1. A Cayley tree is a symmetric tree, constructed starting from a central node of degree *k*. Each node at distance *d* from the central node has degree *k*, until we reach the nodes at distance *P* that have degree one and are called leaves (see the image below for a Cayley tree with *k* = 3 and *P* = 5.)

(a) Calculate the number of nodes reachable in *t* steps from the central node.

The number of nodes reachable in t steps from the central node is given by,

Number of nodes reachable in t steps = $N_t = k(k-1)^{(t-1)}$

So, at **t=1** (one step away) we can reach, $N_1 = 3(3-1)^{(1-1)} = 3(2)^0 = 3$.

Similarly at 2 steps away at **t=2** we can reach, $N_2 = 3(3-1)^{(2-1)} = 3(2)^1 = 6$

Similarly at 3 steps away at **t=3** we can reach, $N_3 = 3(3-1)^{(3-1)} = 3(2)^2 = 12$.

Similarly at 4 steps away at **t=4** we can reach, $N_4 = 3(3-1)^{(4-1)} = 3(2)^3 = 24$.

Similarly at 5 steps away at **t=5** we can reach, $N_5 = 3(3-1)^{(5-1)} = 3(2)^4 = 48$.

Number of nodes = $1 + \sum^{p}_{t=1} k(k-1)^{(t-1)}$

Number of nodes = N = 1+(3+6+12+24+48) =94

Alternatively, we can calculate it using below formula:

Number of nodes = $N = 1 + \sum^{p}_{t=1} k(k-1)^{(t-1)}$

$$N = 1 + [k*((k-1)^{p} - 1)/(k-2)]$$

$$N = 1 + 3*(2^5 - 1)/1$$

$$= 1 + (3*31) = 94$$

(b) Calculate the degree distribution of the network

Given by $P_k = N_k / N$.

where N_k = the number of nodes of degree k and degree is number of adjacent nodes of a given node.

N = Total number of nodes

The number of surface nodes(leaves) (degree = 1) is given by

$Np = k(k-1)^{(p-1)} = 3*(2)^4 = 48$

Here $P_{leafNodes} = N_{leafNodes} / N$

As leaf nodes are having degree 1 so $P_{leafNodes} = P_1 = 3*(2)^4 / 94 = 48/94$

For the nodes having degree 2,

As there are no nodes having the degree 2 So, $P_2 = 0$

For Nodes having degree 3

$N_{centralNode} = 1$

$N_1 = 3(3\text{-}1)^{(1\text{-}1)} = 3(2)^0 = 3$ $N_2 = 3(3\text{-}1)^{(2\text{-}1)} = 3(2)^1 = 6$

$N_3 = 3(3\text{-}1)^{(3\text{-}1)} = 3(2)^2 = 12$ $N_4 = 3(3\text{-}1)^{(4\text{-}1)} = 3(2)^3 = 24$

So, for nodes having degree 3, can be calculated as,

$P_3 = N_{centralNode} + N_1 + N_2 + N_3 + N_4 = 1+3+6+12+24/94 = 46/94$

Or simply we can say,

$P_3 = 1 - P_1 = 1- 48/94 = 46/94$

Generally, the degree distribution is given using binomial and in larger networks this is given with poisson distribution.

So, the degree distribution of network is:

$$P_k = Bin((N-1), p) = \binom{N-1}{k} p^k (1-p)^{(N-1-k)}$$

$$P_k = \binom{N-1}{k} p^k (1-p)^{(N-1-k)}$$

(c) Calculate the diameter d_{max}.

$d_{max} = \ln N / \ln \langle k \rangle = \ln (94) / \ln (3) = 4.1354$

(d) Find an expression for the diameter d_{max} in terms of the total number of nodes N.

Consider a random network with average degree ‹k›. A node in this network has on average:

‹k› nodes at distance one (d=1).

‹k›2 nodes at distance two (d=2).

‹k›3 nodes at distance three (d =3).

...

