ASSIGNMENT 2

Vishal Gadiraju
01672049

1. Network Characteristics:

```
import snap
import numpy as np
import matplotlib.pyplot as plt
# Setup
erdosRenyi = None
smallWorld = None
collabNet = None

# Problem 1.1
def genErdosRenyi(N=5242, E=14484):
    Graph = snap.TUNGraph.New()
    # Adding the nodes
    for i in range(N):
        Graph.AddNode(i)
    # Adding the edges
    while Graph.GetEdges() < E:
        nodes = np.random.choice(N,2,replace = False)
        if not Graph.IsEdge(nodes[0],nodes[1]):
            Graph.AddEdge(nodes[0],nodes[1])
    return Graph

def genCircle(N=5242):
    Graph = snap.TUNGraph.New()
# Add nodes
    for i in range(N):
        Graph.AddNode(i)
# Add edges
    for i in range(N-1):
        Graph.AddEdge(i,i+1)
    Graph.AddEdge(N-1,0)
    return Graph

def connectNbrOfNbr(Graph, N=5242):
# Add edges
    for i in range(N-2):
        Graph.AddEdge(i,i+2)

    Graph.AddEdge(N-3,N-1)
    Graph.AddEdge(N-2,0)
    Graph.AddEdge(N-1,1)
    return Graph

def connectRandomNodes(Graph, M=4000):
```

```
# Add edges
    target_numedges = Graph.GetEdges() + M
    num_nodes = Graph.GetNodes()
    while Graph.GetEdges() < target_numedges:
        nodes = np.random.choice(num_nodes,2,replace = False)
        if not Graph.IsEdge(nodes[0],nodes[1]):
            Graph.AddEdge(nodes[0],nodes[1])
    return Graph
def genSmallWorld(N=5242, E=14484):
    Graph = genCircle(N)
    Graph = connectNbrOfNbr(Graph, N)
    Graph = connectRandomNodes(Graph, 4000)
    return Graph
def loadCollabNet(path):
    Graph = snap.LoadEdgeList(snap.PUNGraph, path, 0, 1, '\t')
    snap.DelSelfEdges(Graph)
    return Graph
def getDataPointsToPlot(Graph):
    odegree_list = []
    for NI in Graph.Nodes():
        odegree_list.append(NI.GetOutDeg())
    from collections import Counter
    cnt = Counter(odegree_list)
    X_cnt = cnt.keys()
    Y_cnt = cnt.values()
    X, Y = [], []
    for i in range(len(odegree_list)):
        X.append(i)
        if i in X_cnt:
            idx = X_cnt.index(i)
            Y.append(Y_cnt[idx])
        else:
            Y.append(0)
    return X, Y
def Q1_1():
    global erdosRenyi, smallWorld, collabNet
    erdosRenyi = genErdosRenyi(5242, 14484)
    smallWorld = genSmallWorld(5242, 14484)
    collabNet = loadCollabNet("ca-GrQc.txt")
    x_erdosRenyi, y_erdosRenyi = getDataPointsToPlot(erdosRenyi)
    plt.loglog(x_erdosRenyi, y_erdosRenyi, color = 'y', label = 'Erdos Renyi Network')
    x_smallWorld, y_smallWorld = getDataPointsToPlot(smallWorld)
    plt.loglog(x_smallWorld, y_smallWorld, linestyle = 'dashed', color = 'b', label = 'Small
World Network')
    x_collabNet, y_collabNet = getDataPointsToPlot(collabNet)
    plt.loglog(x_collabNet, y_collabNet, linestyle = 'dotted', color = 'r', label =
'Collaboration Network')
    plt.xlabel('Node Degree (log)')
    plt.ylabel('Node Proportion with the Given Degree (log)')
```

```
    plt.title('Degree Distribution of Erdos Renyi, Small World, and Collaboration Networks')
    plt.legend()
    plt.show()
# Execute code for Q1.1
Q1_1()
# Problem 1.2
# Find max degree of all 3 graphs for plotting (add 2 for padding)
maxdeg = max([erdosRenyi.GetNI((snap.GetMxDegNId(erdosRenyi))).GetDeg(),
                smallWorld.GetNI((snap.GetMxDegNId(smallWorld))).GetDeg(),
                collabNet.GetNI((snap.GetMxDegNId(collabNet))).GetDeg()]) + 2
# Erdos Renyi
def calcQk(Graph, maxDeg=maxdeg):
    q_k = np.zeros(maxDeg)
    ks, ns = getDataPointsToPlot(Graph)
    total_nodes = np.sum(ns)
    ks ,ns = ks[:maxDeg], ns[:maxDeg]
    ps = [x/float(total_nodes) for x in ns]
    denom_k = np.sum([x*y for x,y in zip(ks,ps)])
    numerator = [x*y for x,y in zip(ks[1:],ps[1:])] # k+1*Pk+1
    numerator.append(0)
    q_k = numerator/denom_k
    return q_k

def calcExpectedDegree(Graph):
    ed = 0.0
    ks, ns = getDataPointsToPlot(Graph)
    total_nodes = Graph.GetNodes()
    ps = [x/float(total_nodes) for x in ns]
    ed = np.sum([x*y for x,y in zip(ks,ps)])
    return ed

def calcExpectedExcessDegree(Graph, qk):
    eed = 0.0
    eed = np.sum([x*y for x,y in zip(qk,range(len(qk)))])
    return eed

def Q1_2_a():
    print 'Output for Q1_2_a :'
    print ' '
    qk_erdosRenyi = calcQk(erdosRenyi, maxdeg)
    qk_smallWorld = calcQk(smallWorld, maxdeg)
    qk_collabNet = calcQk(collabNet, maxdeg)

    plt.loglog(range(maxdeg), qk_erdosRenyi, color = 'y', label = 'Erdos Renyi Network')
    plt.loglog(range(maxdeg), qk_smallWorld, linestyle = 'dashed', color = 'r', label =
'Small World Network')
    plt.loglog(range(maxdeg), qk_collabNet, linestyle = 'dotted', color = 'b', label =
'Collaboration Network')
```

```
    plt.xlabel('k Degree')
    plt.ylabel('Excess Degree Distribution')
    plt.title('Excess Degree Distribution of Erdos Renyi, Small World, and Collaboration
Networks')
    plt.legend()
    plt.show()

    # Calculate Expected Degree
    ed_erdosRenyi = calcExpectedDegree(erdosRenyi)
    ed_smallWorld = calcExpectedDegree(smallWorld)
    ed_collabNet = calcExpectedDegree(collabNet)
    print 'Expected Degree for Erdos Renyi: %f' % ed_erdosRenyi
    print 'Expected Degree for Small World: %f' % ed_smallWorld
    print 'Expected Degree for Collaboration Network: %f' % ed_collabNet
    print ''
    print ''

    # Calculate Expected Excess Degree
    eed_erdosRenyi = calcExpectedExcessDegree(erdosRenyi, qk_erdosRenyi)
    eed_smallWorld = calcExpectedExcessDegree(smallWorld, qk_smallWorld)
    eed_collabNet = calcExpectedExcessDegree(collabNet, qk_collabNet)
    print 'Expected Excess Degree for Erdos Renyi: %f' % (eed_erdosRenyi)
    print 'Expected Excess Degree for Small World: %f' % (eed_smallWorld)
    print 'Expected Excess Degree for Collaboration Network: %f' % (eed_collabNet)
    print ''
    print ''

# Execute code for Q1.2a
Q1_2_a()

# Problem 1.3 - Clustering Coefficient

def calcClusteringCoefficient(Graph):
    C = 0.0
    c_arrays =[]
    for NI in Graph.Nodes():
        k = NI.GetOutDeg()
        if k>=2:
            k_denom = k*(k-1)/2
            neibor_nodes =[]
            connected_neibor_edges = 0
            for Id in NI.GetOutEdges():
                neibor_nodes.append(Id)
            for id_n1 in range(len(neibor_nodes)):
                for id_n2 in range(id_n1+1,len(neibor_nodes)):
                    n1 = neibor_nodes[id_n1]
                    n2 = neibor_nodes[id_n2]
                    connected = Graph.IsEdge(n1,n2)
                    if connected:
```

