NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. VIth Sem

Subject Code: CS-331

Subject Name: Social Network Analysis Lab

Submitted By:

Name : Subhojit Ghimire

Sch. Id. : 1912160

Branch : CSE – B

Under the Guidance of:

Dr. Anupam Biswas

Assistant Professor

Department of Computer Science and Engineering

LIST OF EXPERIMENTS

AIM I: TO ANALYZE THE NETWORK DISTRIBUTION BY INCORPORATING THE FOLLOWING CENTRALITY MEASURES AND MENTION THE SIGNIFICANCE OF EACH OF THE MEASURES.

CENTRALITY MEASURES: DEGREE CENTRALITY, BETWEENNESS CENTRALITY, EIGENVECTOR CENTRALITY.

DATASETS: KARATE CLUB NETWORK, FOOTBALL.

THEORY:

1. **Degree Centrality:** It finds the most central vertex in terms of number of connections. It can be calculated using the following formula:

$$C_D(v_i) = \frac{\deg(v_i)}{n-1}$$

where, n is the number of nodes in the network

2. **Betweenness Centrality:** It finds the vertices that appear maximum number of times in the shortest path between two vertices. It can be calculated using the following formula:

$$C'_B(v_i) = \frac{C_B(v_i)}{\frac{(n-1)(n-2)}{2}}$$

where, $C_B(v_i) = \sum_{v_s = v_i = v_t \in V,\ s<t} \frac{\sigma_{s,t}(v_i)}{\sigma_{s,t}}$ is node betweenness

and, n is the number of nodes in the network

3. **Eigenvector Centrality:** It finds the vertices that are connected to important vertices. It can be calculated using the following formula:

$$C_E(v_i) = \frac{1}{\lambda} \sum_{v_j \in V} A_{i,j} C_E(v_j)$$

where, $C_E(v_j)$ is the Eigenvector Centrality of node 'j'

and, λ is a constant such that Ax = λx; x is eigenvector of the adjacency matrix A.

CODE:

```
import matplotlib.pyplot as plt
import networkx as nx

import urllib.request
import io
import zipfile

# FOOTBALL CLUB
url = "http://www-personal.umich.edu/~mejn/netdata/football.zip"
sock = urllib.request.urlopen (url)
sockRead = io.BytesIO (sock.read ())
sock.close ()

zf = zipfile.ZipFile (sockRead)
txt = zf.read ("football.txt").decode ()
gml = zf.read ("football.gml").decode ()
gml = gml.split ("\n") [1:]

FC = nx.parse_gml (gml)

# KARATE CLUB
KC = nx.karate_club_graph ()

# Graph Node Setting
options = {
    "node_color": "red",
    "node_size": 50,
    "linewidths": 0,
    "width": 0.1,
}

# DEGREE CENTRALITY
CD_KC = nx.degree_centrality (KC) # Degree Centrality for Karate Club
print ("\nDEGREE CENTRALITY: KARATE CLUB\n")
print (['%s %0.2f'%(node, CD_KC [node]) for node in CD_KC])

CD_FC = nx.degree_centrality (FC) # Degree Centrality for Football Club
print ("\nDEGREE CENTRALITY: FOOTBALL CLUB\n")
print (['%s %0.2f'%(node, CD_FC [node]) for node in CD_FC])


# BETWEENNESS CENTRALITY
CB_KC = nx.betweenness_centrality (KC, normalized = True, endpoints = False) #
Betweenness Centrality for Karate Club
print ("\nBETWEENNESS CENTRALITY: KARATE CLUB\n")
print (['%s %0.2f'%(node, CB_KC [node]) for node in CB_KC])
```

```
CB_FC = nx.betweenness_centrality (FC, normalized = True, endpoints = False) #
Betweenness Centrality for Football Club
print ("\nBETWEENNESS CENTRALITY: FOOTBALL CLUB\n")
print (['%s %0.2f'%(node, CB_FC [node]) for node in CB_FC])

# EIGENVECTOR CENTRALITY
CE_KC = nx.eigenvector_centrality (KC) # Eigenvector Centrality for Karate
Club
print ("\nEIGENVECTOR CENTRALITY: KARATE CLUB\n")
print (['%s %0.2f'%(node, CE_KC [node]) for node in CE_KC])

CE_FC = nx.eigenvector_centrality(FC) # Eigenvector Centrality for Football
Club
print ("\nEIGENVECTOR CENTRALITY: FOOTBALL CLUB\n")
print (['%s %0.2f'%(node, CE_FC [node]) for node in CE_FC])

# NETWORK GRAPHS
plt.figure (figsize = (15, 15))
plt.title ("KARATE CLUB NETWORK")
nx.draw_networkx (KC, with_labels = True, **options)

plt.figure (figsize = (15, 15))
plt.title ("FOOTBALL CLUB NETWORK")
nx.draw_networkx (FC, with_labels = True, **options)

plt.show ()
```

OUTPUT:

```
PS C:\Users\subho> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab1.py"

DEGREE CENTRALITY: KARATE CLUB

['0 0.48', '1 0.27', '2 0.30', '3 0.18', '4 0.09', '5 0.12', '6 0.12', '7 0.12', '8 0.15', '9 0.06', '10 0.09', '11 0.03', '1
2 0.06', '13 0.15', '14 0.06', '15 0.06', '16 0.06', '17 0.06', '18 0.06', '19 0.09', '20 0.06', '21 0.06', '22 0.06', '23 0.
15', '24 0.09', '25 0.09', '26 0.06', '27 0.12', '28 0.09', '29 0.12', '30 0.12', '31 0.18', '32 0.36', '33 0.52']

DEGREE CENTRALITY: FOOTBALL CLUB

['BrighamYoung 0.11', 'FloridaState 0.11', 'Iowa 0.11', 'KansasState 0.11', 'NewMexico 0.10', 'TexasTech 0.11', 'PennState 0.
11', 'SouthernCalifornia 0.11', 'ArizonaState 0.10', 'SanDiegoState 0.10', 'Baylor 0.09', 'NorthTexas 0.09', 'NorthernIllinoi
s 0.09', 'Northwestern 0.10', 'WesternMichigan 0.09', 'Wisconsin 0.11', 'Wyoming 0.10', 'Auburn 0.10', 'Akron 0.10', 'Virgini
aTech 0.10', 'Alabama 0.10', 'UCLA 0.10', 'Arizona 0.10', 'Utah 0.10', 'ArkansasState 0.09', 'NorthCarolinaState 0.10', 'Ball
State 0.09', 'Florida 0.10', 'BoiseState 0.08', 'BostonCollege 0.10', 'WestVirginia 0.10', 'BowlingGreenState 0.10', 'Michiga
n 0.10', 'Virginia 0.09', 'Buffalo 0.10', 'Syracuse 0.10', 'CentralFlorida 0.07', 'GeorgiaTech 0.10', 'CentralMichigan 0.10',
 'Purdue 0.10', 'Colorado 0.10', 'ColoradoState 0.09', 'Connecticut 0.06', 'EasternMichigan 0.10', 'EastCarolina 0.10', 'Duke
 0.10', 'FresnoState 0.10', 'OhioState 0.10', 'Houston 0.10', 'Rice 0.10', 'Idaho 0.08', 'Washington 0.10', 'Kansas 0.09', 'S
outhernMethodist 0.11', 'Kent 0.09', 'Pittsburgh 0.10', 'Kentucky 0.09', 'Louisville 0.09', 'LouisianaTech 0.09', 'LouisianaM
onroe 0.07', 'Minnesota 0.10', 'MiamiOhio 0.10', 'Vanderbilt 0.10', 'MiddleTennesseeState 0.08', 'Illinois 0.10', 'Mississipp
iState 0.10', 'Memphis 0.10', 'Nevada 0.11', 'Oregon 0.10', 'NewMexicoState 0.10', 'SouthCarolina 0.10', 'Ohio 0.09', 'IowaSt
ate 0.10', 'SanJoseState 0.10', 'Nebraska 0.10', 'SouthernMississippi 0.09', 'Tennessee 0.10', 'Stanford 0.10', 'WashingtonSt
ate 0.10', 'Temple 0.10', 'Navy 0.10', 'TexasA&M 0.10', 'NotreDame 0.10', 'TexasElPaso 0.10', 'Oklahoma 0.10', 'Toledo 0.08',
 'Tulane 0.10', 'Mississippi 0.10', 'Tulsa 0.11', 'NorthCarolina 0.10', 'UtahState 0.08', 'Army 0.10', 'Cincinnati 0.10', 'Ai
rForce 0.09', 'Rutgers 0.09', 'Georgia 0.09', 'LouisianaState 0.09', 'LouisianaLafayette 0.07', 'Texas 0.10', 'Marshall 0.09'
, 'MichiganState 0.10', 'MiamiFlorida 0.09', 'Missouri 0.09', 'Clemson 0.09', 'NevadaLasVegas 0.11', 'WakeForest 0.09', 'Indi
ana 0.10', 'OklahomaState 0.09', 'OregonState 0.09', 'Maryland 0.10', 'TexasChristian 0.10', 'California 0.10', 'AlabamaBirmi
ngham 0.09', 'Arkansas 0.09', 'Hawaii 0.10']
```

Fig.: Degree Centralities for every node in Karate Club and Football Club Network

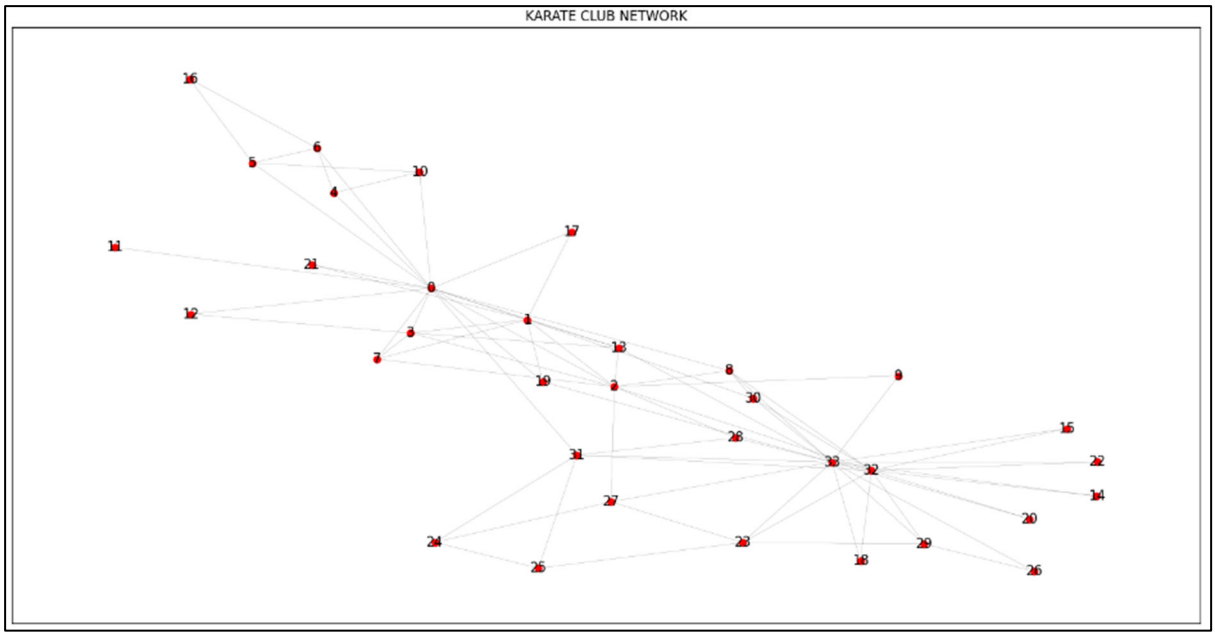
```
BETWEENNESS CENTRALITY: KARATE CLUB

['0 0.44', '1 0.05', '2 0.14', '3 0.01', '4 0.00', '5 0.03', '6 0.03', '7 0.00', '8 0.06', '9 0.00', '10 0.00', '11 0.00', '1
2 0.00', '13 0.05', '14 0.00', '15 0.00', '16 0.00', '17 0.00', '18 0.00', '19 0.03', '20 0.00', '21 0.00', '22 0.00', '23 0.
02', '24 0.00', '25 0.00', '26 0.00', '27 0.02', '28 0.00', '29 0.00', '30 0.01', '31 0.14', '32 0.15', '33 0.30']

BETWEENNESS CENTRALITY: FOOTBALL CLUB

['BrighamYoung 0.03', 'FloridaState 0.02', 'Iowa 0.01', 'KansasState 0.02', 'NewMexico 0.01', 'TexasTech 0.01', 'PennState 0.
02', 'SouthernCalifornia 0.01', 'ArizonaState 0.01', 'SanDiegoState 0.01', 'Baylor 0.01', 'NorthTexas 0.01', 'NorthernIllinoi
s 0.01', 'Northwestern 0.02', 'WesternMichigan 0.01', 'Wisconsin 0.02', 'Wyoming 0.02', 'Auburn 0.02', 'Akron 0.01', 'Virgini
aTech 0.01', 'Alabama 0.02', 'UCLA 0.02', 'Arizona 0.01', 'Utah 0.01', 'ArkansasState 0.02', 'NorthCarolinaState 0.02', 'Ball
State 0.02', 'Florida 0.01', 'BoiseState 0.02', 'BostonCollege 0.01', 'WestVirginia 0.02', 'BowlingGreenState 0.01', 'Michiga
n 0.01', 'Virginia 0.01', 'Buffalo 0.01', 'Syracuse 0.02', 'CentralFlorida 0.01', 'GeorgiaTech 0.01', 'CentralMichigan 0.03',
 'Purdue 0.01', 'Colorado 0.01', 'ColoradoState 0.01', 'Connecticut 0.01', 'EasternMichigan 0.01', 'EastCarolina 0.01', 'Duke
 0.01', 'FresnoState 0.01', 'OhioState 0.02', 'Houston 0.02', 'Rice 0.01', 'Idaho 0.01', 'Washington 0.01', 'Kansas 0.01', 'S
outhernMethodist 0.02', 'Kent 0.00', 'Pittsburgh 0.01', 'Kentucky 0.01', 'Louisville 0.01', 'LouisianaTech 0.03', 'LouisianaM
onroe 0.01', 'Minnesota 0.02', 'MiamiOhio 0.02', 'Vanderbilt 0.01', 'MiddleTennesseeState 0.02', 'Illinois 0.02', 'Mississipp
iState 0.01', 'Memphis 0.01', 'Nevada 0.01', 'Oregon 0.01', 'NewMexicoState 0.02', 'SouthCarolina 0.02', 'Ohio 0.01', 'IowaSt
ate 0.02', 'SanJoseState 0.01', 'Nebraska 0.01', 'SouthernMississippi 0.01', 'Tennessee 0.01', 'Stanford 0.01', 'WashingtonSt
ate 0.01', 'Temple 0.01', 'Navy 0.03', 'TexasA&M 0.01', 'NotreDame 0.03', 'TexasElPaso 0.01', 'Oklahoma 0.01', 'Toledo 0.01',
 'Tulane 0.01', 'Mississippi 0.01', 'Tulsa 0.02', 'NorthCarolina 0.02', 'UtahState 0.01', 'Army 0.02', 'Cincinnati 0.02', 'Air
Force 0.01', 'Rutgers 0.01', 'Georgia 0.01', 'LouisianaState 0.01', 'LouisianaLafayette 0.01', 'Texas 0.01', 'Marshall 0.01'
, 'MichiganState 0.01', 'MiamiFlorida 0.02', 'Missouri 0.02', 'Clemson 0.01', 'NevadaLasVegas 0.02', 'WakeForest 0.00', 'Indi
ana 0.02', 'OklahomaState 0.01', 'OregonState 0.00', 'Maryland 0.01', 'TexasChristian 0.01', 'California 0.01', 'AlabamaBirmi
ngham 0.01', 'Arkansas 0.01', 'Hawaii 0.01']
```

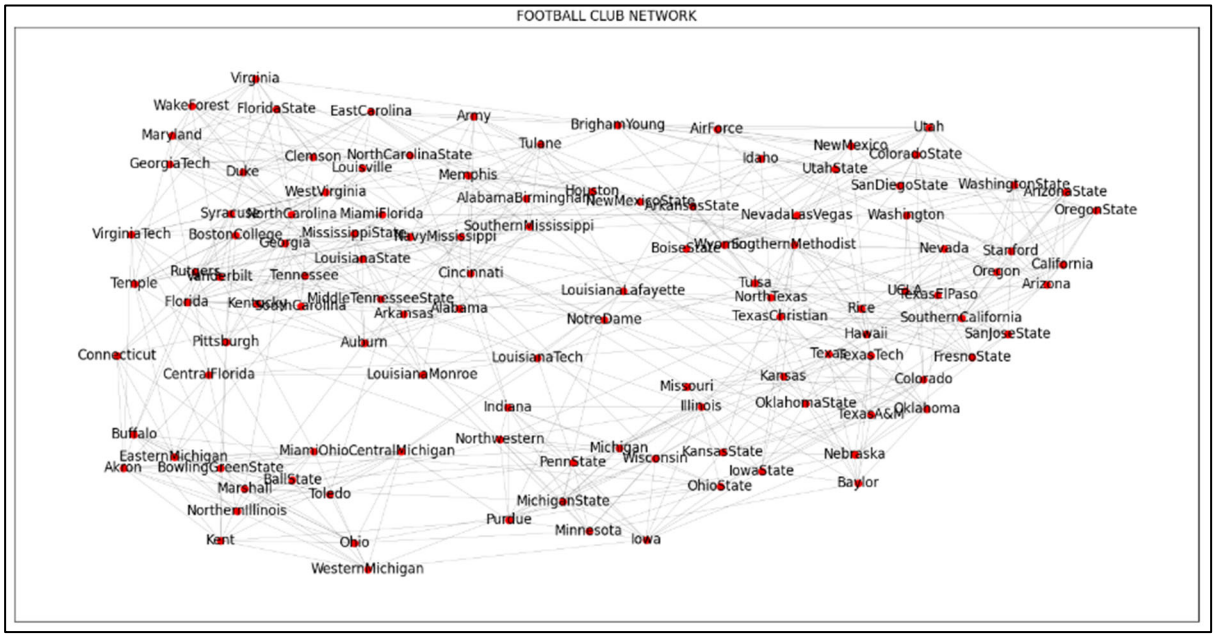
Fig.: Betweenness Centralities for every node in Karate Club and Football Club Network

```
EIGENVECTOR CENTRALITY: KARATE CLUB

['0 0.36', '1 0.27', '2 0.32', '3 0.21', '4 0.08', '5 0.08', '6 0.08', '7 0.17', '8 0.23', '9 0.10', '10 0.08', '11 0.05', '1
2 0.08', '13 0.23', '14 0.10', '15 0.10', '16 0.02', '17 0.09', '18 0.10', '19 0.15', '20 0.10', '21 0.09', '22 0.10', '23 0.
15', '24 0.06', '25 0.06', '26 0.08', '27 0.13', '28 0.13', '29 0.13', '30 0.17', '31 0.19', '32 0.31', '33 0.37']

EIGENVECTOR CENTRALITY: FOOTBALL CLUB

['BrighamYoung 0.11', 'FloridaState 0.10', 'Iowa 0.12', 'KansasState 0.11', 'NewMexico 0.10', 'TexasTech 0.10', 'PennState 0.
11', 'SouthernCalifornia 0.12', 'ArizonaState 0.11', 'SanDiegoState 0.11', 'Baylor 0.09', 'NorthTexas 0.08', 'NorthernIllinoi
s 0.07', 'Northwestern 0.11', 'WesternMichigan 0.08', 'Wisconsin 0.12', 'Wyoming 0.10', 'Auburn 0.08', 'Akron 0.08', 'Virgini
aTech 0.08', 'Alabama 0.08', 'UCLA 0.11', 'Arizona 0.11', 'Utah 0.10', 'ArkansasState 0.08', 'NorthCarolinaState 0.09', 'Ball
State 0.07', 'Florida 0.08', 'BoiseState 0.07', 'BostonCollege 0.09', 'WestVirginia 0.09', 'BowlingGreenState 0.08', 'Michiga
n 0.11', 'Virginia 0.08', 'Buffalo 0.08', 'Syracuse 0.09', 'CentralFlorida 0.06', 'GeorgiaTech 0.09', 'CentralMichigan 0.08',
 'Purdue 0.10', 'Colorado 0.10', 'ColoradoState 0.10', 'Connecticut 0.05', 'EasternMichigan 0.07', 'EastCarolina 0.09', 'Duke
 0.09', 'FresnoState 0.12', 'OhioState 0.11', 'Houston 0.09', 'Rice 0.12', 'Idaho 0.07', 'Washington 0.10', 'Kansas 0.09', 'S
outhernMethodist 0.12', 'Kent 0.08', 'Pittsburgh 0.09', 'Kentucky 0.08', 'Louisville 0.08', 'LouisianaTech 0.08', 'LouisianaM
onroe 0.05', 'Minnesota 0.10', 'MiamiOhio 0.08', 'Vanderbilt 0.08', 'MiddleTennesseeState 0.06', 'Illinois 0.11', 'Mississipp
iState 0.08', 'Memphis 0.08', 'Nevada 0.13', 'Oregon 0.11', 'NewMexicoState 0.09', 'SouthCarolina 0.08', 'Ohio 0.08', 'IowaSt
ate 0.10', 'SanJoseState 0.12', 'Nebraska 0.10', 'SouthernMississippi 0.08', 'Tennessee 0.08', 'Stanford 0.11', 'WashingtonSt
ate 0.10', 'Temple 0.09', 'Navy 0.09', 'TexasA&M 0.10', 'NotreDame 0.10', 'TexasElPaso 0.12', 'Oklahoma 0.10', 'Toledo 0.07',
 'Tulane 0.09', 'Mississippi 0.08', 'Tulsa 0.12', 'NorthCarolina 0.09', 'UtahState 0.08', 'Army 0.09', 'Cincinnati 0.09', 'Air
Force 0.09', 'Rutgers 0.08', 'Georgia 0.07', 'LouisianaState 0.07', 'LouisianaLafayette 0.06', 'Texas 0.10', 'Marshall 0.08'
, 'MichiganState 0.11', 'MiamiFlorida 0.08', 'Missouri 0.09', 'Clemson 0.08', 'NevadaLasVegas 0.11', 'WakeForest 0.08', 'Indi
ana 0.11', 'OklahomaState 0.09', 'OregonState 0.10', 'Maryland 0.09', 'TexasChristian 0.11', 'California 0.11', 'AlabamaBirmi
ngham 0.07', 'Arkansas 0.07', 'Hawaii 0.12']
```

Fig.: Eigenvector Centralities for every node in Karate Club and Football Club Network

NETWORK GRAPHS FOR KARATE CLUB AND FOOTBALL CLUB





AIM II: TO GENERATE THE FOLLOWING MENTIONED RANDOM GRAPHS AND ANALYSE THE PROS AND CONS IN THE CONTEXT OF LARGE-SCALE GRAPHS.

GRAPH SETS: **i. Erdos-Renyi Model (using network library)**

ii. Albert-Barabasi Model (using network library)

iii. LFR Graph (using LFR graph source code)

THEORY:

4. **Erdos-Renyi Model:** This model is used for generating random graphs. This model can be used in the probabilistic method to prove the existence of graphs satisfying various properties.

   The code of the graph which has been used as a function of the network library is:

   G = nx.erdos_renyi_graph (n, p, seed = None, directed = False)

   where, **n (int)** is the number of nodes to form up the network graph.
   p (float) is the probability for edge creation by each node. Its value is $0 \leq p \leq 1$.
   seed (int, optional) is the seed for random number generator. Default is None.
   directed (bool, optional) is the function that return a directed graph if set to True.

5. **Albert-Barabsai Model:** This model is used for generating random scale-free network with power-law degree distribution. This model tries to explain the existence of such nodes in real network which are thought to be approximately scale-free but contain few nodes with unusually high degree as compared to the other nodes of the network.

   The code of the graph which has been used as a function of the network library is:

   G = nx.barabasi_albert_graph (n, m, seed = None)

   where, **n (int)** is the number of nodes to form up the network graph.
   m (int) is the number of edges to attach from a new node to the existing nodes.
   seed (int, optional) is the seed for random number generator. Default is None.

6. **LFR** (**Lancichinetti–Fortunato–Radicchi benchmark) Graph:** The LFR algorithm is used to generate benchmark networks, i.e., artificial networks that resemble real-world networks. They have a priori known communities and are used to compare different community detection methods.

   The code of the graph which has been used as a function of the network library is:
   G = LFR_benchmark_graph (n, tau1, tau2, mu, average_degree = None, min_degree = None, max_degree = None, min_community = None, max_community = None, tol = 1e-07, max_iters = 500, seed=None)
   where, **n (int)** is the number of nodes in the created graph.
   tau1 (float) is the power law exponent for the degree distribution of the created graph. Its value should be tau1 > 1.
   tau2 (float) is the power law exponent for the community size distribution in the created graph. Its value should be tau2 > 1.
   mu (float) is the fraction of inter-community edges incident to each node. Its value is $0 \leq mu \leq 1$.

average_degree (float) is the desired average degree of nodes in the created graph. Its value is 0 ≤ average_degree ≤ n.
min_degree (int) is the minimum degree of nodes in the created graph. Its value is 0 ≤ min_degree ≤ n.
max_degree (int) is the maximum degree of nodes in the created graph. Its value is 0 ≤ min_degree < max_degree ≤ n. By default, its value is set to n.
min_community (int) is the minimum size of the communities in the graph. If not specified, this is set to min_degree.
max_community (int) is the maximum size of communities in the graph. If not specified, this is set to n.
tol (float) is the tolerance when comparing floats, specifically when comparing average degree values.
max_iters (int) is the maximum number of iterations to try to create the community sizes, degree distribution and community affiliations.
seed (int, optional) is the seed for random number generation state. Default is None.

The significance of these models is that these models generate random network graph for study purpose.

In Erdos-Renyi network, we assign N nodes and then connect each pair with probability p. This means that no one node will have much higher degree than any other. The probability for a link between any two nodes is the same. It is used to determine weather the links in a real-world network are formed due to random interactions of nodes or due to the preference of nodes to communicate or attach to certain nodes.

In Barabsai-Albert network, we assign N nodes, but to create them, we first start with a small set of connected nodes. Then we add nodes one at a time till we get N nodes. When we add a node, we connect it to a small number of existing nodes with probability proportional to the degree of the existing node. As a result, nodes with higher degree (the earlier nodes) tend to get even higher degree.

The advantage of Lancichinetti–Fortunato–Radicchi benchmark over other methods is that it accounts for the heterogeneity in the distributions of node degrees and of community sizes. They have a priori known communities and are used to compare different community detection methods.