‹k›d nodes at distance d.

the expected number of nodes up to distance d from our starting node is

$N_{(d)} \approx 1+\langle k\rangle+\langle k\rangle^2+...+\langle k\rangle^d = \langle k\rangle^{d+1}-1 \,/\, \langle k\rangle-1$

$N_{(d)}$ must not exceed the total number of nodes, N, in the network. Therefore, the distances cannot take up arbitrary values. We can identify the maximum distance, d_{max}, or the network's diameter by setting

$N_{(dmax)} \approx N$

Assuming that ‹k› » 1, we can neglect the (-1) term in the nominator and the denominator, obtaining

$\langle k\rangle_{dmax} \approx N$

Therefore, the diameter of a random network follows

$d_{max} \approx \ln(N) \,/\, \ln\langle k\rangle$

which represents the mathematical formulation of the small world phenomenon. The key, however, is its interpretation: the expected number of nodes up to distance d from our starting node i

(e) Does the network display the small-world property?

The small-world property is defined as the property of a network where any 2 randomly nodes can be reached from one another by ~log(N) hops. For these trees, most of the nodes are connected to each other. On contrary, in Cayley tree many nodes are not interconnected. So, for our Cayley tree number of hops for small-world tree will be deduced as → log(94) = 1.97~ 2.

So, we can clearly that all randomly selected nodes can't be connected to each other or traveled in just 2 hops. So, it is evident enough to prove that given network of Cayley tree with degree (k=3) and distance (P=5) does not display the small-world property.

2. (a) The mean neighbor degree of a node is the average degree of its neighbors. We define MND(G) to be the average mean neighbor degree across all the nodes of a network. In particular, where n and m are the number of nodes and edges in the network G respectively. Show that this is

$$\mathrm{MND}(G) = \frac{1}{2m}\sum_{u=1}^{n}\sum_{v=1}^{n} k_v A_{uv}$$

equivalent to (k^2)/(k)

Answer:

$$\begin{aligned}
\langle k_v \rangle &= \frac{1}{2m}\sum_{u=1}^{n}\sum_{v=1}^{n} k_v A_{uv} \\
&= \frac{1}{2 * \frac{1}{2}\sum_{i=1}^{n} k_i}\sum_{u=1}^{n} \vec{k}\vec{A}_u \\
&= \frac{1}{\sum_{i=1}^{n} k_i}\sum_{u=1}^{n} \vec{k}\vec{A}_u \\
&= \frac{\sum_{u=1}^{n} k_u^2}{\sum_{i=1}^{n} k_i} \\
&= \frac{\frac{1}{n}\sum_{u=1}^{n} k_u^2}{\frac{1}{n}\sum_{i=1}^{n} k_i} \\
&= \frac{\langle k^2 \rangle}{\langle k \rangle}
\end{aligned}$$

Thus, the mean neighbor degree (MND) is average square degree divided by average degree

.

.

.

.

2. (b) The friendship paradox occurs when MND(G) is greater than (k). In terms of a social network, this means that, on average, each of your friends have more friends than you. For each node in the UC Irvine student network, determine the average degree of its neighbors and make a scatterplot with node degree on the x-axis and mean neighbor degree divided by node degree on the y-axis. Include a horizontal line separating points that display the friendship paradox and points that do not.

```
#imports Required
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from itertools import chain
from networkx.utils import cumulative_distribution, discrete_sequence
import math
import random
import scipy.io
```

This function is to calculate mean neighbor degree over the range

```
#This function is to calculate mean neighbor degree over the range

def is_friend_paradox(G):
  plot_data_x = []
  plot_data_y = []
  #calculate average neighbor degree
  avg_neighbor_degree = nx.average_neighbor_degree(G)
  mean_neighbour_degree  = 0
  k = 0
  n = len(G.nodes)
  for val in avg_neighbor_degree:
    mean_neighbour_degree += avg_neighbor_degree[val]
    k += G.degree(val)
    mean_neighbour_degree /= n
    k /= n
    plot_data_y.append(mean_neighbour_degree/k)
    plot_data_x.append(k)
  return plot_data_x, plot_data_y
```

.

.

.

This function is used to plot the meanneighbor degree/degree on Y axis vs degree on X-axis

```
# plot the meanneighbor degree/degree on Y axis vs degree on X-axis
def plot_graph_linear_log_axis(plot_data_x,plot_data_y,linearLog):
  # create plot on linear axis
  fig, ax = plt.subplots(figsize=(12, 6))
  ax.scatter(plot_data_x, plot_data_y)
  # add horozontal line
  ax.axhline(y=1, c="red", linewidth=2.4, zorder=0)
  ax.set_title('UC Irvine student network data - Linear Scale')
  ax.set_xlabel('Degree')
  ax.set_ylabel('Mean Neighbor Degree / Degree')
  if linearLog:
      plt.yscale('log')
      plt.xscale('log')
      ax.set_title('UC Irvine student network data - Log Scale')
  plt.show()
```

.

