ES404 Assignment-1

Yashraj J Deshmukh
21110245
https://github.com/YYashraj/NCS

February 18, 2024

Solution 1:

Q. What network are you working on?

I am personally interested to work on a network dealing with the patterns of bird migrations.

Q. What are its nodes and links?

The nodes of our network will be important bird areas (across the globe), and the links will be directed edges between the nodes, depicting the movement from one area to another. The links will have two attributes, namely species of the bird and number of birds using that migratory channel.

Q. How large is it?

In our project we plan to focus on only select species of migratory birds. We may cover few avian species flying over the American continent and rest from the Asian continent. If we consider five species of birds for our network we could have around 10,000 nodes, each representing potential migratory hotspots for these species.

Q. Can it be mapped out?

Data collection is a challenging task for this project that we have taken up. Luckily, we had found a website which had data about few species' range and important bird areas. But they only lend it for non-commercial use so we need to request this data from them and hope that it suits the needs of mapping out our network.

For the project proposal we worked on a .csv file which had movement data on birds in USA. In that we were able to locate the important IBA clusters over the country. We hope we can extend this work- by adding connections between these hotspots once we find some good data for it.

Q. Why do you care about it? What are the most interesting questions that can be asked to/ answered from this network?

The inspiration for this project struck me while I was on the hostel terrace, where I witnessed a flock of birds gracefully circling the skies above our campus. This sight made me ponder over the annual migration of birds and how it might have been affected in the past years, mainly due to climate change and the impact on environment that we humans have done.

The main answer that we want to find from this network is the potential migratory path of a species over the coming years or the change in the previous path due to some ecological hindrance. We could delete a node in our network (real-world equivalent would be something like a lake drying up) and see where the flock would travel instead, or may be delete a cluster of nodes (real-world equivalent could be something like a state becoming too polluted or its weather becomes unsustainable) and observe how the migration route will change. Once we have a good understanding of how these two cases will be handled what we can do further is that we can use the climate data for these nodes over the past years to predict which of these are prone to becoming unhabitable in the upcoming years, and from that recognise the potential new hotspots that these aviators may choose. Once identified, we can rally our efforts towards preservation of the critical hotspots as well as maintaining the likely new stops in the migration path, hence making our network help a very crucial environmental problem.

Solution 2:

A is the $N \times N$ adjacency matrix of an undirected weighted network, without self-loops. We assume that the given matrix A is of the form:

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between } i^{\text{th}} \text{ and } j^{\text{th}} \text{ nodes} \\ 0 & \text{otherwise} \end{cases}$$

1 be the column vector of N elements, all equal to 1, i.e, $1 = (1, 1, 1, ..., 1)^T$.

2.1 Degree Vector k

We need to compute vector **k** whose elements are degrees k_i of all nodes $i = 1, 2, 3, \dots, N$. This can be easily calculated as value of k_i is equal to number of edges that i^{th} node has, which is nothing but the number of 1s along the i^{th} row in the adjacency matrix. Hence,

$$k = A \cdot 1$$

2.2 Total number of links L in the network

The number of edges in a network is equal to half of the sum of degrees of each node. So,

$$L = \frac{1}{2}\, k \cdot 1^T$$

2.3 The number of triangles T present in the network

We know that A^α gives a matrix where A_{ij} is the total number of distinct paths possible to reach the j^{th} node from the i^{th} node in distance α.

This means that in matrix A^3, the diagonal entries A_{ii} would store the distinct paths of length 3 in the network that begin and end at the i^{th} node, which is nothing but the number of triangles!

But one such triangle in the network will be counted more than once. For the node-i itself a triangle will get counted twice since the network is undirected. And the same triangle will be calculated thrice for all of the participating nodes.

So the actual number of unique triangles in the network is,

$$T = \frac{1}{6}\, \text{trace}(A^3)$$

2.4 Total Neighbor Degree Vector ($k_{\mathbf{nn}}$)

The i^{th} value in (k_{nn} is the sum of degrees of neighbours of the i^{th} node. These could be calculated if in all rows of A the 1s would get substituted by the degree of the column-node, then we can simply add them. So,

$$k_{\text{nn}} = A \cdot k$$

2.5 Total Second Neighbor Degree Vector ($k_{\mathbf{nn}}$)

We can calculate this vector if we derive a matrix M from A^2, which has the number of length-2 paths between nodes, by ensuring that diagonal of A^2 is zero and then reduce all non-zero entries in the matrix to 1. Then we can simply multiply the matrix M with degree vector k to get k_{nn}.

To compute this k_{nn}, we need to follow this procedure:

1. Calculate A^2.
2. Generate the degree matrix, $D = I_N \cdot k$
3. To turn all diagonal entries to zero, subtract D from A^2.
$$\hat{A}^2 = (A^2) - D$$
4. Construct a matrix M such that,
$$M_{ij} = \begin{cases} 1 & \text{if } (\hat{A}^2)_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$
5. Obtain k_{nn} by:
$$k_{\text{nn}} = M \cdot k$$

Test run on a 4-node network

Let our network be defined by the following adjacency matrix:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Then, Degree Vector,

$$\mathbf{k} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \end{pmatrix}$$

We know that the number of links in the network is 5.