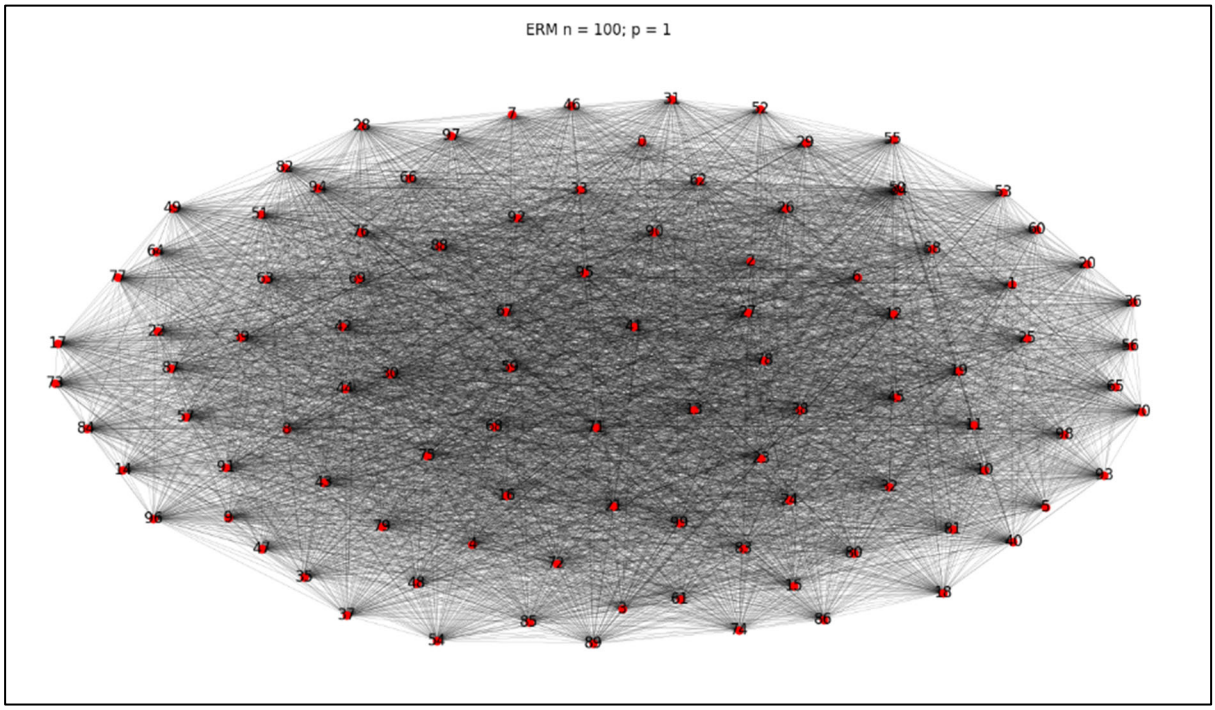
CODE:

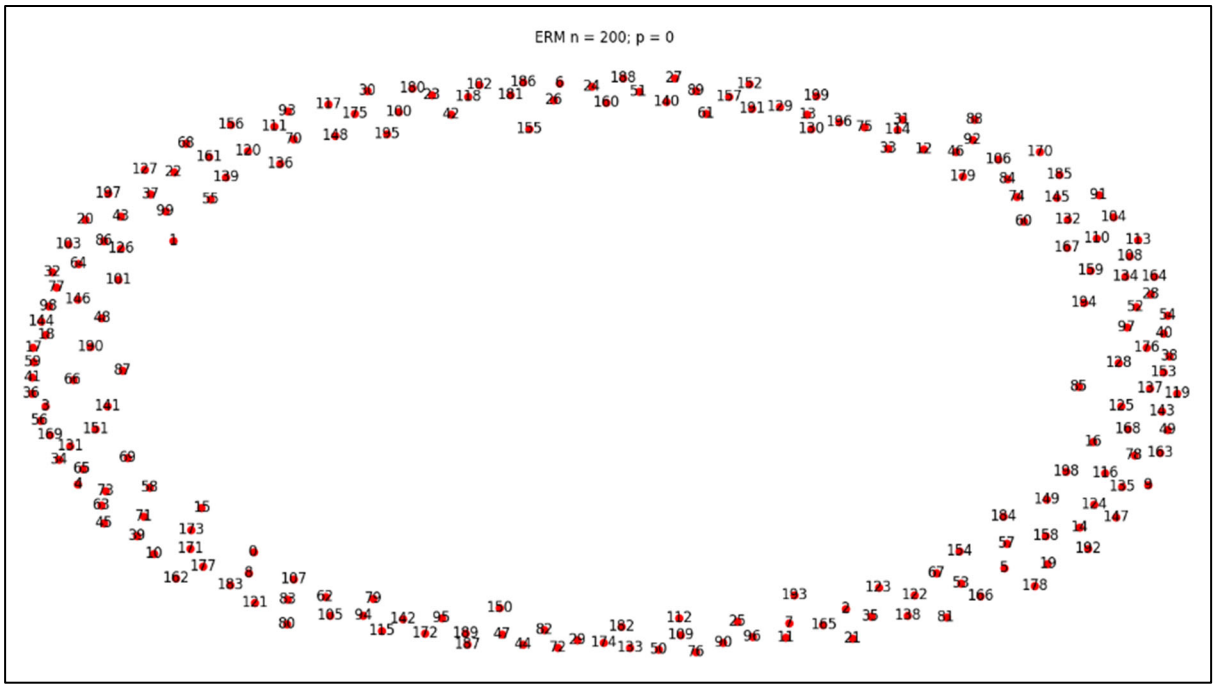
```
import matplotlib.pyplot as plt
import networkx as nx
from networkx.algorithms.community import LFR_benchmark_graph

# Graph Node Setting
options = {
    "node_color": "red",
    "node_size": 50,
    "linewidths": 0,
    "width": 0.1,
}

# ERDOS-RENYI MODEL
ERM_1 = nx.erdos_renyi_graph (100, 0)
ERM_2 = nx.erdos_renyi_graph (200, 0.02)
ERM_3 = nx.erdos_renyi_graph (300, 0.2)
ERM_4 = nx.erdos_renyi_graph (400, 1)

plt.figure ()
plt.title ("ERM n = 100; p = 0")
nx.draw (ERM_1, with_labels = True, **options)

plt.figure ()
plt.title ("ERM n = 200; p = 0.02")
nx.draw (ERM_2, with_labels = True, **options)

plt.figure ()
plt.title ("ERM n = 300; p = 0.2")
nx.draw (ERM_3, with_labels = True, **options)

plt.figure ()
plt.title ("ERM n = 400; p = 1")
nx.draw (ERM_4, with_labels = True, **options)

plt.show ()

# ALBERT-BARABASI MODEL
ABM_1 = nx.barabasi_albert_graph (100, 15)
ABM_2 = nx.barabasi_albert_graph (200, 5)
ABM_3 = nx.barabasi_albert_graph (300, 20)
ABM_4 = nx.barabasi_albert_graph (400, 10)

plt.figure ()
plt.title ("ABM n = 100; m = 20")
nx.draw (ABM_1, with_labels = True, **options)
```

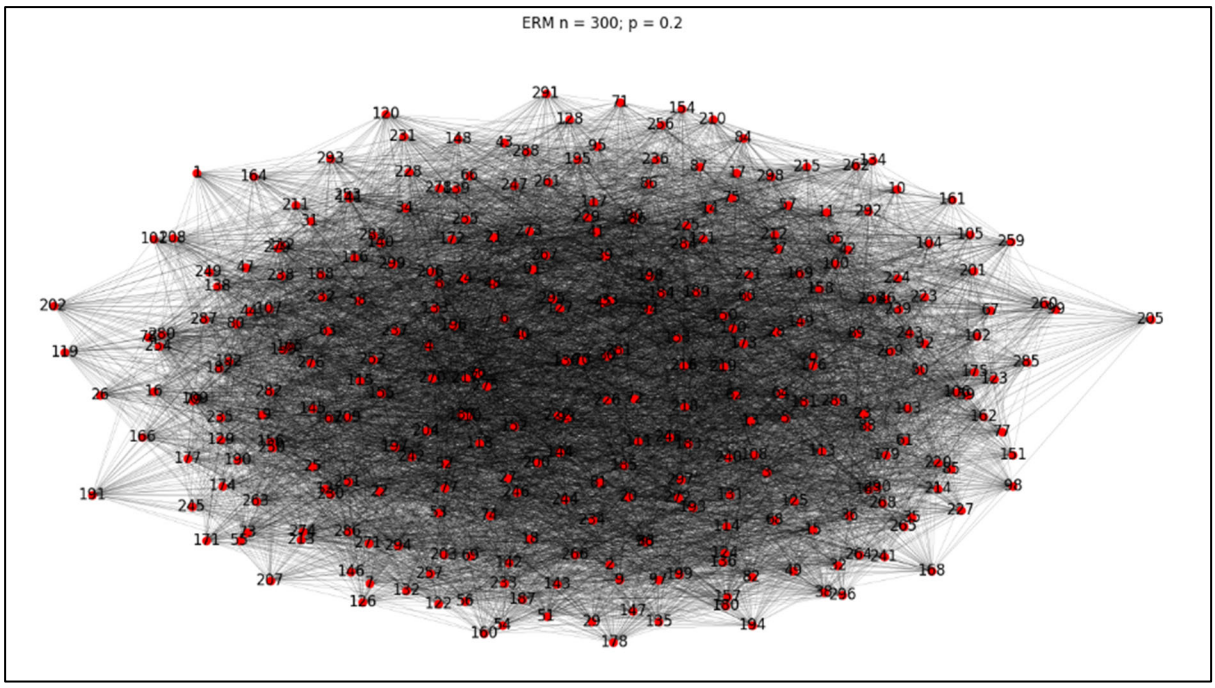
```
plt.figure ()
plt.title ("ABM n = 200; m = 15")
nx.draw (ABM_2, with_labels = True, **options)

plt.figure ()
plt.title ("ABM n = 300; m = 10")
nx.draw (ABM_3, with_labels = True, **options)

plt.figure ()
plt.title ("ABM n = 400; m = 5")
nx.draw (ABM_4, with_labels = True, **options)

plt.show ()

# LANCICHINETTI-FORTUNATO-RADICCHI BENCHMARK
LFR_1 = LFR_benchmark_graph (n = 250, tau1 = 3, tau2 = 1.5, mu = 0.1,
average_degree = 5, min_community = 20, seed = 10)
LFR_2 = LFR_benchmark_graph (n = 250, tau1 = 2, tau2 = 1.1, mu = 0.1,
min_degree = 20, max_degree = 70, max_iters = 1000, seed = 10)
LFR_3 = LFR_benchmark_graph (n = 350, tau1 = 2, tau2 = 1.1, mu = 0.1,
min_degree = 20, max_degree = 50, max_iters = 5000, seed = 20)
LFR_4 = LFR_benchmark_graph (n = 450, tau1 = 2, tau2 = 1.1, mu = 0.1,
min_degree = 20, max_degree = 50, max_iters = 2500, seed = 10)

plt.figure ()
plt.title ("LFR n = 250; tau1 = 3; tau2 = 1.5; mu = 0.1; average_degree = 5;
min_community = 20; seed = 10")
nx.draw (LFR_1, with_labels = True, **options)

plt.figure ()
plt.title ("LFR n = 250; tau1 = 2; tau2 = 1.1; mu = 0.1; min_degree = 20;
max_degree = 50; max_iters = 1000; seed = 10")
nx.draw (LFR_2, with_labels = True, **options)

plt.figure ()
plt.title ("LFR n = 350; tau1 = 2; tau2 = 1.1; mu = 0.1; min_degree = 20;
max_degree = 50; max_iters = 5000; seed = 20")
nx.draw (LFR_3, with_labels = True, **options)

plt.figure ()
plt.title ("LFR n = 450; tau1 = 2; tau2 = 1.1; mu = 0.1; min_degree = 20;
max_degree = 50; max_iters = 2500; seed = 10")
nx.draw (LFR_4, with_labels = True, **options)

plt.show ()
```

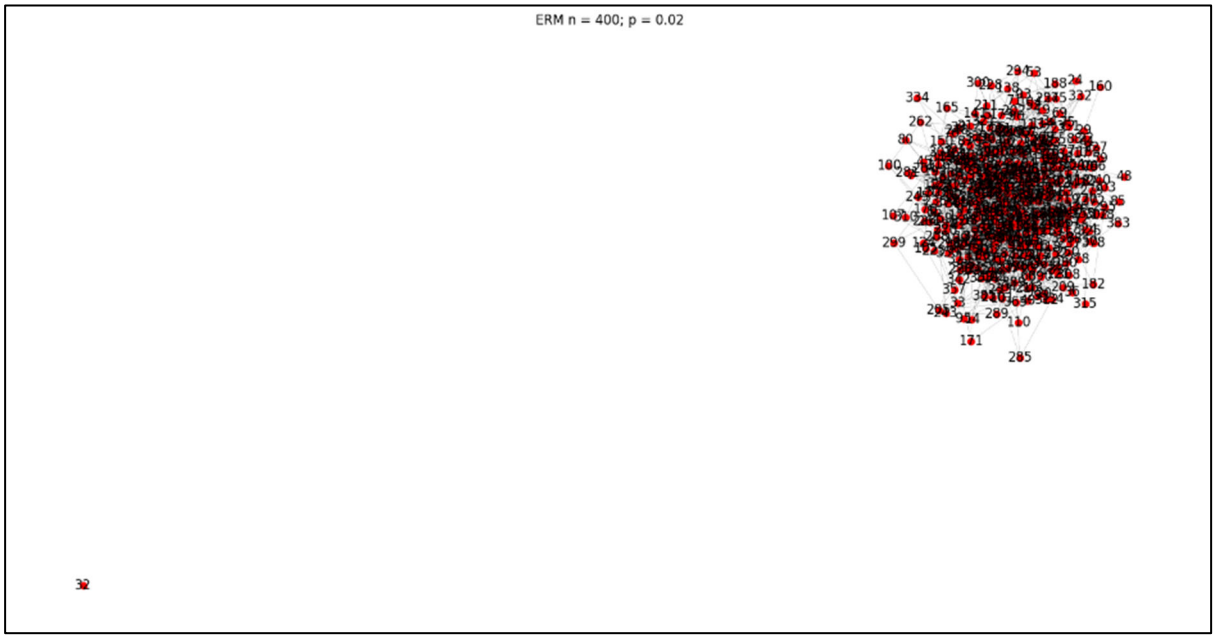
OUTPUT AND OBSERVATIONS:

ERDOS-RENYI MODEL







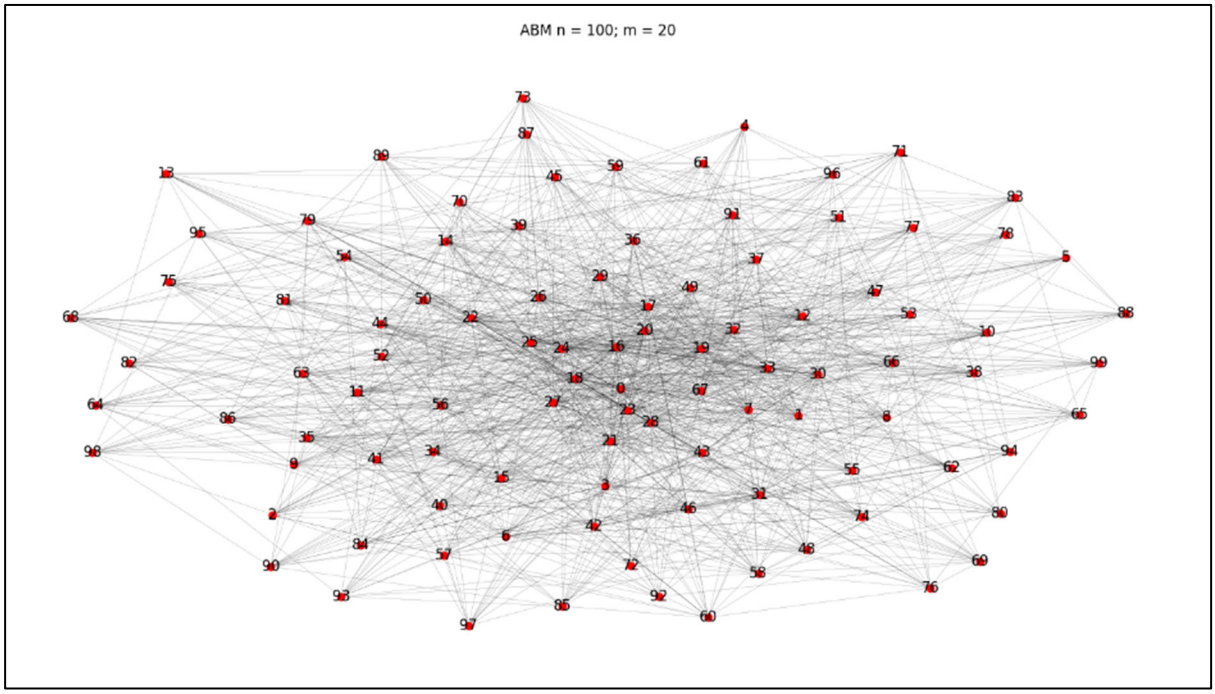


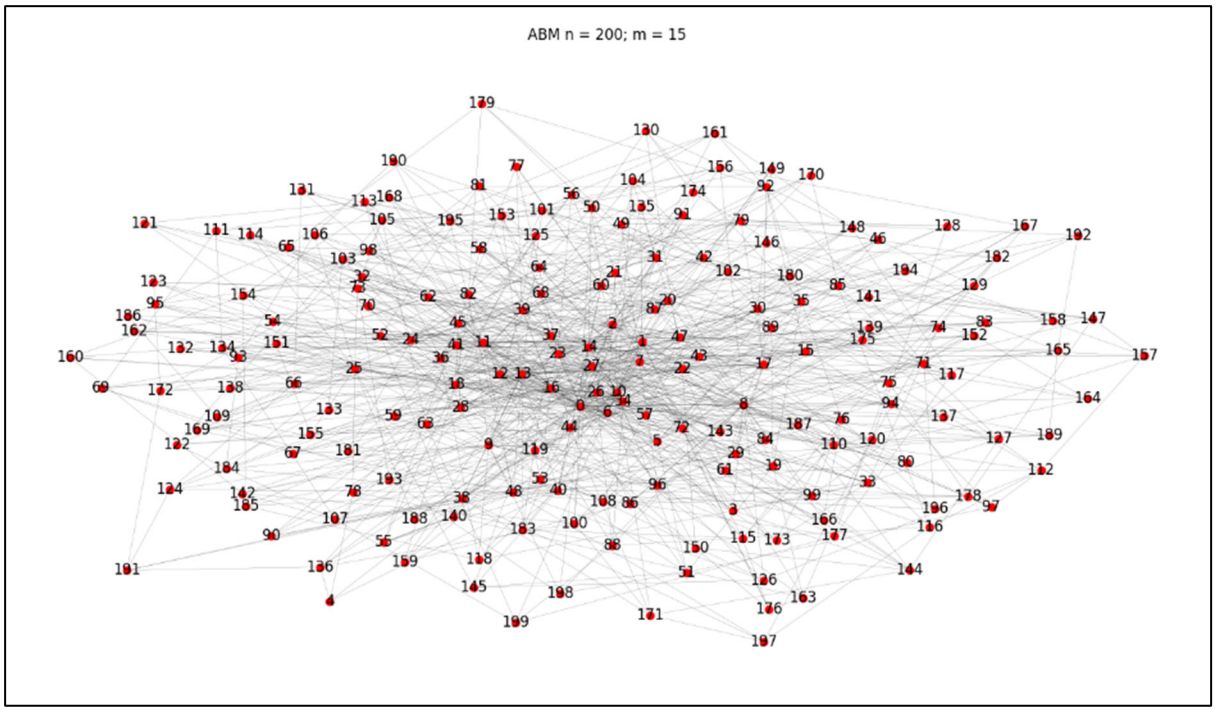
Observations:

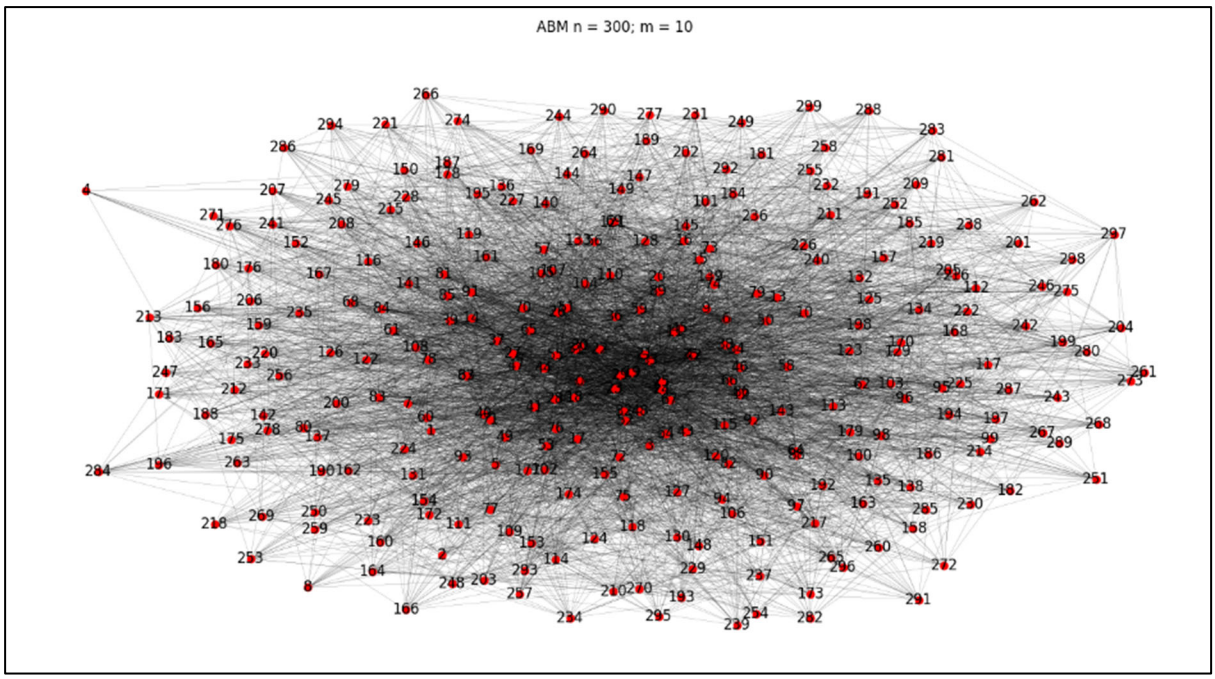
1. It resembles small-world network, i.e., it has more highly connected nodes as compared to that predicted with random model.

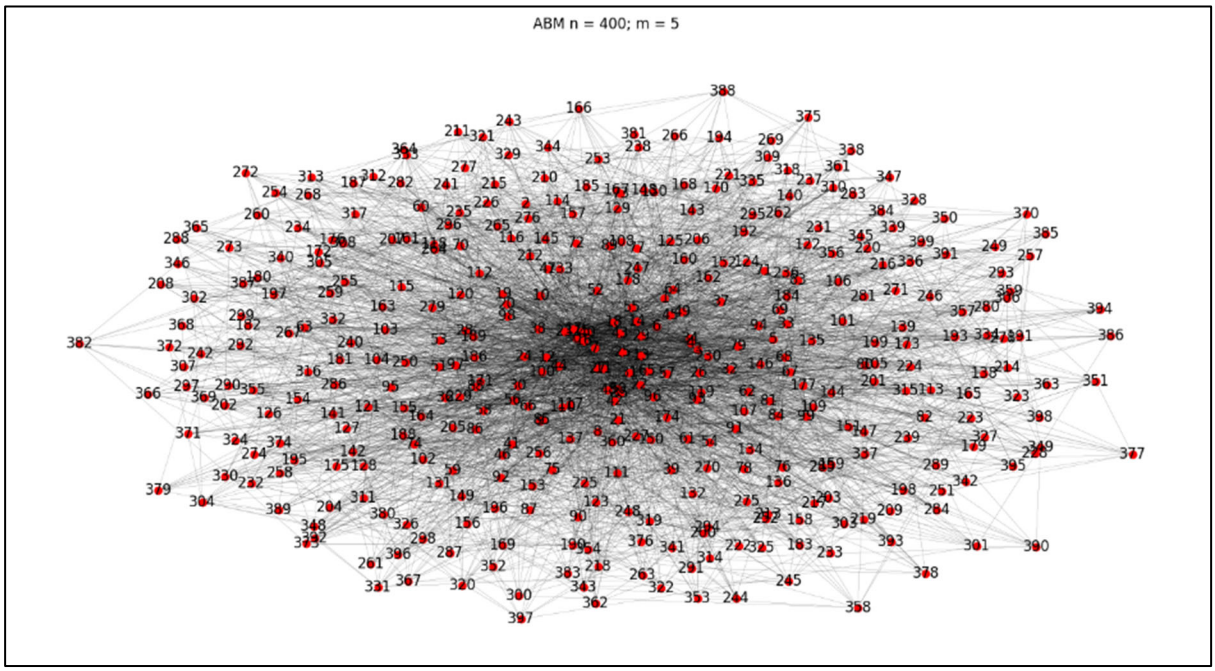
2. There is a possibility that a node is not connected to any other nodes, and also the possibility exists such that a node is connected to every other node in the network. This possibility increases and decreases as we vary the values for number of nodes and the probability of edge creation.