.

Load UC Irvine student network data : https://toreopsahl.com/datasets/#online_social_network

```
# Load UC Irvine student network data : https://toreopsahl.com/datasets/#online_soci
G = nx.read_edgelist("/content/OClinks_w_chars.txt",data=[('weight', int)])
plot_data_x,plot_data_y = is_friend_paradox(G)
```

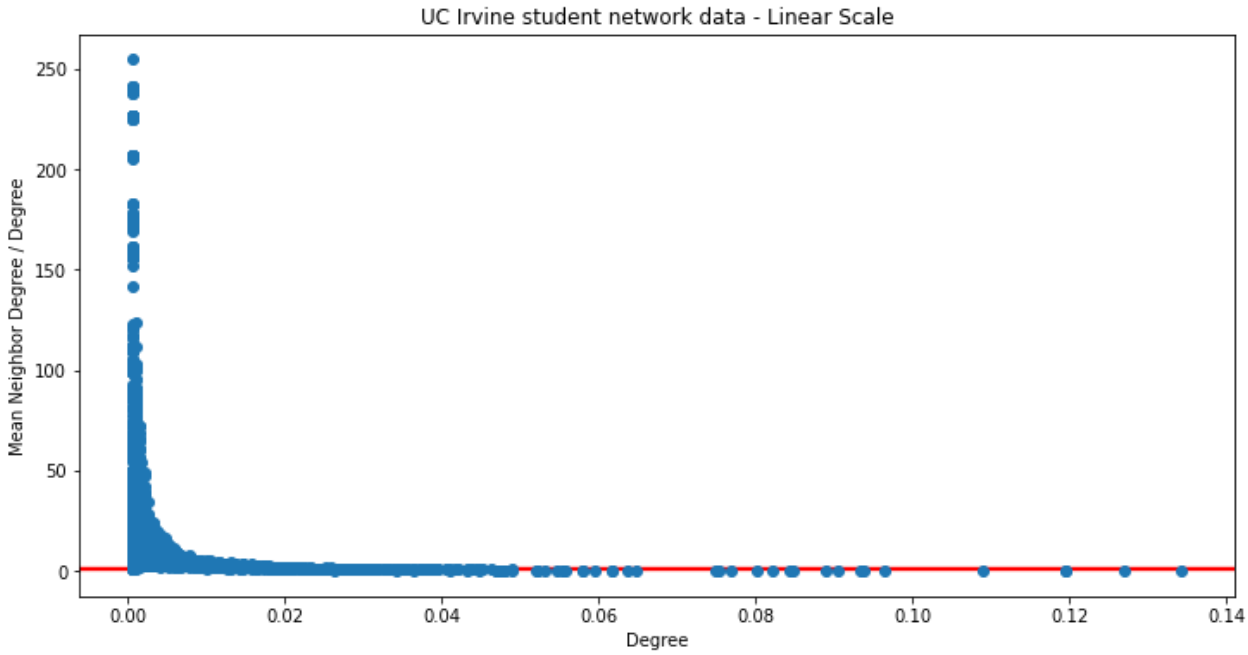
.

.

.

Plotting UC Irvine student network data on linear axis

```
# Plotting UC Irvine student network data on linear axis
plot_graph_linear_log_axis(plot_data_x,plot_data_y,False)
```



Included the horizontal line in red color at y = 1 separating points that display the friendship paradox and points that do not.

.

.

