GUENIN–CARLUT Avel - TD3 Complex Networks

November 28, 2018

1 Complex Networks - TD 3

1.1 Avel GUÉNIN--CARLUT

1.2 04/12/2018

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

1.3 I - The Watts-Strogatz random graph model

We will first build a Watts-Strogatz model from scratch, then demonstrate its scale free and small world properties in two separate parts.

1.3.1 I.1 - Graph construction

```
In [2]: def build_WattsStrogatz(N,half_k,beta):

            ## implementing 2*half_k-uniform ring graph
            G = nx.Graph()
            for i in range(N):
                G.add_node(i)
            for i in range(N):
                for j in range(half_k):
                    G.add_edge(i,(i+j+1)%N)


            ## picking edges for rewiring with probability beta
            rewires = []
            for edge in G.edges():
                if rd.random() < beta:
                    rewires.append(edge)
```

```
            ## adding new links
            for edge in rewires:

                ## picking origin node
                if (edge[1]-edge[0]) < half_k + 1 :
                    ## if edge[1] is half_k or less after edge[0] it is edge[0] where the edge
                    origin_node = edge[0]
                elif (edge[0]-edge[1]+N) < half_k + 1 :
                    ## otherwise, edge must have been originated from edge[1] and edge[1]-N is
                    origin_node = edge[1]

                ## link origin node to new neighbor
                candidate_nodes = [n for n in G.node() if (not(i==n) and not(n in G.neighbors(i
                G.add_edge(origin_node,rd.choice(candidate_nodes))


            ## removing old links
            for edge in rewires:
                G.remove_edge(edge[0],edge[1])


            return(G)
```

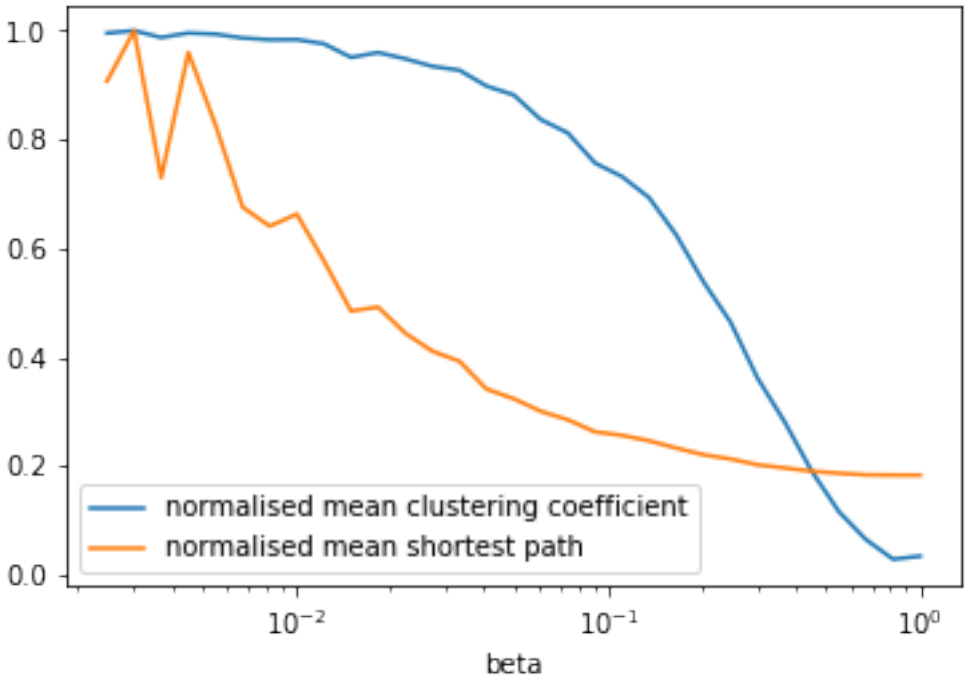
1.3.2 I.2 - Clustering propertiy

```
In [8]: beta_range = [np.exp(-x/5) for x in range(0,31)]
        N = 300
        half_k = 3

        I = 5

        clustering_plot = []
        mean_distance_plot = []

        ## clustering and diameter are computed for different values of beta and averaged over
        for beta in beta_range:
            print("En cours : beta = " + str(beta))
            clustering = []
            mean_distance = []
            for i in range(I):
                G = build_WattsStrogatz(N,half_k,beta)
                clustering.append(nx.average_clustering(G))
                mean_distance.append(nx.average_shortest_path_length(G))
            clustering_plot.append(np.mean(clustering))
            mean_distance_plot.append(np.mean(mean_distance))
```

```
En cours : beta = 1.0
En cours : beta = 0.818730753078
```

```
En cours : beta = 0.670320046036
En cours : beta = 0.548811636094
En cours : beta = 0.449328964117
En cours : beta = 0.367879441171
En cours : beta = 0.301194211912
En cours : beta = 0.246596963942
En cours : beta = 0.201896517995
En cours : beta = 0.165298888222
En cours : beta = 0.135335283237
En cours : beta = 0.110803158362
En cours : beta = 0.0907179532894
En cours : beta = 0.0742735782143
En cours : beta = 0.0608100626252
En cours : beta = 0.0497870683679
En cours : beta = 0.0407622039784
En cours : beta = 0.0333732699603
En cours : beta = 0.0273237224473
En cours : beta = 0.0223707718562
En cours : beta = 0.0183156388887
En cours : beta = 0.0149955768205
En cours : beta = 0.0122773399031
En cours : beta = 0.0100518357446
En cours : beta = 0.00822974704902
En cours : beta = 0.00673794699909
En cours : beta = 0.00551656442076
En cours : beta = 0.00451658094261
En cours : beta = 0.00369786371648
En cours : beta = 0.00302755474538
En cours : beta = 0.00247875217667
```

```
In [9]: plt.semilogx(beta_range,clustering_plot/max(clustering_plot),label="normalised mean clu
        plt.semilogx(beta_range,mean_distance_plot/max(mean_distance_plot),label="normalised me

        plt.xlabel("beta")
        plt.legend()
        plt.show()
```