ALBERT-BARABASI MODEL







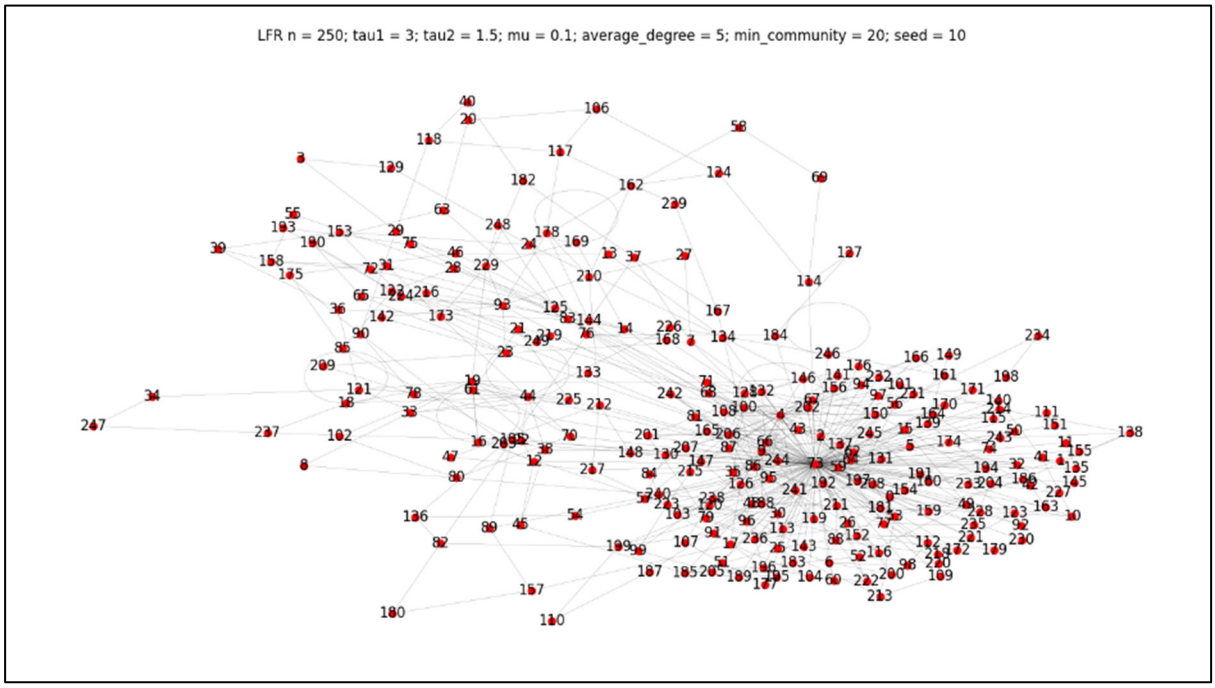


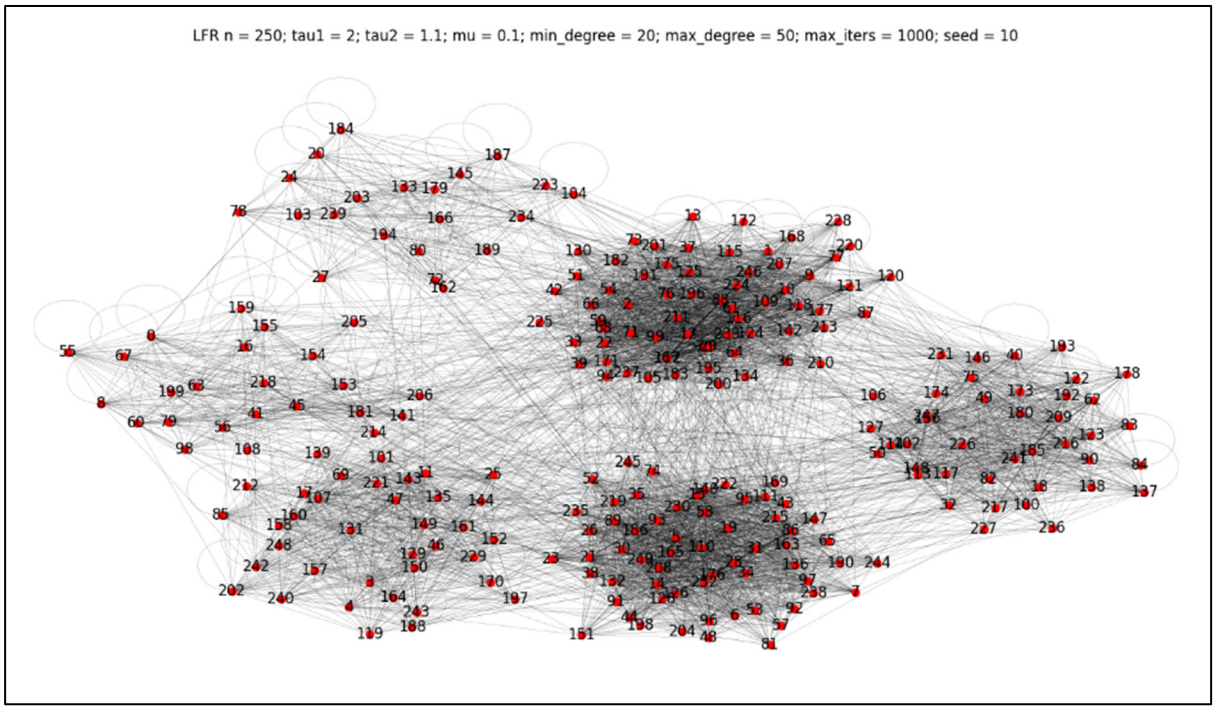
Observations:

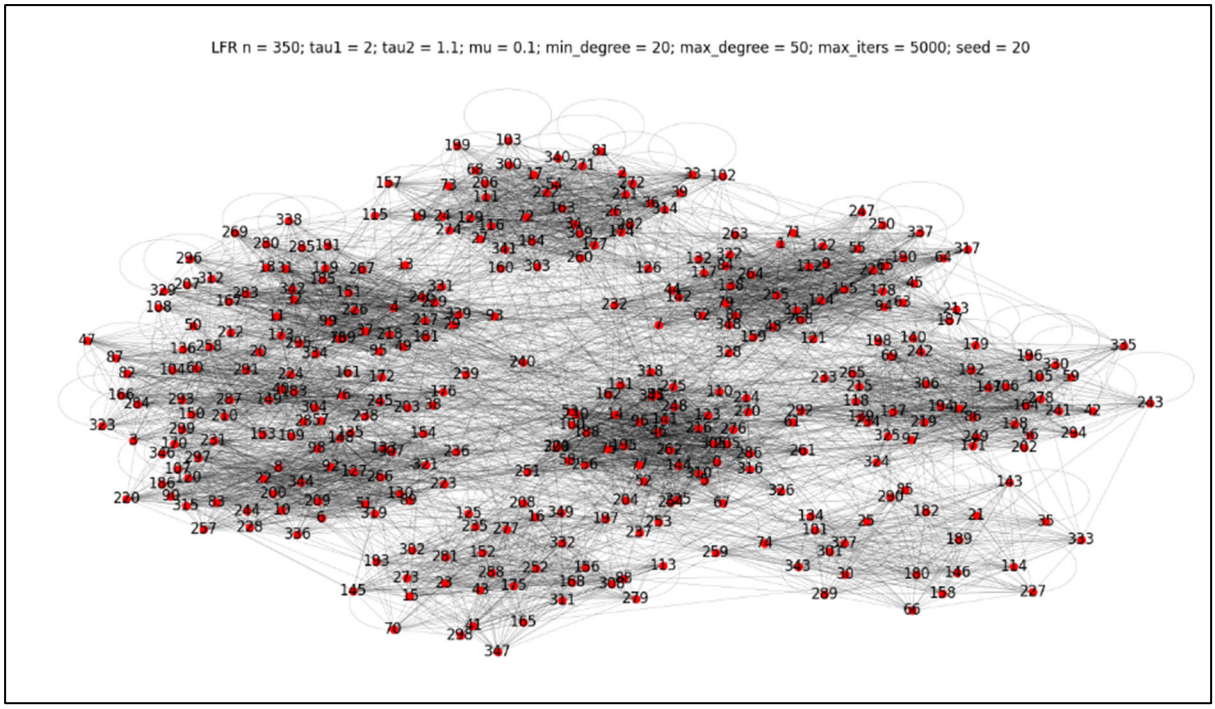
1. The degree of each node increases following the power law.

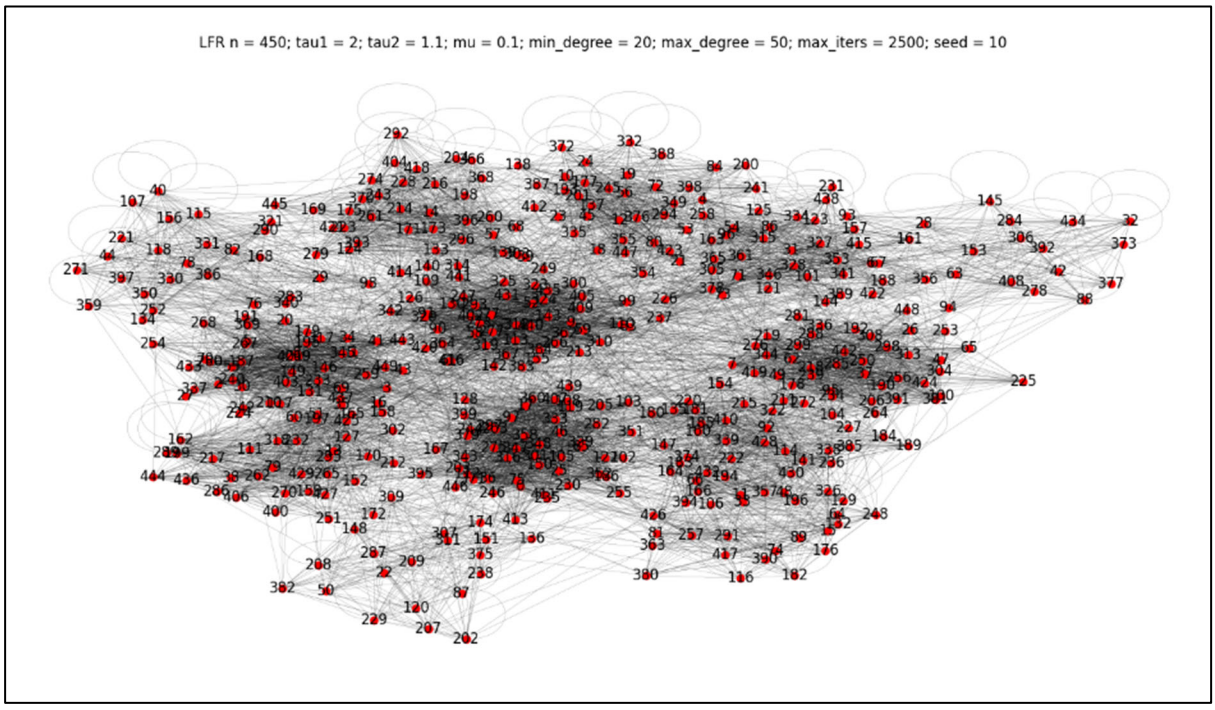
2. Each new node has more nodes to link than the previous nodes. In other words, with time, each node competes for links with an increasing pool of nodes.

LANCICHINETTI-FORTUNATO-RADICCHI BENCHMARK









Observations:

1. It resembles the real-world network, i.e., smaller communities distributed all over the place but connected with each other to form a network.

AIM III: USE THE GEPHI TOOL TO DO THE FOLLOWING OPERATIONS:

INSTALL PLUGIN WHEREVER REQUIRED AND USE THE FOLLOWING LAYOUTS TO SET THE GRAPH SHAPE AND SUMMARIZE THE SIGNIFICANCE OF EACH LAYOUT.

- **LAYOUT:** CIRCULAR LAYOUT, GRAPHVIZ.
- **COMPUTE THE FOLLOWING USING STATISTICS AND METRICS FRAMEWORK OF THE GEPHI TOOL:** AVERAGE DEGREE, GRAPH DENSITY, MODULARITY, CLUSTERING COEFFICIENT, PAGERANK, DBSCAN, GIRVAN-NEWMAN ALGORITHM, LEIDEN ALGORITHM, AVERAGE CLUSTERING COEFFICIENT, EIGENVECTOR CENTRALITY AND AVERAGE PATH LENGTH.
- SHOW THE USAGE (IMPORTING AND EXPORTING) OF THE GRAPH FORMATS SUCH AS GEXF, CSV, GML AND GRAPHML.

THEORY:

1. **Layout:** Layout algorithms give the shape to the graph. Gephi provides state-of-the-art algorithms layout algorithms, both for efficiency and quality.
2. **Circular Layout:** It draws nodes in a circle ordered by ID, a metric (degree, betweenness centrality, etc.) or by an attribute. It is used to show a distribution of nodes with their links.
3. **Graphviz Layout:** Graphviz consists of a graph description language named the DOT language and a set of tools that can generate and/or process DOT files.
4. **DOT:** It is a command-line tool to produce layered drawings of directed graphs in a variety of output formats, such as (PostScript, PDF, SVG, annotated text and so on).

IMPORTING/EXPORTING GRAPH FORMATS:

Graph representation file formats like GEXF, CSV, GML and GraphML, along with various other unmentioned graph formats achieve a single objective: storing dataset in the form of nodes and edges. There is no standardised graph format as each graph formats have their own speciality and useful over the other in some specific way. Some format can handle dataset and vertices in billions while the others cannot. Some are dedicated large-scale graph analytics frameworks, while the others are small-scale graph analytics frameworks.

GEFX: Graph Exchange XML Format is a language for describing complex network structures, their associated data and dynamics. It is an XML-based file format for storing undirected or directed graph.

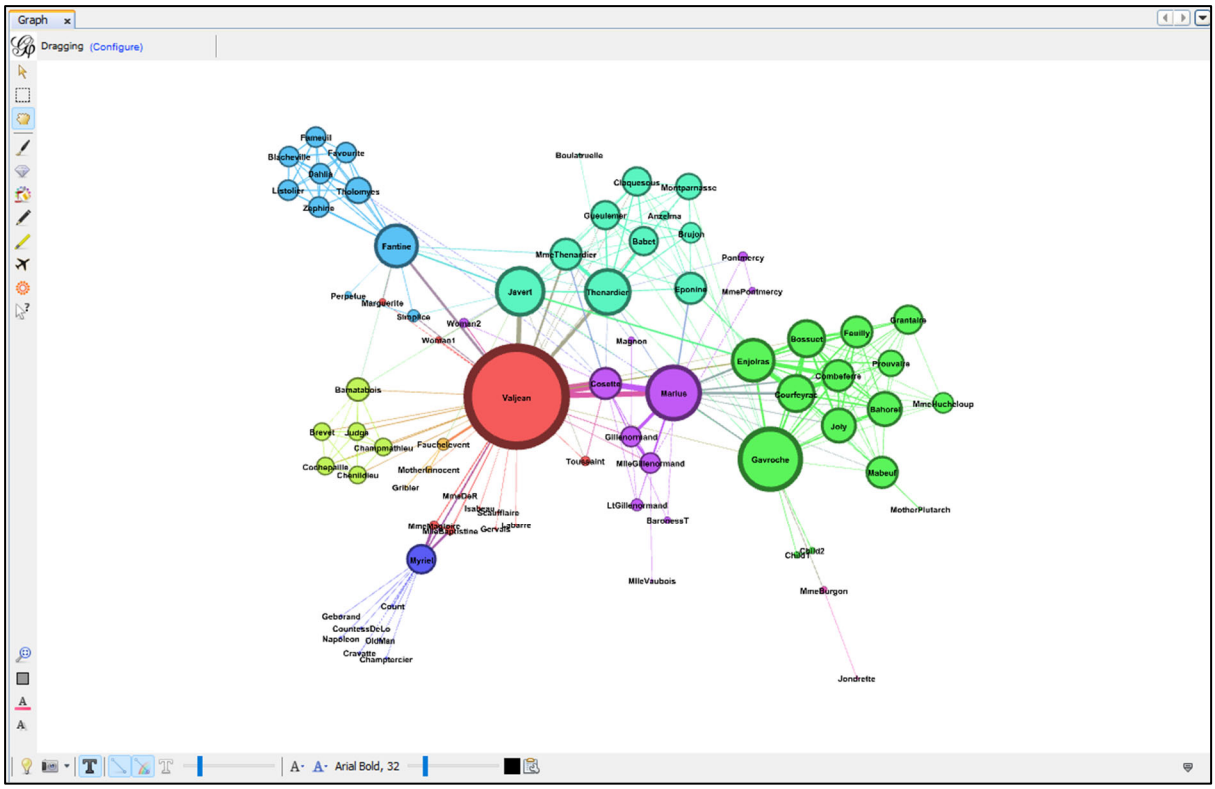
CSV: Comma Separated Value stores input data that is separated by commas. Standard graph data can be exported as .csv format. By default, graphs imported from CSV are directed graphs, but the user can select undirected in the import report dialog.

GML: Graph Modelling Language is a text file format supporting network data with a very easy syntax. Like GEFX, GML is an XML dialect. It supports the entire range of possible graph structure constellations, i.e., it can preserve all relevant hierarchical information like any inner-edges from the hierarchy, the structure of any inner graphs, and whether a node containing an inner graph is a group node or a folder node.

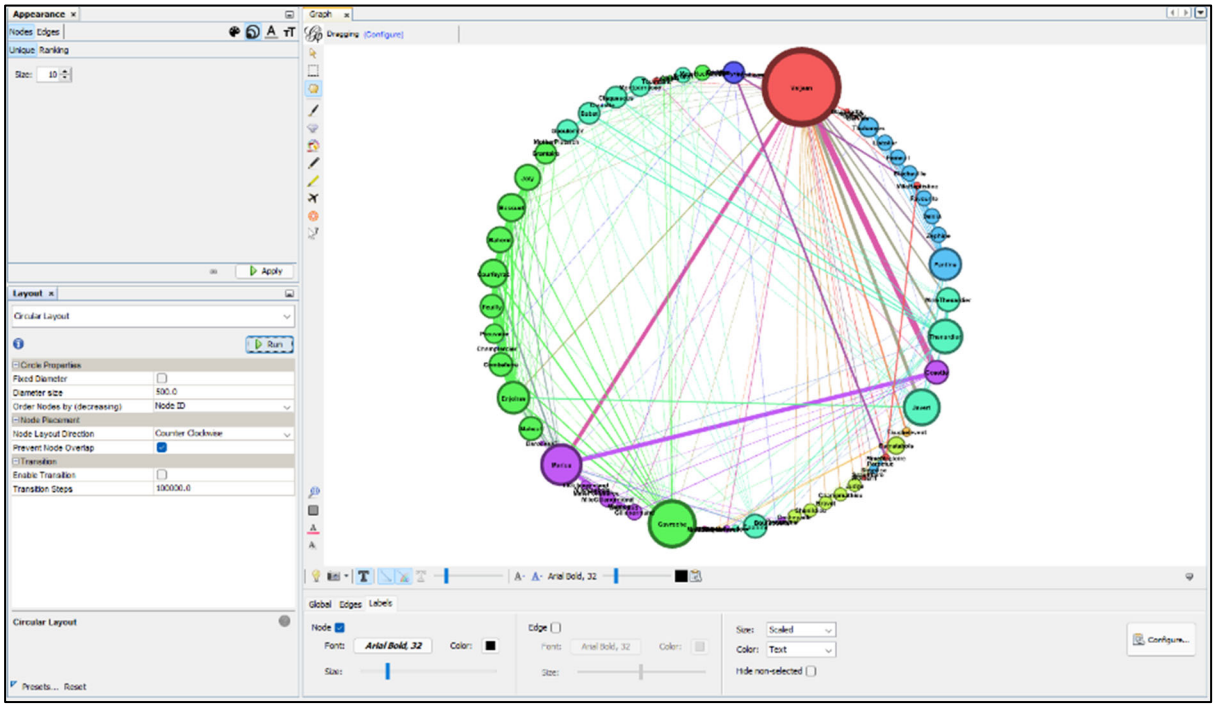
GraphML: It uses an XML-based syntax and supports the entire range of possible graph structure constellations including directed, undirected, mixed graphs, hypergraphs, hierarchical graphs, graphical representations, references to external data and application-specific attributes.

SAMPLE GRAPH FILES USED IN THIS PROJECT: “Les Miserables.gefx”, “Power Grid.gml”, “Airlines.graphml”, “Zachary’s Karate Club.csv”, “Jazz Musicians Network.net”, “Coauthorships in Network Science.gml” and a Gephi Generated Random Graph.

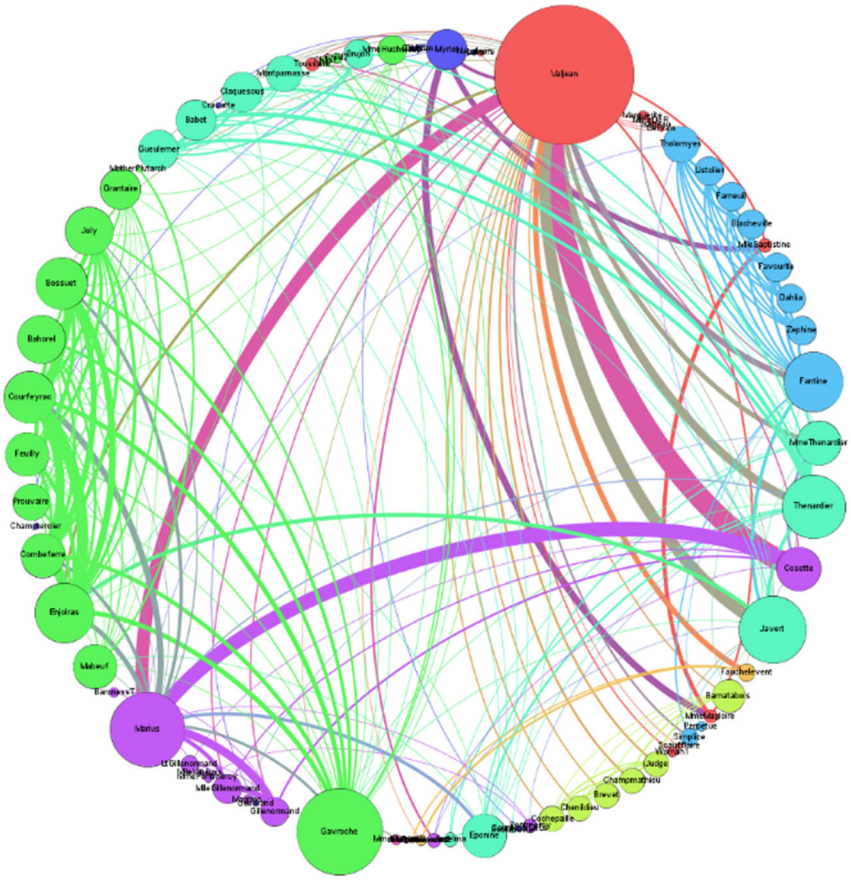
GRAPH FILE NAME: “Les Miserables.gexf”



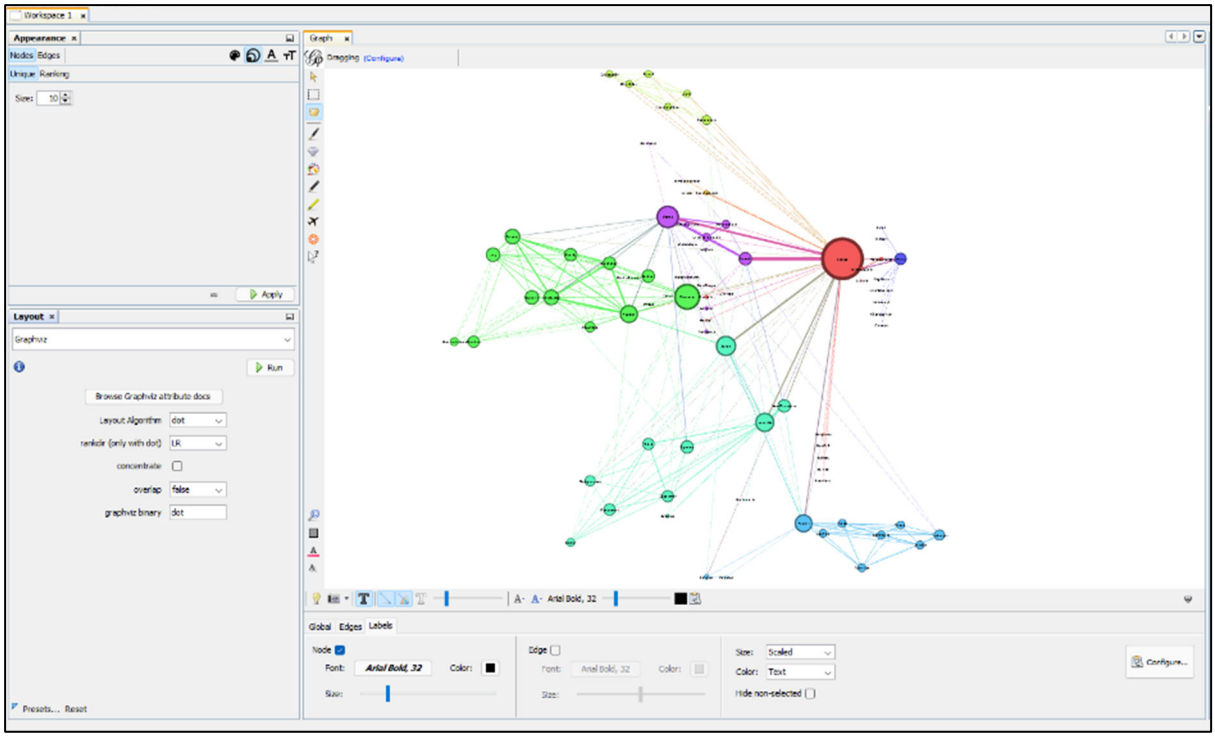
CIRCULAR LAYOUT: Les Miserables



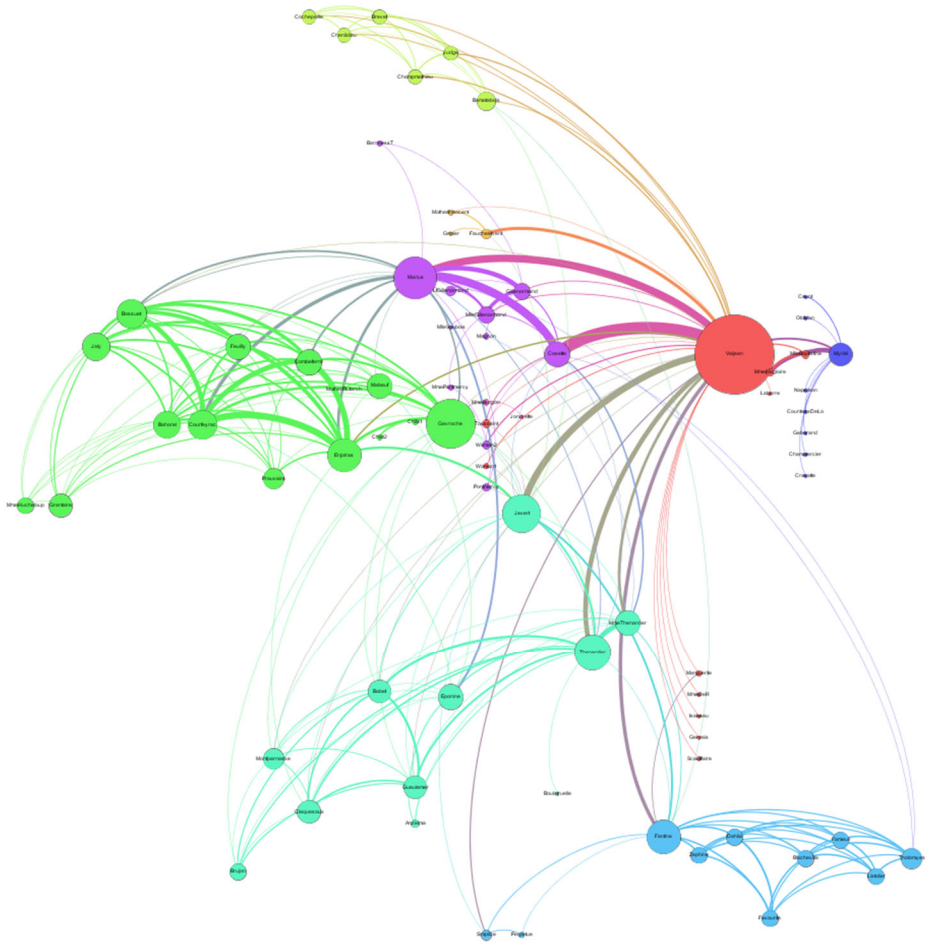
PREVIEW: CIRCULAR LAYOUT: Les Miserables (Exported as png)



GRAPHVIZ LAYOUT: Les Miserables



PREVIEW: GRAPHVIZ LAYOUT: Les Miserables (Exported as png)



