FINDING STEADY STATE OF NETWORK FLOW USED FOR OPTIMIZING SEARCH ENGINE RESULTS

VATSAL SHAH AU2040019 USHMAY PATEL AU2040253
DEEP PATEL AU2040250 NIHAAR PATEL AU2040182

ABSTRACT:
Each and Every system in the modern world holds a saturation point above which there is no possible way to optimize the system**.** So using concepts of Markov chain and network flow the aim is to optimize browsing results. A Markov chain or Markov process is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. A network flow is a directed graph where each edge has a capacity and each edge receives a flow, in graph theory. The aim is to improve the efficiency of search engine results by using the previous state achieved.

INTRODUCTION:

The network flow is a graph that connects nodes and edges and forms a flow system. It works as a transport system using network chains that have very limited space. Examples are the traffic problem on roads and the water system of the city through the connection of pipes. The nodes are connected in pairs and make a flow of data and maintain a net amount of flow. The concept of network flow is applied at many places like in a network of wiring, the connection of sewage pipes, and fluid networks. In this case, we are specifically using the Markov Matrix to illustrate the stochastic process between two nodes.

BACKGROUND:

We all are familiar with web browsing and internet surfing. Have you ever wondered how Google responds so quickly to our search and provides us with results? Network flow efficiency is the reason why this is possible. Search engines spend large amounts of money on algorithms and equations to achieve maximum network flow efficiency. We are primarily using a steady-state analysis of network flow. Steady-state analysis of network flow means analyzing a flow in the network until it reaches a steady-state.

MOTIVATION:

Problems related to Network flow are considerably used in real-life scenarios. One of the most prominent motivations was to understand the approach behind the well-known PageRank algorithm that's used by Google to rank pages. With ever-rising competition in ranking web- pages on a search engine there arises a prominent question on how these pages are ranked.

Whenever we look up something on a search- engine we get millions and millions of websites and webpages as the results. Since there are millions of pages that are recovered as the result it becomes of high significance to figure out which are the most relevant pages so that the user shouldn't have to waste time looking up for results and pass through web pages and should get the asked result at the top. This process of ranking pages based on whether the user after searching significant terms relates to them stays on a webpage or is directed to some other web page. or simply moving to another webpage is useful info. This was the primary boost to discover some subtle stuff about how the theory of linear algebra is used to rank the web page.

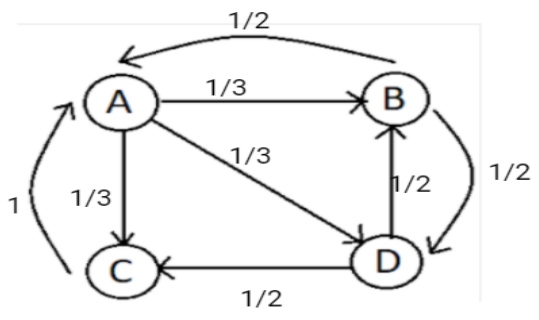
LITERATURE SURVEY:

We have applied the project in Python because it consumes less time and is more effective and readable. We have used libraries like NumPy and pandas. We have used concepts of matrix and eigenvectors. With python, we have reduced our time for running the program.

How Pagerank works?

Pagerank is defined as "A classical method used to arrange the web page according to its objective and the usage of terms involved in it on WWW by using any link data structure". Pagerank is a "Vote" by all other pages on the web about how important a page is. A link to the page counts as a vote of support.

Connection of web pages

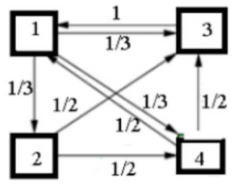


$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1/2 & 1 & 0 \\ B & 1/3 & 0 & 0 & 1/2 \\ C & 1/3 & 0 & 0 & 1/2 \\ D & 1/3 & 1/2 & 0 & 0 \\ \hline \text{Sum} = & 1 & 1 & 1 & 1 \end{array}$$

In the above figure A,B,C and D are the web pages and the arrows are the link connections between the web pages. We made the above matrices from the figure.

Representation of graphs/network as matrix

Network is a directed graph. Nodes can be represented as spheres and edges can be represented as arrows. A network can be shown as the NxN matrix where N is the number of nodes.



