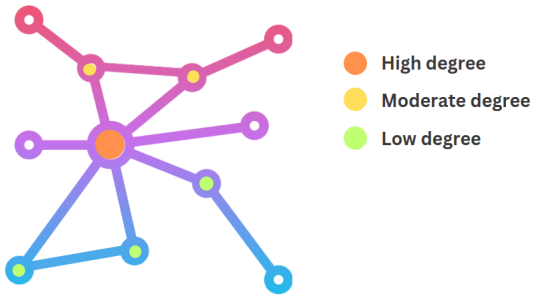
Centrality Metrics in Graph Node Networks

By: Gargie Tambe, Asish Bharadwaj, Chetan Vellanki

June 2023

Introduction

Centrality measures play a critical role in the field of graph analytics as they provide valuable insights into the importance and influence of individual nodes within a network. By quantifying the centrality or "centrality" of nodes, researchers can identify key entities that have a significant impact on network dynamics, control the flow of information, or act as vital connectors within the network structure.



SOCIAL NETWORKING DEGREE CENTRALITY

The concept of centrality encompasses multiple perspectives, allowing researchers to evaluate the significance of nodes from various angles. This versatility is particularly valuable, as it enables the analysis of complex systems such as human brain networks or social networks, where understanding the role and importance of individual nodes is crucial for comprehending the overall system dynamics.

This review aims to explore and evaluate the key characteristics and limitations of commonly used centrality measures, focusing on Degree Centrality, Betweenness Centrality, and Eigenvector Centrality. Additionally, we will demonstrate how centrality measures can be effectively combined with linear algebraic concepts to gain a deeper understanding of the operation and behavior of complex systems.

Furthermore, we will delve into real-world scenarios and practical applications of centrality analysis, showcasing how these measures have been successfully utilized in diverse fields such as social network analysis, disease spread modeling, recommendation systems, and urban planning.

Types of Centrality Metrics

- **Degree Centrality**

  Degree centrality is a fundamental measure in network analysis that quantifies the importance of a node within a graph based on its degree, which represents the number of connections it has to other nodes. It serves as a simple and intuitive indicator of node prominence, with nodes having a higher degree of centrality considered more central or influential in the network. Degree centrality is widely used in various domains, including social network analysis, transportation networks, and biological networks, providing valuable insights into key nodes and network structure. By considering the immediate connections of a node, degree centrality offers a valuable perspective on node importance without implying any negative connotations.

  In a given graph $G(V, E)$, for a given vertex v, the degree centrality is defined as the following -

  $$C_D(v) = deg(v)$$

  Time complexity: Calculating the degree centrality for all nodes in a graph, it takes $\Theta(V^2)$ time in a dense adjacency matrix and for edges, $\Theta(E)$ time in a sparse matrix representation.

  Let $X(Y, Z)$ be the $|Y|$ node connected graph, with $|Y|$ vertices and $|Z|$ edges, that maximizes the following quantity (where $y*$ is node with highest degree centrality in $|X|$:

  $$H = \sum_{j=1}^{|Y|} [C_D(y*) - C_D(y_j)]$$

  So, the degree centrality of the graph G will be:

  $$C_D(G) = \frac{\sum_{i=1}^{|V|} [C_D(v*) - C_D(V_i)]}{H}$$

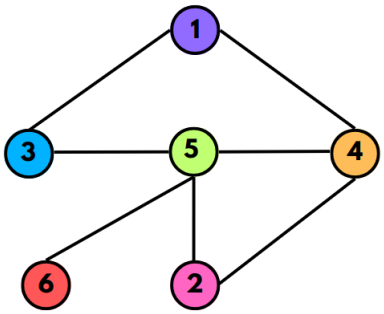
  The value of H is maximized when graph X contains one central node to which all other nodes are connected (star topology). In that case,

  $$H = (n-1) \times ((n-1) - 1) = n^2 - 3n + 2$$

To summarize,

$$DC(v) = \frac{Number\ of\ edges\ connected\ to\ node\ v}{Total\ number\ of\ nodes\ -1}$$

Let's consider the following graph -



The degree of each of the nodes would be as follows -

Node	Degree
1	2
2	2
3	2
4	3
5	4
6	1

For each node, the degree centrality would be calculated as follows -
$DC(v)$ = Fraction of nodes connected to it

