Assignment 2 Q1

March 22, 2021

0.1 Q.1.

Implement Watts and Strogatz's small-world network model. Compute and plot the 'scaled clustering coefficient' and 'scaled characteristic path length' of Watts and Strogatz network models with increasing value of rewiring probability. Choose the values of , suitably. Exact replication of the below-shown plot is expected.

0.1.1 Libraries Import

[1]:
```
import random
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

[2]:
```
# Set the initial values
n = 1000     # n is the number of nodes in the graph
k = 10       # k is the degree of each node in the regular graph
```

[3]:
```
"""
Function         : drawKRegular
Input Parameters : Number of nodes, n and value of the parameter, k(defined for␣
 ↪k-Regular Graph)
Purpose          : To create a k-Regular Graph
Returns          : A k-Regular Graph
"""

def drawKRegular(n, k):
    node_list = [item for item in range(0, n)]              # Defines the nodes
    G = nx.Graph()                                          # Makes the graph␣
 ↪with no nodes and edges
    G.add_nodes_from(node_list)                             # adds the nodes␣
 ↪from the node_list defined above into the graph

    for i in range(n):                                      # for each node
        for j in range(1,int(k/2)+1):                       # for connecting␣
 ↪each node to the j nearest neighbours(the edge is undirected, therefore k/2)
            G.add_edge(i, (i+j)%n)                          # adds the edge into␣
 ↪the graph
```

```
    return G                                                # returns the␣
↪k-Regular Graph
```

```
[4]: G = drawKRegular(n, k)                                 # Calls the␣
↪drawKRegular function and returns the graph
L0 = nx.average_shortest_path_length(G)                     # L0 stores the␣
↪characteristic path length of the k-Regular Graph
C0 = nx.average_clustering(G)                               # C0 stores the␣
↪average clustering coefficient of the k-Regular graph
print("Characteristic Path length:", L0)
print("Clustering Coefficient:", C0)
```

```
Characteristic Path length: 50.450450450450454
Clustering Coefficient: 0.6666666666666636
```

```
[5]: """
Function          : runSmallWorld
Input Parameters : Probability value(p), k-Regular Graph(G) and value of the␣
↪parameter(k) (defined for k-Regular Graph)
Purpose           : This function creates the graphs i.e adds the randomness␣
↪into the graph through the process of rewiring
                    on the basis of probability from being a k-Regular Graph(at␣
↪p = 0) to completely random graph(at p = 1)
                    and storing characteristic path length and clustering␣
↪coefficient
Returns           : Characteristic Path length and Clustering coefficient
"""

def runSmallWorld(p, G, k):
    for j in range(1,int(k/2)+1):                           # For each edge␣
↪(since it is undirected therefore k/2)
        node_list = list(G.nodes())                         # Fetch the list of␣
↪nodes in the graph
        for node in node_list:                              # For each node␣
↪present in the graph
            num = random.uniform(0, 1)                      # Generate a random␣
↪number between 0 and 1
            random_vertex = random.randint(0,n-1)
            if num <= p:
                if random_vertex != node and not G.has_edge(random_vertex,␣
↪node):      # To prevent multi edges
                    G.add_edge(random_vertex, node)         # Rewire the␣
↪nearest edge defined by j to the current node
                    G.remove_edge(node, (node+j)%n)         # Remove the edge␣
↪present
```

```python
    char_path_length = nx.average_shortest_path_length(G)    # Calculate the␣
↪characteristic path length
    clustering_coefficient = nx.average_clustering(G)        # Calculate the␣
↪average clustering coefficient
    return char_path_length, clustering_coefficient          # Returns the␣
↪characteristic path length and clustering coefficient
```

[6]:
```python
"""
Function         : generateProbList
Input Parameters : None
Purpose          : To create a list of probability values
Returns          : A list containing all the probability values the small world␣
↪network is to be experimented upon
"""
def generateProbList():
    prob_list = []                                           # Define an empty␣
↪probability list
    i = 4                                                    # Initialise the␣
↪counter
    while i >= 1:                                            # Run the loop for␣
↪4 times
        prob_list.extend([item/(pow(10, i)) for item in range(1, 10, 1)])    #␣
↪Add the values into the probability list
        i -= 1                                               # Decrement the␣
↪value of the counter
    prob_list.append(1.0)                                    # Add the last␣
↪value into the list
    return prob_list                                         # Returns the␣
↪probability list
```

[7]:
```python
prob_list = generateProbList()                               # Generates␣
↪Probability List on which the parameters will be computed
print("prob_list:", prob_list)
num_iter = 50                                                # Specify Number␣
↪of iterations for each Value of Probability
final_char_path_length_list = []                             # Initialize␣
↪Characteristic Path length List
final_clust_coeff_list = []                                  # Initialize␣
↪Clustering Coefficient List

for p in prob_list:                                          # For each␣
↪Probability Value
    print("\nprob:", p)
    char_path_length_list = []                               # Initialize␣
↪internal Characteristic Path length list
```