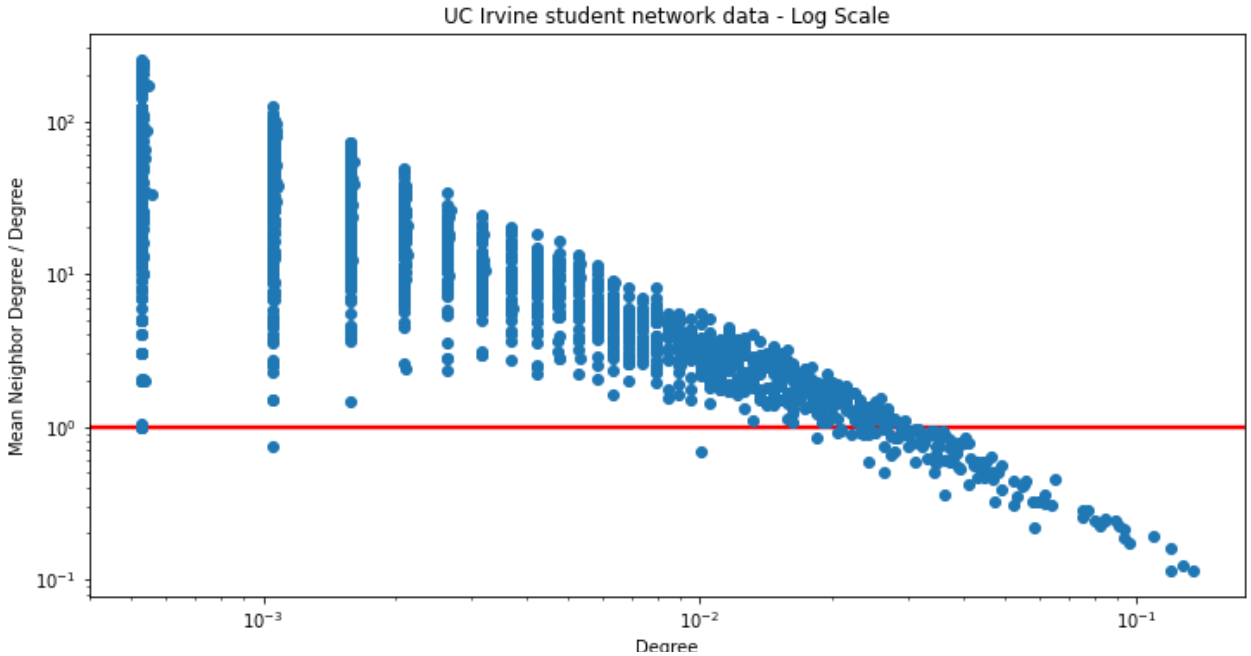
.

.

.

Plotting UC Irvine student network data on Log axis

```
1 # Plotting UC Irvine student network data on Log axis
2 plot_graph_linear_log_axis(plot_data_x,plot_data_y,True)
```



Included the horizontal line in red color at y = 1 separating points that display the friendship paradox and points that do not.

2. (c) Does the UC Irvine student network display the friendship paradox? Why might this paradox not be so surprising? Given part (a), consider what must be true of G for this paradox to be absent.

The UC Irvine Student network does display friendship paradox as there are few points falling in that criteria. The friendship paradox arises because of numbers of friends people have are distributed in a way that follows a power law rather than an ordinary linear relationship. So most people have a few friends while a small number of people have lots of friends. It's this second small group that causes the paradox. People with lots of friends are more likely to number among your friends in the first place. And when they do, they significantly raise the average number of friends that your friends have. That's the reason that, on average, your friends have more friends than you do.

For example, consider a 10000-node network consisting of a complete graph with one edge removed. In such a network almost all of the nodes—9998 of them—have a degree larger than the average of their neighbours, but there are two outliers that substantially skew the distribution so that over the whole network nodes are still less popular than their neighbours on average. So, it is evident that friendship paradox would be present across almost all networks with common characteristics.

For friendship paradox to be absent in G, I can say that all nodes should have degree 0 (with no edges) but it can't be represented as regular graph or network. Second choice would be a graph with all nodes of degree 2 (basically a ring) .

3.(a)Write a function configModel(degSeq) taking a degree sequence as input and returning a graph generated according to the configuration model as described in chapter 4.8 of Network Science.

Consider the set of graphs for which all vertices have degree 1 or 3 and n = 10^4 nodes. Let p1 be the probability that a node is of degree 1 and p3 = 1 − p1 be the probability that a node is of degree 3. Use configModel(degSeq) to make a figure showing the mean fractional size of the largest component for values of p1 from 0 to 1 in steps of 0.01. Estimate the value of p1 at which a phase transition occurs and the giant component disappears.