Using the previous formula $\frac{1}{2}\ k \cdot 1^T$, we have $L = 0.5 \times (3 + 2 + 3 + 2) = 5$.

We know that the number of triangles in the network is 2. From the formula we get,

$$A^3 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}^3 = \begin{pmatrix} 4 & 5 & 5 & 2 \\ 5 & 2 & 5 & 2 \\ 5 & 5 & 4 & 5 \\ 5 & 2 & 5 & 2 \end{pmatrix}$$

Plugging into the formula we get, $T = \frac{1}{6}\ \mathrm{Tr}\begin{pmatrix} 4 & 5 & 5 & 2 \\ 5 & 2 & 5 & 2 \\ 5 & 5 & 4 & 5 \\ 5 & 2 & 5 & 2 \end{pmatrix} = \frac{1}{6}\ (4+2+4+2) = 2$

Neighbour degree vector, $k_{\mathrm{nn}} = A \cdot k = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \\ 7 \\ 6 \end{pmatrix}$

For the degree vector of second neighbors:

$$A^2 = \begin{pmatrix} 3 & 1 & 2 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 1 & 2 \end{pmatrix}$$

$$\hat{A}^2 = A^2 - D = \begin{pmatrix} 3 & 1 & 2 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 1 & 2 \end{pmatrix} - \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

$$=> M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$k_{\mathrm{nn}} = M \cdot k = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 7 \\ 8 \\ 7 \\ 8 \end{pmatrix}$$

Solution 3:

3.1 Network Generation

We generate three (random) networks with 500 nodes using the *erdos_renyi_graph()* method from the NetworkX library.

```
n = 500

# Subcritical network with p < 1 / (n - 1)
p_subcritical = 0.75 / (n - 1)
subcritical_network = nx.erdos_renyi_graph(n, p_subcritical)

# Critical network with p = 1 / (n - 1)
p_critical = 1 / (n - 1)
critical_network = nx.erdos_renyi_graph(n, p_critical)

# Supercritical network with p > 1 / (n - 1)
p_supercritical = 3 / (n - 1)
supercritical_network = nx.erdos_renyi_graph(n, p_supercritical)
```

I also created a connected regime just to observe and compare few properties of this network with the rest.

```
# connected regime
p_cr = 10 / (n - 1)
cr_network = nx.erdos_renyi_graph(n, p_cr)
```

I also computed the average clustering coefficient (ACC) for these networks for some initial observation about the properties of network.

```
Average clustering coefficient of subcritical network: 0.0
Average clustering coefficient of critical network: 0.0033333333333333333
Average clustering coefficient of supercritical network: 0.009295238095238096
Average clustering coefficient of connected regime network: 0.022407427608511207
```

Theoretically, the ACC for ER-network is approximately equal to the edge probability (p) of the network. The values that we observed aren't equal to that, but these values change every time we re-run our code, generating different networks each time. Even though the values differ, in most of the cases the ACC increased as we increased p for the network.

3.2 Network Visualization

For further look into the characteristics of the network, I decided to take a look at degrees of the nodes and the number of edges across the networks.

The degrees for different networks ranged between these values:

```
Degree range of subcritical network: 0 - 5
Degree range of critical network: 0 - 4
Degree range of supercritical network: 0 - 8
Degree range of connected regime network: 2 - 21
```

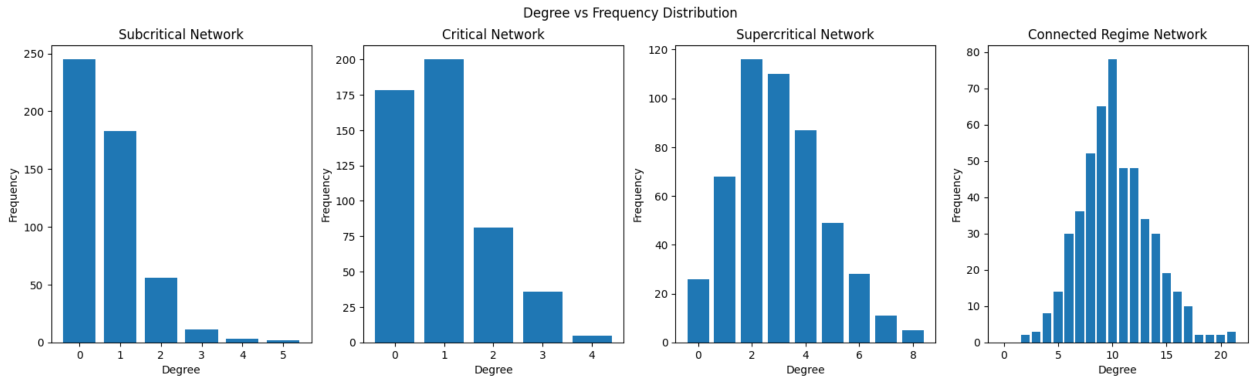


Figure 1: Bar plot for degree and frequency for each network.

Theoretically, we know that probability $p = \langle k \rangle/(N-1)$, where $\langle k \rangle$ is the average degree and N is the number of nodes in the network.
From the above graphs too it is clear that even the observed average degrees are nearly equal to the theoretical ones. This would mean the numerator that we plugged in the probability should be equal to the network's average degree.