STATISCTICS: Les Miserables

Average Degree: 6.597

Graph Density (Undirected): 0.087

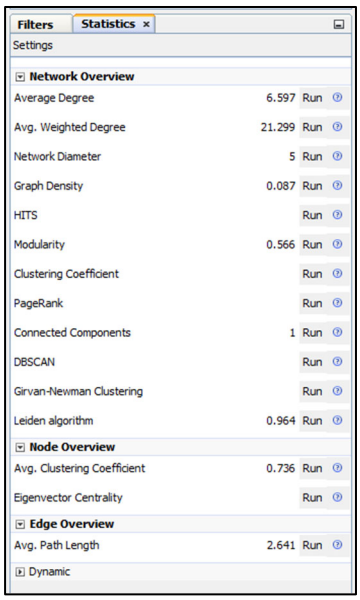
Modularity: 0.557

Clustering Coefficient (Triangle Method): 0.499

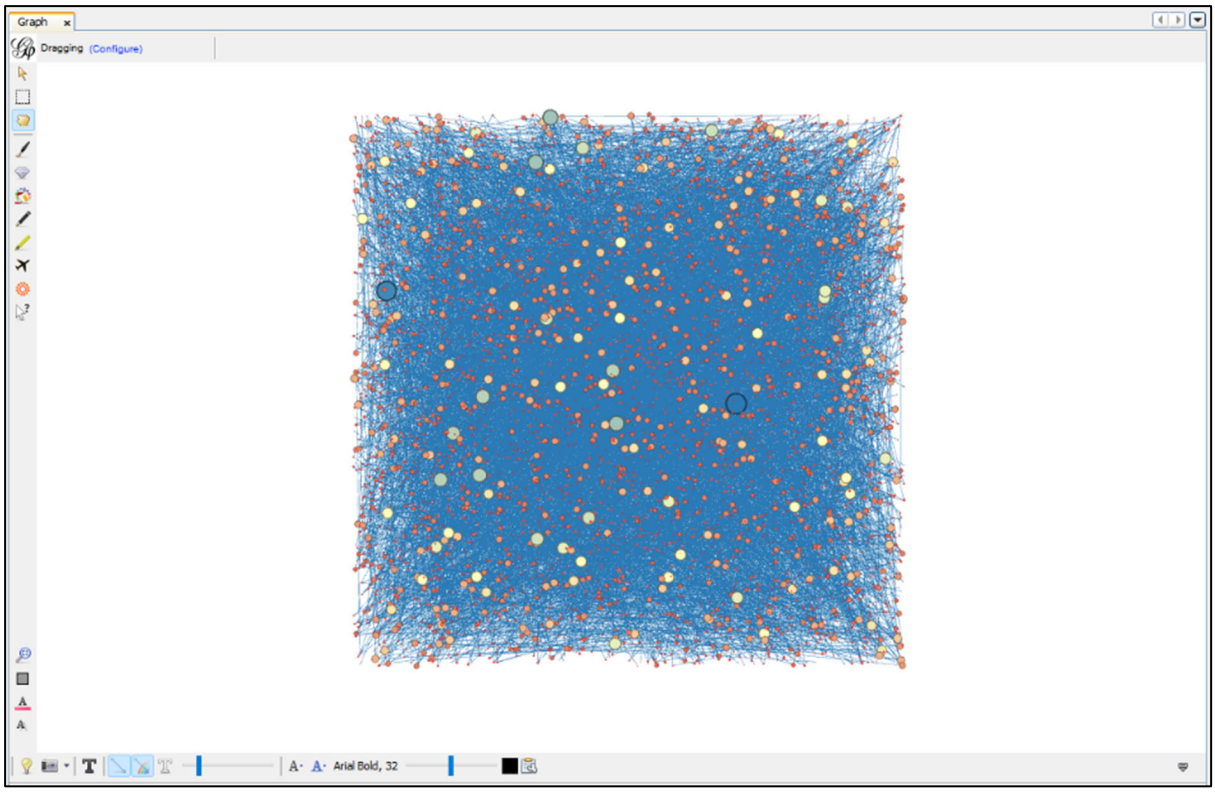
Leiden Algorithm: 0.964

Average Clustering Coefficient: 0.736

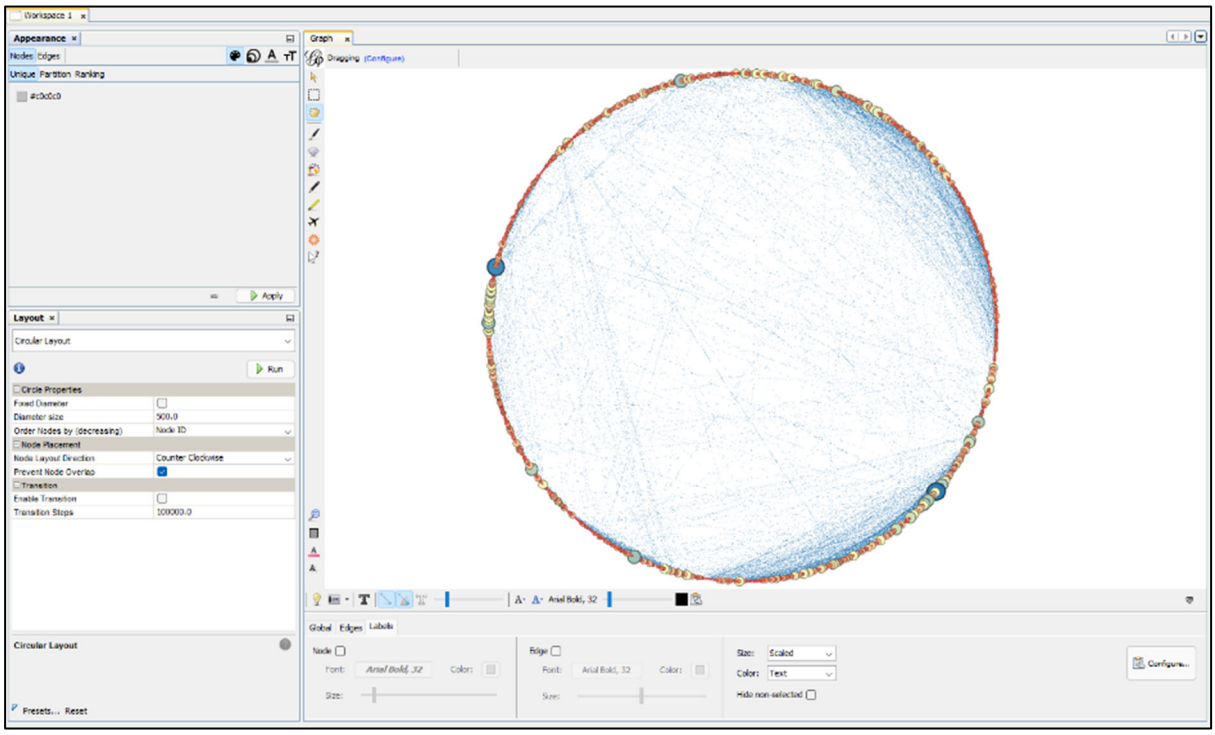
Average Path Length: 2.641



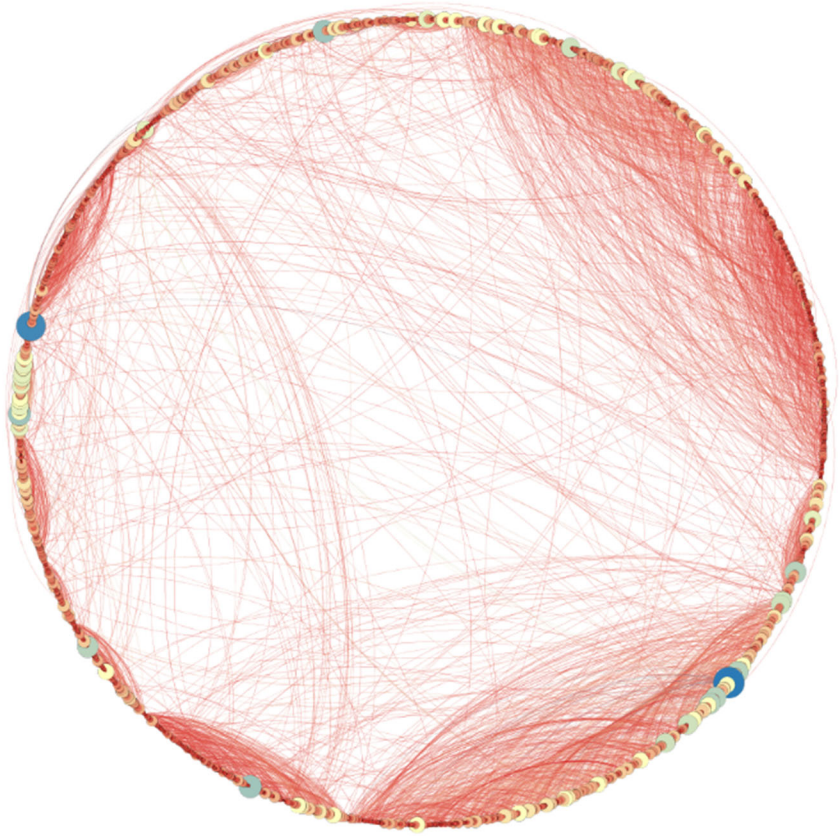
GRAPH FILE NAME: “Power Grid.gml”



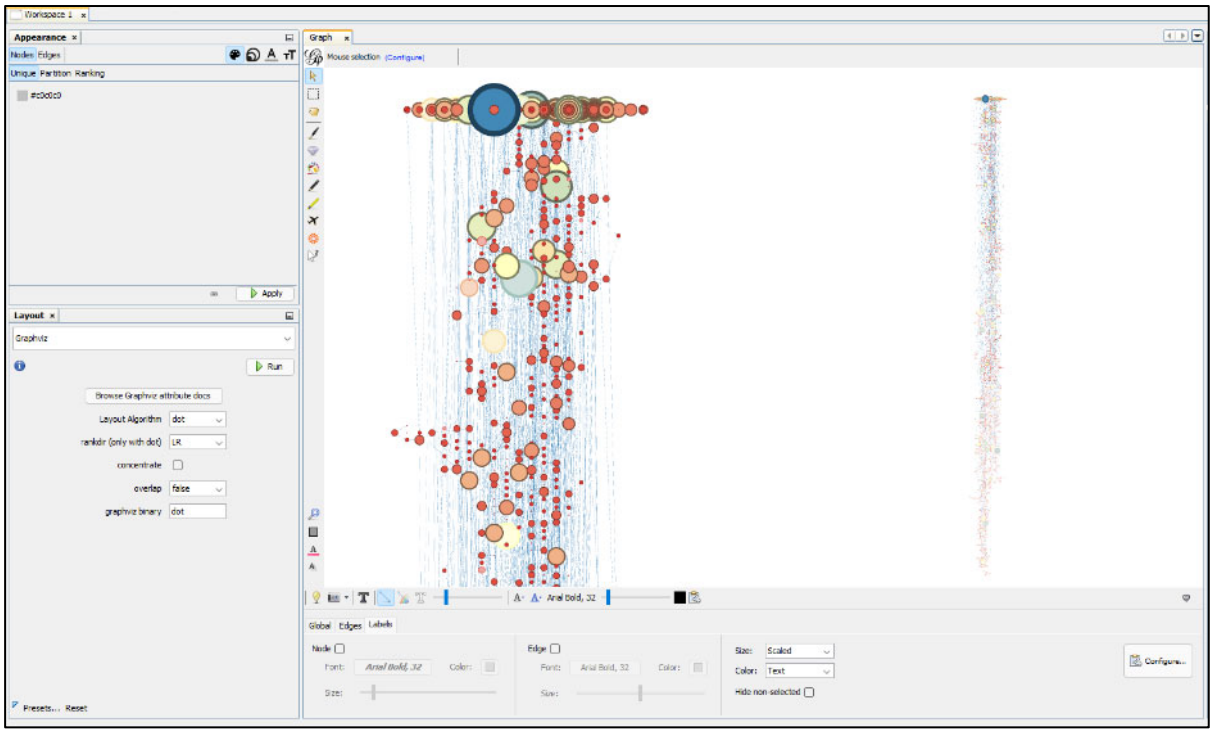
CIRCULAR LAYOUT: Power Grid



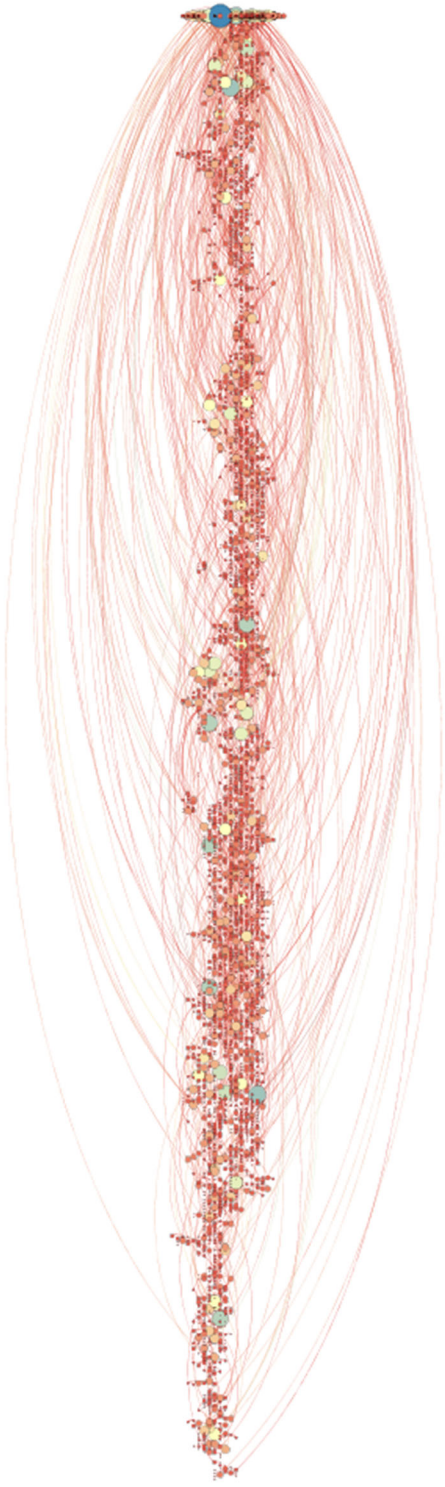
PREVIEW: CIRCULAR LAYOUT: Power Grid (Exported as png)



GRAPHVIZ LAYOUT: Power Grid



PREVIEW: GRAPHVIZ LAYOUT: Power Grid (Exported as png)



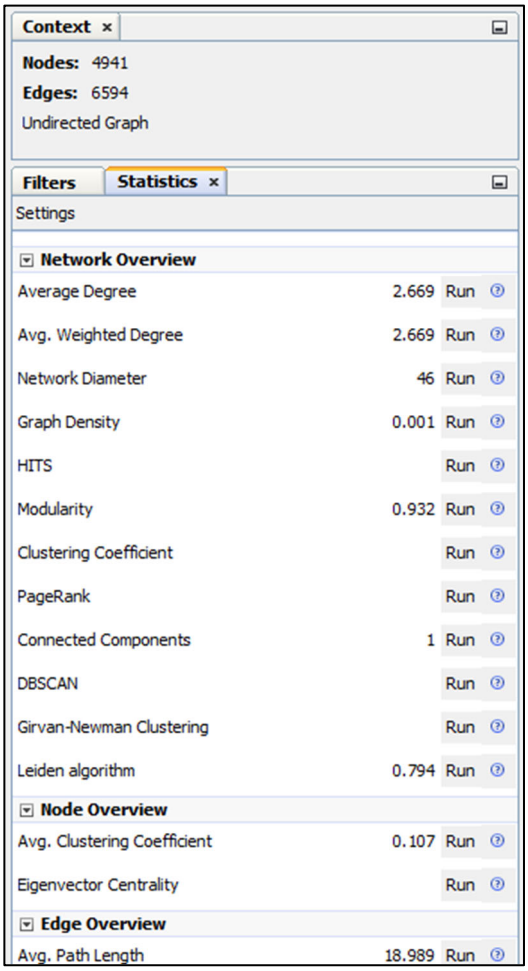
STASTICS: Power Grid

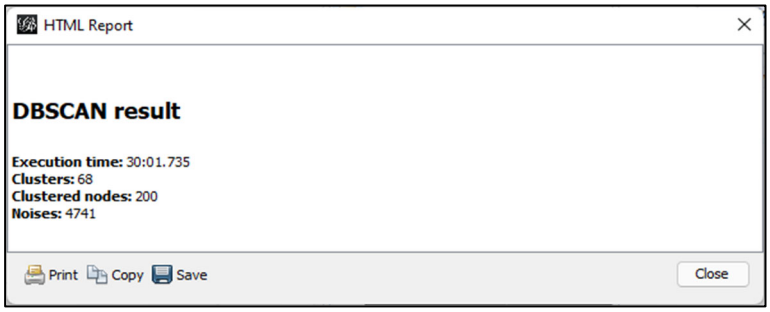
PageRank Report

Parameters:

Epsilon = 0.001
Probability = 0.85

Results:



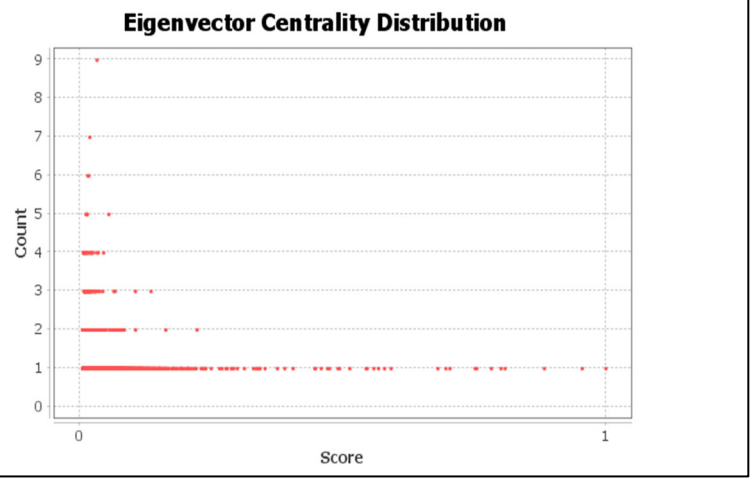


Eigenvector Centrality Report

Parameters:

Network Interpretation: undirected
Number of iterations: 100
Sum change: 0.6473580145499714

Results:



Girvan-Newman Report

Parameters:

Respect edge type for shortest path betweeness: yes
Respect parallel edges for shortest path betweeness: no

Respect edge type for modularity computation: no
Respect parallel edges for modularity computation: no

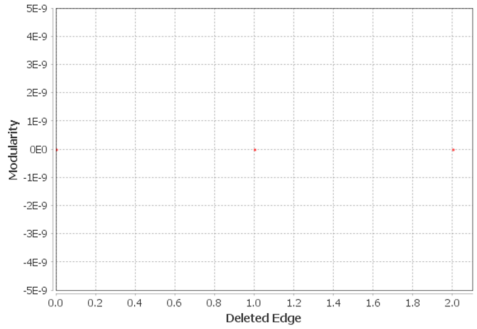
Processed Graph Data

Nodes: 4941
Edges 6594

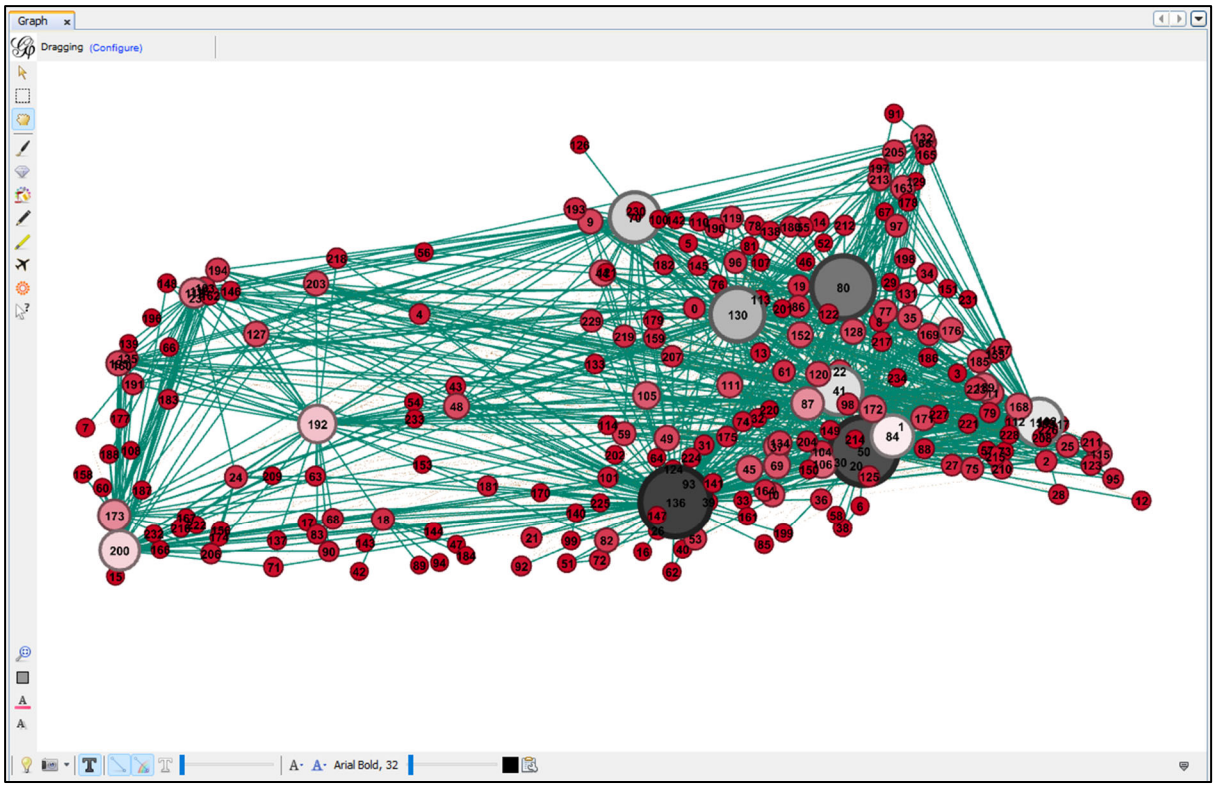
Processing time: 26.179 sec.

Communities

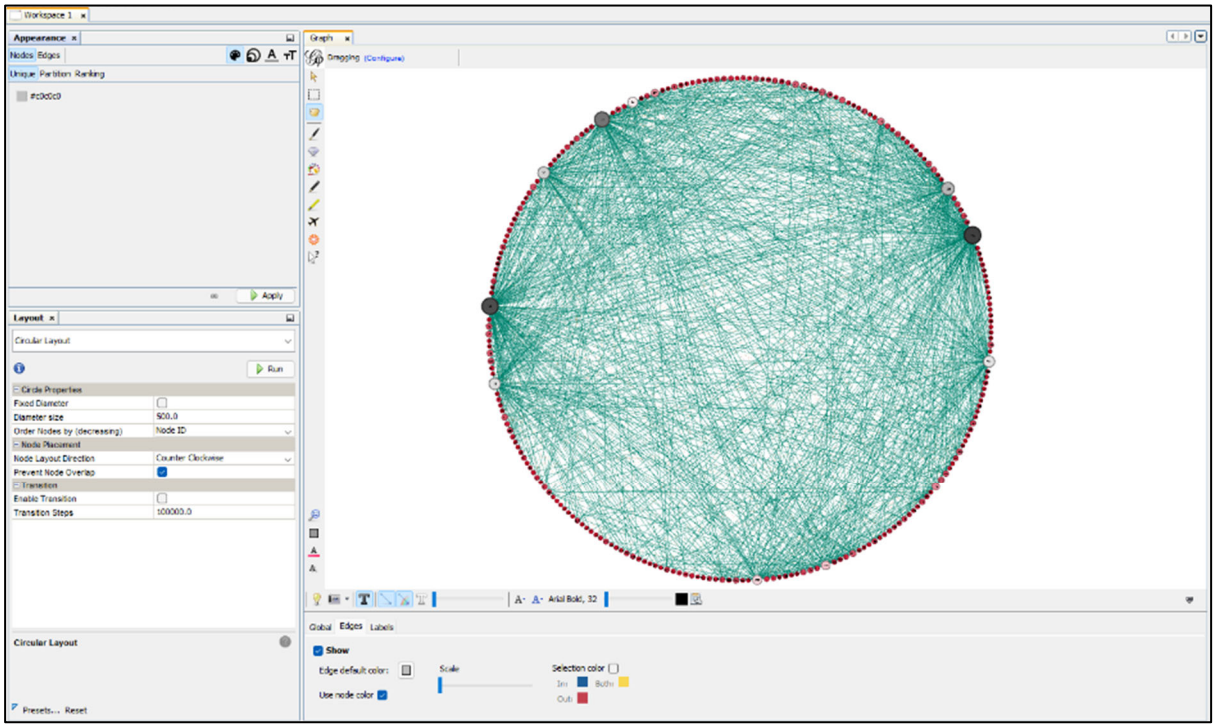
Number of communities: 1
Maximum found modularity: 0.0



GRAPH FILE NAME: “Airlines.graphml”



CIRCULAR LAYOUT: Airlines



PREVIEW: CIRCULAR LAYOUT: Airlines

STATISTICS: Airlines

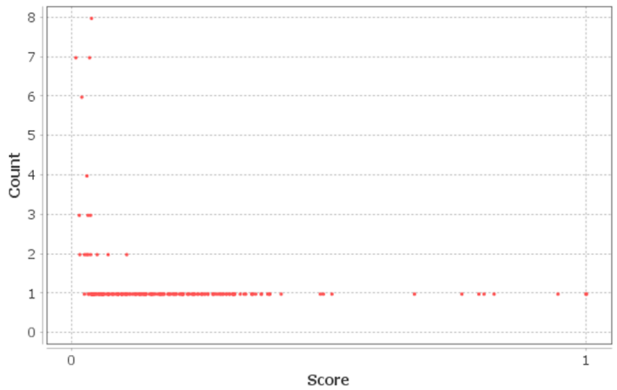
Eigenvector Centrality Report

Parameters:

Network Interpretation: undirected
Number of iterations: 100
Sum change: 0.0035883631010529106

Results:

Eigenvector Centrality Distribution



DBSCAN result

Execution time: 30:00.015
Clusters: 0
Clustered nodes: 0
Noises: 235

Context

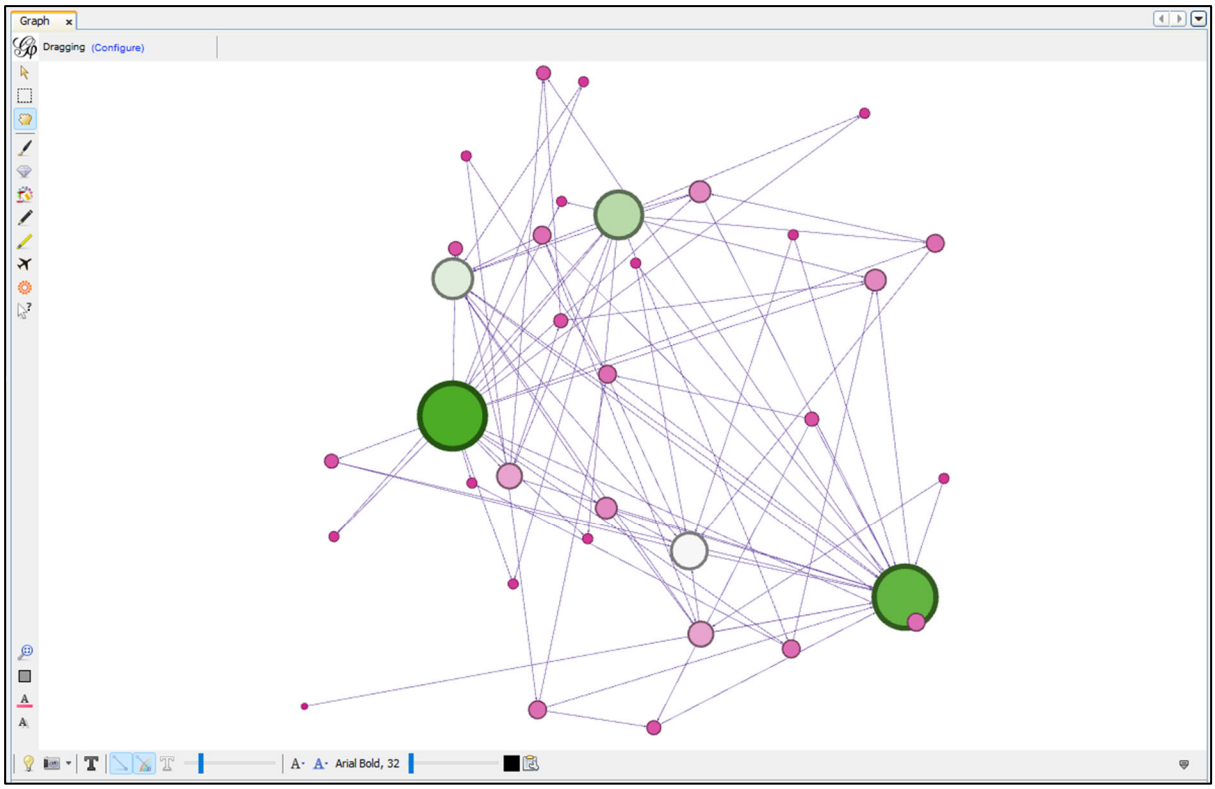
Nodes: 235
Edges: 1297
Undirected Graph

Filters | **Statistics**

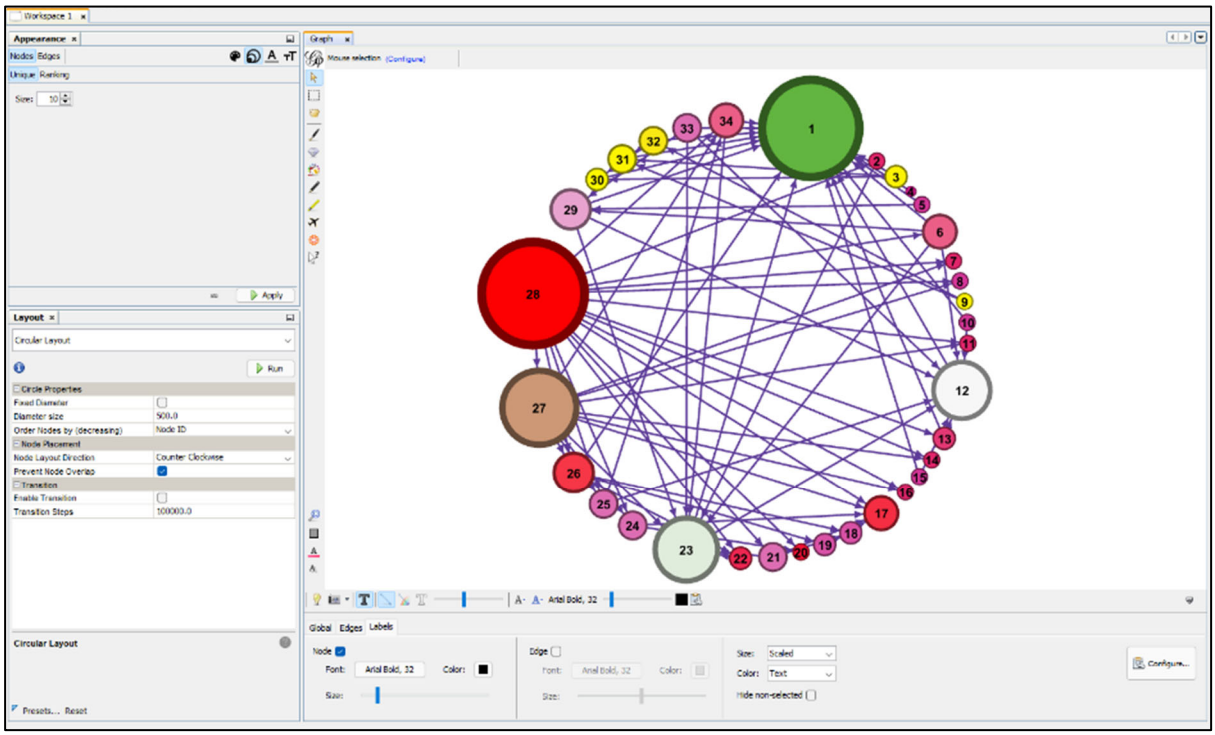
Settings

Network Overview		
Average Degree	11.038	Run
Avg. Weighted Degree	17.881	Run
Network Diameter	4	Run
Graph Density	0.047	Run
HITS		Run
Modularity	0.245	Run
Clustering Coefficient		Run
PageRank		Run
Connected Components	1	Run
DBSCAN		Run
Girvan-Newman Clustering		Run
Leiden algorithm	0.875	Run
Node Overview		
Avg. Clustering Coefficient	0.652	Run
Eigenvector Centrality		Run
Edge Overview		
Avg. Path Length	2.318	Run
Dynamic		