Node	Degree	Centrality
1	2	$2/5 = 0.4$
2	2	$2/5 = 0.4$
3	2	$2/5 = 0.4$
4	3	$3/5 = 0.6$
5	4	$4/5 = 0.8$
6	1	$1/5 = 0.2$

Code in Python for calculating Degree Centrality -

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph([(1, 3), (1, 4), (2, 4), (2, 5), (3, 5), (4, 5), (5, 6)])
nx.draw(G, with_labels=True)
nx.degree_centrality(G)
```

Output would be -

```
{1: 0.4, 3: 0.4, 4: 0.6000000000000001, 2: 0.4, 5: 0.8, 6: 0.2}
```

The same formula can be applied for a graph with directed edges or edges having edge weights in order to calculate the most "significant" or influential nodes in a graph.

By utilising this metric, it is possible to identify influential individuals in a network, including those with the most extensive contact networks, those transitioning rapidly between peers, and those with access to the most essential information.

- **Betweenness Centrality**

  Betweenness centrality is a metric that assesses the significance of a node in a network based on its frequency in the geodesic distance or shortest path between all pairs of nodes. It identifies nodes that serve as connectors between distinct groups within the network, playing a pivotal role in controlling the flow of interactions or information. Although originating from social network analysis, betweenness centrality finds applicability in various network types, including social, transportation, and computer networks.

  The betweenness centrality of a vertex v in a graph can be calculated using the following formula, which involves linear algebra:

$$BC(v) = \sum_{u,w \ \epsilon \ V} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

  where V represents the set of all nodes in the graph, σ_{uw} denotes the total number of shortest paths between node u and w, and $\sigma_{uw}(v)$ is the total number of shortest paths between nodes u and w that pass through vertex v.

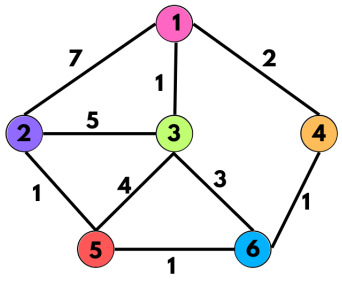
  Following steps are followed to calculate the betweenness centrality of a vertex v in a graph -

  1. Calculate unique pairs of nodes (unique edges): Start by considering all possible pairs of nodes in the graph. These pairs represent the edges or connections between nodes.
  2. Determine the shortest paths: Find the shortest paths between all pairs of nodes in the graph. This can be done using algorithms such as Dijkstra's algorithm or Floyd-Warshall algorithm. In case of a weighted graph, the paths should be minimal weighted paths.
  3. Calculate the sum of all fractions of shortest paths through vertex v: For each pair of nodes (u, w) calculate the number of shortest paths between them (σ_{uw}) and the number of paths that pass through the

vertex v ($\sigma_{uw}(v)$). Then, sum up the fractions $\frac{\sigma_{uw}}{\sigma_{uw}(v)}$ for all pairs of nodes in which vertex v lies on the shortest path.

4. Normalize the betweenness centrality (BC): If you are comparing networks of different sizes, it is common practice to normalize the betweenness centrality scores. This normalization takes into account the fact that larger networks tend to have higher centrality scores. There are various normalization techniques, such as dividing each node's betweenness centrality by the maximum possible betweenness centrality value.

Let's consider the following example to calculate Betweenness Centrality,



Pseudocode to find Betweenness Centrality -

```
function betweenness_centrality(graph):
    for each node in the graph:
        // Initialize
        shortest_paths_count = {}
        dependency = {}
        stack = {}
        distance = {}
        num_shortest_path = {}

        // BFS to find shortest path in the graph and to count number of
        shortest paths
        distance[node] = 0
        num_shortest_paths[node] = 1

        queue = {}
        enqueue(queue, node)
        while the queue is not empty:
            current_node = dequeue(queue)
            push(stack, current_node)

            for each neighbour in neighbours(current_node):
```

```
                if neighbour is not visited:
                    mark neighbour as visited
                    enqueue(queue, neighbour)
                    distance[neighbor] = distance[current_node] +
                    edge_weight(current_node, neighbor)
                    num_shortest_paths[neighbor] = num_shortest_paths[neighbor]
                    + num_shortest_paths[current_node]

        for each node in the graph:
            dependency[node] = 0.0

        // Calculate dependency values
        while stack is not empty:
            current_node = pop(stack)

            for each neighbor in neighbors(current_node):
                if distance[neighbor] = distance[current_node] +
                edge_weight(current_node, neighbor):
                    dependency[neighbor] = dependency[neighbor] +
                    (num_shortest_paths[neighbor] /
                    num_shortest_paths[current_node]) *
                    (1 + dependency[current_node])

            betweenness[current_node] = betweenness[current_node] +
            dependency[current_node]