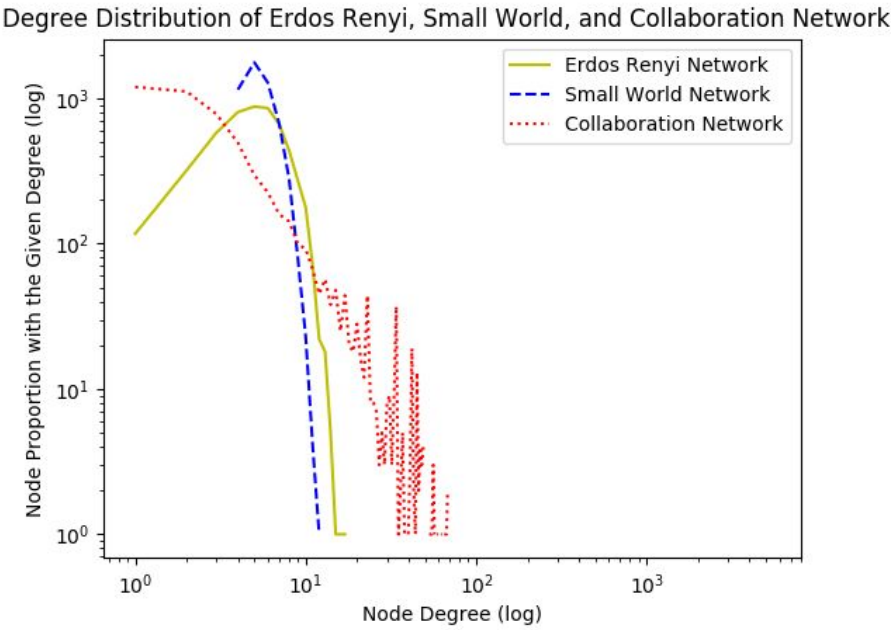
```
                        connected_neibor_edges +=1
            c = connected_neibor_edges/float(k_denom)
            c_arrays.append(c)
        else:
            c_arrays.append(0)
    C = np.mean(c_arrays)
    return C

def Q1_3():
    C_erdosRenyi = calcClusteringCoefficient(erdosRenyi)
    C_smallWorld = calcClusteringCoefficient(smallWorld)
    C_collabNet = calcClusteringCoefficient(collabNet)
    print 'Output for Q1_3:'
    print ' '
    print('Clustering Coefficient for Erdos Renyi Network: %f' % C_erdosRenyi)
    print('Clustering Coefficient for Small World Network: %f' % C_smallWorld)
    print('Clustering Coefficient for Collaboration Network: %f' % C_collabNet)
# Execute code for Q1.3
Q1_3()
```