This function is used to generate the graph with the given degree sequence using the configuration model

```
#generate a graph with the given degree sequence using the configuration model
def configModel(degSeq):
  n = len(degSeq)
  G = nx.empty_graph(n, create_using=None)
  # If empty, return the null graph immediately.
  if n == 0:
    return G
  stublist = list(chain.from_iterable([n] * d for n, d in enumerate(degSeq)))
  # Choose a random balanced stublist, which gives a random pairing of nodes.
  # In this implementation, we shuffle the list and then split it in half.
  n = len(stublist)
  half = n // 2
  random.shuffle(stublist)
  out_stublist, in_stublist = stublist[:half], stublist[half:]
  G.add_edges_from(zip(out_stublist, in_stublist))

  #To remove parallel edges:
  G = nx.Graph(G)

  #To remove self loops:
  G.remove_edges_from(nx.selfloop_edges(G))

  #get gaint component from graph
  giantSizes = max(nx.connected_components(G), key=len)
  giantC = G.subgraph(giantSizes)
  return giantC
```

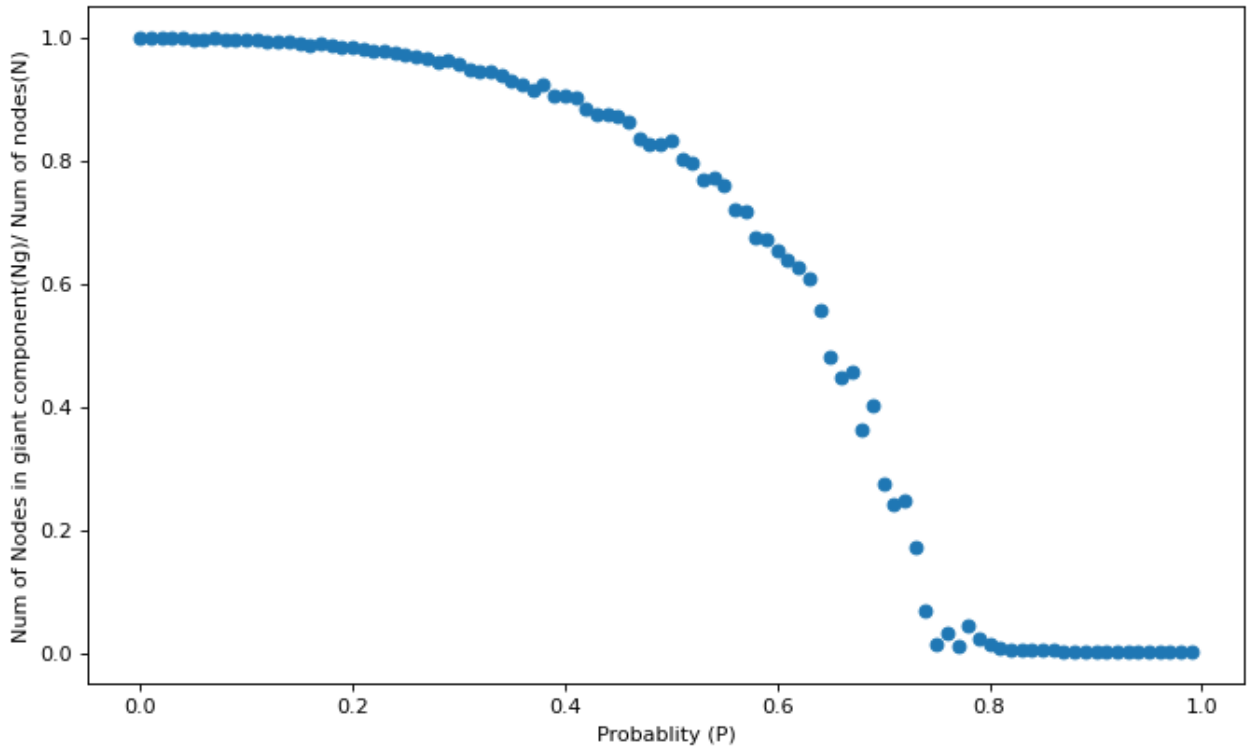
Consider the set of graphs for which all vertices have degree 1 or 3 and n = 10^4 nodes.

Let p1 be the probability that a node is of degree 1 and p3 = 1 − p1 be the probability that a node is of degree 3.