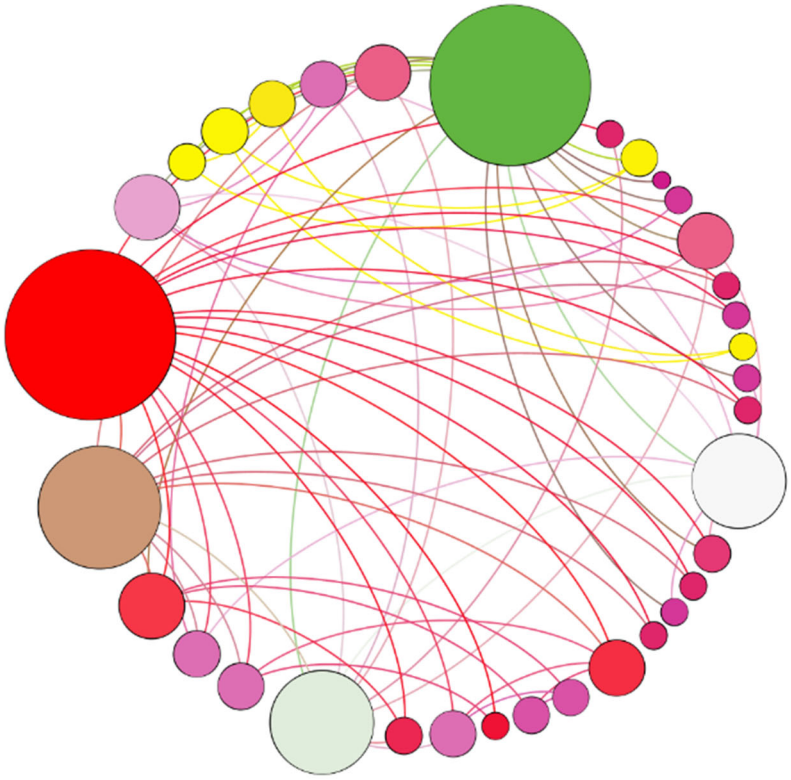
GRAPH FILE NAME: "Zachary's Karate Club.csv"



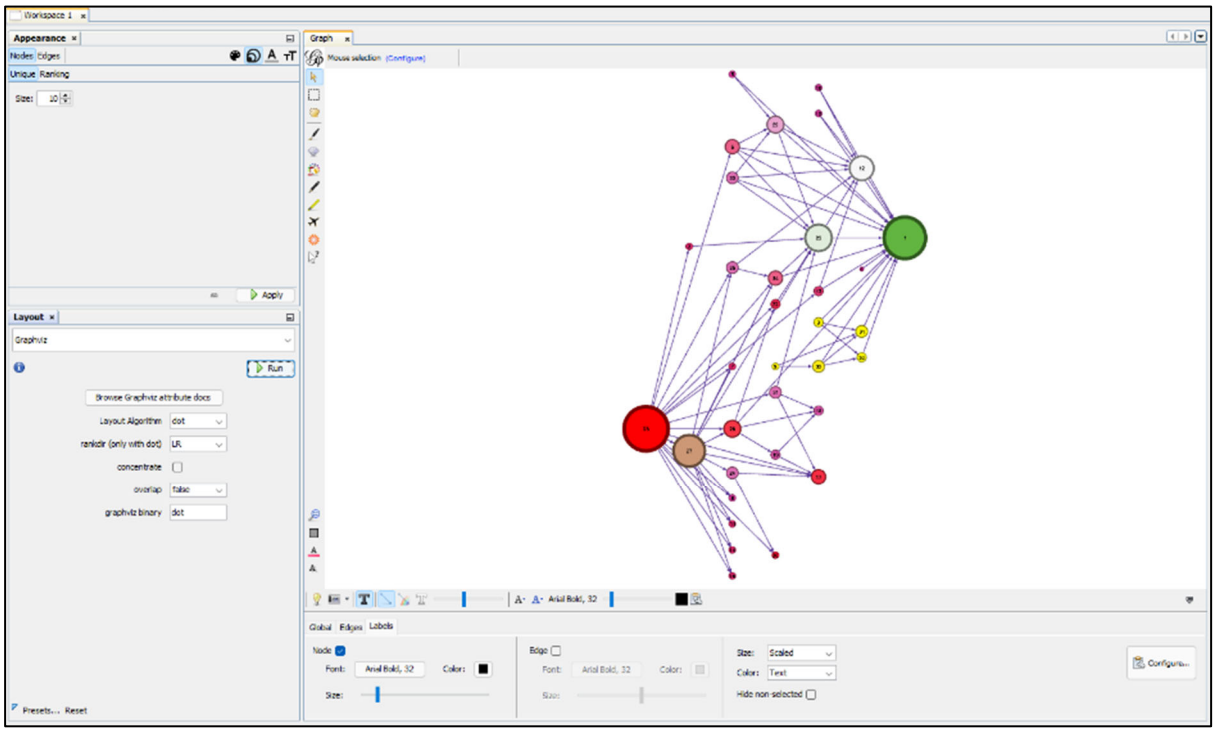
CIRCULAR LAYOUT: Zachary's Karate Club



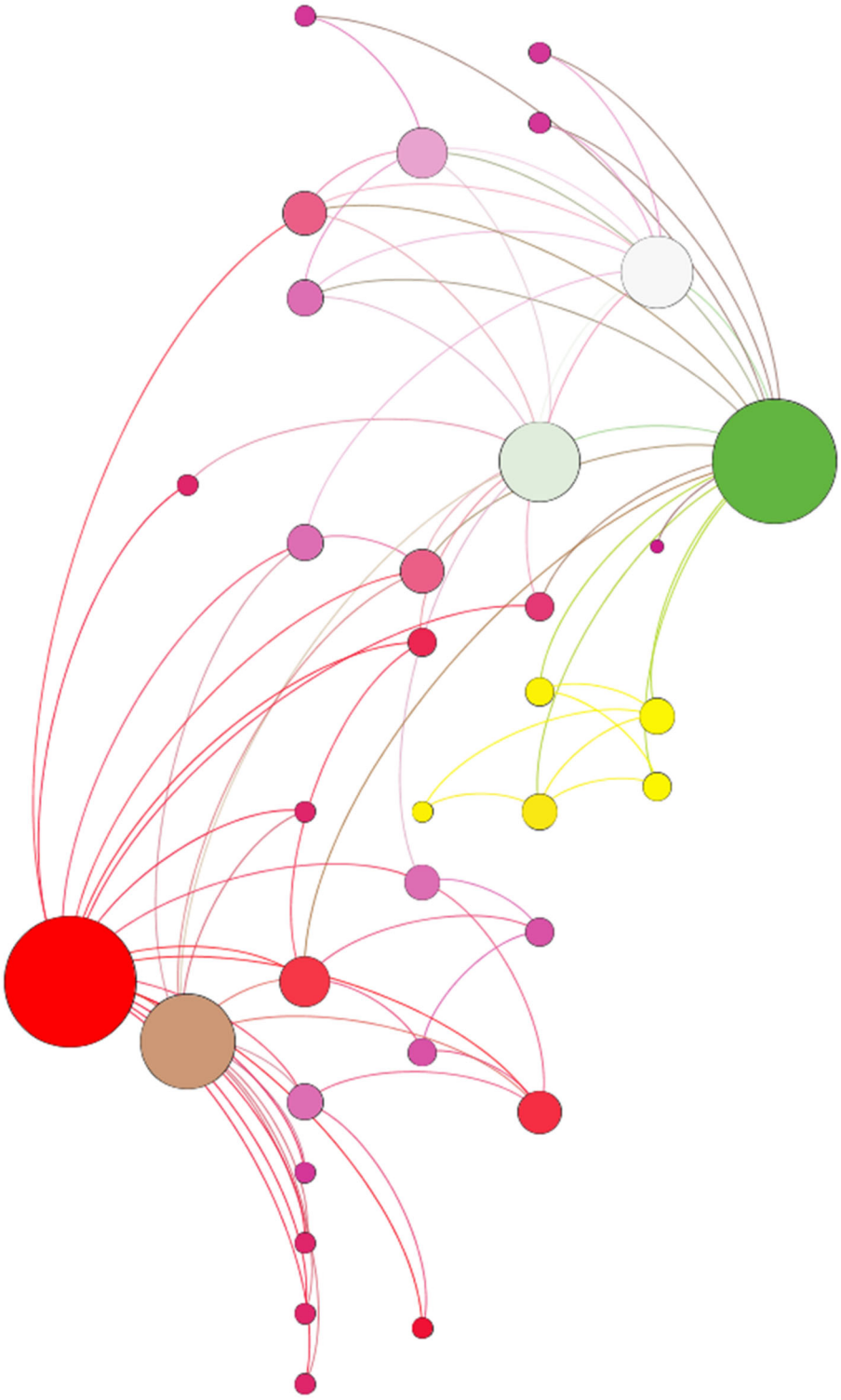
PREVIEW: CIRCULAR LAYOUT: Zachary's Karate Club (Exported as png)



GRAPHVIZ LAYOUT: Zachary's Karate Club



PREVIEW: GRAPHVIZ LAYOUT: Zachary's Karate Club (Exported as png)



STATISTICS: Zachary's Karate Club

Clustering Coefficient (Triangle Method): 0.256

Girvan-Newman Report

Parameters:

Respect edge type for shortest path betweeness: yes
Respect parallel edges for shortest path betweeness: no

Respect edge type for modularity computation: yes
Respect parallel edges for modularity computation: no

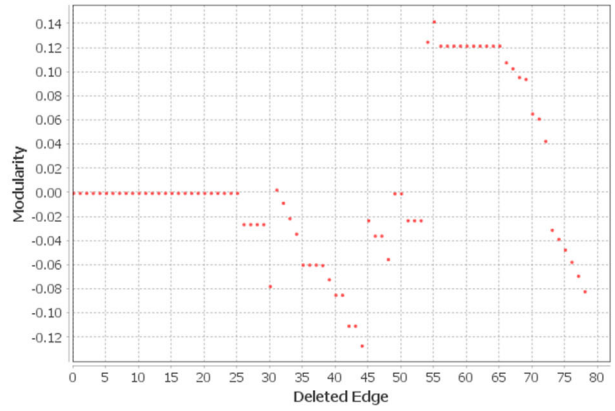
Processed Graph Data

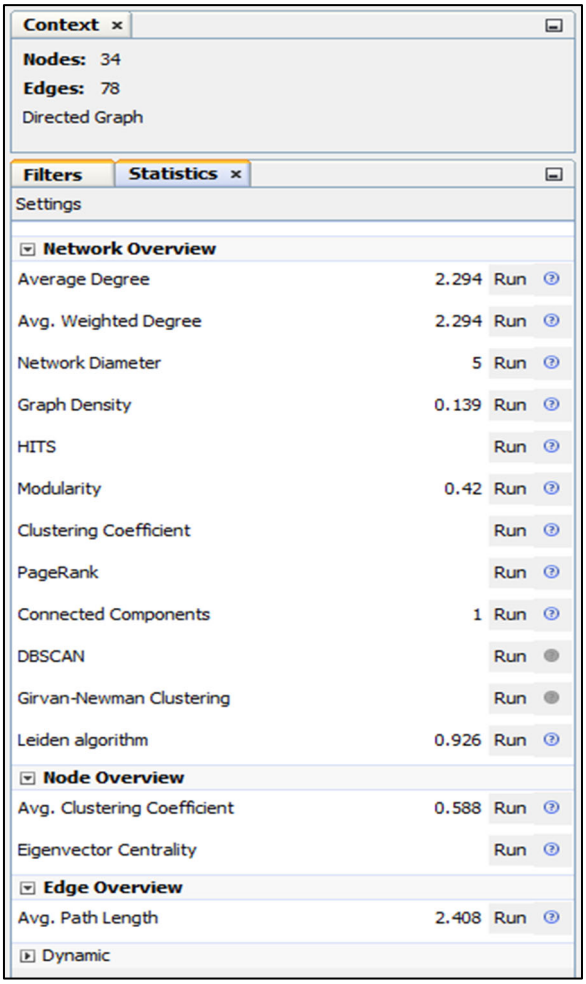
Nodes: 34
Edges 78

Processing time: 0.062 sec.

Communities

Number of communities: 20
Maximum found modularity: 0.14201184





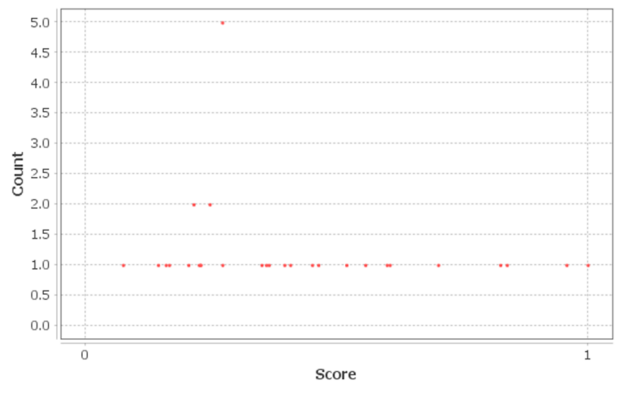
Eigenvector Centrality Report

Parameters:

Network Interpretation: undirected
Number of iterations: 100
Sum change: 9.783386901414531E-4

Results:

Eigenvector Centrality Distribution



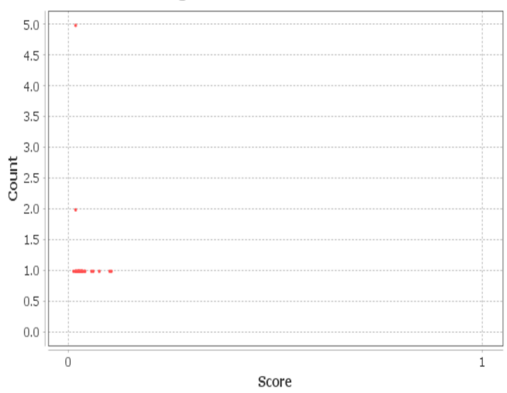
PageRank Report

Parameters:

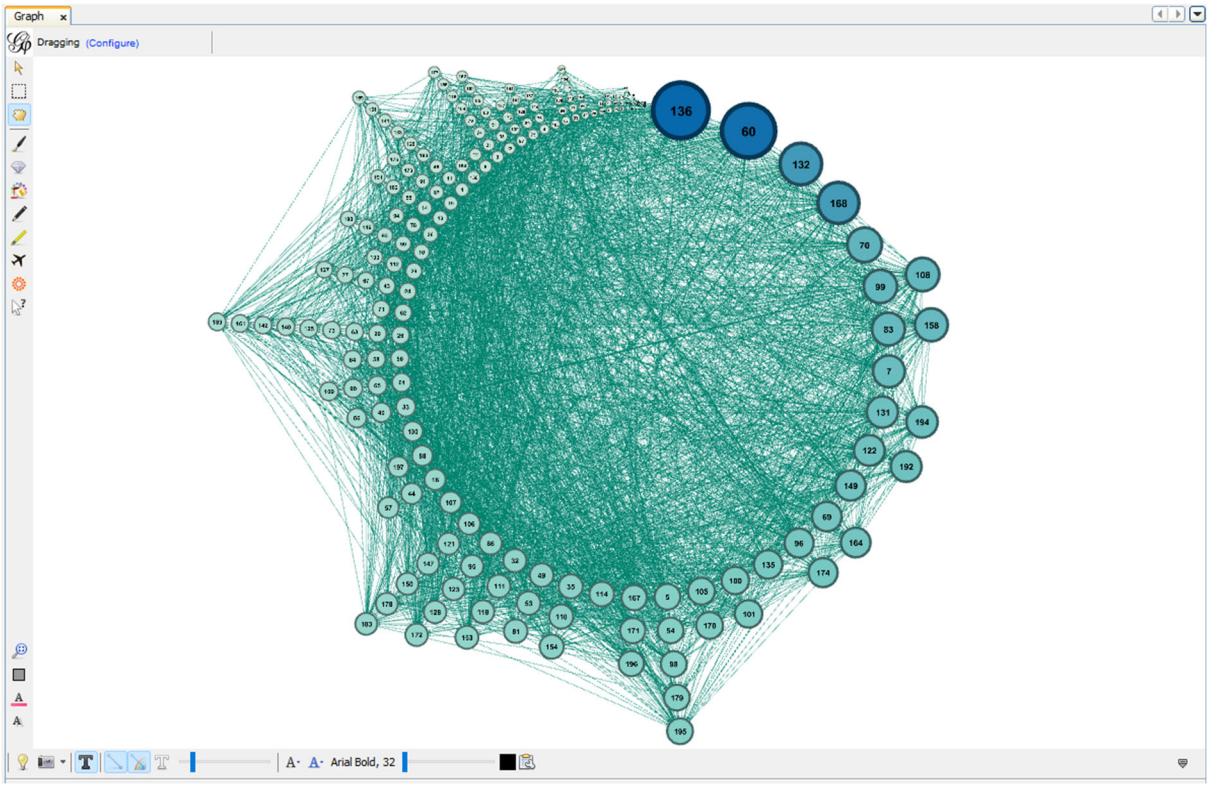
Epsilon = 0.001
Probability = 0.85

Results:

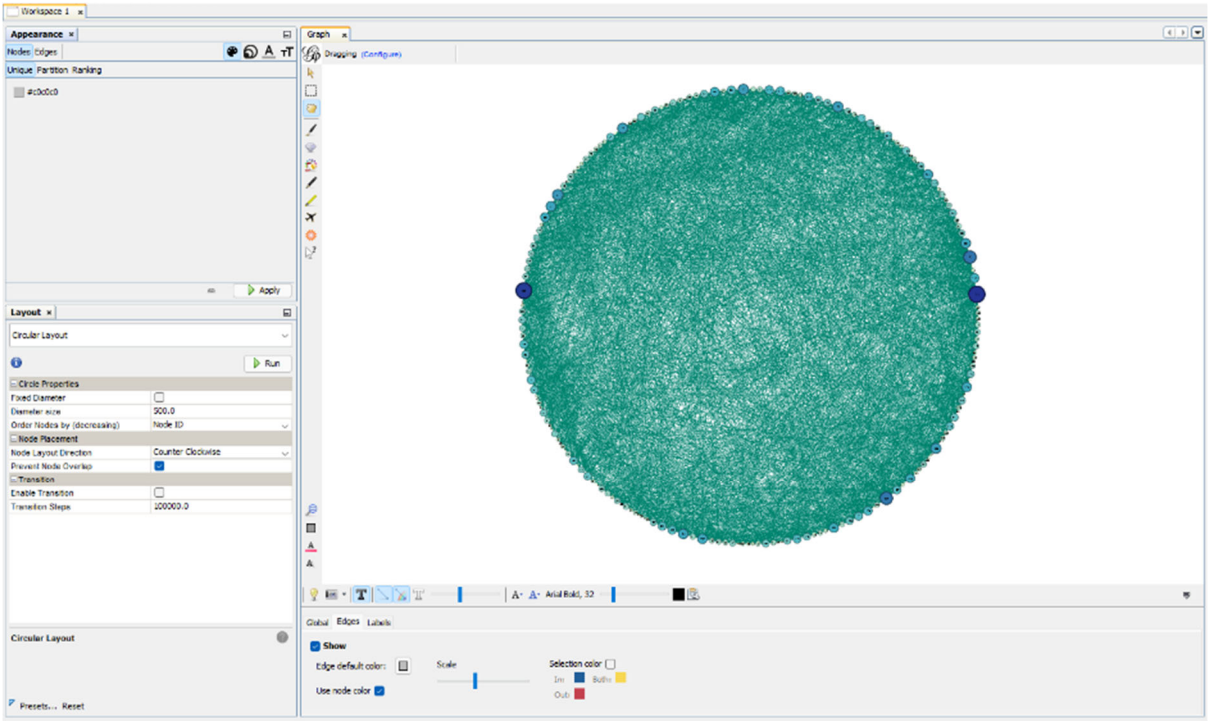
PageRank Distribution



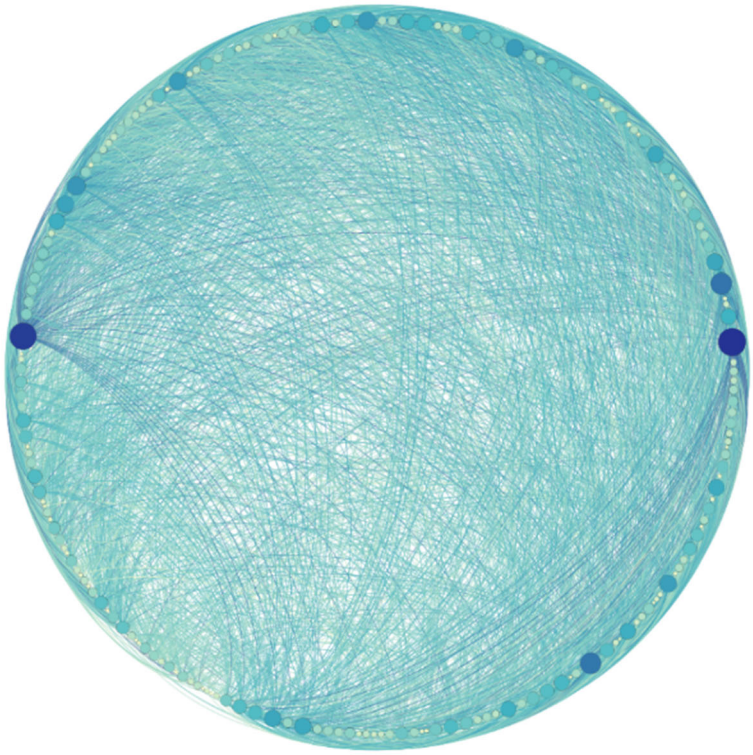
GRAPH FILE NAME: “Jazz Musicians Network.net”



CIRCULAR LAYOUT: Jazz Musicians Network

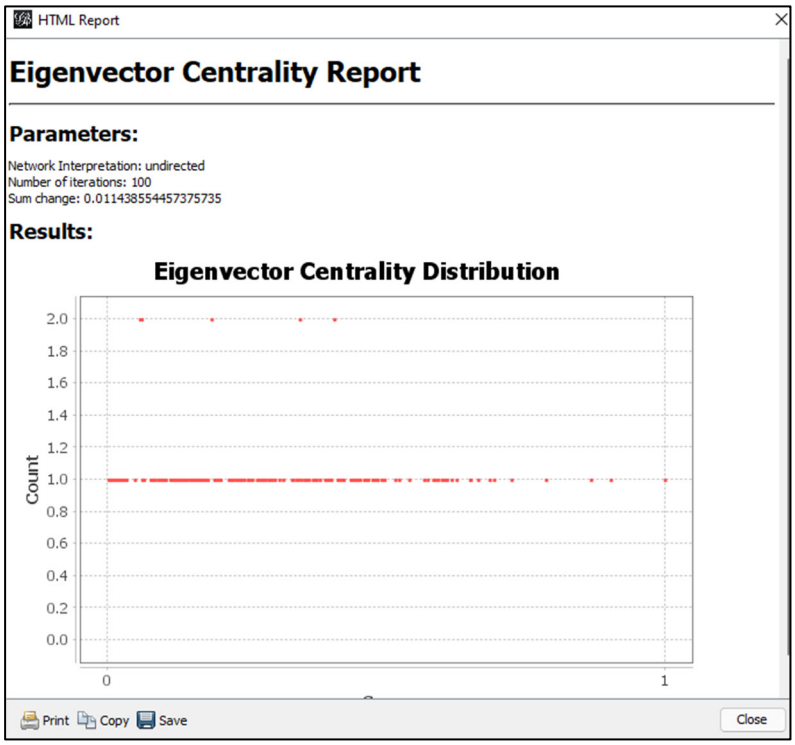


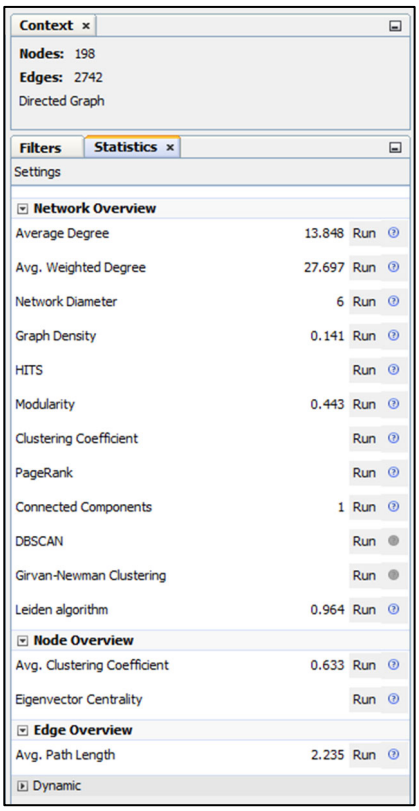
PREVIEW: CIRCULAR LAYOUT: Jazz Musicians Network (Exported as png)