Output:

Output for Q1_1

Difference between the collaboration networks degree distribution and the degree distributions from the random graph model

Ans: Degree distributions of collaboration networks are highly skewed and quite difficult from the distribution of the random graph model.



Output for Q1_2_a:

Expected Degree for Erdos Renyi: 5.526135
Expected Degree for Small World: 5.526135
Expected Degree for Collaboration Network: 5.526135

Expected Excess Degree for Erdos Renyi: 5.502624
Expected Excess Degree for Small World: 4.799365
Expected Excess Degree for Collaboration Network: 15.870409

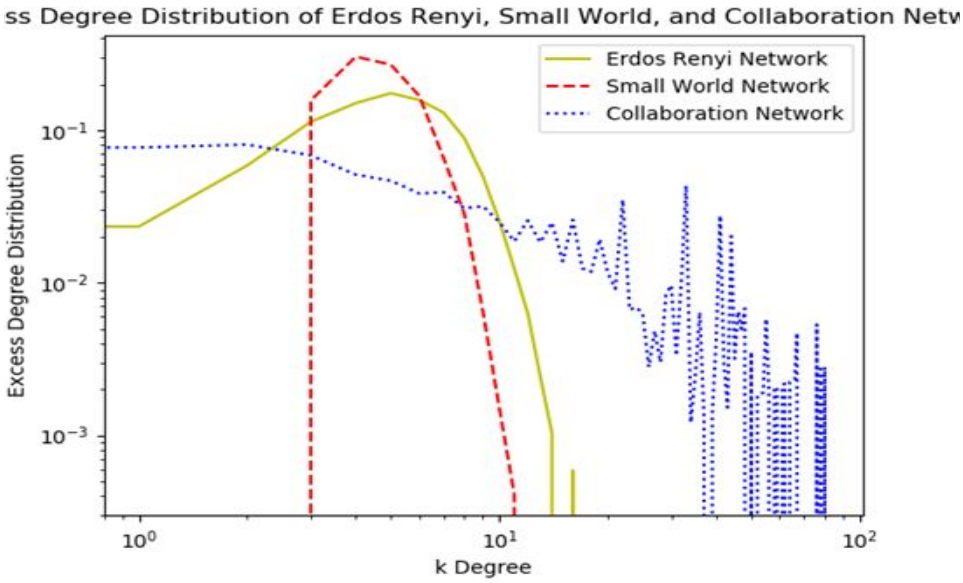
Difference in the distribution of the degree and excess degree distributions for the collaboration network?

Ans: The distribution for the collaboration network follows a power law and the most frequent degrees are the smallest degrees whereas the other distributions are peaked between the smallest and largest degrees.
The expected excess degree distribution is much larger than the expected degree as on average, the co-authors have more co-authors.

Output for Q1_2_b:

Proof showing how to calculate $\{q_k\}$ in terms of $\{p_k\}$
The probability that a random edge goes to a node of degree k + 1 is
Q_k, calculated as the number of edges that go to a node of degree k + 1 over the total number of edges.
The number of edges that go to a node of degree "k + 1" **= $n(k+1)p_{k+1}$**

Since the expected degree is $\Sigma_k kp_k$, the total number of edges **= $n\Sigma_k kp_k$**
Therefore, $q_k = (n(k+1)p_{k+1})/(n\Sigma_k kp_k)$, or after cancelling **n, $q_k = ((k+1)\ p_{k+1})/(\Sigma_k kp_k)$**



Output for Q1_3

Clustering Coefficient for Erdos Renyi Network: 0.000883
Clustering Coefficient for Small World Network: 0.284580
Clustering Coefficient for Collaboration Network: 0.529636

Why this network has the largest average clustering coefficient?

Co-authors also have co-authors to each other, since they also want to be in the field and, having more than two authors on a paper adds triangles to graph.Therefore, this network has the largest average clustering coefficient.