The exact average degree values for these networks are:

```
Mean degree of subcritical network: 0.7
Mean degree of critical network: 0.98
Mean degree of supercritical network: 3.016
Mean degree of connected regime network: 10.3
```

And the number of edges in the three networks of concern is: (175, 245, 754)

Now let's move on to visualising these three networks.
The nodes in the visualisation have distinct appearance as per their degree. The node sizes are proportional to their degrees. It follows a function of the form: $y = 30x + 20$ where x is the degree. The degree-colour relation can be inferred from the legend in the plots below.

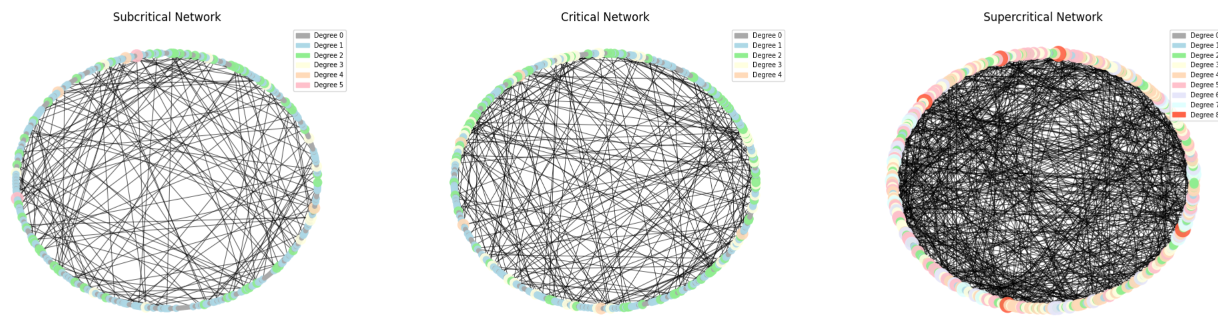


Figure 2: Network Visualisation using *circular_layout*

As the density of the circular plot grows it implies much stronger inter-connectivity between the nodes of the network, meaning that there is a greater number of edges in the network and the average degree too is more.

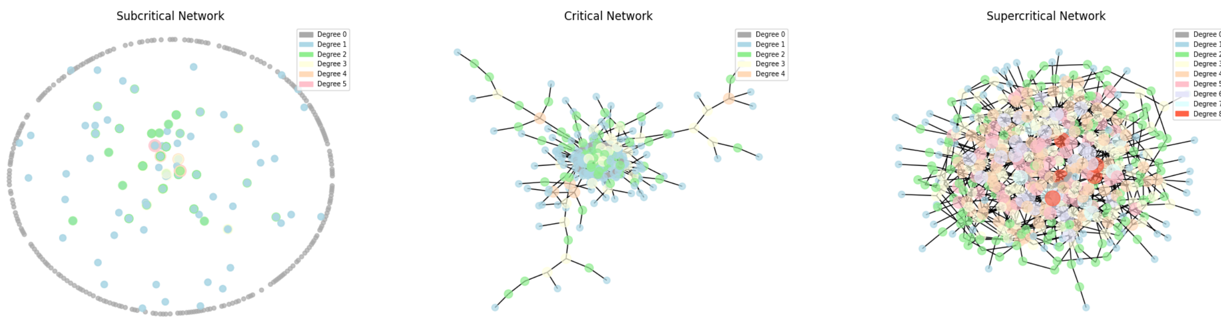


Figure 3: Network Visualisation using *kamada_kawai_layout*

From these plots few things are made clear. In subcritical network, most of the nodes don't have any connection (edge), and even the connected nodes are sparse and scattered. Critical network is better in this sense; the number of zero-degree nodes has decreased and with the increase in number of expected edges in the network, majority of the nodes have atleast one connection through them. This would also result in a few of them showing some clustering amongst them. Supercritical network has very less number of zero degree nodes as compared to N. Since most of the nodes have degree greater than one, this network shows better clustering amongst its nodes.

3.3 Degree Distribution Plotting

Plotting the degree distributions of the networks on linear and log-log scales yield the following graphs:

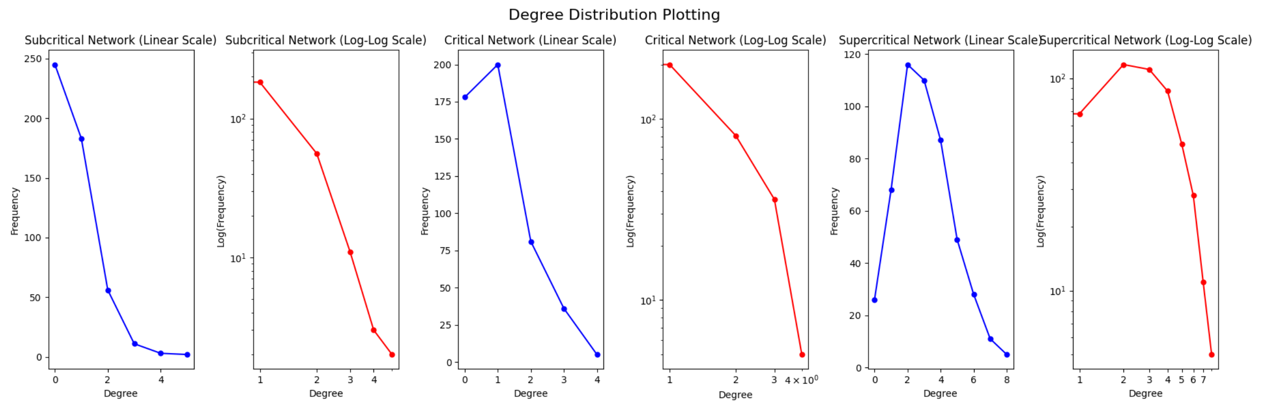


Figure 4: Degree Distribution Plotting

In the above plots, the graph touches its peak near the mean degree. It reaches this point somewhat in as an increasing polynomial function and after mean it drops in some what exponentially decreasing fashion. This hints that the degree distribution follows the Poisson distribution, which we delve further into in the next section.