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

Steady state of network flow

The stabilised flow of the network results in such a situation that the relative data, number of visitors at the nodes are formed according to the potential. Percentage of nodes/data is unchanged irrespective of the flow after a certain number of iterations.

For example: Matrix= [[0.7, 0.2], [0.3, 0.8]]

This matrix can be shown as the following network:
In this graph the flow of population at A is divided into two parts where 70% of the population will stay at A and 30% will move to B. Whereas for B 20% will move to A and 80% stays at B.

Suppose Initial Population = 100 at A and B

Total=200

In the first iteration of the flow the respective populations are: 90 and 110

We go on iterating till a point where the exchange of flow does not affect the

population of both the nodes which means:

30%A=20%B

At Steady state solution: [40.003381, 59.996619]

Final Population: 80 and 120

The dominant eigenvector of this matrix is: (2,3) which asserts our fact.

1) Ranks on a given node are always dependent on other webpages

Formula for calculating the rank of a node: R(j+1)=A*r(j)

According to our assumption as stated earlier our starting r =

[1/n,1/n,1/n,1/n,1/n.........] for n nodes

Then we find out the eigenvalues as eigen values and eigen vectors.

Now we decompose the matrix to $A = SDS^{-1}$ where S contains the eigenvectors;

Whereas D is diagonal matrix containing eigenvalues

Now we scale it to kth degree so that $A^k = SD^kS^{-1}$

Now once we get $A^k \cdot U_0 = U_k$ which was our equation

After a certain number of iterations when the value starts to plateau, we can say that the network has attained a steady state.

This means the vector that we finally get is the vector that represents the rank of

the nodes in the given graph, which is useful in ranking them from the highest to the lowest.

Steady State:

A process in Markov model is generally dependent on the past values. The given values are all transition probabilities.

In a network flow we start off with the assumption that larger volumes of data flow through a node that is highly connected. But a higher connectivity does not necessarily imply higher volumes of data or higher number of visitors. What matters is how well is the node connected with other nodes of higher importance. We intend to do that with our approach.

This approach works because the probability of number of visitors/data going from one node to another depends upon the previous iteration. We capitalize on the fact that rank of a node only depends on the previous rank. So, we find the eigenvalues and eigenvectors to exponentiate in order to iterate. Then once we start iterating, we find out that the value becomes still after a certain number of iterations. This is because eigenvalues play an important role in network stability. The buckling loads of objects to construction sites to vibrational measurements everything obtains a stability at the eigen values. This happens because the sum of probability is 1 and in Markov Matrix the sum of probabilities of a column generally give out the values of dominant eigenvalue.

Here in our case the dominant eigenvalues is 1.

Suppose A has n EigenVectors that are Linearly Independent

$$v1, v2, v3, \ldots\ldots\ldots\ldots vn$$
$$\lambda1, \lambda2, \lambda3, \ldots\ldots\ldots\ldots \lambda n$$

Any vector in the sub space can be represented as

$$x = C_1v_1 + C_2v_2 + C_3v_3 \cdots + C_nv_n$$

We keep on Multiplying A and get to the following result:

$$\mathrm{A}v_1 = \lambda_1 v_1$$

$$\mathrm{A}x = C_1\lambda_1v_1 + C_2\lambda_2v_2 + C_3\lambda_3v_3 + \cdots + C_n\lambda_nv_n$$

$$\mathrm{A}^2x = C_1\lambda_1^2v_1 + C_2\lambda_2^2V_2 + \cdots + C_n\lambda_n^2v_n$$
$$A^nx = c_1\lambda_1^nv_1 + C_2\lambda_2^nV_2 + \cdots + C_n\lambda_n^nV_n$$

$$A^nx = \lambda_1^n\,(c_1v_1\,)$$
$$A^kn = C_1\lambda_1\,kv_1$$

$$A^kx = \lambda^2\,(C_1v_1 + C_2v_2\,(\,\frac{\lambda2}{\lambda1}\,)^{2\cdots}\,C_nv_n\,(\,\frac{\lambda2}{\lambda1}\,)^k\,)$$

The dominant eigenvalue we take is common and then we keep on iterating and get stabler values. It is a good approach that provides stability i.e. steady state.

The eigenvector related to the dominant eigenvector is the steady-state of the system. It is indeed an eigenvector of the matrix as expected in the results.