2. Bowtie Structure of Non-Web Networks:

```
import snap
import os
import numpy as np
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
os.chdir('C:/Python27/HW2')
'''
load directed edge list
'''
Epinions = snap.LoadEdgeList(snap.PNGraph, 'soc-Epinions1.txt', 0, 1, '\t')
Emails = snap.LoadEdgeList(snap.PNGraph, 'Email-EuAll.txt', 0, 1, '\t')
'''
get max scc and nodes list of each graph
'''
EpiMxScc = snap.GetMxScc(Epinions)
EmailMxScc = snap.GetMxScc(Emails)
EpiMxScc_nodes = [x.GetId() for x in EpiMxScc.Nodes()]
EmailMxScc_nodes = [x.GetId() for x in EmailMxScc.Nodes()]
'''
BFS on forward and backward computing
'''
'''
forward bfs trees and nodes list
'''
EpiBfsTreeOut_9809 = snap.GetBfsTree(Epinions, 9809, True, False)
EpiBfsTreeOut_1952 = snap.GetBfsTree(Epinions, 1952, True, False)
EpiBfsTreeOut_1952_nodes = [x.GetId() for x in EpiBfsTreeOut_1952.Nodes()]
EpiBfsTreeOut_9809_nodes = [x.GetId() for x in EpiBfsTreeOut_9809.Nodes()]
EmailBfsTreeOut_189587 = snap.GetBfsTree(Emails, 189587, True, False)
EmailBfsTreeOut_675 = snap.GetBfsTree(Emails, 675, True, False)
EmailBfsTreeOut_189587_nodes = [x.GetId() for x in EmailBfsTreeOut_189587.Nodes()]
EmailBfsTreeOut_675_nodes = [x.GetId() for x in EmailBfsTreeOut_675.Nodes()]
'''
backward bfs trees and nodes list
'''
EpiBfsTreeIn_9809 = snap.GetBfsTree(Epinions, 9809, False, True)
EpiBfsTreeIn_1952 = snap.GetBfsTree(Epinions, 1952, False, True)
EpiBfsTreeIn_1952_nodes = [x.GetId() for x in EpiBfsTreeIn_1952.Nodes()]
EpiBfsTreeIn_9809_nodes = [x.GetId() for x in EpiBfsTreeIn_9809.Nodes()]
EmailBfsTreeIn_189587 = snap.GetBfsTree(Emails, 189587, False, True)
EmailBfsTreeIn_675 = snap.GetBfsTree(Emails, 675, False, True)
EmailBfsTreeIn_189587_nodes = [x.GetId() for x in EmailBfsTreeIn_189587.Nodes()]
EmailBfsTreeIn_675_nodes = [x.GetId() for x in EmailBfsTreeIn_675.Nodes()]
'''
2.1
'''
print 'Output for Q2_1'
```

```
print ''
print 'For Epinions'
print ' '
a = len(set(EpiMxScc_nodes).intersection(set(EpiBfsTreeOut_9809_nodes)))
b = len(set(EpiMxScc_nodes).intersection(set(EpiBfsTreeIn_9809_nodes)))
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 9809 forward link
BFS Tree'\
       % (a)
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 9809 backward link
BFS Tree'\
% (b)
if a < b :
       print 'Node 9809: Out component'
elif a > b :
               print 'Node 9809: In component'
else :
       print 'Node 9809: SCC component'

print ' '

c = len(set(EpiMxScc_nodes).intersection(set(EpiBfsTreeOut_1952_nodes)))
d = len(set(EpiMxScc_nodes).intersection(set(EpiBfsTreeIn_1952_nodes)))
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 1952 forward link
BFS Tree'\
       % (c)
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 1952 backward link
BFS Tree'\
% (d)
if c < d :
       print 'Node 1952: Out component'
elif c > d :
               print 'Node 1952: In component'
else :
       print 'Node 1952: SCC component'
print ' '
print 'For Email'
print ' '
e = len(set(EmailMxScc_nodes).intersection(set(EmailBfsTreeOut_189587_nodes)))
f = len(set(EmailMxScc_nodes).intersection(set(EmailBfsTreeIn_189587_nodes)))
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 189587 forward
link BFS Tree'\
       % (e)
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 189587 backward
link BFS Tree'\
% (f)
if e < f :
       print 'Node 189587: Out component'
elif e > f :
               print 'Node 189587: In component'
```

```
else :
        print 'Node 189587: SCC component'
print ' '
g = len(set(EmailMxScc_nodes).intersection(set(EmailBfsTreeOut_675_nodes)))
h = len(set(EmailMxScc_nodes).intersection(set(EmailBfsTreeIn_675_nodes)))
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 675 forward link
BFS Tree'\
        % (g)
print 'There are %i nodes intersecting Epinion Max Scc Component and Node 675 backward link
BFS Tree'\
% (h)
if g < h :
        print 'Node 675: Out component'
elif g > h :
                print 'Node 675: In component'
else :
        print 'Node 675: SCC component'
print ' '
'''
2.2
'''
print 'Output for Q2_2'
def Bfs_Nodes_Cnt(graph,nodeid,forward=True):
        BfsTree = snap.GetBfsTree(graph, nodeid, forward, not forward)
        BfsTree_nodescnt = BfsTree.GetNodes()
        return BfsTree_nodescnt
perc = np.linspace(0,100,100)
Epi_Fwd_Bfs_NodeCnt = []
Epi_Bck_Bfs_NodeCnt = []
Rnd = snap.TRnd(42)
Rnd.Randomize()
