SNA PROJECT

Barabási-Albert Algorithm

Velicity

Uggirala Rudhvi (20UCS213)

Chinnaga Somasekhar (20UCS056)

Mihir Bagadia (20UCS116)

Abstract:

In this report we discuss about these following topics related to a scale free network generated using the Barabasi-Albert (BA) algorithm with 10,000 nodes.

- Initialization of network
- Plotting of the degree distribution
- Highest degree node
- Analysis of the node entry time vs degree
- Visualization of centrality measures
- Calculation of giant component ratio
- Information passing experiment to test efficiency of information spreading with different probabilities.

Overall, this report provides a comprehensive analysis of scale free network generated using the BA algorithm and its properties.

Pre-Requisites:

Basic Knowledge in SNA, Python, and its libraries

Libraries Used:

NetworkX, NumPy, random, matplotlib

```
import networkx as nx
import numpy as np
import random as rd
import matplotlib.pyplot as plt
```

Introduction

- Scale-free networks are complex systems that exhibit unique structural properties, such as power law used for the distribution of links to nodes. Understanding the properties of scale-free networks is crucial in various fields, including social networks, biological networks, etc.

- **The Barabasi-Albert (BA) algorithm** is a widely used model for generating scale-free networks, which incorporates the concept of preferential attachment. In this report, we implement the BA algorithm to generate a scale-free network denoted as G with 10,000 nodes.

The report is organized as follows.

- First for generating the scale-free network G, we provide details about implementation of the BA algorithm including the initialization.
- Then, we present the results of the analysis, including plots of the degree distribution, node entry time vs. degree, and centrality measures visualization.
- We also discuss about the findings of the giant component ratio and the information passing experiment.
- Finally, we conclude with conclusions drawn from the analysis of the generated scale-free network G.

Problem Statement

Implement the BA Algorithm to generate the scale-free network S over 100,000 nodes. You may assume the initialization (for BA) as per your wish but adhering to the characteristics laid about by the BA Model. State your initialization clearly.

The Barabasi-Albert (BA) algorithm

The algorithm starts with a small initial network and then iteratively adds new nodes with preferential attachment, where the probability of connecting to an existing node is proportional to its degree.

Initialization:

For this implementation, we will start with an initial network of 100 nodes,

```
initial_nodes = 100
final_nodes = 10000
m = 3
```

We have taken our initial graph as Wheel Graph. This initial network will serve as the seed network for the BA algorithm.

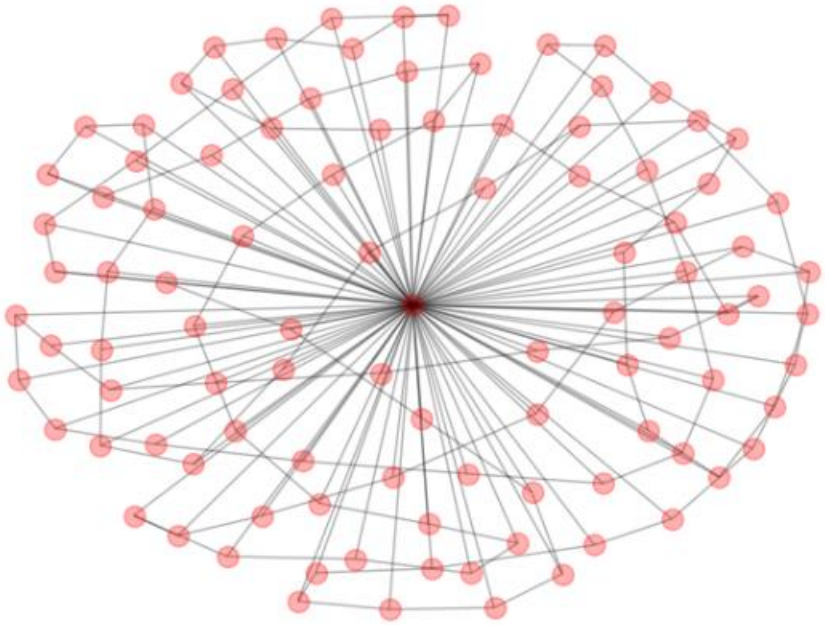
We have also initialized the time of at which the wheel graph nodes entered the network as t=0.

```
G = nx.wheel_graph(initial_nodes)

max_degree = 1
max_degree_node = 0

total_degree = 0
node_times = []
for node, d in G.degree():
    total_degree += G.degree(node)
    node_times.append(0)

    if d>max_degree:
        max_degree=d
        max_degree_node = node

nodes_probability = [G.degree(node) / total_degree for node in G.nodes()]


# print(node_times)
print(f"Node {max_degree_node} has maximum degree of the initialized graph")
# print(node_times[max_degree_node])

COLOR = '#FF0000'
nx.draw(G, alpha = .3, node_color = COLOR, node_size=100)
```

```
Node 0 has maximum degree of the initialized graph
```