As you can see above, both clustering coefficient and mean distance drop as beta grows.

However, as mean distance drops faster than clustering coefficient, there is a beta range where clustering remains high (comparably to the ring network limit) while mean distance is low (comparably to the ER limit). With our parameters, it is 0.03 to 0.3.

Hence, WS networks account for both clustering and short mean distances for specific beta ranges.

1.3.3 I.3 - Small world property

```
In [6]: beta_range = [np.exp(-x/2) for x in range(0,7)]
        N_range = [int(np.floor(np.exp(x/3))) for x in range (10,22)]
        half_k = 3

        I = 3

        clustering_plots = dict()
        mean_distance_plots = dict()
        for beta in beta_range:
            print("En cours : beta = " + str(beta))
            clustering_plots[beta] = []
            mean_distance_plots[beta] = []
            for N in N_range:
                print("N = " + str(N))
                clustering = []
                mean_distance =  []
```

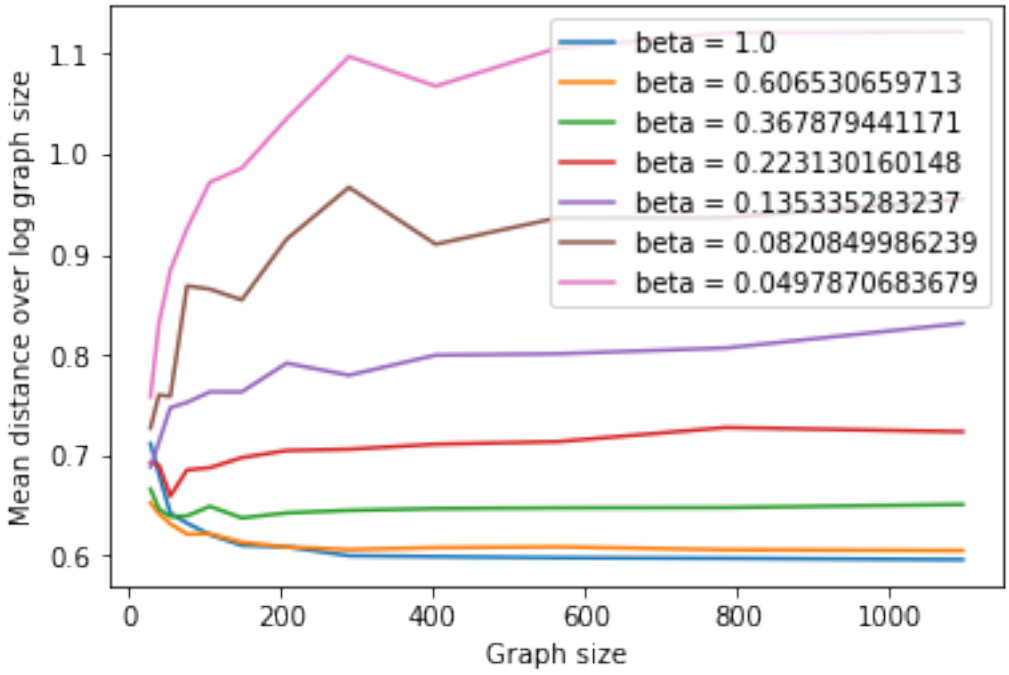
```
                for i in range(I):
                    G = build_WattsStrogatz(N,half_k,beta)
                    clustering.append(nx.average_clustering(G))
                    mean_distance.append(nx.average_shortest_path_length(G))
                clustering_plots[beta].append(np.mean(clustering))
                mean_distance_plots[beta].append(np.mean(mean_distance))
```

```
En cours : beta = 1.0
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : beta = 0.606530659713
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : beta = 0.367879441171
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : beta = 0.223130160148
N = 28
```

```
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : beta = 0.135335283237
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : beta = 0.0820849986239
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : beta = 0.0497870683679
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
```

```
N = 785
N = 1096
```

```
In [7]: for beta in beta_range:
            plot = [mean_distance_plots[beta][i]/np.log(N_range[i]) for i in range(len(N_range)
            plt.plot(N_range,plot,label = "beta = " + str(beta))

        plt.legend()
        plt.ylabel("Mean distance over log graph size")
        plt.xlabel("Graph size")
        plt.show()
```