for i in range(0,100):
        NID = Epinions.GetRndNId(Rnd)
    Epi_Fwd_Bfs_NodeCnt.append(Bfs_Nodes_Cnt(Epinions,NID,forward=True))
    Epi_Bck_Bfs_NodeCnt.append(Bfs_Nodes_Cnt(Epinions,NID,forward=False))
Email_Fwd_Bfs_NodeCnt = []
Email_Bck_Bfs_NodeCnt = []
Rnd = snap.TRnd(43)
Rnd.Randomize()
for i in range(0,100):
        NID = Emails.GetRndNId(Rnd)
        Email_Fwd_Bfs_NodeCnt.append(Bfs_Nodes_Cnt(Emails,NID,forward=True))
    Email_Bck_Bfs_NodeCnt.append(Bfs_Nodes_Cnt(Emails,NID,forward=False))
'''
Sort
'''
Epi_Fwd_Bfs_NodeCnt.sort()
Epi_Bck_Bfs_NodeCnt.sort()
Email_Fwd_Bfs_NodeCnt.sort()
```

```
Email_Bck_Bfs_NodeCnt.sort()
fig = plt.figure(1, (7,4))
ax = fig.add_subplot(1,1,1)
ax.plot(perc, Epi_Fwd_Bfs_NodeCnt)
fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)
plt.title('Epinions: Reachability using outwardlinks')
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached')
plt.show()
plt.yscale('log')
plt.plot(perc, Epi_Fwd_Bfs_NodeCnt)
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached(log)')
plt.title('Epinions: Reachability using outwardlinks')
plt.show()
fig = plt.figure(1, (7,4))
ax = fig.add_subplot(1,1,1)
ax.plot(perc, Epi_Bck_Bfs_NodeCnt)
fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)
plt.title('Epinions: Reachability using inwardlinks')
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached')
plt.show()
plt.yscale('log')
plt.plot(perc, Epi_Bck_Bfs_NodeCnt)
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached(log)')
plt.title('Epinions: Reachability using inwardlinks')
plt.show()
fig = plt.figure(1, (7,4))
ax = fig.add_subplot(1,1,1)
ax.plot(perc, Email_Fwd_Bfs_NodeCnt)
fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)
plt.title('Email: Reachability using outwardlinks')
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached')
plt.show()
plt.yscale('log')
plt.plot(perc, Email_Fwd_Bfs_NodeCnt)
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached(log)')
plt.title('Email: Reachability using outwardlinks')
plt.show()
```

```
fig = plt.figure(1, (7,4))
ax = fig.add_subplot(1,1,1)
ax.plot(perc, Email_Bck_Bfs_NodeCnt)
fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
xticks = mtick.FormatStrFormatter(fmt)
ax.xaxis.set_major_formatter(xticks)
plt.title('Email: Reachability using inwardlinks')
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached')
plt.show()
plt.yscale('log')
plt.plot(perc, Email_Bck_Bfs_NodeCnt)
plt.xlabel('fraction of starting nodes')
plt.ylabel('number of nodes reached(log)')
plt.title('Email: Reachability using outwardlinks')
plt.show()
'''
2.3
'''
def Components_compute(graph):
        total_size = graph.GetNodes()
        WCC_Components = snap.TCnComV()
        snap.GetWccs(graph, WCC_Components)
        largest_wcc_size = WCC_Components[0].Len()
        scc_component = snap.GetMxScc(graph)
        scc_nodes = [x.GetId() for x in scc_component.Nodes()]
        scc_size =  scc_component.GetNodes()
        print 'Size of SCC: %i nodes' % (scc_size)
        '''
        Get a random node from SCC and perform BFS on inward and outward directions
        '''
        Rnd = snap.TRnd(42)
        Rnd.Randomize()
        NID = scc_component.GetRndNId(Rnd)
        BfsTree_Out = snap.GetBfsTree(graph, NID, True, False)
        BfsTree_Out_nodes = [x.GetId() for x in BfsTree_Out.Nodes()]
        BfsTree_In = snap.GetBfsTree(graph, NID, False, True)
        BfsTree_In_nodes = [x.GetId() for x in BfsTree_In.Nodes()]
        '''
        Subtract the SCC nodes from BFS trees we got In and Out components nodes
        '''
        in_size = len(set(BfsTree_In_nodes) - set(scc_nodes))
        print 'Size of IN component: %i nodes' % (in_size)
        out_size = len(set(BfsTree_Out_nodes) - set(scc_nodes))
        print 'Size of OUT component: %i nodes' % (out_size)
    '''
        largest WCC - SCC - In - Out == Tendrills
        '''
    tendrills_size = largest_wcc_size - scc_size - in_size - out_size
```

```
        print 'Size of TENDRILS: %i nodes' % (tendrills_size)
        disconnected_size = total_size - largest_wcc_size
        num_disconnected_components = WCC_Components.Len() - 1
        print 'Size of DISCONNECTED: %i nodes' % disconnected_size
        print ' '
print 'Output of Q2_3'
print ''
print 'For Emails'
print ''
Components_compute(Emails)
print 'For Epinions'
print ''
Components_compute(Epinions)
```