Proof using row operations:

$$m = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

This is an input matrix of R3. Now we write the characteristic equation for the matrix

$$m = \begin{bmatrix} a-\lambda & d & y \\ b & e-\lambda & h \\ c & f & i-\lambda \end{bmatrix}$$

Here applying the row operation we add the elements.

$$m = \begin{bmatrix} a+b+c-\lambda & d+e+f-\lambda & y+h+i-\lambda \\ b & e-\lambda & h \\ c & f & i-\lambda \end{bmatrix}$$

$$s = a+b+c$$
$$s = d+e+f$$
$$s = g+h+i$$

$$m = \begin{bmatrix} s-\lambda & s-\lambda & s-\lambda \\ b & e-\lambda & h \\ c & f & i-\lambda \end{bmatrix}$$

Therefore, we can take s- λ common and it becomes the factor of the determinant. So, if s=1 then one of the eigenvalues of the input matrix will be 1.

Detailed Linear Algebra Concepts in context of our code:

• Matrix Multiplication
• Eigenvalues and Eigenvectors
• Inverse of a Matrix
• Jacobi Algorithm
• Diagonalization of a matrix

Matrix Multiplication:

In Calculation of eigenValues of Given Matrix
In Diagonalization to Multiply S, D and S^{-1} Matrices
In finding Inverse of a matrix to multiply row reduced matrix

Eigen-Values:
Jacobi Method is used to find them. Then EigenValues are placed into a diagonal matrix while diagonalization plays an important role as the core topic of our code revolves around eigen centrality at the steady-state.

EigenVectors using Jacobi:
Jacobi method is an iterative method which is used to find out Eigenvalues and Eigenvectors of real symmetric matrices only. To understand this method let's take a symmetric matrix A. The First step is that we find the largest non-diagonal element. Since this is symmetric there will be two of them. They can be represented as a[p][q] and a[q][p].Now we define a rotation matrix J1 in such a way that sin Θ will be at position A[q][p] and A[p][q] and cos Θ will be at position a[p][p] and a[q][q] and all other diagonal elements will be one and all non-diagonal element will be zero.

Now we will multiply the A with J1^T and then again multiply the product of these two matrices with the J matrix. Let the obtain matrix be A1(A1=J1t*A*J). Now A1 will be a symmetric matrix. If A1 is not a diagonal matrix, then perform the Jacobi method on it and obtain another rotation matrix J2, and then again try to diagonalize the matrix A2 by multiplying it with the transpose of J2 and then multiplying this product with J2. If the obtained matrix is a non diagonalized matrix then repeat the same process until the resultant matrix is diagonalized. The resulting matrix consists of EigenValues as the diagonal elements Eigenvectors are simply the multiplication of J1*J2*…. Jn i.e. all the rotation matrices.

$$A' = J(p_1 q_1 \theta)^t A J(p_1 q_1 \theta)$$

$$\phi = \cot 2\theta = (Aqq - App)/2Apq$$

$$\tan\theta = \Phi + \sqrt{(\varnothing + 1)} = I_1 * I_2 * I_3 \cdots$$

Inverse of a Matrix:
Invertible Matrix Theorem states that every n x n is row equivalent to an n x n Identity Matrix. The matrix A-Lambda(I) must be non-invertible in order to find eigenvalues is a subtle concept we use. For the second part we use Inverse function in order to invert the matrix S. We do that by row reduction. We start off with AX=I where I is Identity matrix. Now we row reduce it to echelon form using row operations till a point that the augmented matrix [A | I] is converted [E(A),A^{-1}].

Equations:

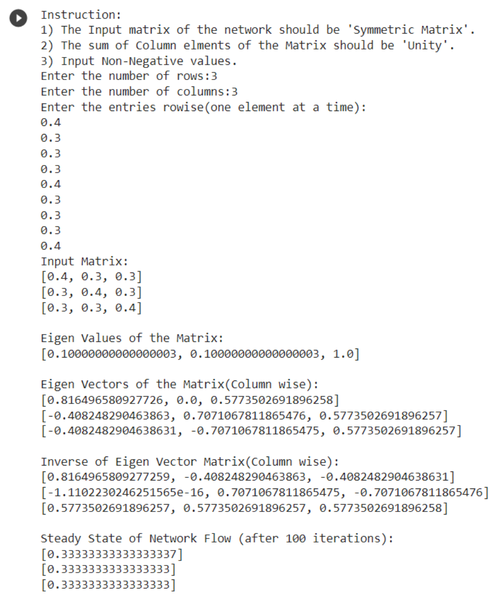
$$AA^{-} = I$$
$$Ax = I$$
$$Ax_1 = e_1$$
$$Ax_2 = e_2$$
$$Ax_3 = e_3$$
$$\cdots$$
$$Ax_n = e_n$$
$$[A|e_j]$$
$$[I|x]$$
$$x \longrightarrow A^{-1}$$