3.4 Statistical Analysis

We are required to conduct a p-value test to determine if the degree distribution follows a Poisson process.

Our null-hypothesis, from the observations of degree distribution, is H_0 : The network follows a Poisson process. This also means that our alternative hypothesis is to reject this null hypothesis.

For our experimentation, we will set the significance level, α, to be 0.005 (standard value). The p-value would be the output of a chi-square goodness-of-fit test when we assume that our null hypothesis is true.

The results are as follows:

```
For Subcritical Network:
Chi-square statistic: 9.5891155206948
p-value: 0.08775063209371169
Fail to reject the null hypothesis. The degree distribution could follow a Poisson distribution.

For Critical Network:
Chi-square statistic: 4.9467851345504235
p-value: 0.2928013852545619
Fail to reject the null hypothesis. The degree distribution could follow a Poisson distribution.

For Supercritical Network:
Chi-square statistic: 1.316334281052223
p-value: 0.9953541286990043
Fail to reject the null hypothesis. The degree distribution could follow a Poisson distribution.
```

This result is because p-value from each tests were above significance level. Moreover, we observe that the network behaves more strongly like Poisson as we increase probability.

Even though we have gotten a good score to say that our network follows Poisson process, but by definition p-test is usually used to reject null hypothesis when the score drops below α; we can, by principle, only say that our null hypothesis was not rejected.

To prove further that our network is indeed following Poisson we can conduct multiple p-test for different hypothesis with the null hypothesis being the network follows that particular distribution. The results for this were as follows:

```
For Subcritical Network:
Poisson distribution p-value: 0.08775063209371169
Power-law distribution p-value: 0.3146705092820228
Exponential distribution p-value: 0.0025481591308446428
Normal distribution p-value: 0.00406677950333255

For Critical Network:
Poisson distribution p-value: 0.2928013852545619
Power-law distribution p-value: 0.25622062647613114
Exponential distribution p-value: 7.453837131807866e-12
Normal distribution p-value: 0.0007197503101610084

For Supercritical Network:
Poisson distribution p-value: 0.9953541286990043
Power-law distribution p-value: 0.0060118806212276655
Exponential distribution p-value: 3.397987016429321e-19
Normal distribution p-value: 1.9257289887562198e-08

For Cr Network:
Poisson distribution p-value: 0.41940337265893124
Power-law distribution p-value: 1.921750434847249e-17
Exponential distribution p-value: 1.5204995214924699e-05
Normal distribution p-value: 0.00012415422875551276
```

This clearly shows that only Poisson qualifies the p-test and hence is what our random networks follow. Note that we also notice Power-law distribution passing the p-test, even though with a smaller value. This is because for lower average degree values the graphs for poisson would look similar to a powerlaw, minus the heavy tail.

3.5 Largest Connected Cluster Analysis

P_o would always be 1 since every node is partaking in the network. So the value that we are calculating is basically just the inverse of the fraction of nodes in the largest connected cluster.

```python
def compute_Po_P_infinity(G):
    # Compute the fraction of nodes in the network (Po)
    Po = len(G.nodes()) / n
    # which will always be equal to 1 since all nodes will be part of the network

    # Compute the fraction of nodes in the largest connected cluster (P_infinity)
    largest_cc = max(nx.connected_components(G), key=len)
    P_infinity = len(largest_cc) / len(G.nodes())

    # Compute Po/P_infinity
    Po_Pinfinity = Po / P_infinity
    return Po_Pinfinity
```

The results of this function for the given networks are:

```
Po/P_infinity for Subcritical Network: 27.77777777777778
Po/P_infinity for Critical Network: 10.204081632653061
Po/P_infinity for Supercritical Network: 1.0615711252653928
Po/P_infinity for Connected Regime Network: 1.0
```

Since the computated value is reciprocal of the fraction of nodes in the largest connected cluster, a value of 1 would mean that the network is fully connected, a value close to 1 would indicate the network being strongly connected, and any further deviation would mean that the graph is not well connected.

What this implies is that subcritical and critical networks are sparsely connected while Supercritical network are widely connected (here almost all nodes are part of LCC). For connected regime the value was 1, implying that the network truly does become fully connected for $\langle k \rangle > \ln N$.