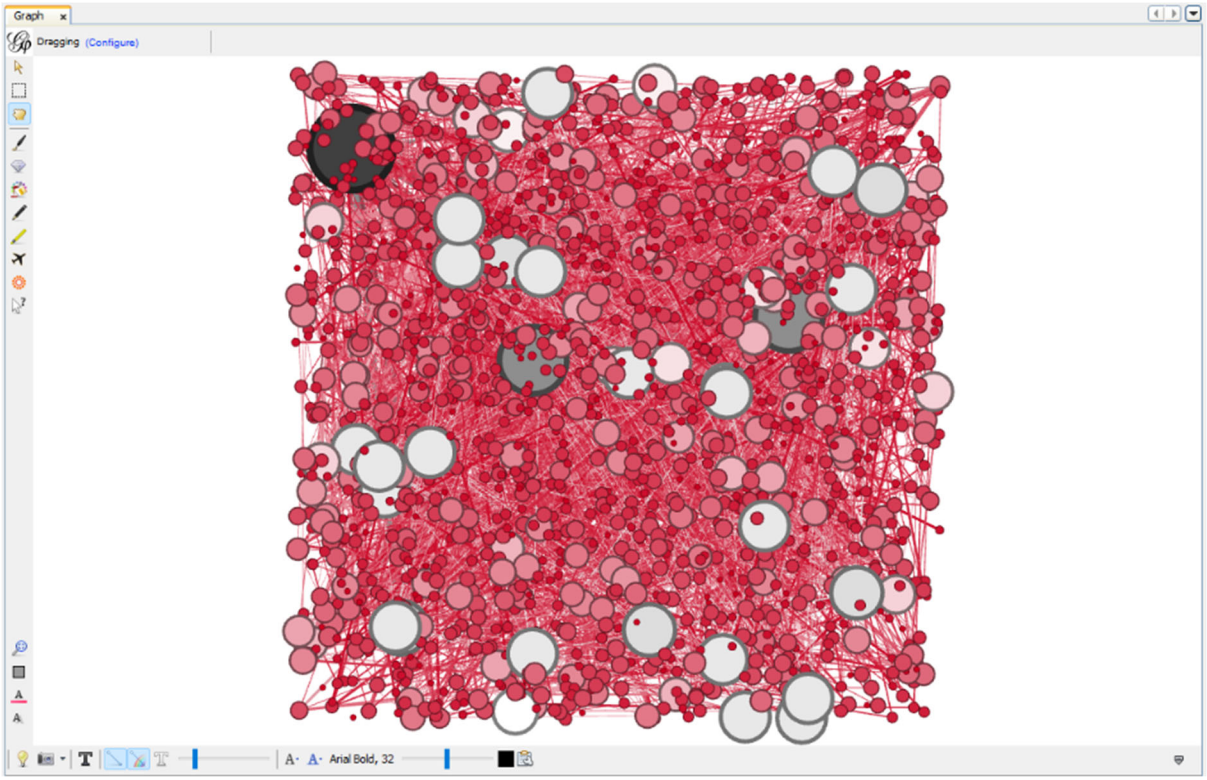
STATISTICS: Jazz Musicians Network

Clustering Coefficient (Triangle Method): 0.520

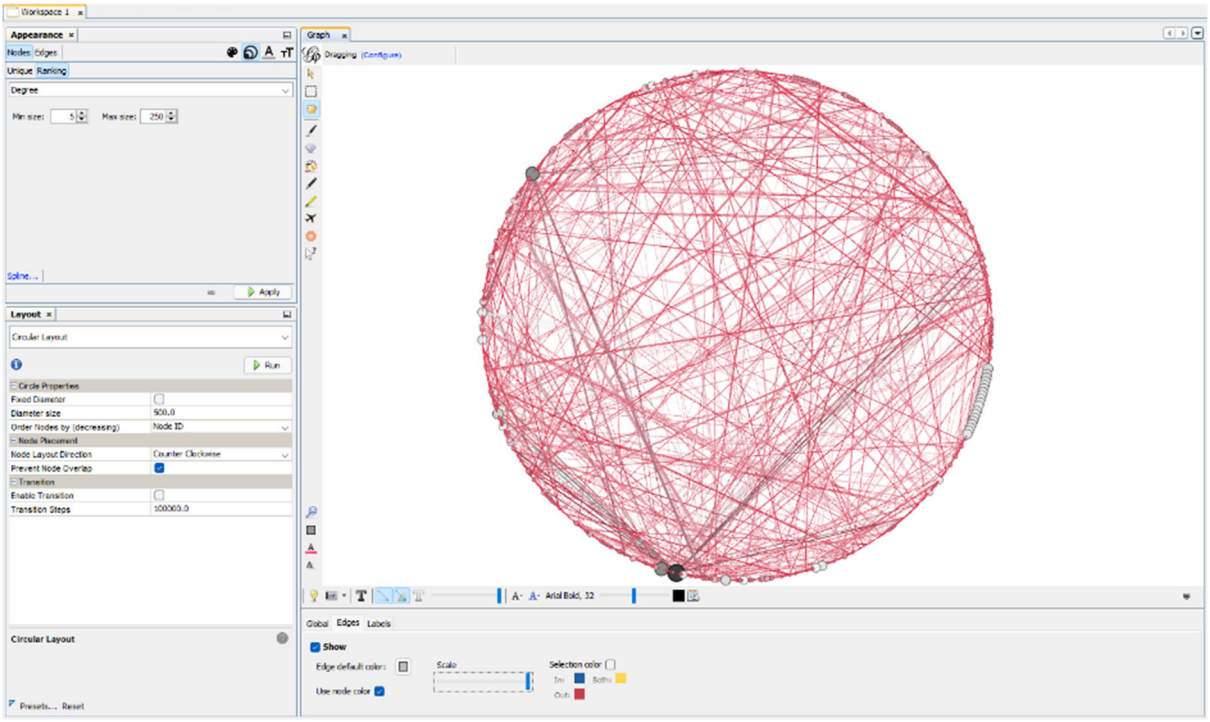




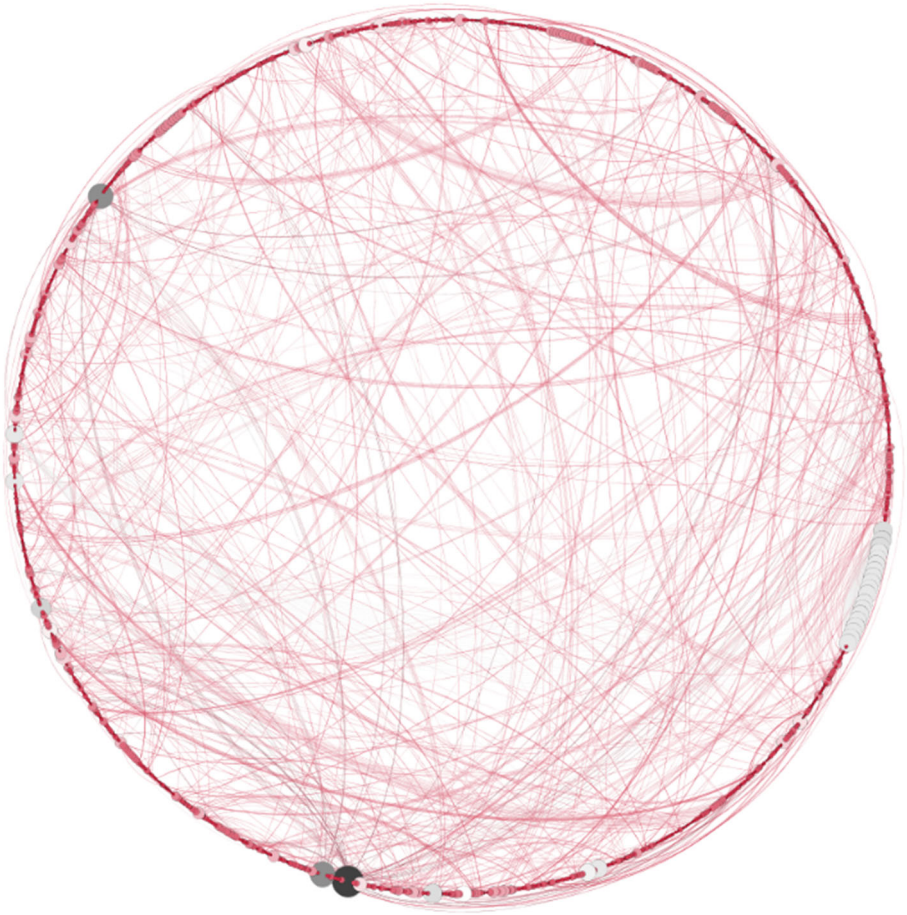
GRAPH FILE NAME: “Coauthorships in Network Science.gml”



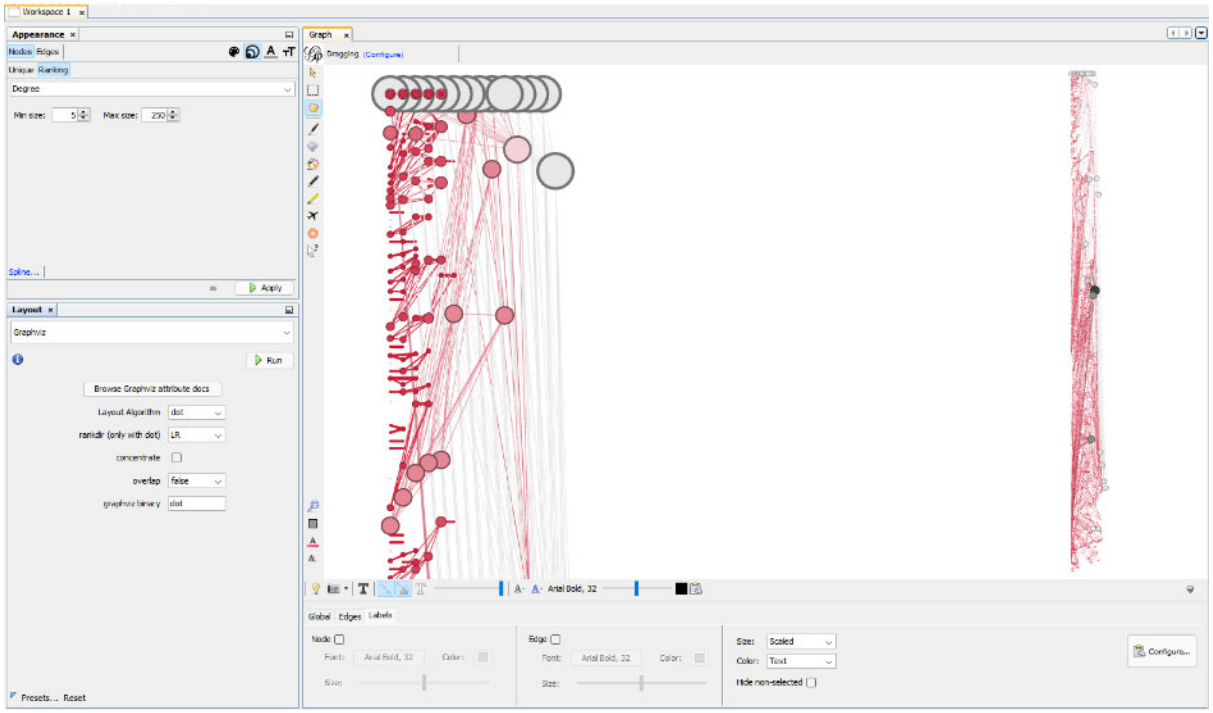
CIRCULAR LAYOUT: Coauthorships in Network Science



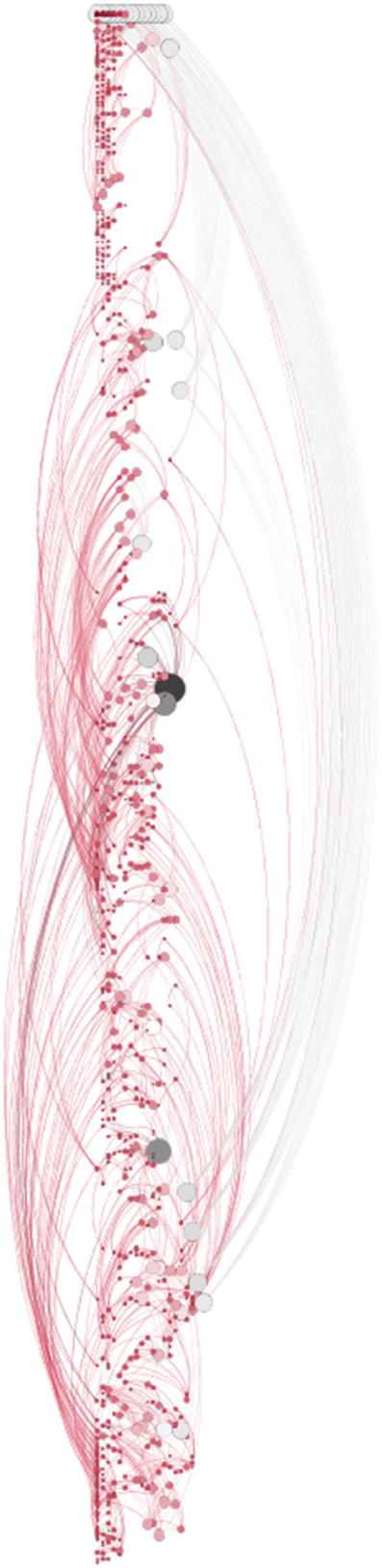
PREVIEW: CIRCULAR LAYOUT: Coauthorships in Network Science (Exported as png)



GRAPHVIZ LAYOUT: Coauthorships in Network Science



PREVIEW: GRAPHVIZ LAYOUT: Coauthorships in Network Science (Exported as png)



STATISTICS: Coauthorships in Network Science

Clustering Coefficient: 0.693

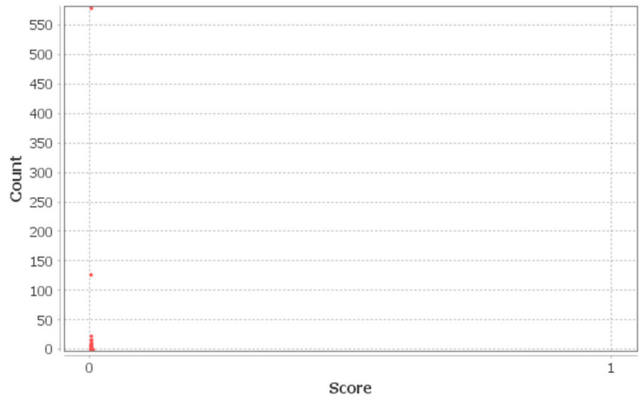
PageRank Report

Parameters:

Epsilon = 0.001
Probability = 0.85

Results:

PageRank Distribution



Girvan-Newman Report

Parameters:

Respect edge type for shortest path betweeness: yes
Respect parallel edges for shortest path betweeness: no

Respect edge type for modularity computation: yes
Respect parallel edges for modularity computation: no

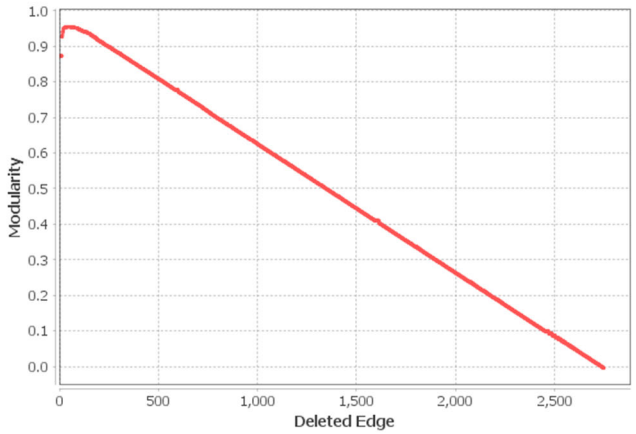
Processed Graph Data

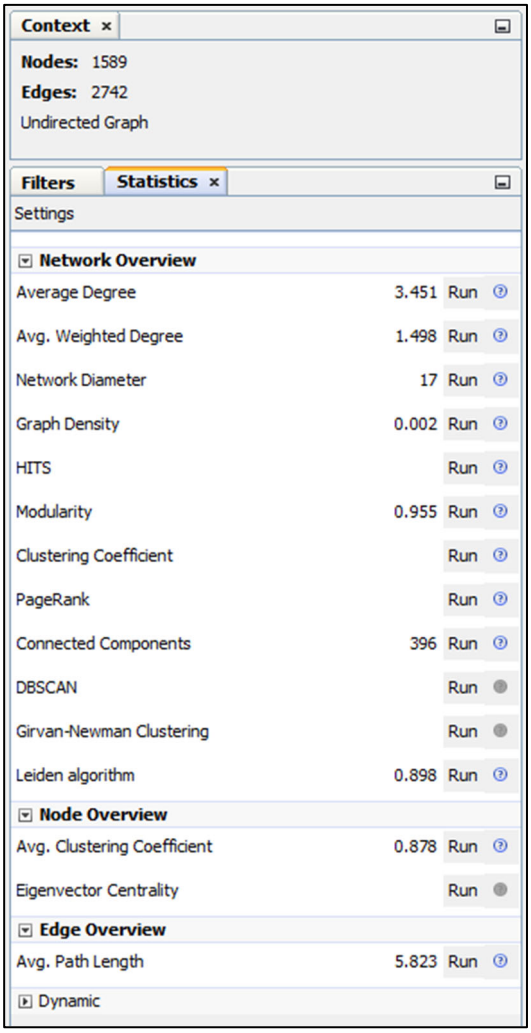
Nodes: 1589
Edges 2742

Processing time: 14.785 sec.

Communities

Number of communities: 405
Maximum found modularity: 0.95792794





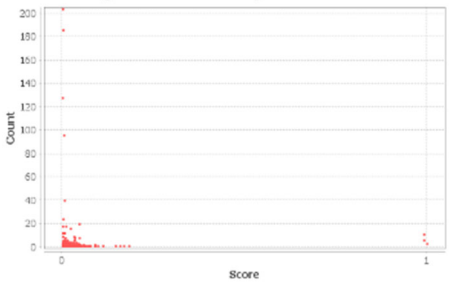
Eigenvector Centrality Report

Parameters:

Network Interpretation: undirected
Number of iterations: 100
Sum change: 0.0916494149317325 1

Results:

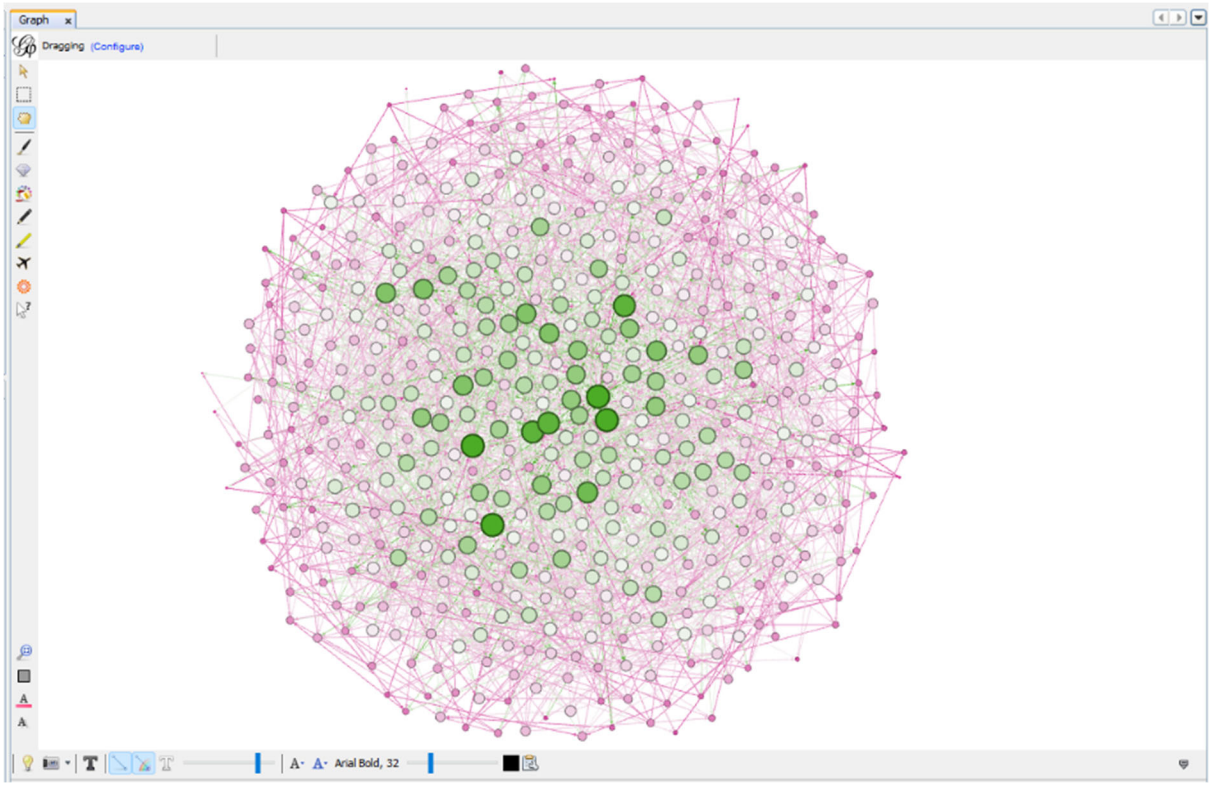
Eigenvector Centrality Distribution



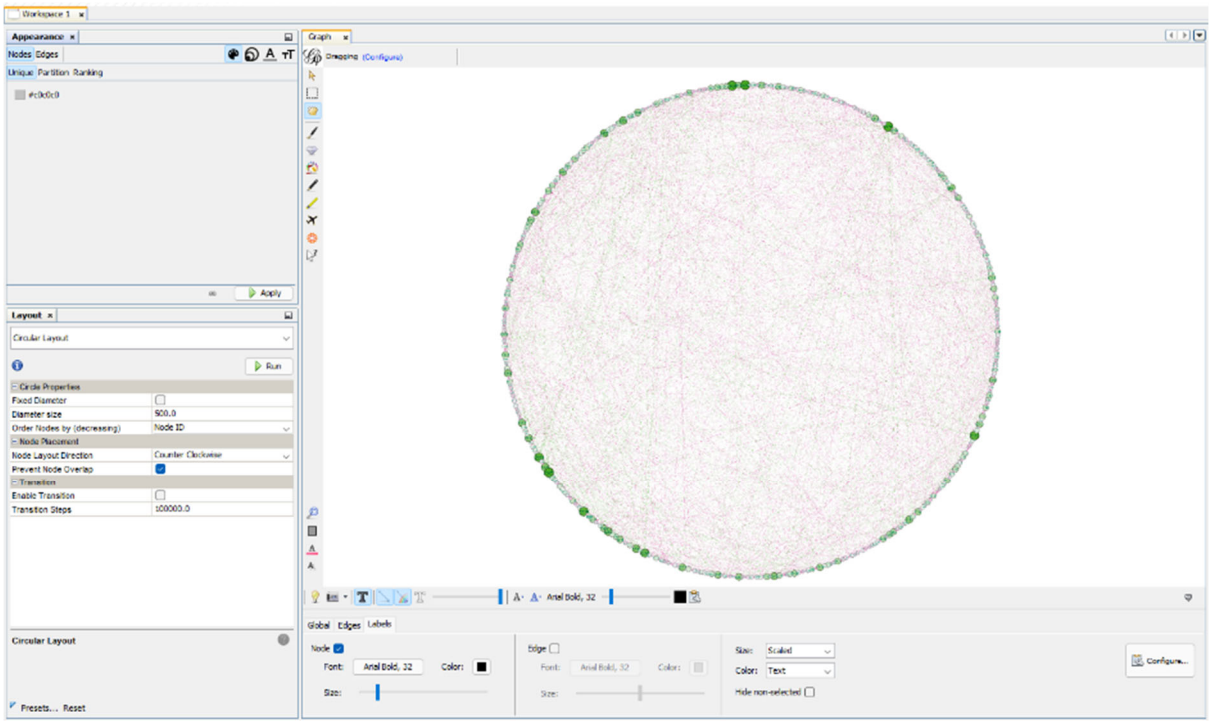
DBSCAN result

Execution time: 30:00.710
Clusters: 0
Clustered nodes: 0
Noises: 1589

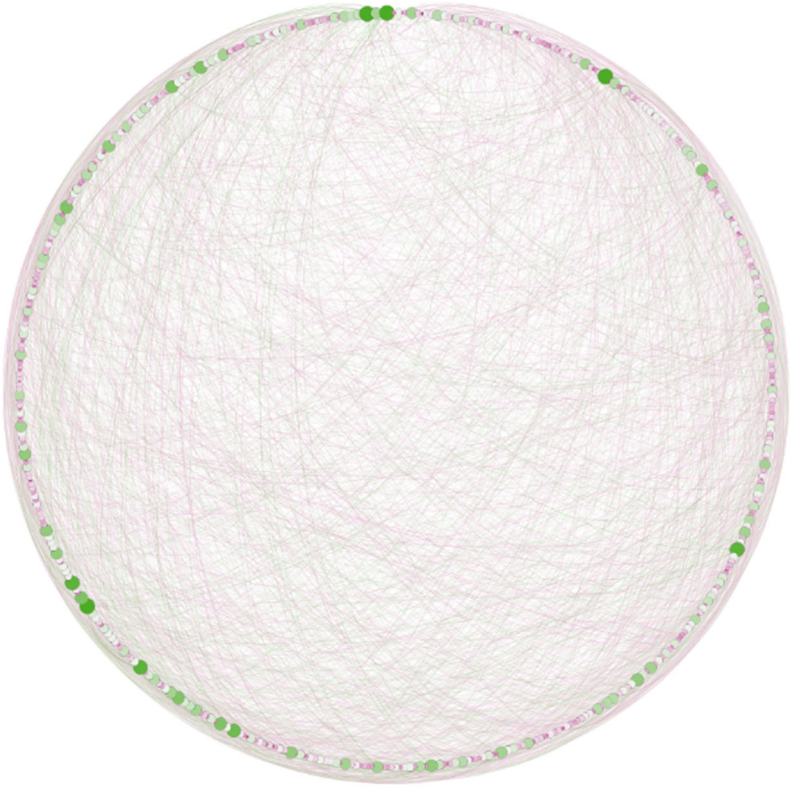
GRAPH FILE NAME: Gephi Generated Random Graph (n = 500, p = 0.02)



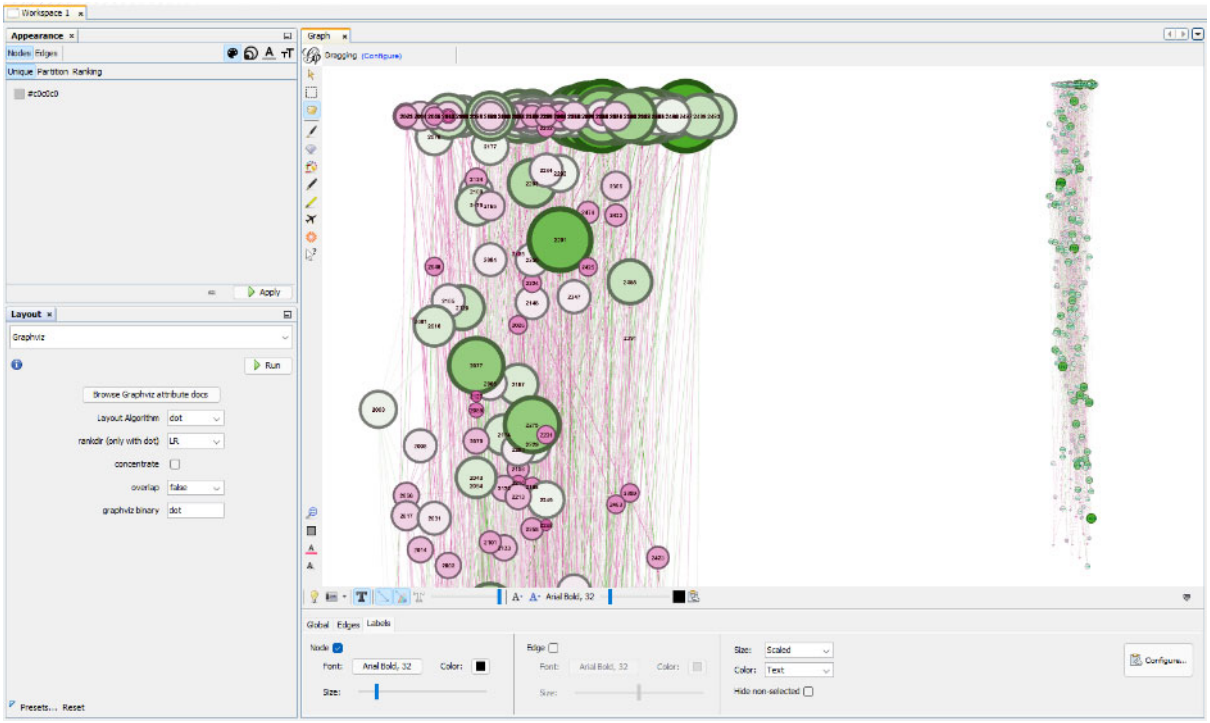
CIRCULAR LAYOUT: Gephi Generated Random Graph (n = 500, p = 0.02)