Row Reduction:
Although we do not have an explicit function defined for row reduction and we do it within the inverse function it becomes an indispensable concept of linear algebra that is used. According to Invertible Matrix theorem, we have the idea of inverting using row reduced echelon form.

Diagonalization (Eigen-Decomposition) And Final Calculations for ranks:
Diagonalization is used in the context of our project to split the input matrix into S, D and S^{-1} so that we can exponentiate the matrix and thus iterate with ease.

$$A = SDS^{-1}$$
$$A(A) = SDS^{-1}\,(SDS^{-1})$$
$$A^2 = SD^2S^{-1}$$
$$A^3 = SD^3S^{-1}$$
$$\cdots$$
$$A^k = SD^kS^{-1}$$
$$A^k \cdot U_0 = U_k$$
$$SD^kS^{-1}U_0 = U_k$$
$$C = S^{-1}U_0$$
$$SD^kC = U_k$$

Result of our codes



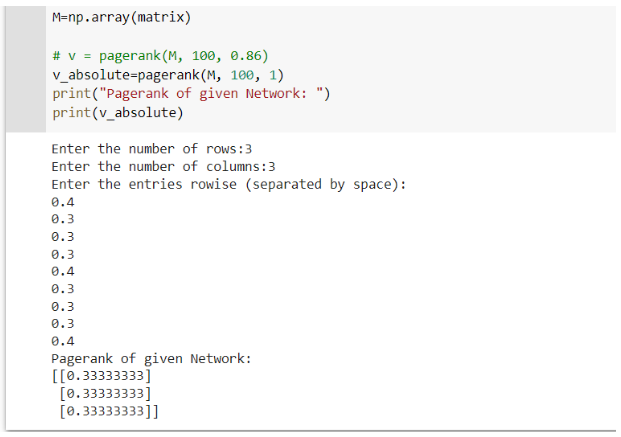
```
Instruction:
1) The Input matrix of the network should be 'Symmetric Matrix'.
2) The sum of Column elments of the Matrix should be 'Unity'.
3) Input Non-Negative values.
Enter the number of rows:3
Enter the number of columns:3
Enter the entries rowise(one element at a time):
0.4
0.3
0.3
0.3
0.4
0.3
0.3
0.3
0.4
Input Matrix:
[0.4, 0.3, 0.3]
[0.3, 0.4, 0.3]
[0.3, 0.3, 0.4]

Eigen Values of the Matrix:
[0.10000000000000003, 0.10000000000000003, 1.0]

Eigen Vectors of the Matrix(Column wise):
[0.816496580927726, 0.0, 0.5773502691896258]
[-0.408248290463863, 0.7071067811865476, 0.5773502691896257]
[-0.4082482904638631, -0.7071067811865475, 0.5773502691896257]

Inverse of Eigen Vector Matrix(Column wise):
[0.8164965809277259, -0.408248290463863, -0.4082482904638631]
[-1.1102230246251565e-16, 0.7071067811865475, -0.7071067811865476]
[0.5773502691896257, 0.5773502691896257, 0.5773502691896258]

Steady State of Network Flow (after 100 iterations):
[0.33333333333333337]
[0.3333333333333333]
[0.3333333333333333]
```



```
M=np.array(matrix)

# v = pagerank(M, 100, 0.86)
v_absolute=pagerank(M, 100, 1)
print("Pagerank of given Network: ")
print(v_absolute)
```

```
Enter the number of rows:3
Enter the number of columns:3
Enter the entries rowise (separated by space):
0.4
0.3
0.3
0.3
0.4
0.3
0.3
0.3
0.4
Pagerank of given Network:
[[0.33333333]
 [0.33333333]
 [0.33333333]]
```

EXPLANATIONS:

This is our main method. Here we invoke our predefined methods and obtain the results.

1) We have predefined matrix multiplication and inverse.