Note: *The Python notebook related to this question is provided in the GitHub repository whose link is at the start of this document. Navigate NCS ⟩ Assignment 1 ⟩ Question 3 to find the .ipynb file containing my code.*

Solution 4:

4.1 Scale-Free Network Generation

There is not a complete in-built method in NetworkX library to generate a graph given the γ for powerlaw distribution. So instead I have written the following function using networkx and powerlaw libraries to create the required networks:

```
# Function to generate power-law distributed degree sequence
def generate_power_law_sequence(num_nodes, gamma):
    degree_sequence = powerlaw.Power_Law(xmin=1, parameters=[gamma]).generate_
    random(num_nodes)
    degree_sequence = np.round(degree_sequence).astype(int)
    # Adjust the number of nodes to make the sum of degrees even
    if sum(degree_sequence) % 2 != 0:
        degree_sequence[np.random.randint(0, num_nodes)] += 1
    return degree_sequence

# Function to create a network with a given degree sequence
def create_network_from_degree_sequence(degree_sequence):
    G = nx.configuration_model(degree_sequence)
    G = nx.Graph(G)  # Remove parallel edges and self-loops
    return G
```

Using these functions we can generate the following networks:

```
num_nodes = 500        # Number of nodes for each network

# Anomalous Regime Network (γ < 2)
gamma_anomalous = 1.5
degree_sequence_anomalous = generate_power_law_sequence(num_nodes, gamma_anomalous)
anomalous_network = create_network_from_degree_sequence(degree_sequence_anomalous)

# Scale-Free Regime Network (2 <= γ <= 3)
gamma_scale_free = 2.5
degree_sequence_scale_free = generate_power_law_sequence(num_nodes, gamma_scale_free)
scale_free_network = create_network_from_degree_sequence(degree_sequence_scale_free)
```

```
# Random Regime Network (γ > 3)
gamma_random = 4
degree_sequence_random = generate_power_law_sequence(num_nodes, gamma_random)
random_network = create_network_from_degree_sequence(degree_sequence_random)
```

4.2 Network Visualization

Let's take a glance at how these networks look:

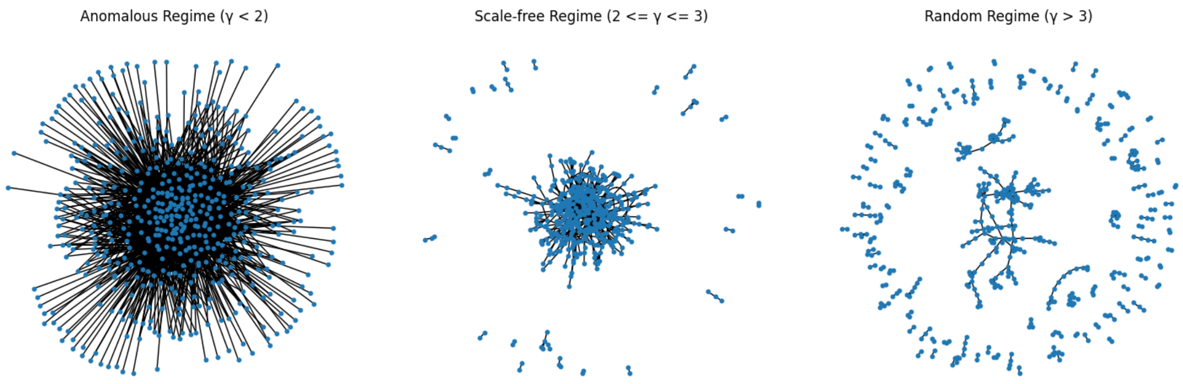
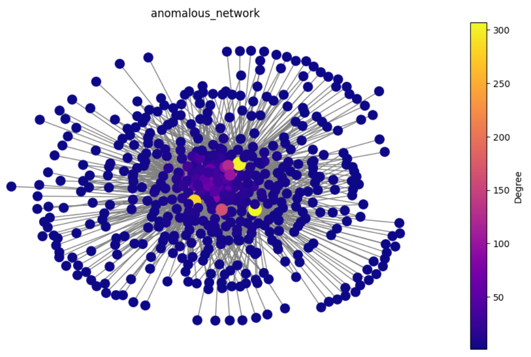


Figure 5: The networks generated from the above defined function

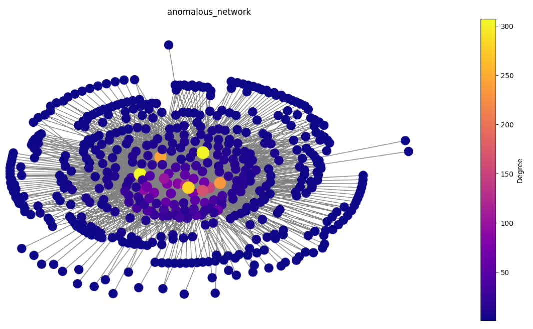
```
Degree range of the Anomalous Regime Network: 1-307
Degree range of the Scale-Free Regime Network: 1-41
Degree range of the Random Regime Network: 1-15

Edges in the networks : (2842, 623, 381)
```

Clearly the anamolous regime has highest degree and number of connections. Scale-free and Random regime have way less connections and density than it.