```
    clust_coeff_list = []                                          # Initialize␣
↪internal Clustering Coefficient list

    for i in range(num_iter):                                      # Draw the small␣
↪world graph for the number of iterations specified
        G = drawKRegular(n, k)                                     # Draws the K␣
↪regular graph each time

        char_path_length, clustering_coefficient = runSmallWorld(p, G, k)  #␣
↪Calls the function to rewire the network
                                                                           #␣
↪and returns the characteristic path length
                                                                           #␣
↪and clustering coefficient of that graph

        char_path_length_list.append(char_path_length)                     #␣
↪Appends the Characteristic Path length
                                                                           # to␣
↪internal Characteristic Path length list
        clust_coeff_list.append(clustering_coefficient)                    #␣
↪Appends the Clustering Coefficient
                                                                           # to␣
↪internal Clustering Coefficient list

    final_char_path_length_list.append(np.mean(char_path_length_list)/L0)  #␣
↪Find the mean of the values and normalize by
                                                                           #␣
↪char path length of K Regular graph and append to final values
    final_clust_coeff_list.append(np.mean(clust_coeff_list)/C0)            #␣
↪Find the mean of the values and normalize by
                                                                           #␣
↪clustering coefficient of K Regular graph and append to final values
print("\nFinal Characteristic Path length:", final_char_path_length_list)
print("Final Clustering Coefficient:", final_clust_coeff_list)
```

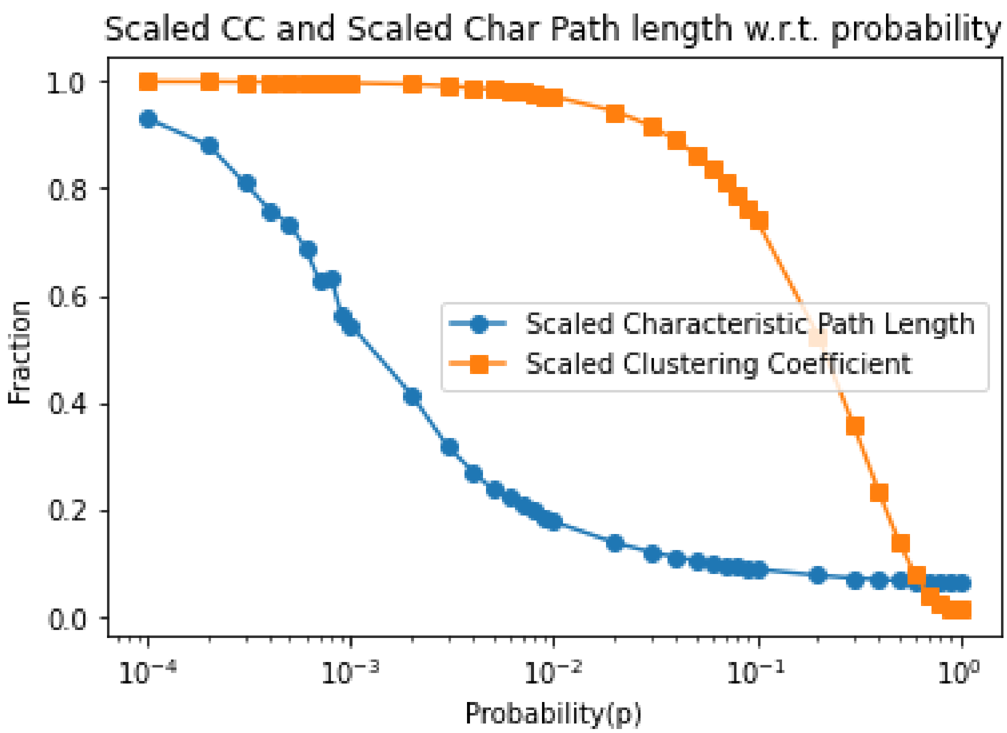
```
prob_list: [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008,
0.0009, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01,
0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
0.7, 0.8, 0.9, 1.0]