OUTPUT :

Output for Q2_1:

For Epinions

There are 0 nodes intersecting Epinion Max Scc Component and Node 9809 forward link BFS Tree

There are 32223 nodes intersecting Epinion Max Scc Component and Node 9809 backward link BFS Tree

Node 9809: Out component

There are **32223** nodes intersecting Epinion Max Scc Component and Node **1952** forward link BFS Tree

There are **0** nodes intersecting Epinion Max Scc Component and Node **1952** backward link BFS Tree

Node 1952: In component

For Email

There are **34203** nodes intersecting Epinion Max Scc Component and Node **189587** forward link BFS Tree

There are **34203** nodes intersecting Epinion Max Scc Component and Node **189587** backward link BFS Tree

Node 189587: SCC component

There are **0** nodes intersecting Epinion Max Scc Component and Node **675** forward link BFS Tree

There are **34203** nodes intersecting Epinion Max Scc Component and Node **675** backward link BFS Tree

Node 675: Out component

Output for Q2_2:

How many nodes can you reach each time, for each of the two traversals? What behaviour do these traversals exhibit and what can you infer from them about the graph structure?

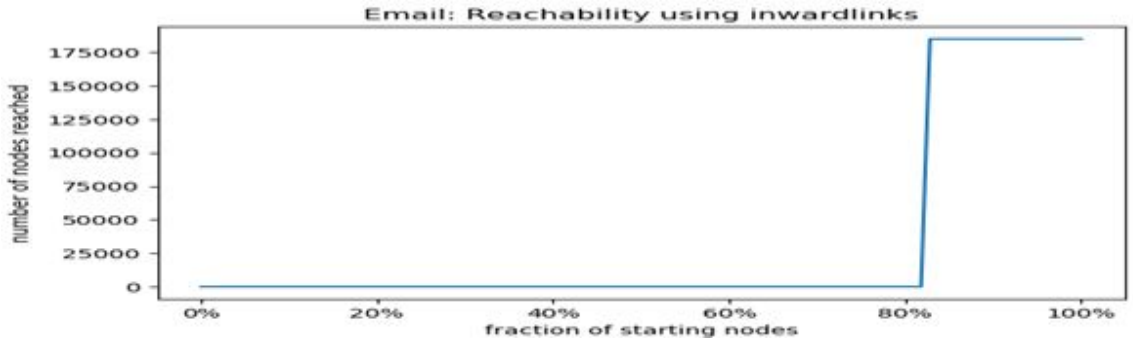
Epinions:
SCC+IN is larger than SCC+OUT, because of the following:

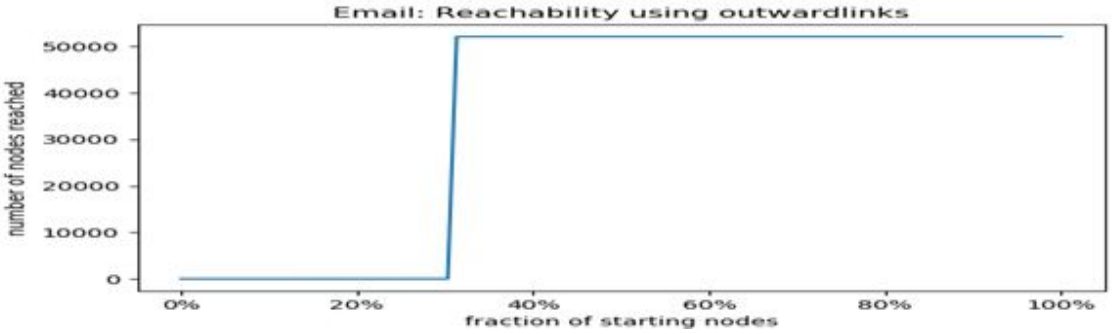
- The maximum no. of nodes reached for in-link BFS is larger than the one for out-link BFS.
- A high proportion of starting nodes can reach most nodes of the network under in-link BFS.
- A smaller proportion of starting nodes can reach the max number of nodes under out-link BFS.