Here the nodes were denoted from 0 to 99.

In the Wheel graph, centre node (I.e., node 0) has the highest degree.



Implementation the BA Algorithm

Growth: At each step we add a new node with m links that connect to m nodes in the network

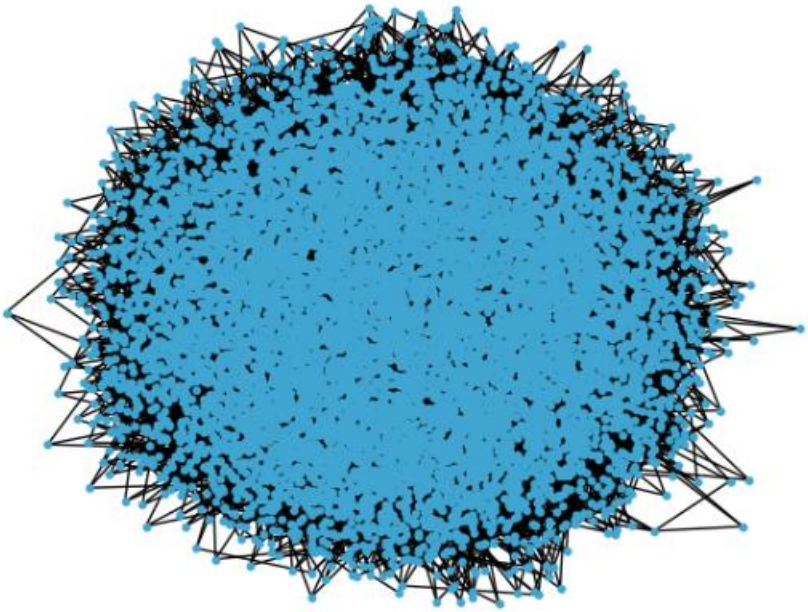
Preferential Attachment: The probability of the link that the new node connects with a node with degree k is.

$$\Pi(k_i) = \frac{k_i}{\sum_j k_j}.$$

```
for new_node in range(initial_nodes, final_nodes):

  # nodes_t = list(G.nodes())

  #choose
  random_probability_node = np.random.choice(G.nodes(),size=m,replace=False, p=nodes_probability)

  # random_probability_node = rd.choices(nodes_t,k=m, weights=nodes_probability)
  node_times.append(new_node - initial_nodes + 1)

  #adding the new_node to the Graph
  G.add_node(new_node)

  #connecting m links from new_node to nodes random_probability_node
  for i in range(m):
    G.add_edge(new_node, random_probability_node[i])


  total_degree = 0
  for node, d in G.degree():
      total_degree += G.degree(node)

      if d>max_degree:
          max_degree=d
          max_degree_node = node

  #updating the every node probability after adding a new_node
  nodes_probability = [G.degree(node) / total_degree for node in G.nodes()]

print(f"Node {max_degree_node} has the highest degree of {max_degree} at time t = {node_times[max_degree_node]}")
```

```
Node 0 has the highest degree of 996 at time t = 0
```

The scale free network with 10,000 nodes generated using BA algorithm.

