**1. Degree Distribution**

**1A) Bipartite Graph**

import networkx as nx

import numpy as np

import matplotlib.pyplot as plt

from networkx.algorithms import bipartite

#characters.txt contains a list of nodes referring to each character

#comics.txt contains a list of nodes referring to each comic book.

#out.txt coins edge-list of characters and comics.

characters = np.loadtxt('characters.txt',dtype=int)

comics = np.loadtxt('comics.txt',dtype=int)

B = nx.Graph() #this will create an empty graph

B.add\_nodes\_from(characters,bipartite=0)

B.add\_nodes\_from(comics,bipartite=1)

h = nx.read\_edgelist('out.txt',create\_using=nx.Graph(),nodetype=int)

B.add\_edges\_from(h.edges)

print(len(B.edges()))

freq = nx.degree\_histogram(B) #this will give the list of frequency of each degree.

degrees = range(len(freq)) # this will give numbers from 0 to highest degree.

p = [] #here p denotes p(k)

a = 1

for i in range(len(freq)):

a = freq[i]/19428

p.append(a)

#semi-log x plot

plt.semilogx(degrees, p, '.')

plt.xlabel("Degree")

plt.ylabel("p(k)")

plt.grid(True)

plt.savefig("normalized.png")

plt.show()

#log-log plot

plt.loglog(degrees, p,'.')

plt.xlabel('Degree')

plt.ylabel('Frequency')

plt.grid()

plt.savefig("Degree\_dis.png")

plt.show()

#semi-log y plot

plt.semilogy(degrees, p, '.')

plt.xlabel('Degree')

plt.ylabel('Frequency')

plt.grid(True)

plt.savefig('Semilog.png')

plt.show()

#linear plot

plt.plot(degrees,p)

plt.xlabel('Degree')

plt.ylabel('Frequency')

plt.grid(True)

plt.savefig('deg\_dist.png')

plt.show()

**1B) Unipartite Graph**

import math as m

characters = np.loadtxt('characters.txt',dtype=int)

comics = np.loadtxt('comics.txt',dtype=int)

B = nx.Graph()

B.add\_nodes\_from(characters,bipartite=0)

B.add\_nodes\_from(comics,bipartite=1)

h = nx.read\_edgelist('out.txt',create\_using=nx.Graph(),nodetype=int)

B.add\_edges\_from(h.edges)

G = bipartite.weighted\_projected\_graph(B,characters) #unipartite projection of bipartite graph

freq = nx.degree\_histogram(G) #this will give the list of frequency of each degree.

degrees = range(len(freq)) # this will give numbers from 0 to highest degree.

p = [] # here p denotes p(k)

a = 1

deg = 0

avg = 0

maximum = 0

loc = 0 #position of the node whose degree is maximum

print(len(G.edges()))

#average-degree calculation and getting the maximum degree

for i in G.nodes():

deg+=G.degree[i]

if(G.degree[i]>maximum):

maximum = G.degree[i]

loc = i

avg = deg/len(G.degree)

for i in range(len(freq)):

a = freq[i]/6486

p.append(a)

print("Average Degree :",avg)

print("Maximum Degree :",maximum,"which corresponds to node :",loc)

#degree distribution plot

#linear scale

plt.plot(degrees,p,'.')

plt.xlabel('Degree(k)')

plt.ylabel('p(k)')

plt.title('Degree Distribution')

plt.grid(True)

plt.savefig('deg\_dist.png')

plt.show()

#loglog scale

plt.loglog(degrees,p,'.')

plt.xlabel('Degree(k)')

plt.ylabel('p(k)')

plt.title('Degree Distribution (loglog scale)')

plt.grid(True)

plt.savefig('deg\_dist\_log.png')

plt.show()

#semi-log x scale

plt.semilogx(degrees,p,'b.')

plt.xlabel('Degree(k)')

plt.ylabel('p(k)')

plt.title('Degree Distribution (semilog x axis)')

plt.grid(True)

plt.savefig('deg\_dist\_semilog.png')

plt.show()

#semi-log y scale

plt.semilogy(degrees,p,'.')

plt.xlabel('Degree(k)')

plt.ylabel('p(k)')

plt.title('Degree Distribution (semilog y axis)')

plt.grid(True)

plt.savefig('deg\_dist\_semilog1.png')

plt.show()

**2. Dictionary to map node numbers with characters**

d = {}

#dict.txt contains node numbers and corresponding characters

with open("dict.txt") as file:

for line in file:

line = line.split(':')

if not line:

continue

d[line[0]] = line[1:]

**3. Robustness of the network**

**3.1.1 Random Node Removal**

import random

nodes = np.loadtxt('characters.txt',dtype=int)

edges = np.loadtxt('uni\_edges.txt',dtype=int)

G = nx.Graph()

G.add\_nodes\_from(nodes)

h = nx.read\_edgelist('uni\_edges.txt',create\_using=nx.Graph(),nodetype=int)

G.add\_edges\_from(h.edges)

per = 0.75

#per variable is used to ask how much of the nodes is to be kept

#or 1-p is the percentage of nodes to be removed

#in this case 25% nodes is removed from the network randomly

while len(G.nodes())>per\*len(nodes):

s = random.randint(1,6486)

if s in G.nodes():

G.remove\_node(s)

giant = max(nx.connected\_components(G),key=len)

cc = G.subgraph(giant)

avg\_path\_length = nx.average\_shortest\_path\_length(cc)

clust\_coeff = nx.average\_clustering(G)

eig\_vec\_cent = nx.eigenvector\_centrality(G)

bet\_cent = nx.betweenness\_centrality(G)

avg\_bet = sum(bet\_cent.values())/len(bet\_cent)

avg\_eig = sum(eig\_vec\_cent.values())/len(eig\_vec\_cent)

deg = 0

avg = 0

maximum = 0

for i in G.nodes():

deg+=G.degree[i]

if(G.degree[i]>maximum):

maximum = G.degree[i]

loc = i

avg = deg/len(G.degree)

print("Average path length :",avg\_path\_length)

print("Average CLustering Coefficient :",clust\_coeff)

print("Average Degree :",avg)

print("Average Betweenness Centrality :",avg\_bet)

print("Average Eigen-vector Centrality :",avg\_eig)