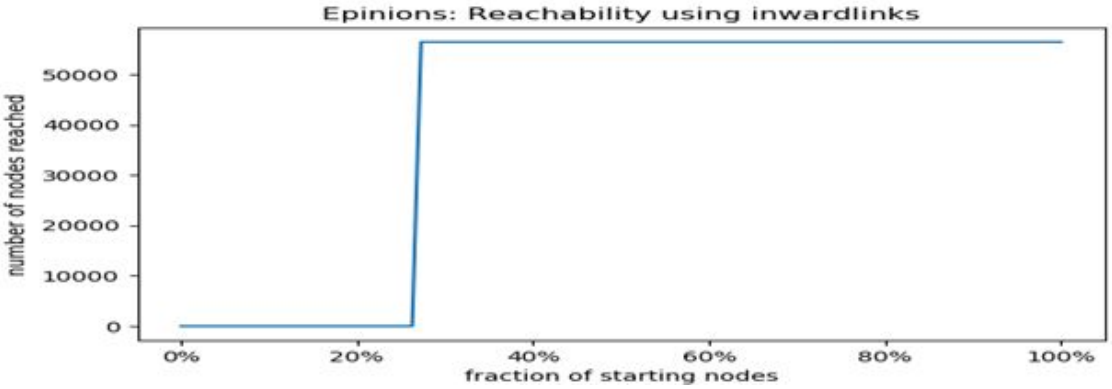
The graphs imply there is a large SCC.

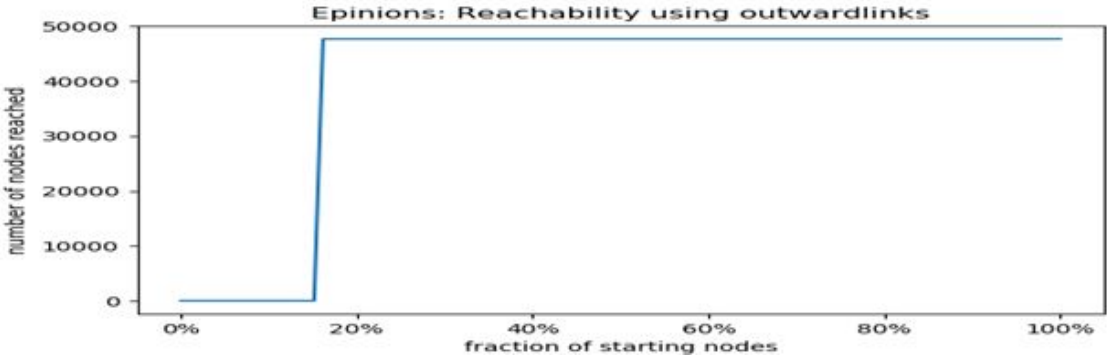
Email:
Large IN i.e because the max number of nodes, corresponding to IN+SCC reached is 175000, comparatively small SCC i.e because the fraction of nodes that can reach all the nodes in IN following in-link BFS is small. SCC+OUT consists of approximately 52000 nodes, and the SCC is larger than OUT since a large number of nodes were able to reach the max number of nodes for out-link BFS.
The plots are accepted with either absolute or log scale for the y-axis.

Graphs With Absolute Scale For Y-Axis:

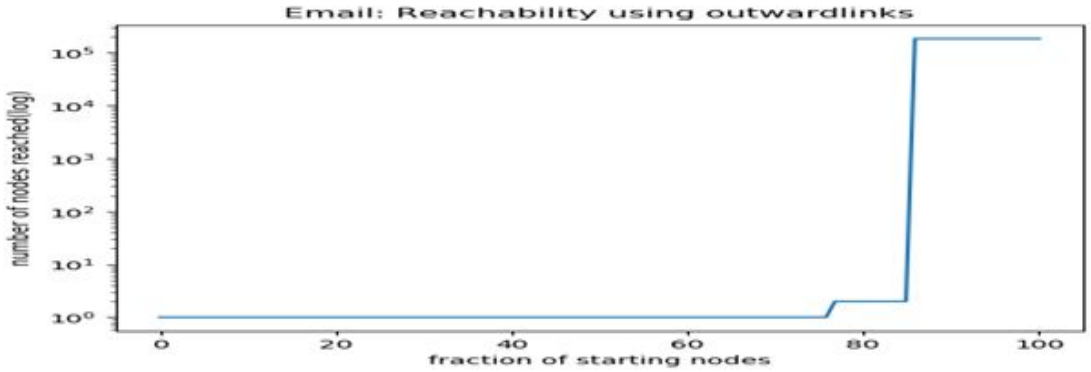


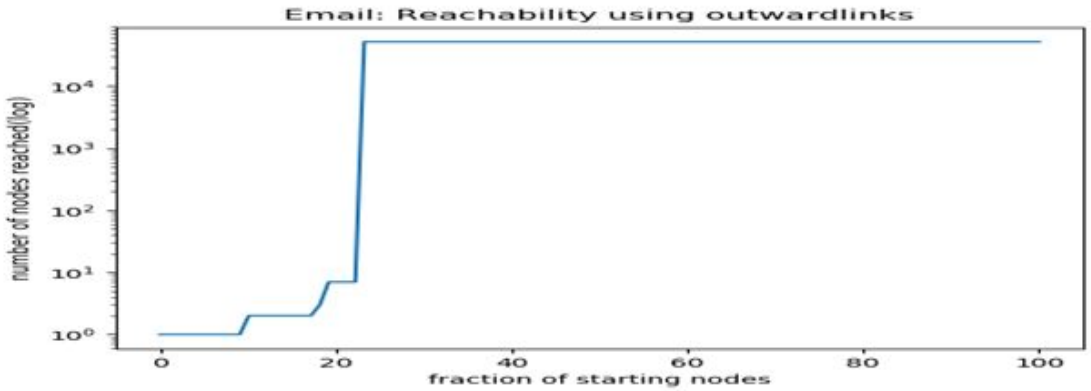

Email: Reachability using outwardlinks
number of nodes reached
50000
40000
30000
20000
10000
0
0%
20%
40%
60%
80%
100%
fraction of starting nodes