We can observe above that, for all tested beta (superior to 0), mean distance growth is capped by log graph size. Lower beta lead to higher constants, and slower convergence to the asymptote. The result seem however to be robust.

Hence, WS networks do not only show lower mean distances as compared to the pure ring network limit, they do show small world property. They notably do so in the range beta < 0.2 where the clustering coefficient is high.

1.4 II - The Barabási-Albert Model

We will first build a Barabàsi-Albert model from scratch, then demonstrate its scale free and small world properties in two separate parts.

1.4.1 II.1 - Graph construction

```
In [87]: def build_BarabasiAlbert(N,m):

             ## build connected starting graph
             G = nx.Graph()
             for k in range(m):
                 G.add_node(k)
             while not(nx.is_connected(G)):
                 edge = rd.sample(G.nodes(),2)
                 G.add_edge(edge[0],edge[1])



             ## add nodes with preferential attachment
             for i in range(m,N):
                 ## define probability distribution for new links
                 relative_degree = [G.degree(j) for j in G.nodes()]
                 relative_degree = [k/sum(relative_degree) for k in relative_degree]

                 ## choose neighbours of incoming nodes
                 link_choice = np.random.choice(G.nodes, size=m, replace=False, p=relative_degr

                 ## add node
                 G.add_node(i)
                 for v in link_choice:
                     G.add_edge(v,i)

             return(G)
```

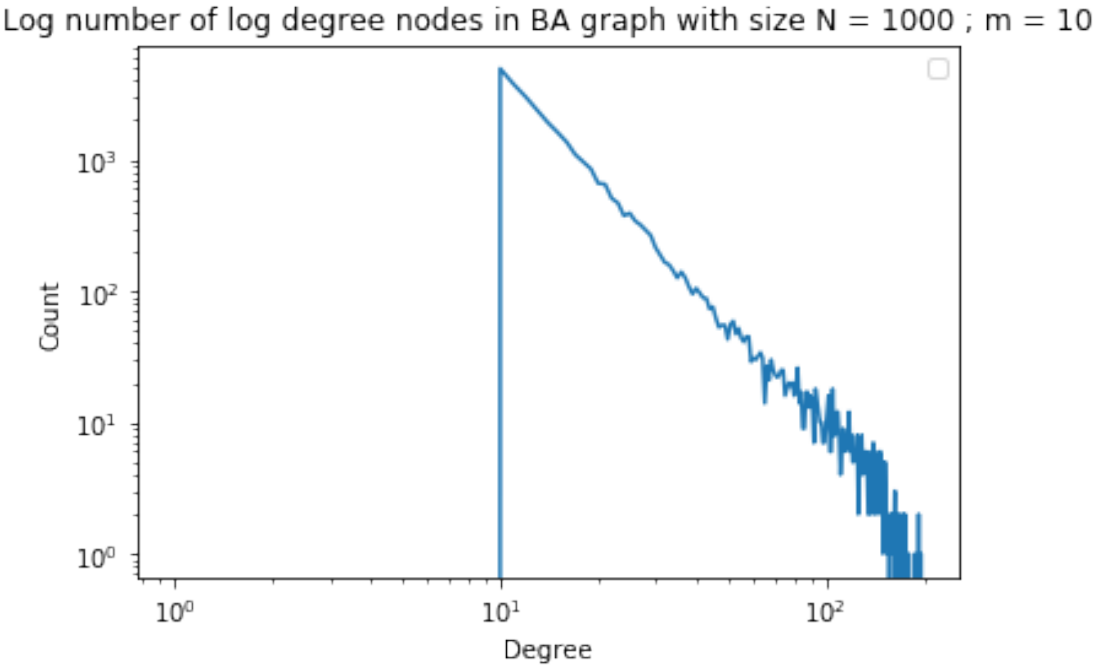
1.4.2 II.2 - Scale-free property

```
In [100]: N = 1000
          m = 10
          I = 30

          degree_sequence = []
          for i in range(I):
              G = build_BarabasiAlbert(N,m)
              degree_sequence += [d for n, d in G.degree()]
```

```
In [116]: k_range = range(max(degree_sequence) + 2)
          plot = []
          for k in k_range:
              plot.append(degree_sequence.count(k))

          plt.loglog(k_range,plot)
          plt.xlabel("Degree")
          plt.ylabel("Count")
```

```
        plt.title("Log number of log degree nodes in BA graph with size N = " + str(N) + " ;
        plt.legend()
        plt.show()
```

```
No handles with labels found to put in legend.
```



We can notice that the degree-rank histogramm draws a line in log log scale (even though it is irregular for high degree values), showing that it follows a power law asymptotically, and therefore that the BA network is scale-free.