Use configModel(degSeq) to make a figure showing the mean fractional size of the largest component for values of p1 from 0 to 1 in steps of 0.01.

Estimate the value of p1 at which a phase transition occurs and the giant component disappears

```
n = 10000 # number of nodes n= 10^4
X =[]
Y =[]
for p in np.arange(0, 1, 0.01):
  # new dgree seq with diff probablity
  degSeq = np.random.choice([1,3], p=[p,1-p], size=n)
  giantC = configModel(degSeq)
  X.append(p)
  Y.append(giantC.number_of_nodes()/n)
plt.figure(figsize=(10, 6), dpi=80)
plt.scatter(X, Y)
plt.xlabel('Probablity (P)')
plt.ylabel('Num of Nodes in giant component(Ng)/ Num of nodes(N)')
plt.show()
```



The value of P1 where the giant component disappers is approx 0.79 - 0.8, at this point probablity of network is very large hence giant component transitions into connected regime comprised of all the nodes, i.e. complete graph. So, in this case giant component disappers at p=0.8

.

3. (b)Write a function degPresRand(G) taking a graph G as input and returning a new network with a degree sequence identical to that of G as described in chapter 4.8 of Network Science. Write your function to perform at least (m/2 log 10 ^7) rewirings as suggested here.

```
def degPresRand(G, rewires=1):
    n = 0
    swapcount = 0
    keys, degrees = zip(*G.degree())  # keys, degree
    cdf = nx.utils.cumulative_distribution(degrees)  # cdf of degree
    discrete_sequence = nx.utils.discrete_sequence
    while swapcount < rewires:
        # if random.random() < 0.5: continue # trick to avoid periodicities?
        #·pick·two·random·edges·without·creating·edge·list
        #·choose·source·node·indices·from·discrete·distribution
        (ui, xi) = discrete_sequence(2, cdistribution=cdf, seed=random)
        if ui == xi:
            continue  # same source, skip
        u = keys[ui]  # convert index to label
        x = keys[xi]
        # choose target uniformly from neighbors
        v = random.choice(list(G[u]))
        y = random.choice(list(G[x]))
        if v == y:
            continue  # same target, skip
        if (x not in G[u]) and (y not in G[v]):  # don't create parallel edges
            G.add_edge(u, x)
            G.add_edge(v, y)
            G.remove_edge(u, v)
            G.remove_edge(x, y)
            swapcount += 1
        n += 1
    return G
```

For above function we can set any rewire value which can be directly passed to function

As asked in 3(b) it supports the rewires (m/2) * log 10^7

.

.

.

3. (c) Using both configModel(degSeq) and degPresRand(G), explore the degree and distance distributions of the undirected versions of Openflights airport network (2016) and the German highway system network (2002). What patterns do you notice? Do you observe any significant deviations in these networks from what we might expect given a graph with a similar degree sequence?

This function is used to calculate the distances

```
#This function is used to calculate the distances
def distCounts(d):
  dists = {}
  for u in d:
    for v in d[u]:
      dists[d[u][v]] = dists.get(d[u][v], 0) + 1
  return dists
```

This function is used to plot network data with distance and count

```
# This function is used to plot network data with distance and count
def plotNetworkDataDistanceVsCount(G, networkDataName):

  print('Nodes', G.number_of_nodes(), 'Edges', G.number_of_edges())
  plt.figure(figsize=(10, 6), dpi=80)
  # Calculate distance
  dists = dict(nx.all_pairs_shortest_path_length(G))
  dists = distCounts(dists)
  (X, Y) = zip(*[(key, dists[key]) for key in dists])
  plt.scatter(X, Y, c='b', label=networkDataName, alpha=0.5)

  # Calculate config model
  H = configModel([G.degree(v) for v in G.nodes()])
  configDists = dict(nx.all_pairs_shortest_path_length(H))
  configDists = distCounts(configDists)
  (X, Y) = zip(*[(key, configDists[key]) for key in configDists])
  plt.scatter(X, Y, c='g', label='Configuration Model', alpha=0.5)

  #rewires the (m/2) * log 10^7
  m= G.number_of_edges();
  rewires_num = int(((m/2) * math.log10(pow(10, 7))))
  # Calculate degree press randomization
  H = degPresRand(G,rewires_num)
  randDists = dict(nx.all_pairs_shortest_path_length(H))
  randDists = distCounts(randDists)
```