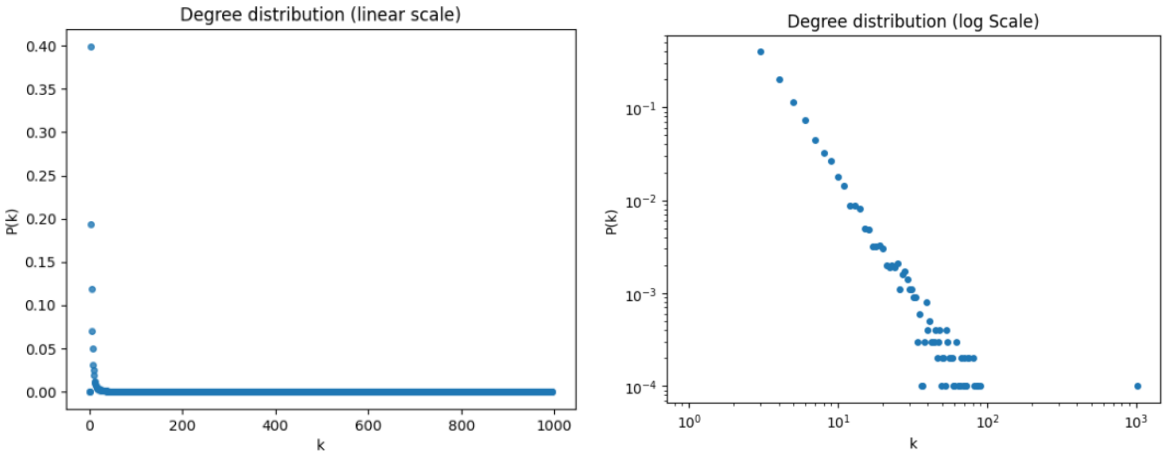


Problem statement:

Plot the degree distribution of the above scale-free network S. Find the node with the highest degree and find at what time interval this node came into the network when you generated the network S

Degree Distribution:

The distinguishing feature of the networks generated by the Barabási-Albert model is their power-law degree distribution.



Highest Degree:

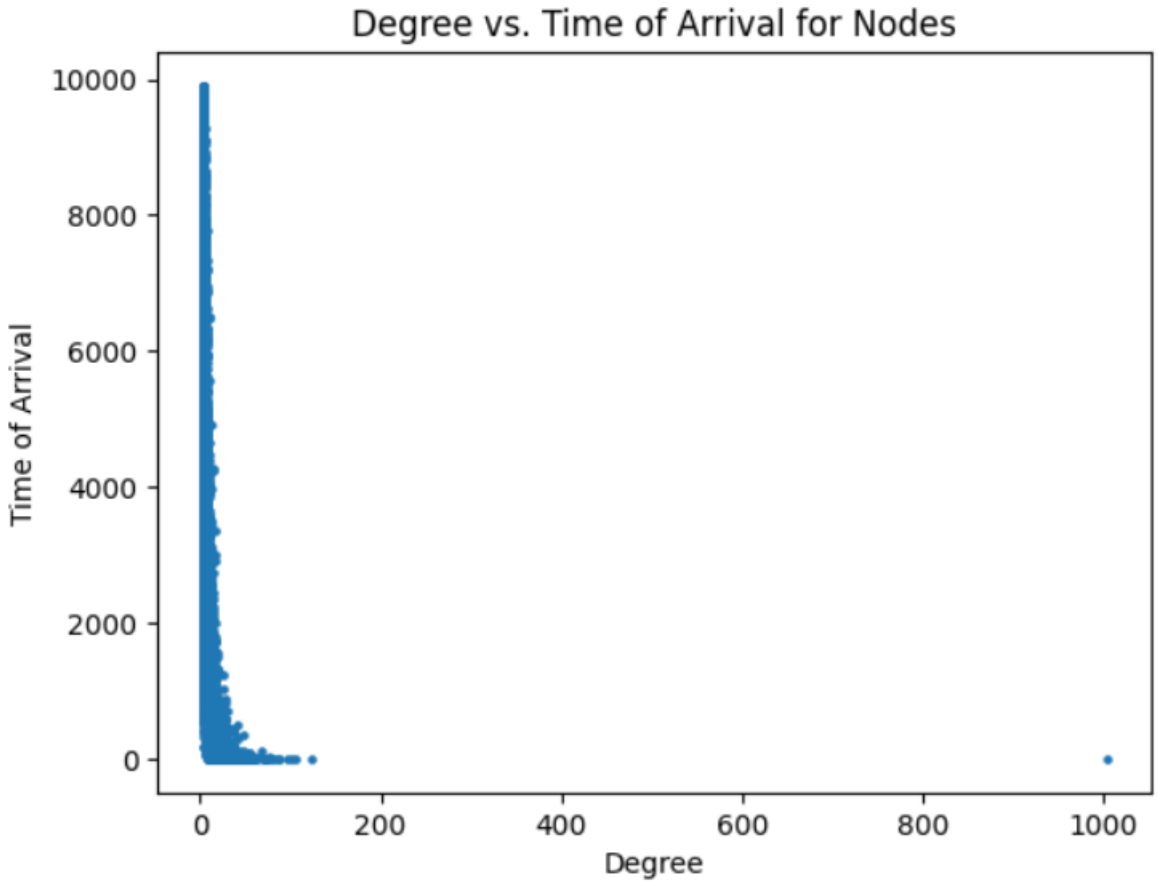
The node that has highest degree is node “0” (centre node in initialized wheel graph) which came at time t = 0 (Present in initialized network)

