COMP336 Assignment1 Part1 Report

1. Load the network.

Option2: 113597493946570654755.edges in ego-gplus.

- The Method 'networkx.read_edgelist()' is used to load the network from the .edges file. The original network is a directed network with 321 nodes and 8558 edges. After adding the ego node, the updated network has 322 nodes and 8879 edges.
- Code with explanation:

```
import networkx as nx
# load the network based on .edges file
g = nx.read_edgelist(edge_file_path, nodetype=int, create_using = nx.DiGraph(),)
# obtain the list of all the nodes
node_list=list(g)
# record the original network info
g_info = nx.info(g)
print (g_info)
# add the ego node
g.add_node(nodeId)
# add edges for the ego node, it is connected to all other nodes
g.add_edges_from((v,nodeId) for v in node_list)
```

2. Visualize the updated network.

Figure 1 below is the picture showing the network via spring layout (dynamic node size proportional to its degree):

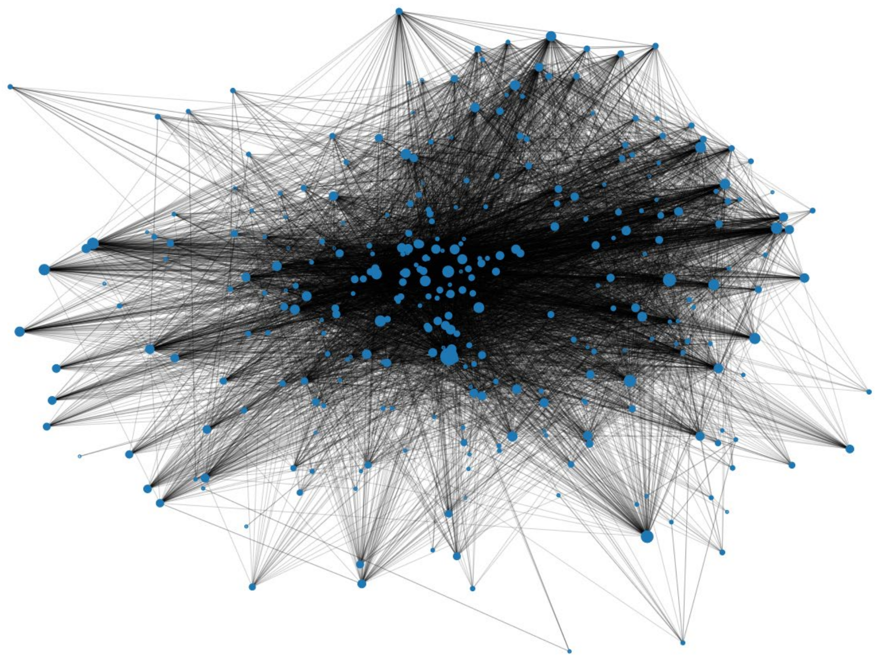

Figure 1

3. Plot the chart showing the in-degree of each user.

Figure 2 below shows the chart: (Its x-axis represents in-degree. Its y-axis shows the fraction of user having the in-degree. Both axes use logarithmic scale.)

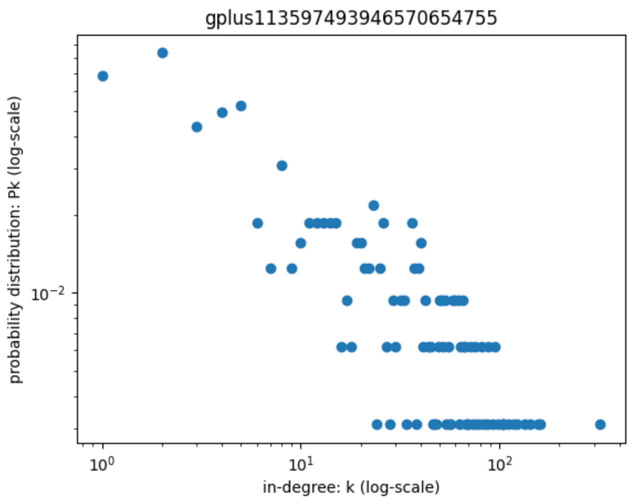

Figure 2

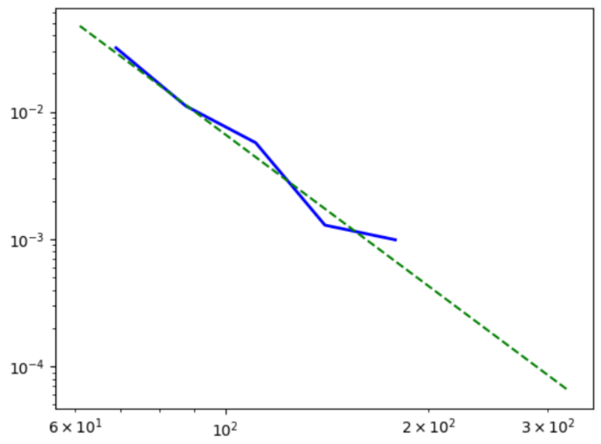

Figure 3

4. Analyze the network and identify the type of the network

As can be seen from figure 1:

- There's no node with 0 degree, thus all the nodes being inter-connected.
- We can see clearly that some outer nodes have small degrees, so the network is not fully connected.
- There're several big nodes in the picture. Since the node size is proportional to its degree, these nodes should be the hubs in this network.