    // Normalize the betweenness centrality values
    for each node in graph:
        betweenness[node] = betweenness[node] / ((graph_size - 1) *
        (graph_size - 2))

    return betweenness
```

Code in Python -

```
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib notebook

G = nx.Graph()
G.add_edge(1, 2, weight = 7)
G.add_edge(1, 3, weight = 1)
G.add_edge(1, 4, weight = 2)
G.add_edge(2, 3, weight = 5)
G.add_edge(5, 2, weight = 1)
G.add_edge(3, 5, weight = 4)
```

```
G.add_edge(3, 6, weight = 3)
G.add_edge(5, 6, weight = 1)
G.add_edge(4, 6, weight = 1)
G.edges(data = True)

nx.draw_networkx(G, with_labels = True)

betweenness = nx.betweenness_centrality(G, weight = 'weight')
print(betweenness)
```

Output -

```
{1: 0.1, 2: 0.0, 3: 0.0, 4: 0.30000000000000004,
5: 0.3666666666666667, 6: 0.4833333333333334}
```

For example, in the given graph, the betweenness centrality would be as follows -

Edge	σ_{uw}	$\sigma_{uw}(v)$	$\frac{\sigma_{uw}(v)}{\sigma_{uw}}$
(1, 2)	1	1	1
(1, 3)	1	0	0
(1, 5)	1	1	1
(1, 6)	1	1	1
(2, 3)	3	0	0
(2, 5)	1	0	0
(2, 6)	1	0	0
(3, 5)	2	0	0
(3, 6)	1	0	0
(5, 6)	1	0	0

Hence betweenness centrality of 4 is: 3

Edge	σ_{uw}	$\sigma_{uw}(v)$	$\frac{\sigma_{uw}(v)}{\sigma_{uw}}$
(1, 2)	1	1	1
(1, 3)	1	0	0
(1, 5)	1	1	1
(1, 4)	1	0	0
(2, 3)	3	1	1/3
(2, 5)	1	0	0
(2, 4)	1	1	1
(3, 5)	2	1	1/2
(3, 4)	1	0	0
(5, 4)	1	1	1

Hence betweenness centrality of 6 is: 4.83

Edge	σ_{uw}	$\sigma_{uw}(v)$	$\frac{\sigma_{uw}(v)}{\sigma_{uw}}$
(1, 2)	1	1	1
(1, 3)	1	0	0
(1, 4)	1	0	0
(1, 6)	1	0	0
(2, 3)	3	2	2/3
(2, 4)	1	1	1
(2, 6)	1	1	1
(3, 4)	1	0	0
(3, 6)	1	0	0
(4, 6)	1	0	0

Hence betweenness centrality of 5 is: 3.67

Edge	σ_{uw}	$\sigma_{uw}(v)$	$\frac{\sigma_{uw}(v)}{\sigma_{uw}}$
(1, 4)	1	0	0
(1, 3)	1	0	0
(1, 5)	1	0	0
(1, 6)	1	0	0
(4, 3)	1	0	0
(4, 5)	1	0	0
(4, 6)	1	0	0
(3, 5)	2	0	0
(3, 6)	1	0	0
(5, 6)	1	0	0

Hence betweenness centrality of 2 is: 0

Edge	σ_{uw}	$\sigma_{uw}(v)$	$\frac{\sigma_{uw}(v)}{\sigma_{uw}}$
(1, 2)	1	0	0
(1, 4)	1	0	0
(1, 5)	1	0	0
(1, 6)	1	0	0
(2, 4)	1	0	0
(2, 5)	1	0	0
(2, 6)	1	0	0
(4, 5)	2	0	0
(4, 6)	1	0	0
(5, 6)	1	0	0

Hence betweenness centrality of 3 is: 0

Edge	σ_{uw}	$\sigma_{uw}(v)$	$\frac{\sigma_{uw}(v)}{\sigma_{uw}}$
(4, 2)	1	0	0
(4, 3)	1	1	0
(4, 5)	1	0	0
(4, 6)	1	0	0
(2, 3)	3	0	0
(2, 5)	1	0	0
(2, 6)	1	0	0
(3, 5)	2	0	0
(3, 6)	1	0	0
(5, 6)	1	0	0

Hence betweenness centrality of 1 is: 1

Maximum possible betweenness centrality for an undirected graph (M) is:

$$\frac{(N-1)\times(N-2)}{2} = \frac{(6-1)\times(6-2)}{2} = 10$$

Vertex	Betweenness Centrality	Normalized BC (= BC/M)
1	1	0.1
2	0	0
3	0	0
4	3	0.3
5	3.67	0.367
6	4.83	0.483

The betweenness centrality metric provides a unique perspective on the significance of nodes compared to other centrality measures. It focuses on individuals who have control over the flow of data throughout the network, including brokers, controllers, intermediaries, gatekeepers, and similar roles. Removing nodes with high betweenness centrality from the network would have the most disruptive impact on the interactions within the network.

- **Eigenvector Centrality**

  Eigenvector Centrality is a metric that assesses the importance of nodes in a graph based on their connections to highly important nodes. It assigns scores to nodes, considering that a node contributes more to its score if it is connected to other nodes with high scores. This measure is calculated using an iterative algorithm that converges to stable scores. Eigenvector Centrality is used in various domains, such as social network analysis, biology, and recommendation systems, to identify influential nodes within a network.

  The Eigenvector Centrality (EVC) of a node in a network can be computed using the adjacency matrix of the network.

For a graph $G(V, E)$ where $|V|$ is the number of vertices of the graph network. $A = (a_{v,t})$ is the adjacency matrix of the graph. $a_{v,t} = 1$ if there exists an edge between vertex v and vertex t and $a_{v,t} = 0$, otherwise.