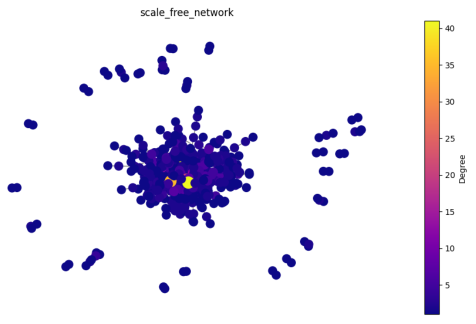


(a) spring layout

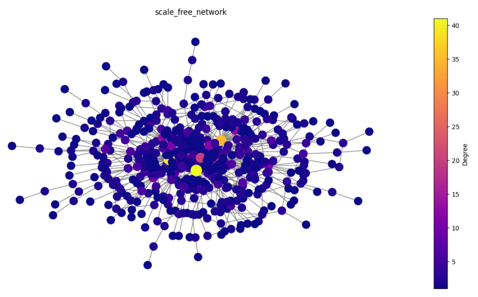


(b) kamada kawaii layout

Figure 6: Anomalous Regime

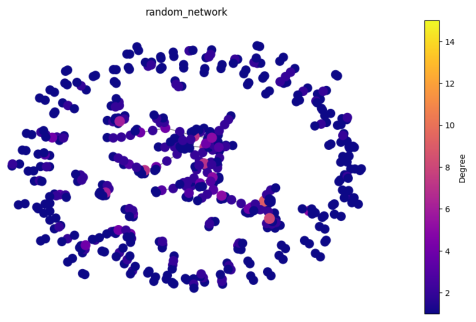


(a) spring layout

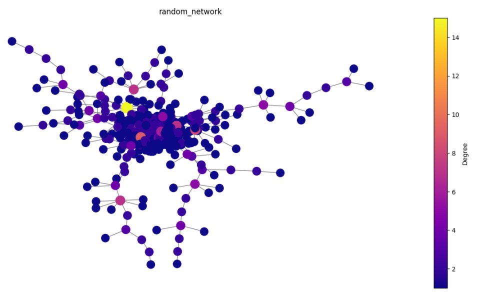


(b) kamada kawaii layout

Figure 7: Scale-Free Regime



(a) spring layout



(b) kamada kawaii layout

Figure 8: Random Regime

4.3 Degree Distribution Plotting

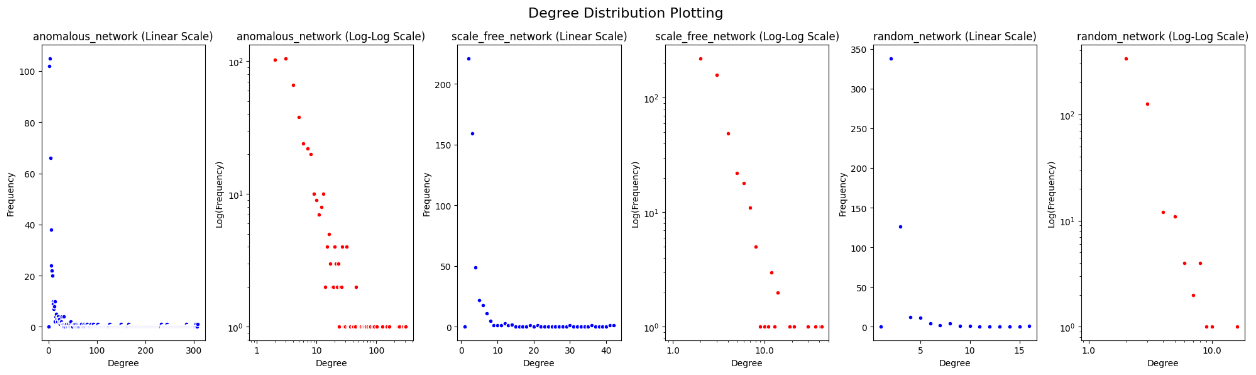


Figure 9: Degree Distribution on Linear and Log-Log scales

Anomalous and Scale-free regimes have a heavy tail in their distribution meaning that there are few nodes even at high degree values. This is not the case for the random network in which distribution dies out after crossing 10.

Between the given anomalous and scale-free networks, the anomalous regime showcase a far superior heterogenity, having a degree range almost seven times that of scale-free.