prob: 0.0001

prob: 0.0002

prob: 0.0003

prob: 0.0004
```

```
prob: 0.0005

prob: 0.0006

prob: 0.0007

prob: 0.0008

prob: 0.0009

prob: 0.001

prob: 0.002

prob: 0.003

prob: 0.004

prob: 0.005

prob: 0.006

prob: 0.007

prob: 0.008

prob: 0.009

prob: 0.01

prob: 0.02

prob: 0.03

prob: 0.04

prob: 0.05

prob: 0.06

prob: 0.07

prob: 0.08

prob: 0.09

prob: 0.1
```

```
prob: 0.2

prob: 0.3

prob: 0.4

prob: 0.5

prob: 0.6

prob: 0.7

prob: 0.8

prob: 0.9

prob: 1.0

Final Characteristic Path length: [0.9305347039682537, 0.8805178674603176,
0.8105782698412698, 0.7592996126984127, 0.7333103452380952, 0.6882033111111111,
0.6272326825396826, 0.6347829753968253, 0.5632785365079365, 0.5448920603174603,
0.413661119047619, 0.3177784666666666, 0.27082538333333334, 0.2369802325396826,
0.2234486428571428, 0.2098205412698413, 0.19957643888888885,
0.18558690158730157, 0.1769644484126984, 0.1378173753968254,
0.12044724047619049, 0.11068726904761904, 0.10427156666666666,
0.09954065000000001, 0.09572622301587301, 0.09270640634920634,
0.09014444523809524, 0.08823729999999999, 0.07675857539682539,
0.07164857857142858, 0.06886557142857141, 0.06710775952380953,
0.06597228253968254, 0.0652670126984127, 0.06492704523809523,
0.06480352619047618, 0.06478223650793652]
Final Clustering Coefficient: [0.9997055606060604, 0.9995067077922077,
0.9991404545454544, 0.9988087878787878, 0.9986409090909091, 0.9984366818181816,
0.9977996515151514, 0.998024831168831, 0.9971515606060605, 0.9968974242424239,
0.9947245497835494, 0.9914755108225105, 0.9882366336996331, 0.9849353549783542,
0.9828081645021638, 0.9803957748917743, 0.9783599843489835, 0.9738771601731594,
0.9714635254745244, 0.9445068664668667, 0.9172680144855159, 0.8899380705960729,
0.8639565266400298, 0.8387195397935439, 0.8127734433899473, 0.7885628020313059,
0.7615240645188185, 0.7406558981019019, 0.5252753833127682, 0.3574612089087402,
0.2322977584113528, 0.140296136776022, 0.07842321668991566,
0.040475245569704096, 0.021831668037329156, 0.015202065401574769,
0.013732367162507317]
```

[12]:
```
# Plots the semi-log graph of Scaled characteristic path length and Scaled␣
↪clustering coefficient
```

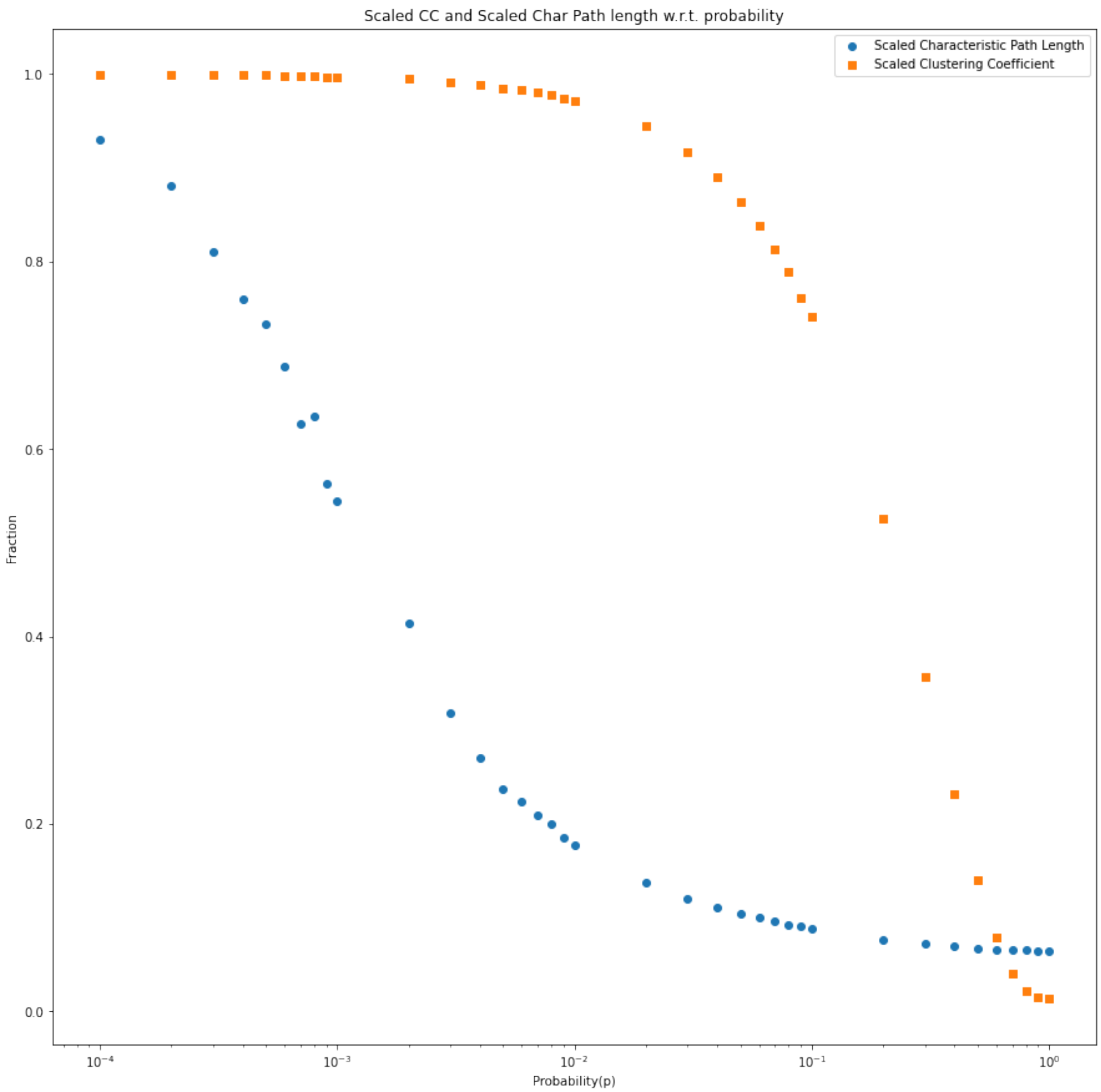
```
plt.semilogx(prob_list, final_char_path_length_list, marker='o', label='Scaled␣
↪Characteristic Path Length') # Semi Log plot of characteristic path length
plt.semilogx(prob_list, final_clust_coeff_list, marker = 's', label='Scaled␣
↪Clustering Coefficient')        # Semi log plot of clustering coefficient
plt.title("Scaled CC and Scaled Char Path length w.r.t. probability")
plt.xlabel("Probability(p)")                        # X-axis is labelled as␣
↪probability
plt.ylabel("Fraction")                              # Y-axis is labelled as the␣
↪Fraction
plt.legend()                                        # Shows the legend of the␣
↪plot
plt.show()                                          # Shows the plot
```



[13]:

```
# Plots the semi-log graph of Scaled characteristic path length and Scaled␣
↪clustering coefficient
# in a better manner with an enlarged figure size and without the line␣
↪connecting the dots

fig = plt.figure(figsize = (15,15))                 # Sets the fig size of the␣
↪plot
plt.rc('axes.formatter', useoffset=False)
```

```python
plt.scatter(prob_list, final_char_path_length_list, marker='o', label='Scaled Characteristic Path Length')  # Plots the scatter plot for Scaled Characteristic Path length
plt.scatter(prob_list, final_clust_coeff_list, marker = 's', label='Scaled Clustering Coefficient')         # Plots the scatter plot for Scaled Clustering Coefficient

ax=plt.gca()
ax.set_xscale('log')                                   # Sets the scale of the X-axis as the log scale
plt.title("Scaled CC and Scaled Char Path length w.r.t. probability")
plt.xlabel("Probability(p)")                           # X-axis is labelled as probability
plt.ylabel("Fraction")                                 # Y-axis is labelled as the Fraction
plt.legend()
plt.show()                                             # Shows the plot
```



Scaled CC and Scaled Char Path length w.r.t. probability
Scaled Characteristic Path Length
Scaled Clustering Coefficient
Fraction
Probability(p)
1.0
0.8
0.6
0.4
0.2
0.0
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}


```
[ ]:
```