The formula for calculating EVC is as follows:

$$x_v = \frac{1}{\lambda} \sum_{t \epsilon M(v)} x_t = \frac{1}{\lambda} \sum_{t \epsilon G} a_{v,t} x_t$$

where $M(v)$ is the set of neighbours of v and λ is the eigenvalue.

Multiplying both LHS and RHS by λ on both sides, we see that $\sum_{t \epsilon G} a_{v,t} x_t$ is the product of the adjacency matrix A with the vector v i.e. Ax. The LHS part of the product of vector x, i.e. λx.

Rearranging, we get, $Ax = \lambda x$.

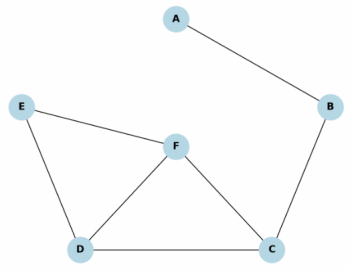
Generally, there exist many values of Eigenvalue λ for which non-zero Eigenvector solutions exist.

But according to the Perron-Frobenius theorem, a particular Eigenvalue should be chosen according to the adjacency matrix. Specifically, for a non-negative square matrix (such as an adjacency matrix) with entries greater than or equal to zero, the Perron-Frobenius theorem states that there exists a non-negative real eigenvalue called the Perron eigenvalue, denoted as λ_{max}, which is greater than or equal to the absolute value of all other eigenvalues.

Furthermore, the Perron eigenvalue λ_{max} has a corresponding non-negative eigenvector called the Perron eigenvector, denoted as v, which has strictly positive entries. In the context of an adjacency matrix, which represents the connections between nodes in a graph, the Perron eigenvalue and the corresponding Perron eigenvector have important interpretations.

The Perron eigenvalue λ_{max} represents the dominant eigenvalue of the adjacency matrix and provides information about the connectivity and structure of the graph. Its magnitude indicates the rate of growth of a random walk on the graph, and it determines the stability and behavior of various dynamic processes on the graph.

For example, consider the following graph -



The adjacency matrix of the above graph would be as follows -

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The degree of each node is -

Node	Degree
A	1
B	2
C	3
D	3
E	2
F	3

Now, representing the degree of all nodes initially before any iteration in the form of a vector -

$$V = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 3 \end{bmatrix}$$

Next, we calculate the Eigenvector centrality of each node.

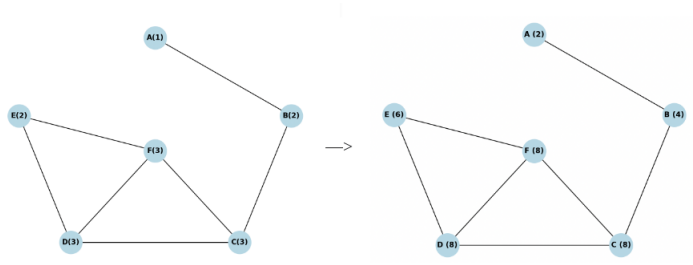
After first iteration,

$$x_1 = Ax_0$$

$$x_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 \\ 2 \\ 3 \end{bmatrix}$$

$$x_1 = \begin{bmatrix} 2 \\ 4 \\ 8 \\ 8 \\ 6 \\ 8 \end{bmatrix}$$

The effect of first iteration of multiplication can be visualised as shown below:



As we see above, the nodes C, D, E have the highest scores of 8 since these nodes are connected to multiple nodes (3) each with high degrees (importance). But, the node A has a least score of only 2 as it was connected only with a single node B. We can observe that the degree of a node after i^{th} iteration is the sum of the degrees of the nodes to which it is connected to before the iteration. For example the degree of F after 1st iteration is,

$$degree(E) + degree(D) + degree(C) = 2 + 3 + 3 = 8$$

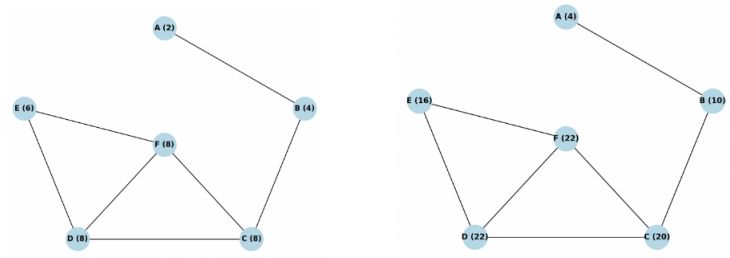
Next, we repeat the same process for the second iteration.

$$x_2 = Ax_1$$

$$x_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \\ 8 \\ 8 \\ 6 \\ 8 \end{bmatrix}$$

$$x_2 = \begin{bmatrix} 4 \\ 10 \\ 20 \\ 22 \\ 16 \\ 22 \end{bmatrix}$$

The effect of second iteration of multiplication can be visualised as shown below:



Elaborating, after the first iteration of multiplication, each node gets it's EVC score from its direct(1st degree) neighbours.

In the second iteration, when we multiply the resultant vector again with the adjacency matrix, each node again gets it's EVC score from its direct neighbours but the difference in the second iteration is that this time, the scores of the direct neighbours have already been impacted by their own direct(1st degree) neighbours previously(from the first iteration of multiplication) which eventually helps the EVC score of any node to be a function of its 2nd degree neighbouring nodes as well.

