824489, 49: 0.024939722913691394, 50: 0.0245827119770140
2,
 51: 0.024263128557312528, 52: 0.023505217517258568, 53: 0.02372431187257
8157,
 54: 0.02312908947188023, 55: 0.02298716954828392, 56: 0.0227022066330039
6,
 57: 0.022060403216132875, 58: 0.021932442105075004, 59: 0.02164328863262
3502]
```

Process finished with exit code 0

Here, the pagerank values are printed along with the rank. Using networkx library in python, a random graph was generated and the values are printed. Here I have set maximum iterations = 100 and tolerance = 10^{-6}.

REFERENCES

- **Books :**

- Google Pagerank and Beyond - Amy. N. Langville and Carl. D. Mayer., Princeton University Press, 2006

- First Course in Graph Theory- Gary Chartrand and Ping Zhang, Western Michigan University.

- Google's PageRank: The Math Behind the Search Engine, Amy. N. Langville and Carl. D. Mayer., Mathematical Intelligencer, 2006

❖ Websites:

- http://www.webworkshop.net/pagerank.html#how_is_pagerank_calculated

- http://pr.efactory.de/e-further-factors.shtml

- https://searchengineland.com/what-is-google-pagerank-a-guide-for-searchers-webmasters-11068

- http://www.geeksforgeeks.org/page-rank-algorithm-implementation/

❖ Papers and Other References:

- D. Aldous and J. A. Fill. Reversible Markov chains and random walks on graphs. 2002. Unfinished monograph, recompiled 2014

- A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank computation and the structure of the web: Experiments and algorithms. In Proceedings of the 11th international conference on the World Wide Web. 2002.

- A. Farahat, T. LoFaro, J. C. Miller, G. Rae, and L. A. Ward. Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization. SIAM Journal on Scientific Computing, 27 (4), pp. 1181{1201, 2006.

- P. Berkhin. A survey on PageRank computing. Internet Mathematics, 2 (1), pp. 73{120, 2005.