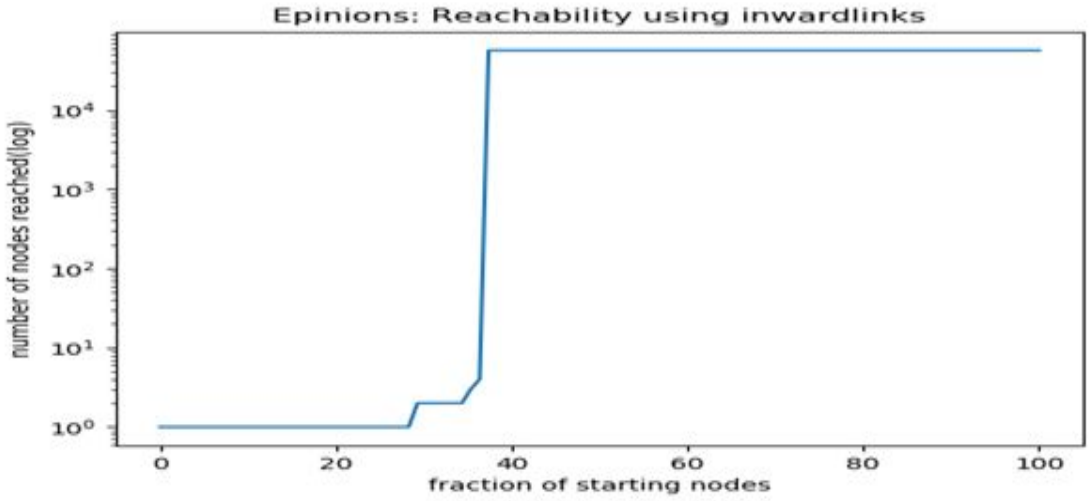

Epinions: Reachability using inwardlinks
number of nodes reached
50000
40000
30000
20000
10000
0
0%
20%
40%
60%
80%
100%
fraction of starting nodes

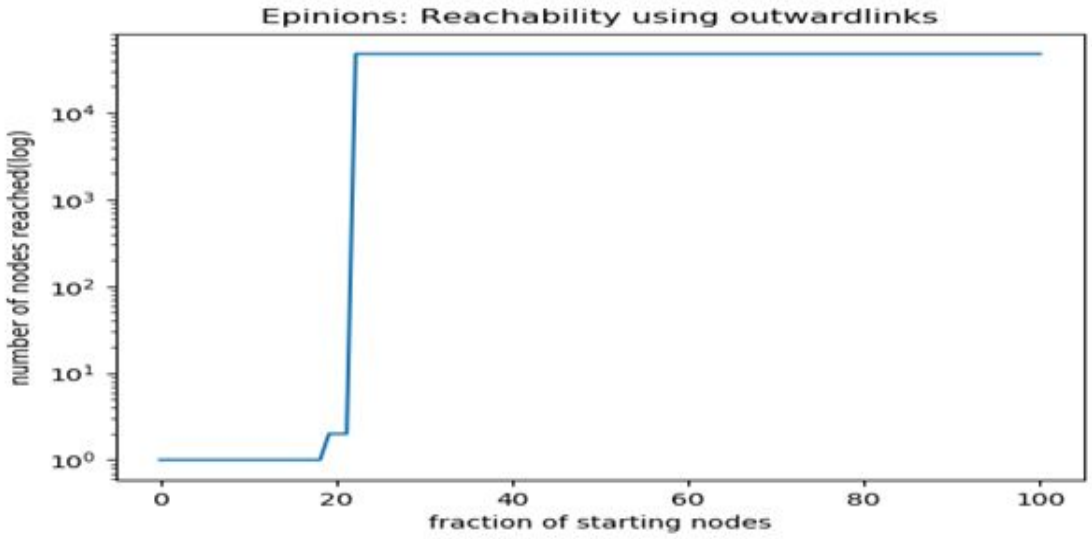

Epinions: Reachability using outwardlinks
number of nodes reached
50000
40000
30000
20000
10000
0
0%
20%
40%
60%
80%
100%
fraction of starting nodes

Graphs With Log Scale For Y-Axis:









Output for Q2_3:

For Emails

Size of SCC: 34203 nodes
Size of IN component: 151023 nodes
Size of OUT component: 17900 nodes
Size of TENDRILS: 21706 nodes
Size of DISCONNECTED: 40382 nodes

For Epinions

Size of SCC: 32223 nodes
Size of IN component: 24236 nodes
Size of OUT component: 15453 nodes
Size of TENDRILS: 3965 nodes
Size of DISCONNECTED: 2 nodes