In subsequent iterations of multiplication, the EVC score of graph nodes keeps getting updated by getting impacted by EVC scores from neighbouring nodes of farther degree (3rd, 4th and so on).

Repeated multiplication makes the EVC score of every node to eventually be a function of or dependent on several degrees of its neighbouring nodes, thereby providing a globally accurate EVC score for each node.Usually the process of multiplying the EVC vector with the adjacency matrix is repeated until the EVC values for nodes in the graph reach an equilibrium or stop showing appreciable change.

If we normalise the vector and then calculate the subsequent normalised EVC vectors, after certain iterations the the EVC values of the nodes would saturate and do not undergo a further change for further iteration.

If we normalise the vector and then calculate the subsequent normalised EVC vectors, after certain iterations the the EVC values of the nodes would saturate and do not undergo a further change for further iteration.

State Of The Art

The paper entitled "GFT centrality: A new node importance measure for complex networks" authored by Rahul Singh, Abhishek Chakraborty, and B.S. Manoj was published in the journal Physica A in 2017. This paper presents a novel centrality measure for networks and highlights the comparative advantages of employing this metric in contrast to traditional centrality measures like Degree Centrality, Betweenness Centrality, and Eigenvector Centrality.

This paper introduces Graph Fourier Transform Centrality (GFT-C), a spectral approach for evaluating the importance of nodes in complex networks. The proposed method utilizes the Graph Fourier Transform (GFT) coefficients of

an importance signal associated with a reference node. The importance signal reflects how other nodes perceive the reference node within the network. By considering the global smoothness or variations of this signal, the method captures the information regarding node importance. The GFT coefficients of the importance signal are then employed to obtain a global perspective on the reference node.

Identifying central nodes is essential for designing efficient communication networks and recognizing key individuals in social networks. GFT-C is introduced as a metric that incorporates both local and global characteristics of a node to quantify its importance in a complex network. The GFT-C of a reference node is estimated based on the GFT coefficients derived from the corresponding importance signal. Comparative analyses demonstrate the superiority of GFT-C over traditional centrality measures such as degree centrality, betweenness centrality, closeness centrality, eigenvector centrality, and Google PageRank centrality across various arbitrary and real-world networks with different degree-degree correlations.

In summary, this paper proposes Graph Fourier Transform Centrality (GFT-C) as a method for assessing node importance in complex networks. GFT-C utilizes the GFT coefficients of an importance signal associated with a reference node, capturing the global smoothness or variations of the signal. The importance signal reflects how other nodes perceive the reference node individually. The GFT-C metric incorporates both local and global characteristics of a node and outperforms traditional centrality measures across diverse networks.

Applications and Related Works