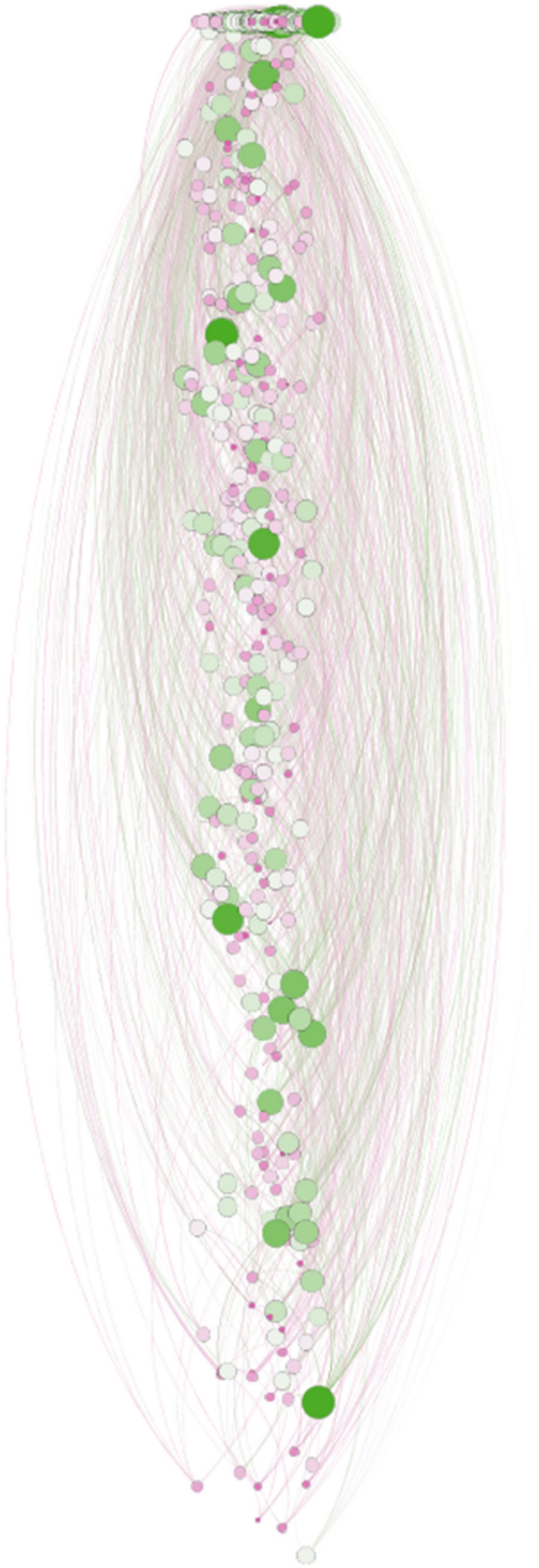
PREVIEW: CIRCULAR LAYOUT: Gephi Generated Random Graph (Exported as png)



GRAPHVIZ LAYOUT: Gephi Generated Random Graph (n = 500, p = 0.02)



PREVIEW: GRAPHVIZ LAYOUT: Gephi Generated Random Graph (Exported as png)



STATISTICS: Gephi Generated Random Graph (n = 500, p = 0.02)

Clustering Coefficient (Triangle Method): 0.023

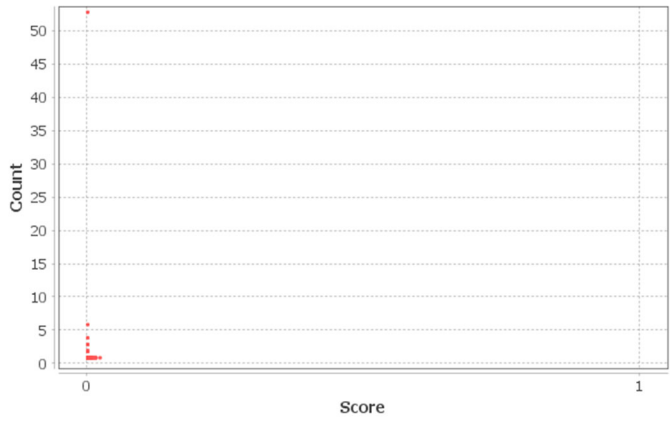
PageRank Report

Parameters:

Epsilon = 0.001
Probability = 0.85

Results:

PageRank Distribution



Girvan-Newman Report

Parameters:

Respect edge type for shortest path betweeness: yes
Respect parallel edges for shortest path betweeness: no

Respect edge type for modularity computation: yes
Respect parallel edges for modularity computation: no

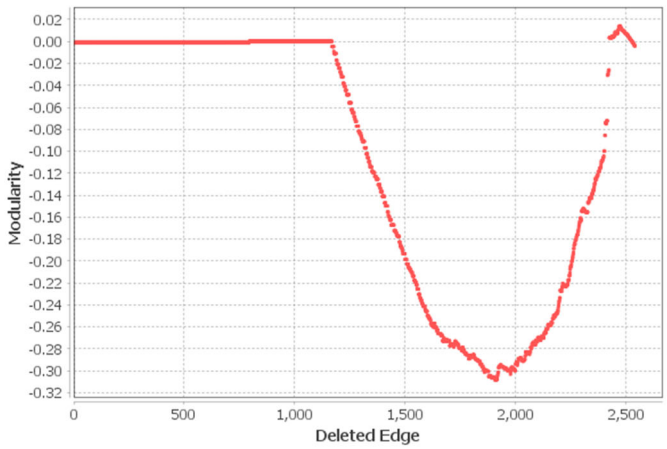
Processed Graph Data

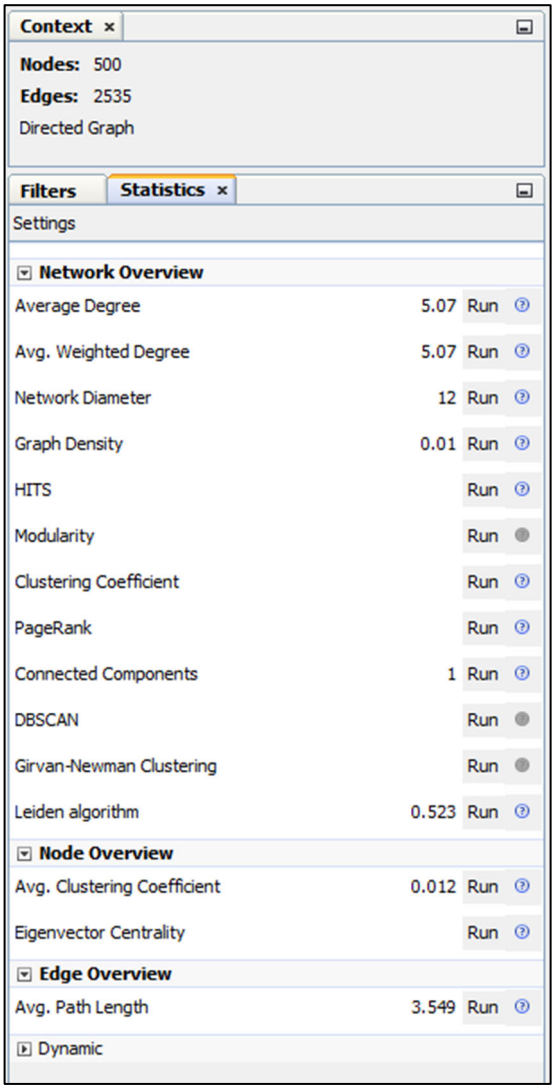
Nodes: 500
Edges 2535

Processing time: 5.932 sec.

Communities

Number of communities: 434
Maximum found modularity: 0.015236629





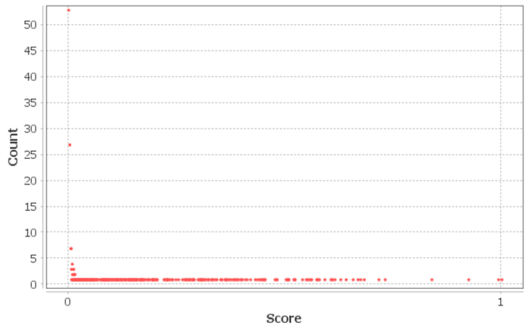
Eigenvector Centrality Report

Parameters:

Network Interpretation: directed
Number of iterations: 100
Sum change: 0.17908370519527342

Results:

Eigenvector Centrality Distribution



AIM IV: TO GENERATE DISJOINT COMMUNITIES WITH LOUVAIN METHOD AND VISUALIZE THE GENERATED COMMUNITIES USING NETWORKX LIBRARY OR GEPHI. (ALTERNATIVELY, CAN ALSO USE ANY OF THE FOLLOWING TOOLS: MATPLOTLIB, PLOTLY, GGPLOT, SEABORN, BOKEH).

THEORY:

1. **Disjoint Community:** It is a community structure where a node can belong to maximum one community.

2. **Louvain Method:** The Louvain method for community detection is a method to extract communities from large networks**.** It is a heuristic method based on modularity optimization. The method is a greedy optimization method that attempts to optimize the "modularity" of a partition of the network. First small communities are found by optimizing modularity locally on all nodes, then each small community is grouped into one node and the first step is repeated.

   This algorithm comprises of two steps. On the first step it assigns every node to be in its own community and then for each node it tries to find the maximum positive modularity gain by moving each node to all of its neighbour communities. If no positive gain is achieved the node remains in its original community**.**

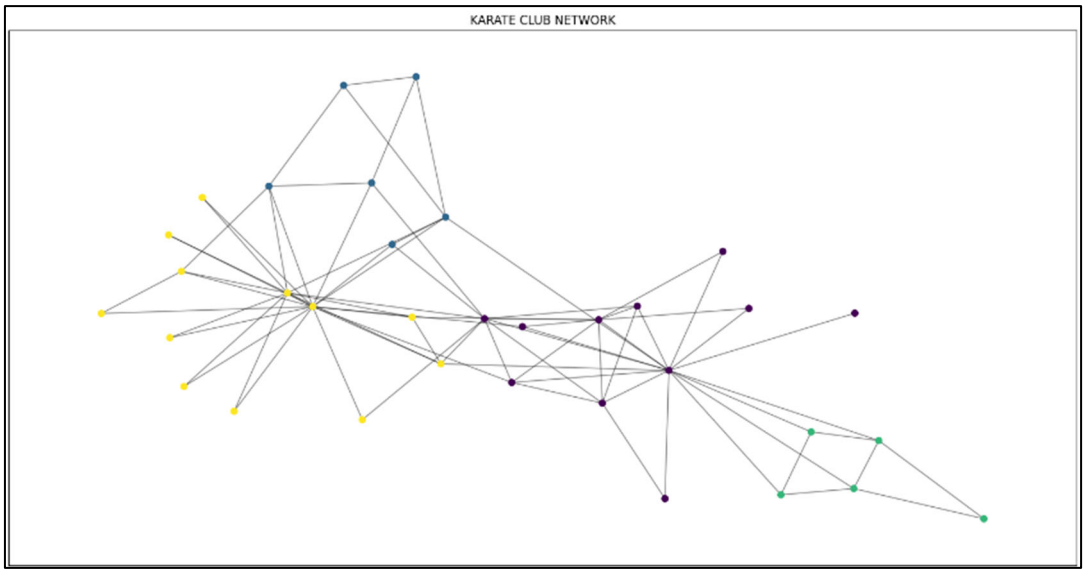
   GRAPHS: Zachary's Karate Club, American College Football and Dolphin Social Network.

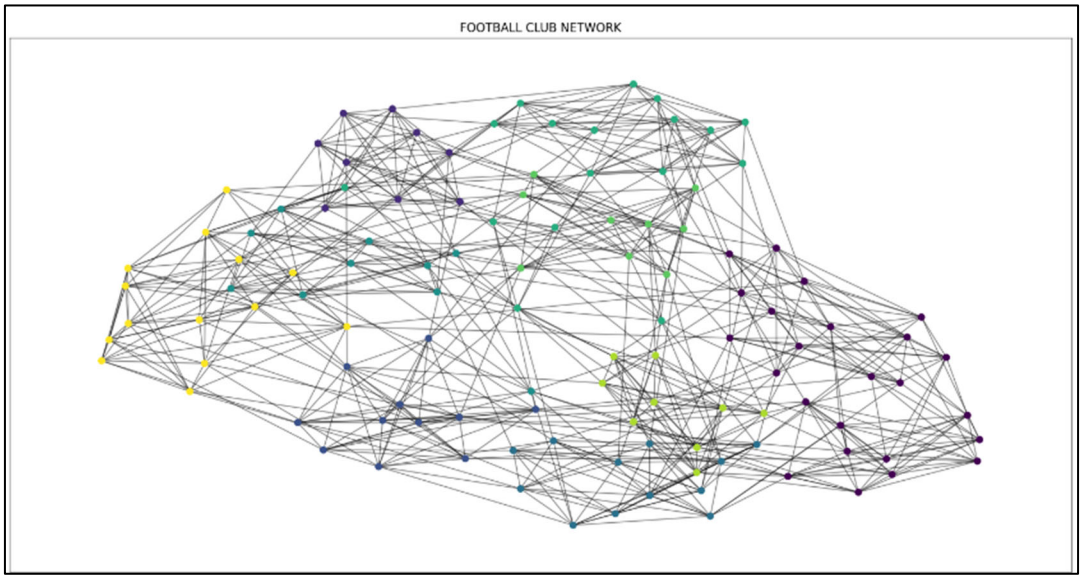
CODE:

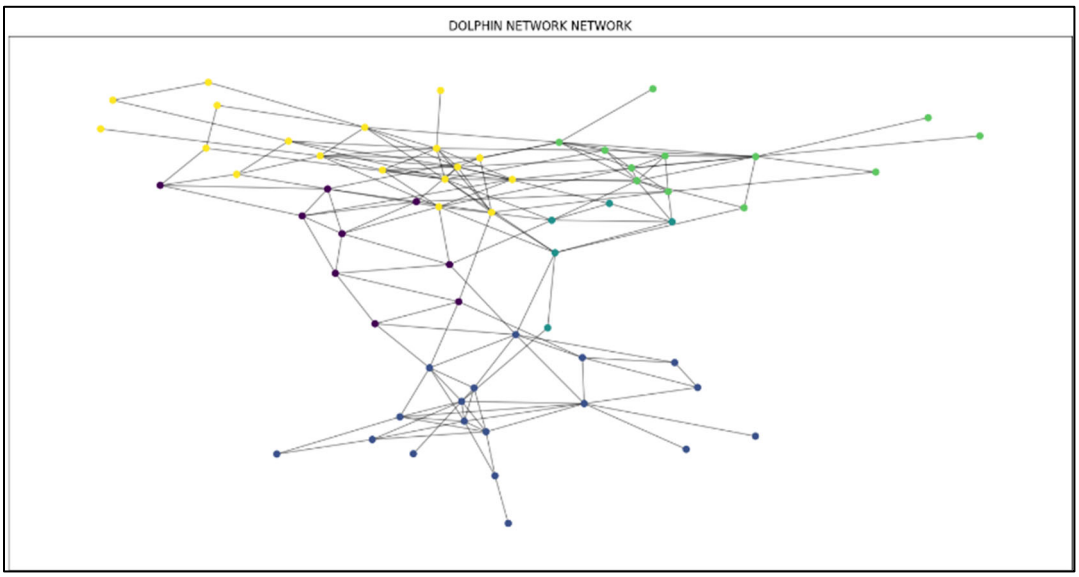
```
import community as community_louvain
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import networkx as nx

import urllib.request
import io
import zipfile

# Loading Karate Club Graph
GG_Karate = nx.karate_club_graph ()

# Loading Football Club Graph
url = "http://www-personal.umich.edu/~mejn/netdata/football.zip"
sock = urllib.request.urlopen (url)
sockRead = io.BytesIO (sock.read ())
sock.close ()

zfFootball = zipfile.ZipFile (sockRead)
txtFootball = zfFootball.read ("football.txt").decode ()
gmlFootball = zfFootball.read ("football.gml").decode ()
gmlFootball = gmlFootball.split ("\n") [1:]
```

```
GG_Football = nx.parse_gml (gmlFootball)

# Loading Dolphin Network Graph
zfDolphin = zipfile.ZipFile ("dolphins.zip")
txtDolphin = zfDolphin.read ("dolphins.txt").decode ()
gmlDolphin = zfDolphin.read ("dolphins.gml").decode ()
gmlDolphin = gmlDolphin.split ("\n") [1:]

GG_Dolphin = nx.parse_gml (gmlDolphin)

#Computing the Best Partition
partitionKarate = community_louvain.best_partition (GG_Karate)
partitionFootball = community_louvain.best_partition (GG_Football)
partitionDolphin = community_louvain.best_partition (GG_Dolphin)

# Drawing the Graph
posKarate = nx.spring_layout (GG_Karate)
posFootball = nx.spring_layout (GG_Football)
posDolphin = nx.spring_layout (GG_Dolphin)

# Colouring the Nodes according to their Partitions
plt.figure (figsize = (15, 15))
plt.title ("KARATE CLUB NETWORK")
cmap = cm.get_cmap ('viridis', max (partitionKarate.values ()) + 1)
nx.draw_networkx_nodes (GG_Karate, posKarate, partitionKarate.keys (),
node_size = 40, cmap = cmap, node_color = list (partitionKarate.values
()))
nx.draw_networkx_edges (GG_Karate, posKarate, alpha = 0.5)

plt.figure (figsize = (15, 15))
plt.title ("FOOTBALL CLUB NETWORK")
cmap = cm.get_cmap ('viridis', max (partitionFootball.values ()) + 1)
nx.draw_networkx_nodes (GG_Football, posFootball,
partitionFootball.keys (), node_size = 40, cmap = cmap, node_color =
list (partitionFootball.values ()))
nx.draw_networkx_edges (GG_Football, posFootball, alpha = 0.5)

plt.figure (figsize = (15, 15))
plt.title ("DOLPHIN NETWORK NETWORK")
cmap = cm.get_cmap ('viridis', max (partitionDolphin.values ()) + 1)
nx.draw_networkx_nodes (GG_Dolphin, posDolphin, partitionDolphin.keys
(), node_size = 40, cmap = cmap, node_color = list
(partitionDolphin.values ()))
nx.draw_networkx_edges (GG_Dolphin, posDolphin, alpha = 0.5)

plt.show ()
```

OUTPUT AND OBSERVATIONS (NETWORKX LIBRARY):







OUTPUT AND OBSERVATIONS (GEPHI):

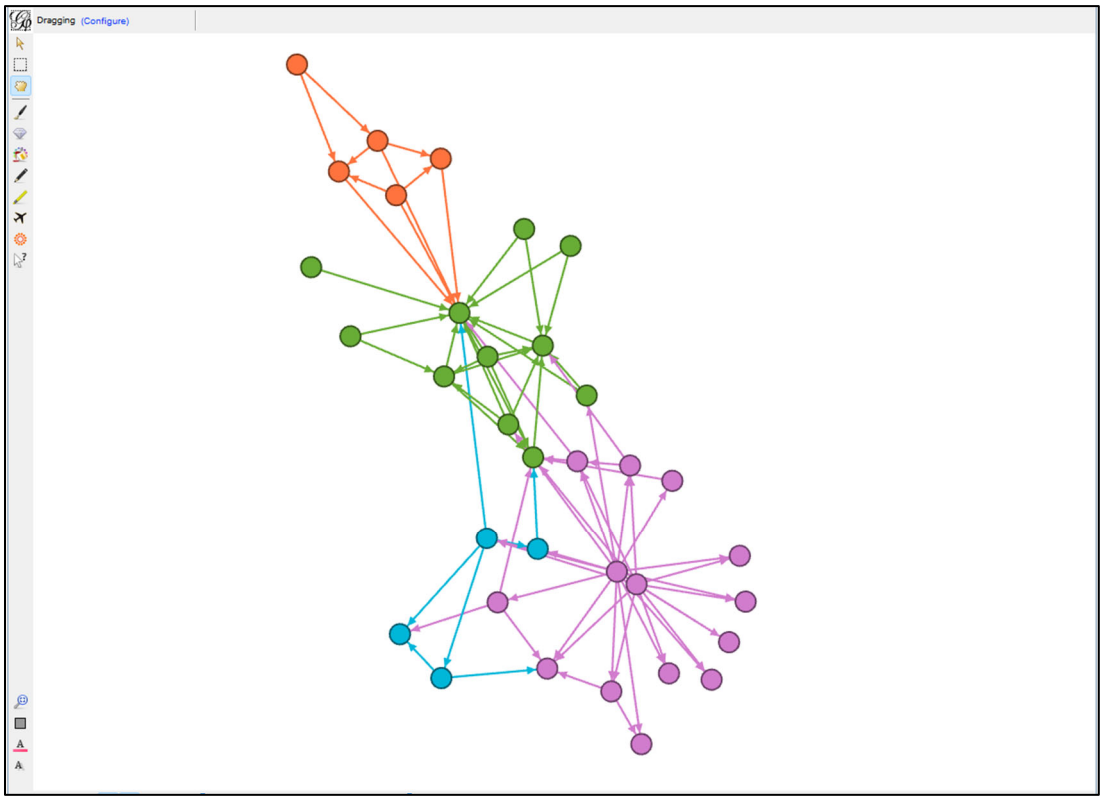


Fig. Zachary's Karate Club

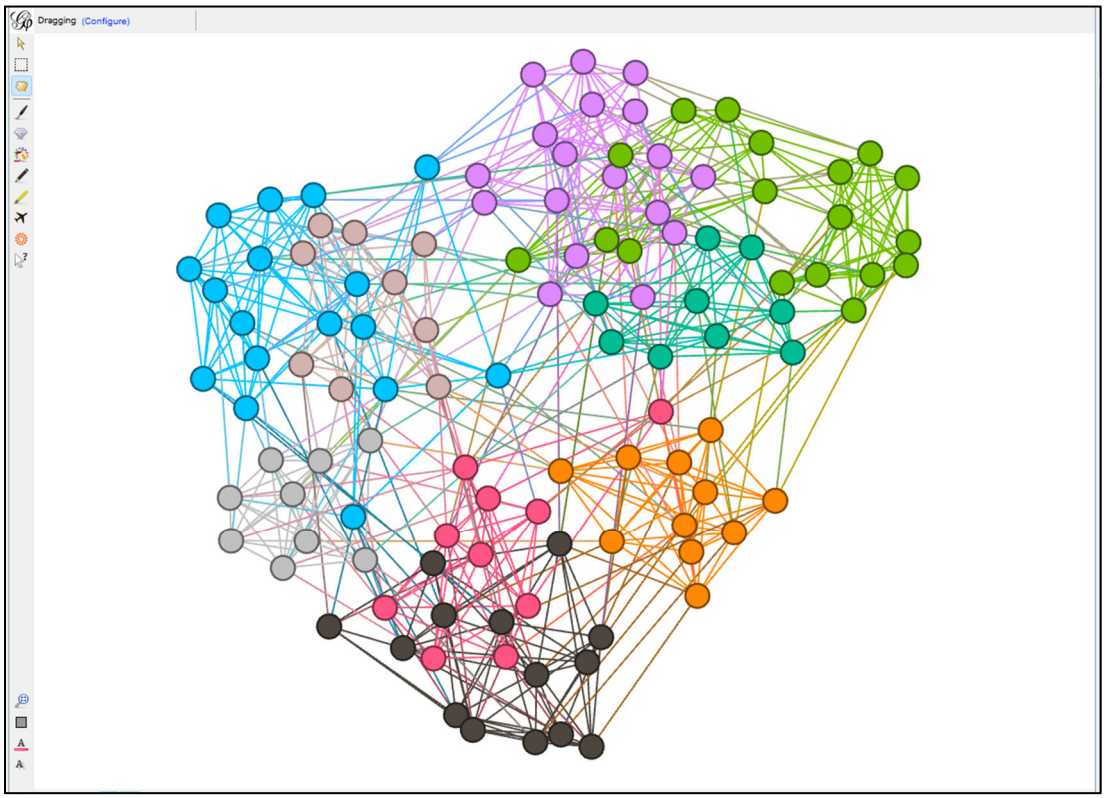


Fig. American College Football

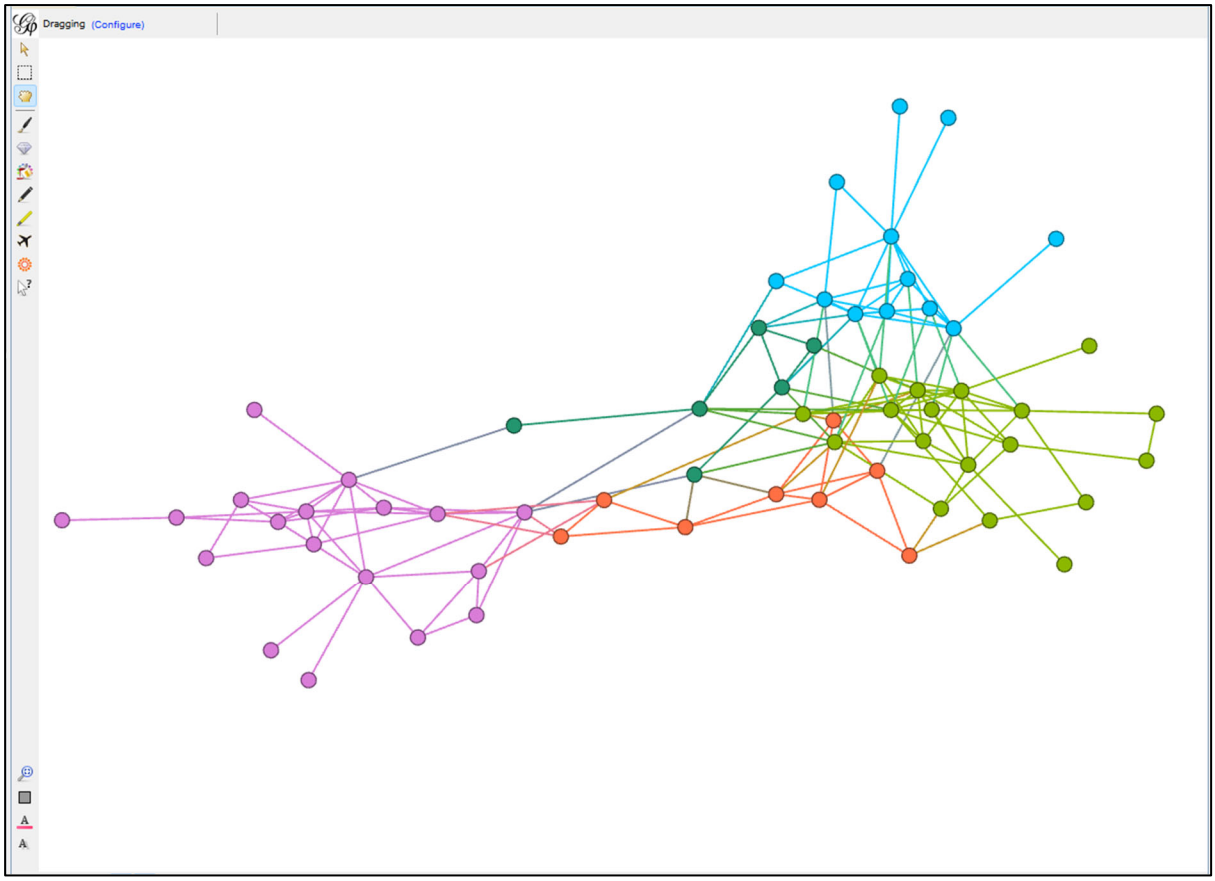


Fig.: Dolphin Social Network

AIM V: TO GENERATE OVERLAPPING COMMUNITIES WITH WLC ALGORITHM METHOD AND VISUALIZE THE GENERATED COMMUNITIES USING NETWORKX LIBRARY OR GEPHI. (ALTERNATIVELY, CAN ALSO USE ANY OF THE FOLLOWING TOOLS: MATPLOTLIB, PLOTLY, GGPLOT, SEABORN, BOKEH).