Problem Statement

Generate a plot where the x-axis is the node degree, and the y-axis is the time interval in which that node entered the network.

Node Degree vs Time Interval:

We generated a plot where the x-axis represents the degree of a node, and the y-axis represents the time interval at which that node entered the network.



This plot provides insights into how the degree of a node with respect to its entry time in the network.

Nodes at time t =0 (initialized nodes) have higher degree compared to the recently added ones

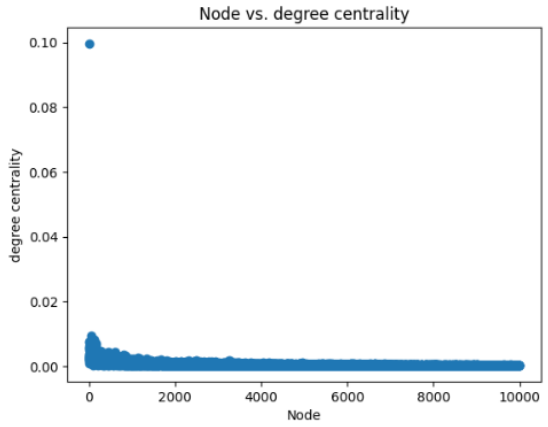
This adheres to the characteristics of the BA model, where nodes with higher degrees are more likely to attract new edges, leading to the emergence of a scale-free network.

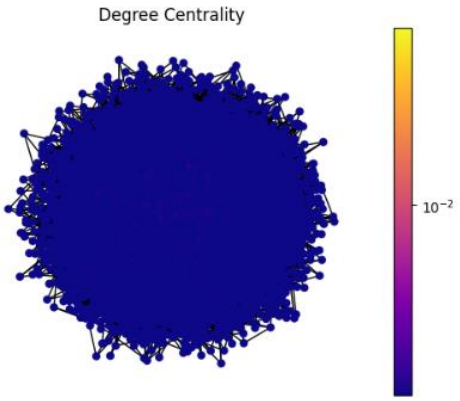
Problem Statement:

Find all the centrality measures of all the nodes in S and through proper visualization depict the values. (Note: visualization is MUST)

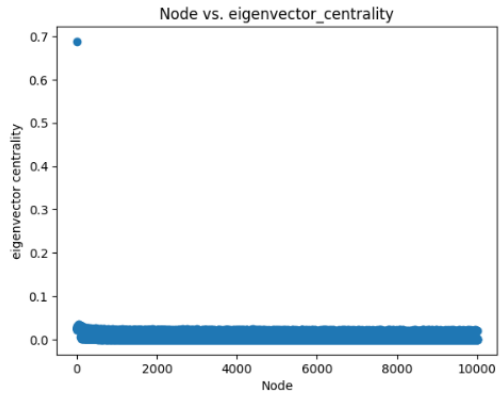
Centrality measures:

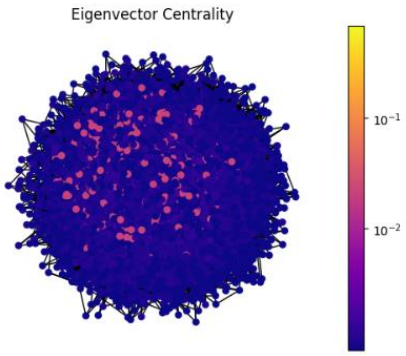
Degree Centrality:



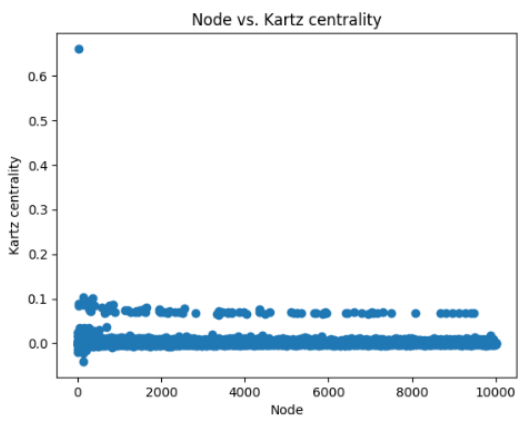


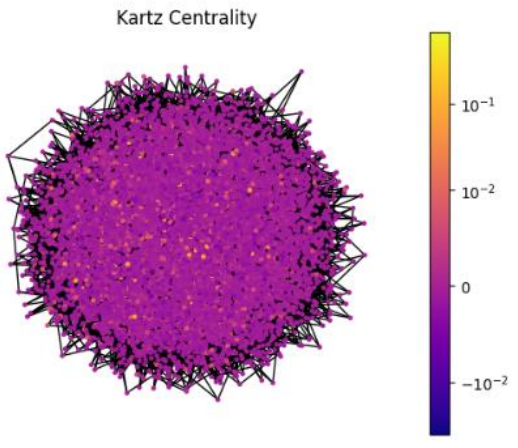
Eigenvector Centrality:



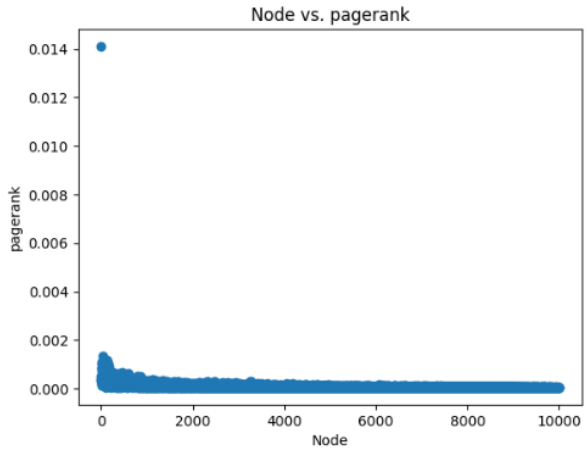


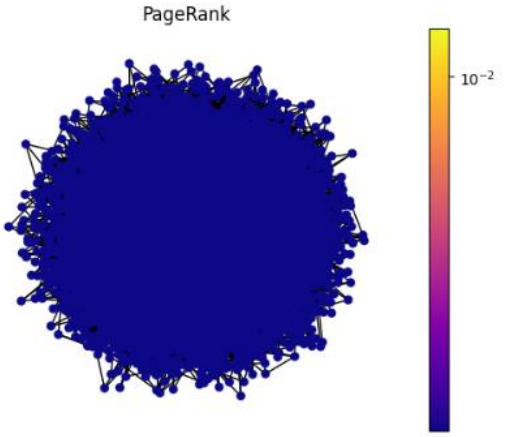
Kartz Centrality:



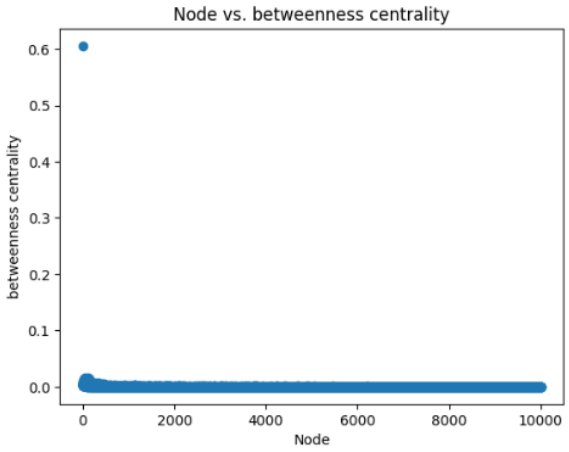


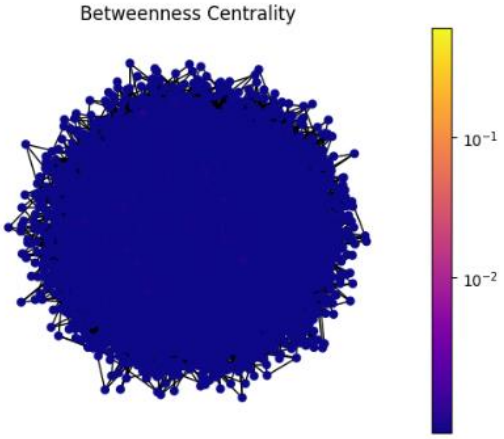
PageRank:



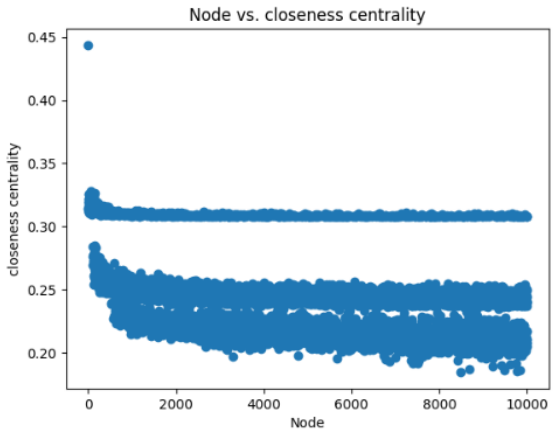


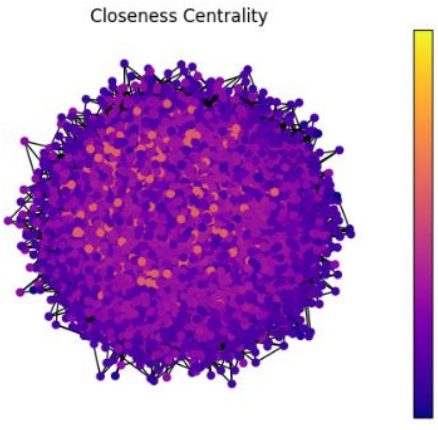
Betweenness Centrality:





Closeness Centrality:





	Nodes				
	1st Highest	**2nd Highest**	**3rd Highest**	**4th Highest**	**5th Highest**
Degree Centrality	0	134	34	100	31
Eigenvector Centrality	0	34	134	100	94
Kartz Centrality	34	144	355	301	35
PageRank	0	134	34	100	61
Betweenness Centrality	0	134	34	94	100
Closeness Centrality	0	134	94	34	61

Problem Statement:

Find the giant component in S as G and find the ratio of the nodes in G to S

Giant Component Ratio:

the **Giant Component Ratio** is the ratio of nodes in the largest connected component to the total number of nodes.

- The first step involves finding the connected components of the given network **G** using the **connected_components()** function from the NetworkX library. It returns a list of sets, where each set contains the nodes belonging to a connected component of the network
- The **max()** function is then used to find the set with the largest number of nodes, which corresponds to the largest connected component

```
# Find the giant component in S
components = nx.connected_components(G)
giant_component = max(components, key=len)

# Compute the ratio of nodes in the giant component to the total number of nodes in S
ratio = len(giant_component) / len(G)

print("Ratio of nodes in giant component to total nodes in S:", ratio)
```

```
Ratio of nodes in giant component to total nodes in S: 1.0
```

We found the Giant Component Ratio (GCR) of a network is 1, it means that the entire network is part of a single connected component, and there are no isolated nodes or disconnected components. In other words, all nodes in the network are reachable from any other node in the network

Problem Statement:

Take a random node in S and assume that it has an information I and it passes this to its neighbour with probability p. Find how many steps are required to pass this information to the maximum number of nodes in S.

a) Do the above for all the probability values p=.25, 0.5, 0.75, 1
b) For each value of p mentioned above repeat the experiment at least 10 times and take the average number of steps required.
c) What is your conclusion? Does this number of steps depend on the probability value?