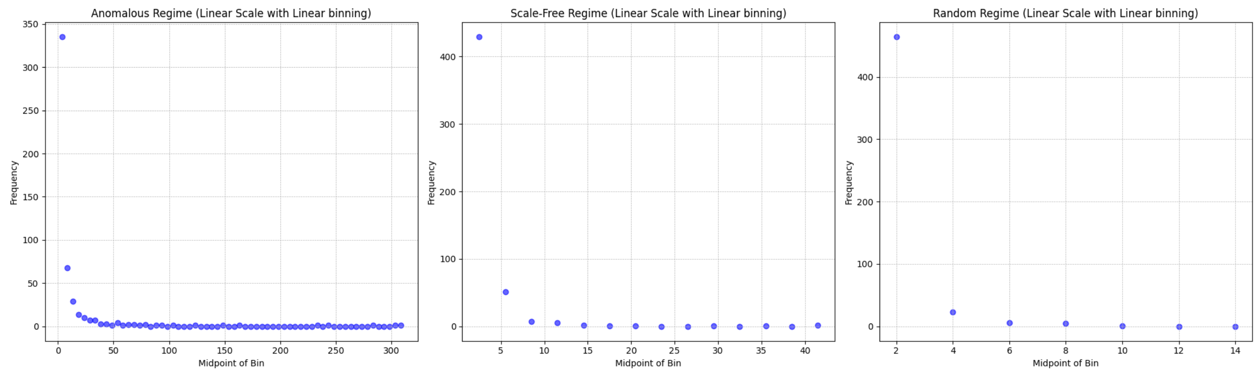


Figure 10: Linear Scale and Linear Binning

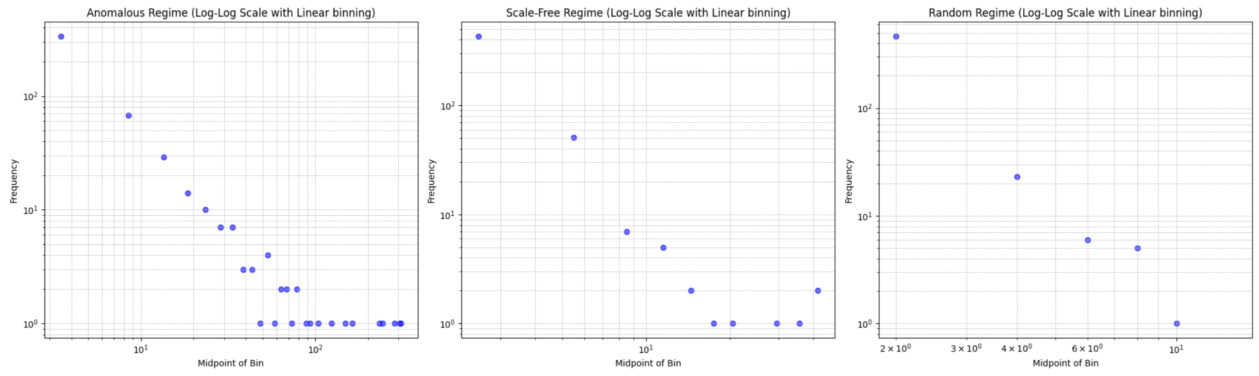


Figure 11: Linear Scale and Log-Log Binning

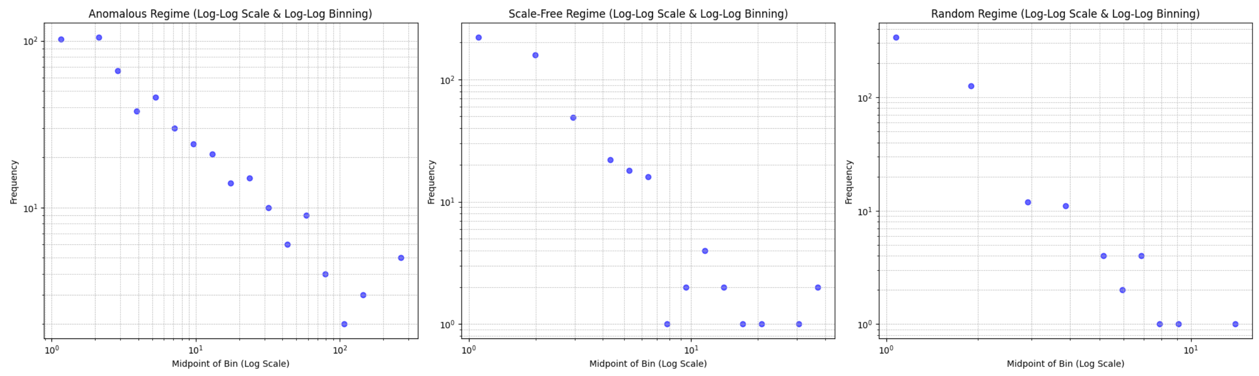


Figure 12: Log-Log Scale and Log-Log Binning

Note: *The Python notebook related to this question is provided in the GitHub repository whose link is at the start of this document. Navigate NCS ⟩ Assignment 1 ⟩ Question 4 to find the .ipynb file containing my code.*

Solution 5:

5.1 Average Degree Calculation

In the subnetwork containing only blue nodes, one blue node has the chance to form a connection with $(N-1)$ other nodes with a probability p.

$$\text{So, the average degree for blue subnetwork would be: } k_{\text{subnetwork}} = p \cdot (N-1)$$

In the full network, a node has probability p to form connection with $(N-1)$ nodes have same colour and probability q to form connection with N nodes of different colour.

$$\text{So, average degree would be } k_{\text{network}} = p \cdot (N-1) + q \cdot N$$

5.2 Minimal p and q for Network Connectivity

For the network to become fully connected, we can break our problem into two sub-parts:

1. Each color subnetwork should be fully connected (find p using this).
2. There should be at least one link between the two subnetworks (find q using this).

For p: Both the subnetworks of the given network are $G(N,p)$ type of random networks. This means that they both would follow properties of an ER-network.

We know that for a random network to be fully connected, its average degree, $\langle k \rangle$, should be greater than $\ln N$. Since probability $p = \frac{\langle k \rangle}{N}$, the minimum value of p for network connectivity will be $\ln N/N$.

For q: a node of one colour, it can form N possible connections with nodes from other colour. So in total there are a total of N^2 inter-colour connections possible in the network.

For the overall network to be fully connected, atleast one of these connections needs to be realised. So the naive value of probability for q is $1/N^2$.

Hence to ensure with high probability that the network is connected, $\mathbf{p} = \frac{\ln \mathbf{N}}{\mathbf{N}}$ and $\mathbf{q} = \frac{\mathbf{1}}{\mathbf{N^2}}$.