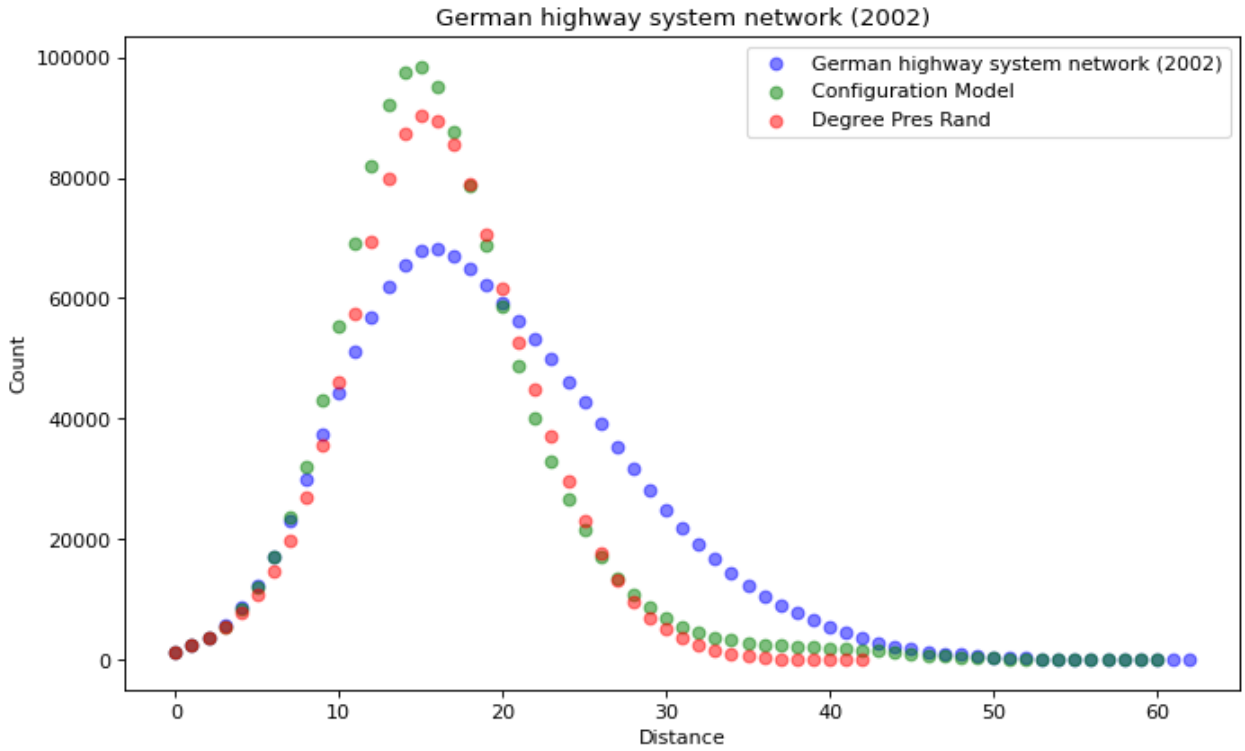
```
  (X, Y) = zip(*[(key, randDists[key]) for key in randDists])
  plt.scatter(X, Y, c='r', label='Degree Pres Rand', alpha=0.5)

  plt.title(networkDataName)
  plt.xlabel('Distance')
  plt.ylabel('Count')
  plt.legend(loc='upper right')
  plt.show()
```

Load German highway system network (2002)

```
#Load German highway system network (2002)
G = scipy.io.loadmat('/content/autobahn.mat')
G = nx.from_numpy_matrix(G['auto1168'])
plotNetworkDataDistanceVsCount(G,'German highway system network (2002)')
```