- Degree centrality analysis in social networking plays a pivotal role in understanding the structure and dynamics of social relationships. In directed networks, there are two kinds of degrees, in-degree $d_{in}(v)$ that is the number of edges ending in v i.e. (u, v) and out-degree $d_{out}(v)$ that is the number of edges by leaving from v i.e. (v, u). An example of out-degree centrality as a measure of importance is an information forwarding network in an organization, where the person that forwards information to the most people would be the most important. The weight of a link can play a role in the calculation of importance, for example when the strength of social relationships is measured in a social network. In weighted networks you can have the measures of weighted in-degree centrality and weighted out-degree centrality, which are the sum of the incoming and outgoing weights of a node respectively.

  By measuring the number of connections a node has in a network, degree centrality provides insights into individual popularity, influence, and information dissemination potential. It helps identify influential hubs, brokers, and opinion leaders who serve as bridges between different communities, facilitating the spread of ideas and social influence. Conversely, it highlights individuals who may be isolated or peripheral, offering insights into

barriers to information diffusion and social integration. Overall, degree centrality analysis informs marketing strategies, community engagement initiatives, and interventions aimed at understanding and leveraging social network dynamics.

- Betweenness centrality has diverse applications, and one notable use case is in the analysis of criminal or terrorist networks. Nodes with high betweenness centrality in such networks often indicate crucial or highly involved actors. These nodes exert control over the flow of data among different groups, serving as bridges for information dissemination. Nodes with high betweenness centrality hold favorable positions in the network, facilitating greater control over information propagation. As a consequence, they become potential single points of failure. Disrupting or removing these nodes can effectively hinder communication and disrupt the network's operations.

- The PageRank algorithm, employed by major companies like Google for website ranking, incorporates eigenvector centrality as a fundamental element. Treating the World Wide Web as a directed graph, PageRank evaluates the significance of web pages based on the quantity and quality of incoming links. By iteratively computing centrality scores using eigenvectors and eigenvalues, PageRank establishes a ranking system that assigns priority to influential pages. This algorithmic application has revolutionized web search, providing more accurate and relevant search results, and enhancing the overall search experience for users.

  Google's PageRank algorithm starts by assigning initial scores to web pages, which are then updated iteratively based on the scores of the linking pages. The algorithm represents the web graph as a stochastic matrix, known as the transition matrix, and computes the principal eigenvector of this matrix to obtain the PageRank scores. This eigenvector represents the stationary distribution of the random walk on the web graph and determines the relative importance of each page. By leveraging eigenvector centrality, PageRank considers the global structure of the web graph and measures the importance of a page based on its connections to other influential pages, revolutionizing website ranking and enhancing search engine results.

References

- Centrality Metrics for Graphs - Works Blog
- Graph Centrality Measures: Types and Explanation - Turing
- Centrality - Wikipedia
- Eigenvector Centrality (Centrality Measure) - GeeksforGeeks

- Degree Centrality (Centrality Measure) - GeeksforGeeks
- Betweenness Centrality (Centrality Measure) - GeeksforGeeks

Work Contribution

- Gargie Tambe: Worked on Degree Centrality, reports and SOTA.
- Asish Bharadwaj: Worked on Betweenness Centrality, presentation and SOTA.
- Chetan Vellanki: Worked on Eigen-Vector Centrality.