It is clear that on changing N both minimum values of p and q would vary. But the effect would be polar opposites for both.

On increasing N, we would also need to increase value of p to keep the subnetwork connected. The difference though is that p only increases logarithmically as compared to N. This is testament to the small-world property of random networks.

On the other hand, on increasing N, the minimum value of q drops quadratically. This implies as long as there is some non-zero probability, even if very little, there's a good chance that atleast one connection is possible amongst the two subnetworks which could make the entire network connected.

5.3 Small-World Property in Snobbish Networks

To observe the connectivity of a snobbish network, we create one with 2000 nodes, $p = 0.01$ $(> \frac{\ln N}{N})$ and $q = 0.0001$ $(\approx \frac{1}{N^2})$.

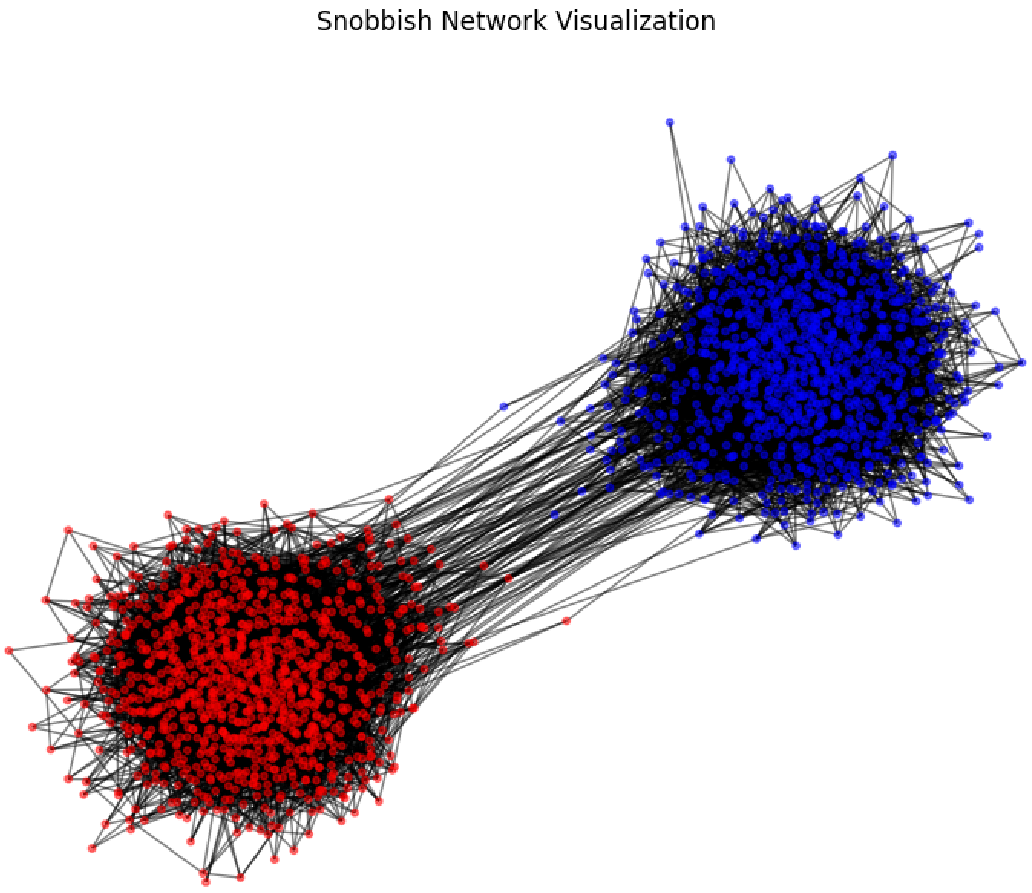


Figure 13: A snobbish network, generated with N=1000, p=0.01, q=0.0001

We observed the following results:

```
Number of intra-color edges: 9898
Number of inter-color edges: 109
Total possible intra-color edges: 999000.0
Total possible inter-color edges: 1000000
Fraction of intra-color edges: 0.009907907907907907
Fraction of inter-color edges: 0.000109

Graph with 2000 nodes and 10007 edges
Average Path Length: 4.038614807403702
Clustering Coefficient: 0.009735153586112705
```

The average path length of the network is around 4 (which is even lesser than $\ln 2000$). This shows that the small-world property is held even by snobbish networks.

And a not so surprising yet note-worthy observation is that the final fraction of both types of edges is nearly the same as the probabilities with which we had constructed this network, further proving that the minimum values found in the previous question should be sufficient to making this type of two-class network completely connected.

The clustering coefficient here is low because even if the subnetworks separately may be well-clustered, the chance of a triangular connection between the two subnetworks is very low, thus reducing the average clustering coefficient for the overall network.

This is the results for individual subnetworks:

```
Red Subnetwork:
Average Path Distance: 3.2598818818818818
Clustering Coefficient: 0.010686074168349716

Blue Subnetwork:
Average Path Distance: 3.219867867867868
Clustering Coefficient: 0.01010062434125903
```

Note: *The Python notebook related to this question is provided in the GitHub repository whose link is at the start of this document. Navigate NCS ⟩ Assignment 1 ⟩ Question 5 to find the .ipynb file containing my code.*