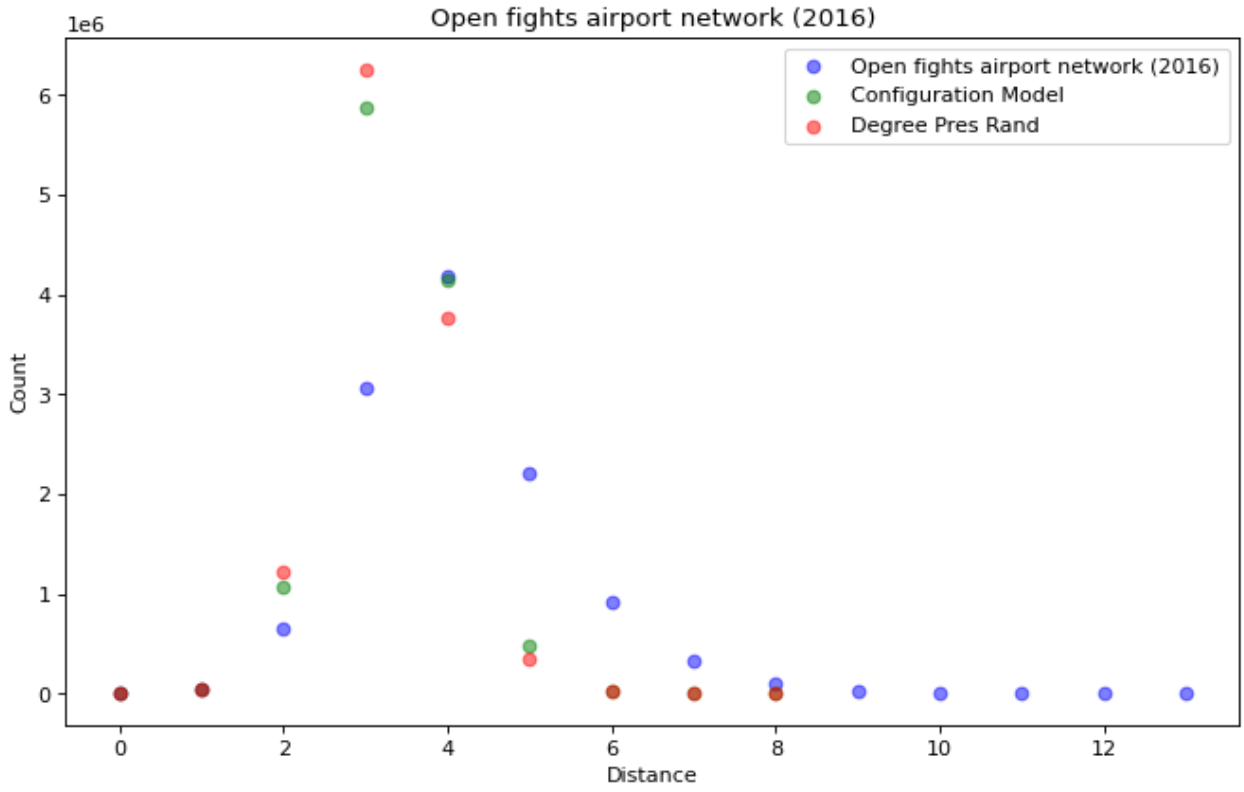
```
Nodes 1168 Edges 1243
```



Load Open fights airport network (2016) http://konect.cc/networks/openflights/

```
#Load Open fights airport network (2016)
G = nx.read_edgelist("/content/out.openflights")
plotNetworkDataDistanceVsCount(G,'Open fights airport network (2016)')
```

```
Nodes 3425 Edges 19257
```



.

Graph 1: German highway system network (2002):

-> ⟨d⟩ seems like close for degree preserving randomization comparable to original data and hence, we can say that this is scale-free network.

-> ⟨d⟩ value for real-network , config model and degree preserving randomization is almost same (approx 15) and can not see much deviation on that part. Given the fact, we can state that both config model and degree-preserving randomization are not changing any degree distribution across the network.

-> As ⟨d⟩ is comparable for all 3 types, we can say that either there are no hubs in the network or hubs are preserved across all 3 types.

Graph 2: Open fights airport network (2016):

-> ⟨d⟩ seems like some deviated for degree preserving randomization (⟨d⟩ approx 3) compared to original data (⟨d⟩ approx 4) and hence, we can say that this is not a scale-free network.

-> ⟨d⟩ value for config model and degree preserving randomization are approximately same or comparable (both approx 4) and are bit left-skewed compared to original data (⟨d⟩ approx 3). It may be safe to state that both config model and degree-preserving randomization changed the degree distribution across the network.

.

✓ 4m 9s completed at 6:04 PM