Information propagation

```
# Select a random node in G and initialize its information
node = rd.choice(list(G.nodes()))
G.nodes[node]['info'] = 'I'

# Define the information propagation function
def propagate_info(G, p):
    steps = 0
    new_info = set()
    old_info = set([node])
    while len(old_info) > 0:
        for n in old_info:
            neighbors = list(G.neighbors(n))
            for neighbor in neighbors:
                if 'info' not in G.nodes[neighbor]:
                    if rd.random() < p:
                        G.nodes[neighbor]['info'] = 'I'
                        new_info.add(neighbor)
        old_info = new_info.copy()
        new_info.clear()
        steps += 1
    return steps
```

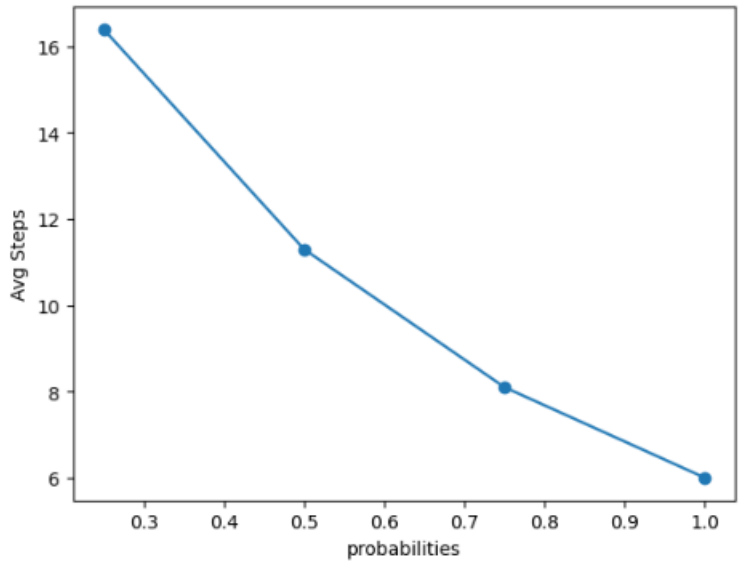
- The function **propagate_info()** that takes the network **G** and a probability value **p** as input and returns the number of **steps** required to pass the information 'I' to the maximum number of nodes in G

```
# Perform the information propagation experiment for different values of p
probabilities = [0.25, 0.5, 0.75, 1]
num_experiments = 10
avg_steps_list = []
for p in probabilities:
    total_steps = 0
    for i in range(num_experiments):
        # Reset the information on all nodes in G
        for n in G.nodes():
            if 'info' in G.nodes[n]:
                del G.nodes[n]['info']
        G.nodes[node]['info'] = 'I'
        # Run the information propagation experiment and record the number of steps
        steps = propagate_info(G, p)
        total_steps += steps
    # Compute and print the average number of steps required
    avg_steps = total_steps / num_experiments
    avg_steps_list.append(avg_steps)
    print(f"Average number of steps to propagate information with p={p}: {avg_steps}")
```

```
Average number of steps to propagate information with p=0.25: 16.4
Average number of steps to propagate information with p=0.5: 11.3
Average number of steps to propagate information with p=0.75: 8.1
Average number of steps to propagate information with p=1: 6.0
```

- Now, we can perform the information propagation experiment for different values of **p** and average the number of steps required over multiple trials

the fact that the average number of steps required decreases as p increases suggests that the network has a high degree of connectivity, and that information can spread quickly throughout the network**.**



Conclusion:

We have performed an information propagation experiment on a scale-free network generated using the BA algorithm. The results show that the network has a high level of connectivity, and a giant component that contains all the nodes in the network. The results of the information propagation experiment suggest that in scale-free networks, the probability of a node passing information to its neighbours plays a crucial role in the speed of information propagation. Therefore, selecting the appropriate probability value is critical for efficient information propagation in the network.

Reference:

https://barabasi.com/f/622.pdf

Code:

https://github.com/Rudhvi-213/Barabasi-Albert-Modelhttps://github.com/Rudhvi-213/Barabasi-Albert-Model