While not shown here, this broad result does not change for varying graph size (N) and entry degree (m).

1.4.3 II.3 - Small world property

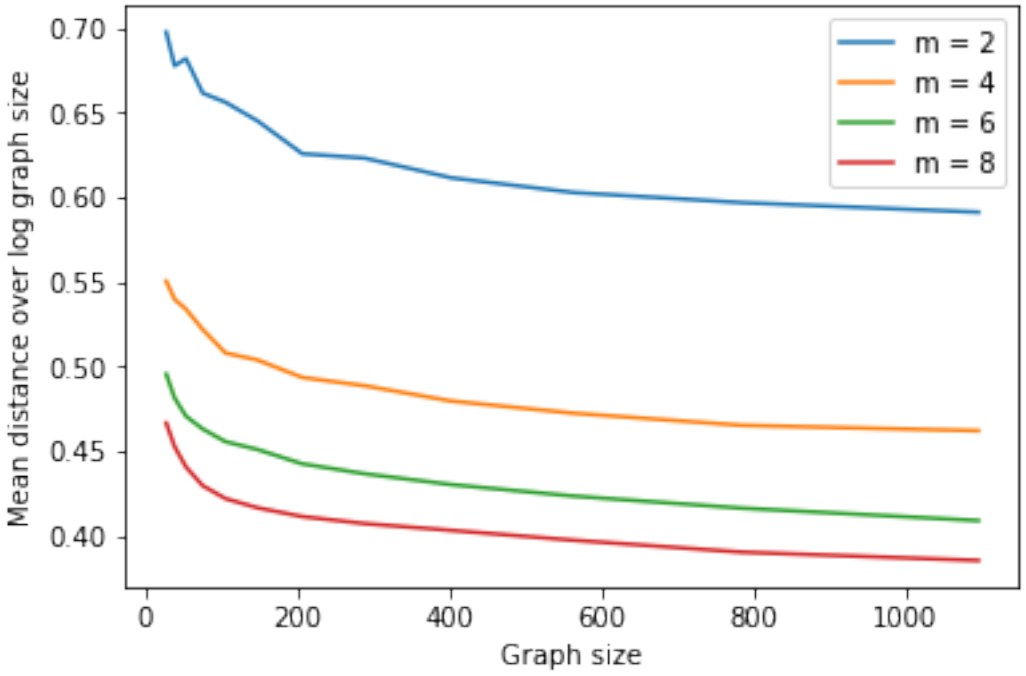
```
In [124]: N_range = [int(np.floor(np.exp(x/3))) for x in range (10,22)]
          m_range = [v * 2 for v in range(1,5)]

          I = 5

          clustering_plots = dict()
          mean_distance_plots = dict()
          for m in m_range:
              print("En cours : m = " + str(m))
              clustering_plots[m] = []
              mean_distance_plots[m] = []
```

```
            for N in N_range:
                print("N = " + str(N))
                clustering = []
                mean_distance =  []
                for i in range(I):
                    G = build_BarabasiAlbert(N,m)
                    clustering.append(nx.average_clustering(G))
                    mean_distance.append(nx.average_shortest_path_length(G))
                clustering_plots[m].append(np.mean(clustering))
                mean_distance_plots[m].append(np.mean(mean_distance))
```

```
En cours : m = 2
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : m = 4
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
En cours : m = 6
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
```

```
N = 785
N = 1096
En cours : m = 8
N = 28
N = 39
N = 54
N = 76
N = 106
N = 148
N = 207
N = 289
N = 403
N = 563
N = 785
N = 1096
```

```
In [125]: for m in m_range:
              plot = [mean_distance_plots[m][i]/np.log(N_range[i]) for i in range(len(N_range))]
              plt.plot(N_range,plot,label = "m = " + str(m))

          plt.legend()
          plt.ylabel("Mean distance over log graph size")
          plt.xlabel("Graph size")
          plt.show()
```



While my hardware forbids me to push my analysis further, mean distance seems to grow slightly slower than log graph size, proving small world property for Barabàsi-Albert graphs.

This result should be stable with changing entry degree value, as it is due to the presence of hubs in the graph, which is inherent to the preferential attachment mechanism. It is indeed verified for an array of m values, with predictibly lower constants as m grows.