As can be seen from figure 2:

- There does exist one super hub in this network, which is the ego node added by ourselves. But it still should not be a hubs-and-spokes topology network, since the general shape of figure 2 does not follow a convex function. Thus the value of alpha should be no greater than 1.
- The network has a few hubs, lots of median-sized nodes, and lots of small nodes. Therefore, the network has preferential attachment. It should be a scale free network, or a random network with sublinear preferential attachment with the value of alpha close to 1.

In conclusion, I think the network is a scale free network.

To verify the conclusion, I used a python library "powerlaw" to fit the in-degree of the nodes. Figure 3 above is the fitting graph.

- The blue line is the original data (log-scale), and the dashed the green line is the fitted power law distribution with γ=3.97. In this case, xmin=61 and sigma=0.43, which shows that the distribution is fitted starting from the 61th data and has a low standard error. This shows the nodes' in-degree follows power law distribution, though not a very standard one.
- I also compare the fitting situation of power law distribution with exponential and stretched exponential distribution, which corresponds to the random network without or with preferential attachment.
- The returned values of R are 0.795 and 0.029. Since R standards for the loglikelihood ratio between the two candidate distributions, it is positive if the data is more likely in power law.

Therefore, the network in general is a scale free network.

Appendix – python code

```
import networkx as nx
import matplotlib.pyplot as plt
import os
import json
import powerlaw
import numpy as np

netname = 'gplus' # twitter or gplus
nodeId = 113597493946570654755
k_sp = 1/1000000000000 # spring layout
k_ns = 0.08 # node size
width = 0.05 # edge width

edge_file_name = netname + str(nodeId)
edge_file_path = "../data/" + netname + "/"+str(nodeId) + ".edges"
output_path = "../output/" + edge_file_name +"/"

if not os.path.exists(output_path):
        os.mkdir(output_path)

# record file size
fsize = os.path.getsize(edge_file_path)
fsize = "File size: {:.2f} KB".format(fsize/1024)
with open(output_path + edge_file_name + '_info.txt', 'wt') as f:
        print (fsize, file=f)
        print (fsize)

# load the network based on .edges file
g = nx.read_edgelist(edge_file_path, nodetype=int, create_using=nx.DiGraph(),)
# obtain a list of all the nodes
node_list=list(g)

# record original network info
g_info = nx.info(g)
with open(output_path + edge_file_name + '_info.txt', 'at') as f:
        print (g_info, file=f)
        print (g_info)

## draw the original network
sp=nx.spring_layout(g, k= k_sp)
plt.axis('off')
nx.draw_networkx(g,pos=sp,with_labels=False, node_size=[(v+1) * k_ns for v in
dict(g.in_degree()).values()], arrows=False, width=width)
plt.savefig(output_path + edge_file_name + '_visial_size_original.svg', format="svg")
# plt.show()
plt.close()

# add the ego node
g.add_node(nodeId)
# add edges for the ego node, it is connected to all other nodes
g.add_edges_from((v,nodeId) for v in node_list)

## update network info
g_info = nx.info(g)
with open(output_path + edge_file_name + '_info.txt', 'at') as f:
        print (g_info, file=f)
        print (g_info)
```

```
## draw the ego network
sp=nx.spring_layout(g, k = k_sp)
plt.axis('off')
nx.draw_networkx(g,pos=sp,with_labels=False, node_size=[(v+1) * k_ns for v in
dict(g.in_degree()).values()], arrows=False, width=width)
plt.savefig(output_path + edge_file_name + '_visial_size_addEgo.svg', format="svg")
# plt.show()
plt.close()

## compute degree centrality
in_degrees=dict(g.in_degree())
in_degree_values=sorted(set(in_degrees.values()))
histogram=[list(in_degrees.values()).count(i)/float(nx.number_of_nodes(g)) for i in
in_degree_values]
indeg_list=in_degrees.values()

## draw the degree
# plt.loglog(in_degree_values, histogram, 'o')
plt.plot(in_degree_values, histogram, 'o')
plt.xlabel("in-degree: k (log-scale)")
plt.ylabel("probability distribution: Pk (log-scale)")
plt.xscale('log')
plt.yscale('log')
plt.title(edge_file_name)
plt.savefig(output_path + edge_file_name + '_indeg.png', format="png")
# plt.show()
plt.close()


# fit the degree
degree_sequence = sorted([d for d in indeg_list], reverse=True)
fit = powerlaw.Fit(np.array(degree_sequence)+1, discrete=True)
fig1 = fit.plot_pdf(color='b', linewidth=2)
fit.power_law.plot_pdf(color='g', linestyle='--', ax=fig1)
# the alpha here is actually the gamma in power law
print('xmin= ', fit.xmin,'alpha= ',fit.power_law.alpha,'    sigma= ',fit.power_law.sigma)
plt.show()

# compare different distributions
R, p = fit.distribution_compare('power_law', 'exponential', normalized_ratio=True)
print (R, p)
R, p = fit.distribution_compare('power_law', 'stretched_exponential', normalized_ratio=True)
print (R, p)
```