2) Eigenvectors and eigenvalues are calculated using the Jacobi method.

3) The Max Element function defined in Jacobi method gives us the largest non-diagonal element of the matrix which plays the decisive role in rotation of the matrix.

4) This method then reduces the off diagonal elements to 0 in order to obtain the eigenvalues using the Formula Off(A') ^2= Frobenius Norm -Square of trace of matrix

5) Eigen Vectors are given by J1*J2*...... Eigen Vectors give S and S^{-1} (by invoking inverse function) Populate D with EigenValues at the Position D[i][j] s.t i=j using for loops.

6) Now we have our S, D, and S^{-1}. Now here in our strategy to reduce time complexity, we apply the fact that while scaling S*D*$S^{-1}$and multiplying it again with S*D*S^{-1} the S*S^{-1}in between the expression makes for an identity matrix.

7) This optimizes our approach as we use the basic multiplication algorithm which takes O(n^3) complexity. We directly scale the values of Lambda Eigenvalues up till k, which the desired number of iterations we would like to multiply S*D*S^{-1}in order to get A^k.

8) Then we multiply (A^k) *P_0 which is our population matrix.

The P0 is the matrix that consists of columns as the populations available at the Nodes/ Webpages. The final multiplication in the mentioned earlier step gives us the Steady State of the Network Flow.

The final outcome will give us the steady state of the network.

Modules Used: Numpy (Only for making arrays, Linear algebra functions are defined separately)

CONCLUSIONS:

Steady state of network flow is an eigenvector of the input matrix.

Page Rank Algorithm gives similar results because of Eigen centrality.

Repetitive analysis helps result in a steady state of network flow and also gives out relevant outcomes for algorithms like Pagerank.

CONTRIBUTIONS:

AU2040250-Deep Patel

Coding

Writing Report

Research about Markov Chain and Steady-state analysis.

Mathematical reason of study about Eigenvalues and Eigenvectors.

Analysis about the Eigen Vector and Eigen values and Diagonalization and inculcating them in the project

AU2040019-Vatsal Shah

Coding

Writing Report

Research about Pagerank Algorithm

Research about Jacobi Algorithm and how eigenvalues are obtained in a diagonal matrix through rotation matrices.

Research about the how steady state of network flow is linked with Page Rank

AU2040253-Ushmay Patel

Coding

Writing Report

Connection of web pages

Research about Pagerank Algorithm

Research about how to inculcate Matrix Multiply in the Code

Researched how Steady state of Network flow is linked with PageRank.

AU2040182-Nihaar Patel

Coding

Writing Report

Research about Markov Chain and Steady state analysis.

Researched how Steady state of Network flow is linked with PageRank.

REFERENCES:

Online help. Steady State Vector of a Markov Chain - Maple Help. (n.d.). Retrieved November 24, 2021, from https://www.maplesoft.com/support/help/maple/view.aspx?path=examples%2FSteadyStateMarkovChain#:~:text=A%20common%20question%20arising%20in,vector%20of%20the%20Markov%20chain.

Math 312 - markov chains, Google's PageRank algorithm. (n.d.). Retrieved November 24, 2021, from https://www2.math.upenn.edu/~kazdan/312F12/JJ/MarkovChains/markov_google.pdf.

Using row reduction to calculate the inverse and the ... (n.d.). Retrieved November 24, 2021, from http://www.math.pitt.edu/~annav/0290H/row_reduction.pdf.

Andrew Chamberlain, P. D. (2017,

October 4). *Using eigenvectors to find steady state population flows*. Medium. Retrieved November 24, 2021, from https://medium.com/@andrew.chamberlain/using-eigenvectors-to-find-steady-state-population-flows-cd938f124764.

Machado Google PageRank Linear Algebra Project - UNCG. (n.d.). Retrieved November 24, 2021, from https://mathstats.uncg.edu/sites/yasaki/publications/machado-google-pagerank-linear-algebra-project.pdf.

D. S. (2019, July 16). *Introduction to markov chains*. Medium. Retrieved November 25, 2021, from https://towardsdatascience.com/introduction-to-markov-chains-50da3645a50d.

Page rank algorithm and Implementation. GeeksforGeeks. (2021, November 6). Retrieved November 25, 2021, from https://www.geeksforgeeks.org/page-rank-algorithm-implementation/.