THEORY:

1. **Overlapping Community:** In network theory, overlapping community detection is one node having multiple community memberships in the networks, meaning, overlapping communities are possible if a node is a member of more than one community. Some of the algorithms for overlapping community detection are local seed selection algorithm, seed set expansion algorithm, speaker listener label propagation algorithm (SPLA) and WLC algorithm.

2. **WLC (Weighted Linear Combination) Algorithm:** It is a local algorithm for overlapping community detection based on clustering coefficient and common neighbour similarity**.**

   The clustering coefficient represents one of the most widespread functions of community's goodness evaluation. This measure is mainly intended to quantify the importance of nodes in a way that the nodes having high values are situated in the centre of the network, whereas the other ones are in the frontiers. There exist the global and the local versions of the clustering coefficient. The global version points us toward an overall indication of clustering in the network where the local version describes the embeddedness of nodes. In this work, we investigate the local clustering coefficient that represents the proportion of relations within the neighbourhoods set of a given node divided by the number of eventual relations between them.

$$Local\ Clustering, LC\ (u) = \frac{2\ |E\ (v,z) \in E\ /\ v,\ z\ \in neighbours\ (u)|}{|neighbours\ (u)|\ .\ |neighbours\ (u) - 1|}$$

   The common neighbours similarity measure is a widely used metric. It estimates the similarity between nodes based on the network structure. Moreover, when the nodes in question share a high number of common neighbours, they are more likely to belong to the same community. Many uses of this measure were proposed like Jaccard similarity and cosine similarity, but in this work, we use the naïve way to compute this metric.

$$Common\ Neighbours\ Similarity, S\ (u,v) = |neighbours\ (u)\ \cap\ nieghbours\ (v)|$$

   Other than the clustering coefficient and the common neighbours similarity measure, one more most importantly used metrics to conceive the new weighted belonging degree is the belonging degree. The belonging degree describes tightness of a node *u* with a community *c*.

$$Belonging\ Degree, B\ (u,c) = \frac{\sum_{v\ \in\ c} W_{u,v}}{k_u}$$

   When all the elements in the neighbourhood set of the node *u* are included in *c*, *B (u, c) = 1*, otherwise, *0 ≤ B (u , c) ≤ 1*.

   REAL WORLD NETWORK DATASETS: Zachary's Karate Club, American College Football and Dolphin Social Network.

CODE:

```
import networkx as nx
import time
import matplotlib.pyplot as plt
from collections import defaultdict

def graphPlot (path, title):

    GG = nx.read_gml (path, label = 'id')
    community = {}
    openResult = open ('results.txt', 'r')
    readLine = openResult.readlines ()

    ii = 0
    for line in readLine:
        aa = list (map (int, line.split ()))
        for xx in range(0, len(aa)):
            aa [xx] = aa [xx] + 1
        community [ii] = aa
        ii = ii + 1

    comDict = defaultdict (lambda: 0)
    comColour = dict ()

    for ii, com in community.items ():
        comColour |= {node: ii + 10 for node in com}
        for node in com:
            comDict [node] = comDict [node] + 1

    pos = nx.spring_layout (GG, k = 0.2, seed = 4572321)

    overlappedNodes = {node for node, n_comm in comDict.items() if n_comm > 1}
    nodeColour = [0 if nn in overlappedNodes else comColour [nn] for nn in GG]

    options = {
        "pos" : pos,
        "with_labels" : False,
        "node_color" : nodeColour,
        "node_size" : 250,
    }
    plt.figure (figsize = (15, 15))
    plt.title (title)
    nx.draw_networkx (GG, **options)
    plt.show ()
```

```
def modu1 (GG, NN, res):
    mm = 0
    for UU in res:
        nn = len (UU)
        SS = GG.subgraph (UU)
        rx = []
        for kx in res:
            if not kx == UU:
                rx.extend (kx)
        ov = list (set (UU).intersection (set (rx)))
        sum1 = 0
        ii = 0
        while ii < len (UU):
            jj = ii + 1
            while jj < len (UU):
                if UU [ii] in ov :
                    oo = SS.degree (UU [ii])
                    oo1 = 0
                    for lx in res:
                        if UU [ii] in lx:
                            SS1 = GG.subgraph (lx)
                            oo1 = oo1 + SS1.degree (UU [ii])
                    al1 = oo / oo1
                else:
                    al1 = 1
                if UU [jj] in ov:
                    ox = SS.degree (UU [jj])
                    ox1 = 0
                    for lx in res:
                        if UU [jj] in lx:
                            SS1 = GG.subgraph (lx)
                            ox1 = ox1 + SS1.degree (UU [jj])
                    al2 = ox / ox1
                else :
                    al2 = 1
                if GG.has_edge (UU [jj], UU [ii]):
                    xx = ((1 - ((GG.degree (UU [ii]) * GG.degree (UU [jj])) /
(2 * NN))) * al1 * al2)
                    sum1 = sum1 + 2 * xx
                else:
                    sum1 = sum1 + 2 * ((0 - ((GG.degree (UU [ii]) * GG.degree
(UU [jj])) / (2 * NN))) * al1 * al2)
                jj = jj + 1
            ii = ii + 1
        mm = mm + sum1
    mm = mm / (2 * NN)
    return (mm)
```

```
def WLC (path):
    tt = []
    tri = []
    GG = nx.read_gml (path, label = 'id')
    nx.write_edgelist (GG, 'tempEdgeList.txt', delimiter = ',')
    GG = nx.read_edgelist ('tempEdgeList.txt', comments = '#', delimiter =
',', nodetype = int, encoding = 'utf-8')
    ns = len (GG.nodes ())
    NN = GG.number_of_edges ()
    tt = []
    den = nx.density (GG)
    re = []
    res = []
    res1 = []
    res2 = []
    rx = []
    ww1 = []
    tps1 = time.time ()
    T11 = list (GG.nodes ())

    ii = 0
    while ii < len (T11):
        cpt1 = 0
        zx = list (GG.neighbors (T11 [ii]))
        aa = len (zx)
        jj = 0
        while jj < aa - 1:
            jj1 = jj + 1
            while jj1 < aa:
                if  GG.has_edge (zx [jj], zx [jj1]):
                   cpt1 = cpt1+1
                jj1 = jj1 + 1
            jj = jj + 1
        if aa > 1:
            ww1.append (2 * cpt1 / (aa * (aa - 1)))
        else:
            ww1.append (0)
        ii = ii + 1

    TT = GG.nodes ()
    while len (TT) > 0:
        nst = []
        SS = GG.subgraph (TT)
        for kk in TT:
            nst.append ([SS.degree (kk), kk])
        nst.sort (reverse = True)
        ll = nst [0][1]
        print ('PROCESSING NODE ', ll)
```

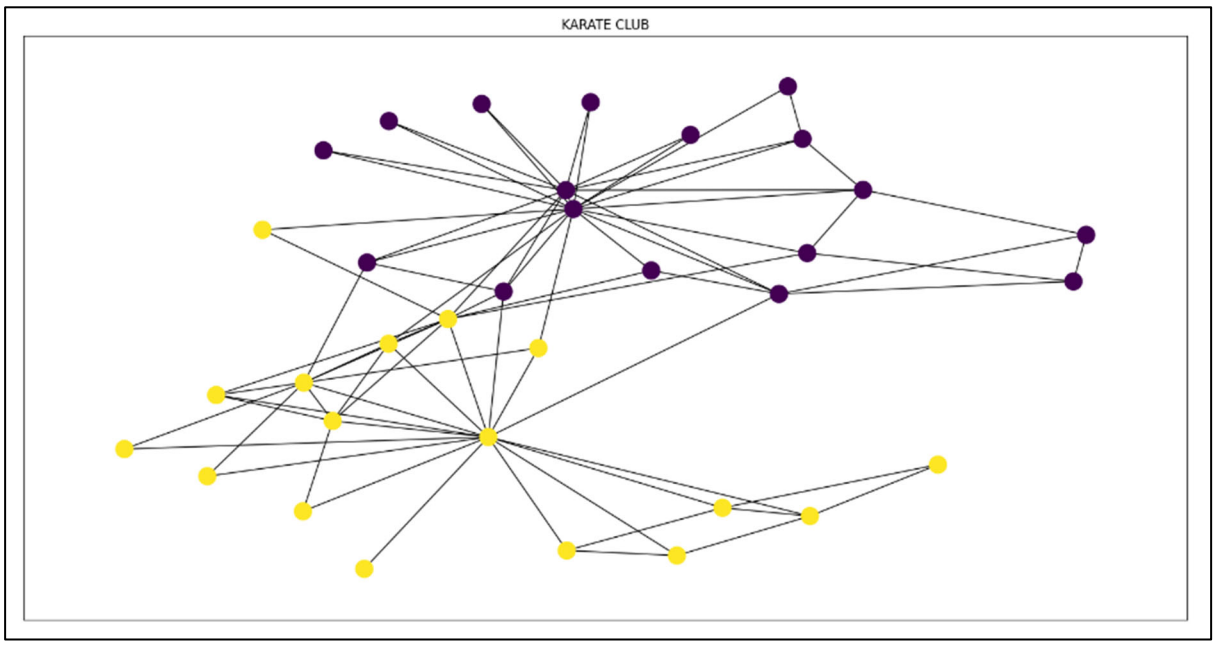
```
        ini = list (set (SS.neighbors (ll)))
        ini.append (ll)
        nn = len (ini)
        nn1 = len (ini)
        bb = True
        while bb == True:
            mm1 = []
            temp = -1
            for rr in ini:
                aa = ww1 [T11.index (rr)]
                xx = list (SS.neighbors (rr))
                wx1 = 0
                wx2 = 0
                if len (xx) > 0:
                    for rx1 in xx:
                        dd1 = ww1 [T11.index (rx1)]
                        dd = (dd1 + len (sorted (nx.common_neighbors (GG, rr,
rx1))))
                        wx1 = wx1 + dd
                        if rx1 in ini:
                            wx2 = wx2 + dd
                    if wx1 > 0:
                        bl = wx2 / wx1
                        if bl < 0.5:
                            ini.remove (rr)
            nn1 = len (ini)
            if  nn1 < nn:
                nn = nn1
                bb = True
            else:
                bb = False
        bb = 1
        while bb == 1:
            xx = []
            for kk in ini:
                xx.extend (GG.neighbors (kk))
                xx = list (set (xx) - set (ini))
            nn = len (ini)
            mm1 = []
            for rr in xx:
                xx1 = list (GG.neighbors (rr))
                wx1 = 0
                wx2 = 0
                if len (xx1) > 0:
                    for rx1 in xx1:
                        dd1 = ww1 [T11.index (rx1)]
                        dd = (dd1 + len (sorted (nx.common_neighbors (GG, rr,
rx1))))
```

```
                        wx1 = wx1 + dd
                        if rx1 in ini:
                            wx2 = wx2 + dd
                    if wx1 > 0:
                        bl = wx2 / wx1
                        if bl >= 0.4:
                            mm1.append (rr)
            ini.extend (mm1)
            nn1 = len (ini)
            if nn1 > nn:
                bb = 1
            else:
                bb = 0
                break
        res.append (ini)
        rx.extend (ini)
        TT = list (set (TT) - set (ini))
        if (len (ini) == 0):
            TT.remove (ll)
    tps2 = time.time ()

    mm = 0
    newFile = open ("results.txt", "w")
    for res1 in res:
        for kk in res1:
            newFile.write (str (kk-1))
            newFile.write (' ')
        newFile.write ('\n')
    newFile.close ()

    mm = modu1 (GG, NN, res)
    print ("OVERLAPPING MODULARITY: ", mm)

# KARATE CLUB
print ('\n\nKARATE CLUB\n')
WLC ('karate.gml')
graphPlot ('karate.gml', 'KARATE CLUB')

# FOOTBALL CLUB
print ('\n\nFOOTBALL CLUB\n')
WLC ("football.gml")
graphPlot ('football.gml', 'FOOTBALL CLUB')

# DOLPHIN NETWORK
print ("\n\nDOLPHINS NETWORK\n")
WLC ("dolphins.gml")
graphPlot ('dolphins.gml', 'DOLPHIN NETWORK')
```

OUTPUT AND OBSERVATIONS (NETWORKX LIBRARY):

// KARATE CLUB

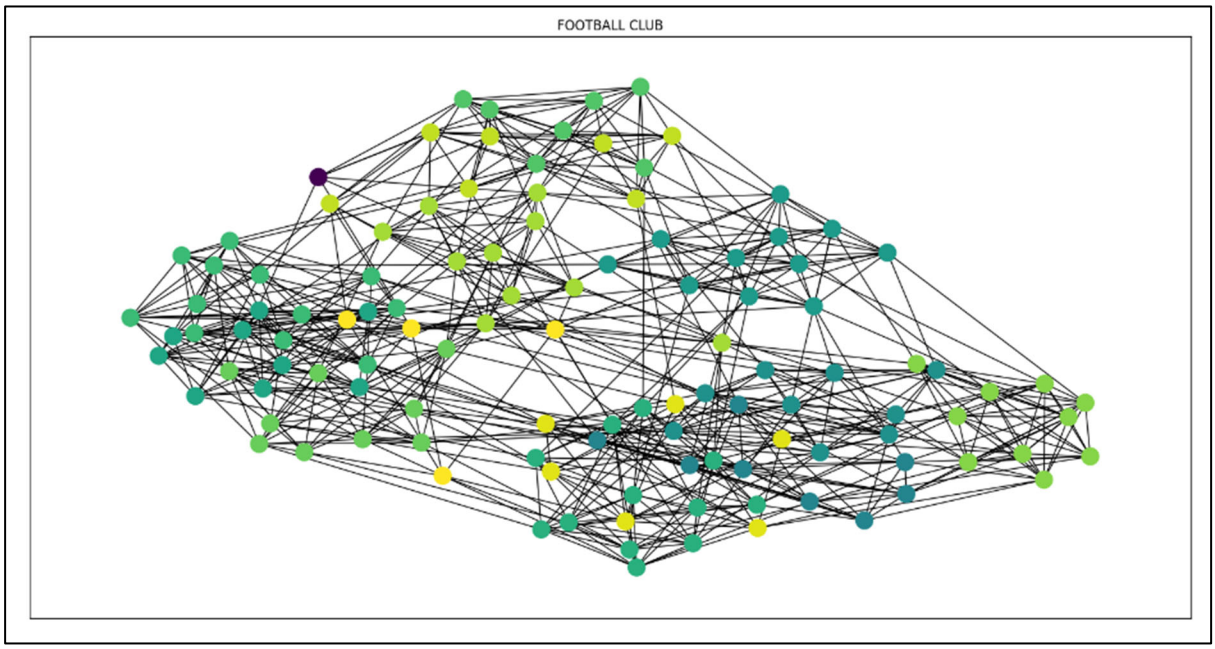
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                   Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab5.py"


KARATE CLUB

PROCESSING NODE  34
PROCESSING NODE  1
OVERLAPPING MODULARITY:  0.421597633136095
```



The Karate Club doesn't have any overlapping node.

// FOOTBALL CLUB

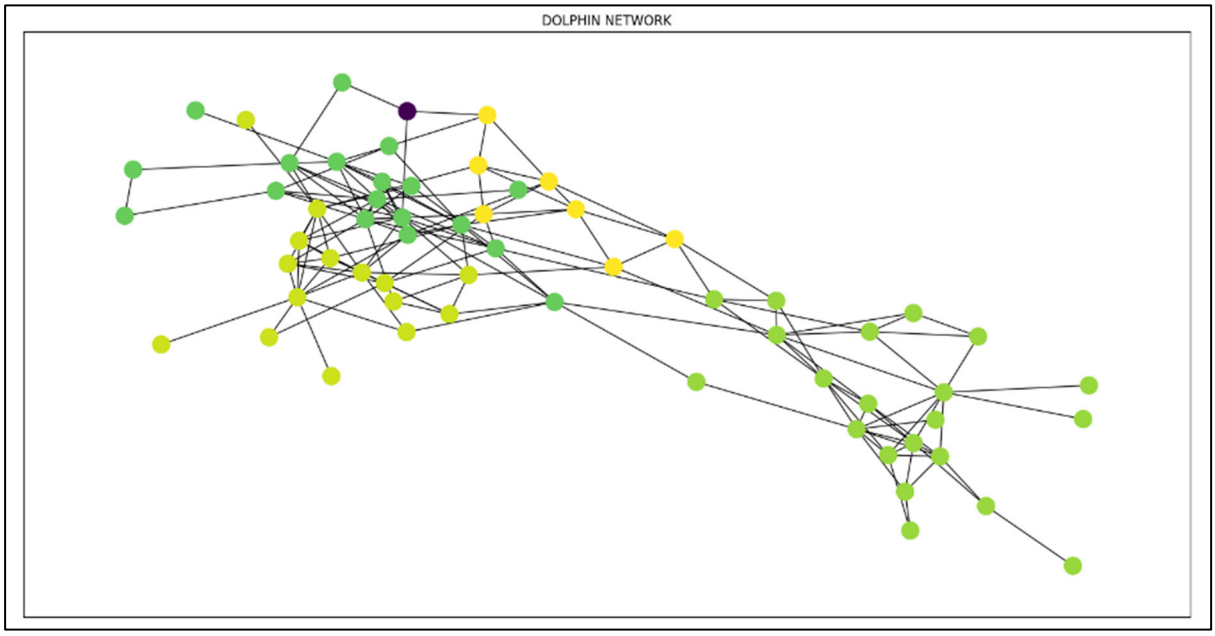
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    Code

FOOTBALL CLUB

PROCESSING NODE  104
PROCESSING NODE  88
PROCESSING NODE  6
PROCESSING NODE  109
PROCESSING NODE  98
PROCESSING NODE  76
PROCESSING NODE  34
PROCESSING NODE  91
PROCESSING NODE  78
PROCESSING NODE  94
PROCESSING NODE  43
PROCESSING NODE  11
PROCESSING NODE  97
OVERLAPPING MODULARITY:  0.5894918154504497
```



The overlapping node in football club has been highlighted with a distinct deep purple colour.

// DOLPHINS NETWORK

```
DOLPHINS NETWORK

PROCESSING NODE  14
PROCESSING NODE  57
PROCESSING NODE  51
PROCESSING NODE  47
OVERLAPPING MODULARITY:  0.5471302559234211
PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA>
```



The overlapping node in dolphin network has been highlighted with a distinct deep purple colour.

AIM VI: TO ANALYSE THE QUALITY OF COMMUNITIES DETECTED WITH WLC METHOD WITH FOLLOWING EXPERIMENTAL SETUP:

QUALITY MEASURES: ANUI, EXTENDED MODULARITY

THEORY:

1. **ANUI:** Average Normalised Unifiability and Isolability, in network theory, is a quality measure analysis of communities. This is evaluated as:

$$Q_{ANUI}\ (G,C) = \frac{Q_{AUI}\ (G,C)}{2} = \frac{Q_{AVI}\ (G,C)}{1 + Q_{AVU}(G,C) \times Q_{AVI}(G,C)}$$

$$where, Q_{AVI}(G,C) = \frac{1}{k}\sum_{i=1}^{k} Isolability\ (C_i)$$

$$and, Q_{AVI}(G,C) = \frac{1}{k}\sum_{i=1}^{k} Unifiability\ (C_i)$$

Unification is a property that unites the members of smaller communities into one community. Two communities are unified into single community if members are significantly connected.

$$Unifiability\ (C_i) = \sum_{j=1}^{k} Unifiability\ (C_i, C_j)$$

Isolation is a property that isolates the members of community from rest of the network and integrates the members of the community This means that connectivity of community with rest of the network should be less and connectivity within the community should be high.

$$Isolability\ (C_i) = \frac{\sum_{u \in C_i v} \delta\ (u,v)}{\sum_{u \in C_i v} \delta\ (u,v) + \sum_{u \in C_i, v \notin C_i} \delta\ (u,v)}$$

2. **Extended Modularity:** Extended modularity or extended overlapping modularity is a quality measure analysis of communities. This is given by,

$$EQ = \frac{1}{2m}\sum_{i}\sum_{v \in C_i, w \in C_i} \frac{1}{O_v O_w}\left[A_{vw} - \frac{k_v k_w}{2m}\right]$$

$where, O_v\ is\ the\ number\ of\ communities\ to\ which\ the\ node\ v\ belongs$
$k_i\ is\ the\ degree\ of\ this\ node$
$and, m\ is\ the\ total\ number\ of\ edges$

DATASETS: Zachary's Karate Club, LFR Graphs for Overlapping Communities.

CODE:

```
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import community as community_louvain
import matplotlib.cm as cm
import math
import networkx as nx
from networkx.algorithms.community import LFR_benchmark_graph
import matplotlib.pyplot as plt
from itertools import combinations

# ANUI
# STRENGTH OF NODE delta(u,v)
def delta (u,v):
  return math.matrix [u][v]

# UNIFIABILITY BETWEEN CLUSTER C[i] and C[j]
def unifiability (GG, Ci, Cj, mat):
    sum1, sum2, sum3 = 0, 0, 0
    for ii in Ci:
        for jj in Cj:
            sum1 += int (mat [[ii], [jj]])
    for ii in Ci:
        for jj in GG:
            sum2 += int (mat [[ii], [jj]])
        for jj in Cj:
            sum2 -= int (mat [[ii], [jj]])
    for ii in Cj:
        for jj in GG:
            sum3 += int (mat [[ii], [jj]])
        for jj in Ci:
            sum3 -= int (mat [[ii], [jj]])
    return sum1 / (sum2 + sum3 - sum1)

# AVERAGE UNIFIABILITY
def AVU (GG, cluster, mat):
    #CALLING UNIFIABILITY FOR ALL CLUSTERS
    sum_unifiability = 0
    for ii in cluster:
        for jj in cluster:
            if ii != jj:
                sum_unifiability += unifiability (GG, ii, jj, mat)
    return sum_unifiability / len (cluster)

# ISOLABILITY OF CLUSTER C[i]:
def isolability (GG, Ci, mat):
```

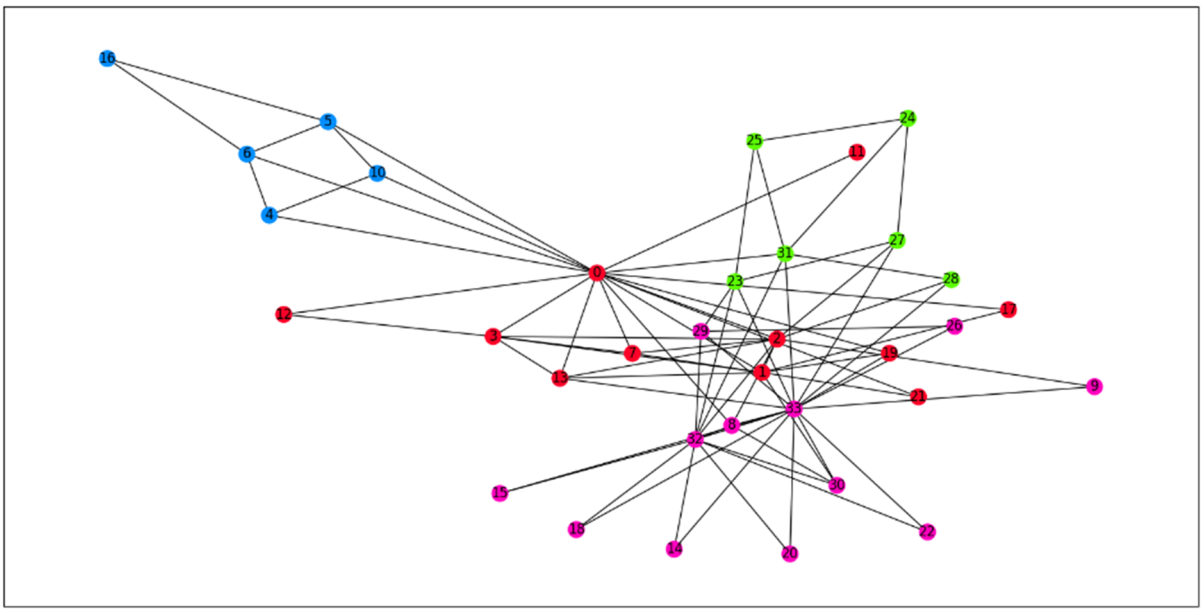
```
    sum1, sum2 = 0, 0
    for ii in Ci:
        for jj in Ci:
            sum1 += int (mat [[ii], [jj]])
    for ii in Ci:
        for jj in GG:
            if ii != jj:
                sum2 += int (mat [[ii], [jj]])
    return sum1 / (sum1 + sum2)

# AVERAGE ISOLABALITY
def AVI (GG, cluster, mat):
    sum = 0
    for ii in cluster:
        sum += isolability (GG, ii, mat)
    return sum / len (cluster)

# ENTENDED MODULARITY
alpha = {}

def ff (xx, pr = 30):
    return 2. * pr * xx - pr

def logistic (xx):
    bb = 1 + np.exp (-ff (xx))
    return 1.0 / bb

def logweight (ii, jj):
    return logistic (alpha [ii]) * logistic (alpha [jj])

def EQ (graph, communities, weight = 'weight', p=30, func = logweight):
    qq = 0.0
    degrees = dict (graph.degree (weight = weight))
    mm = sum (degrees.values ())
    nn = graph.number_of_nodes ()

    for nd in graph.nodes:
        alpha [nd] = 0

    for community in communities:
        for nd in community:
            alpha [int (nd)] = alpha [int (nd)] + 1

    for kk in alpha:
        alpha [kk] = 1./alpha [kk]

    for nd1, nd2 in combinations (graph.nodes, 2):
        if graph.has_edge (nd1, nd2):
```

```
            ee = graph [nd1][nd2]
            wt = ee.get (weight, 1)
        else:
            wt = 0
        for community in communities:
            beta_out = 0.0
            for jj in graph.nodes:
                if jj in community: continue
                if jj == nd1: continue
                if jj == nd2: continue
                beta_out = beta_out + func (nd1, jj)
            beta_out = beta_out / mm

            beta_in = 0.0
            for ii in graph.nodes:
                if ii in community: continue
                if ii == nd1: continue
                if ii == nd2: continue
                beta_in = beta_in + func (ii, nd2)
            beta_in = beta_in / mm

            qq = qq + func (nd1, nd2) * wt - float (beta_in * beta_out *
degrees [nd1] * degrees [nd2] / mm)
    return qq / mm

def readGraph (graphName):
    GG = nx.read_gml (graphName, label = 'id')
    partition = community_louvain.best_partition (GG)
    pos = nx.spring_layout (GG)
    cmap = cm.get_cmap ('rainbow', max (partition.values()) + 1)

    # GRAPH GG TO MATRIX
    mat = nx.to_numpy_matrix (GG)

    # CLUSTER FORMATION
    cluster = [[]]
    maxPartitionVal = 0
    for ii in partition:
        if partition[ii] > maxPartitionVal:
            maxPartitionVal = partition[ii]
    for ii in range (maxPartitionVal):
        cluster += [[]]
    for ii in partition:
        cluster [partition[ii]].append (ii)

    # CLUSTERS AND THEIR NODES
    print ("CLUSTER AND THEIR NODES")
    var = 1
```

```
    for ii in cluster:
        print ("Cluster ", var, "= ", ii)
        var += 1

    # AVERAGE UNIFIABILITY OF GRAPH
    AVU_G = AVU (GG, cluster, mat)
    print ("AVERGAE UNIFIABILITY = ", AVU_G)
    # AVERAGE ISOLABILITY
    AVI_G = AVI (GG, cluster, mat)
    print ("AVERAGE ISOLABILITY = ", AVI_G)
    # AVERAGE UNIFIABILITY AND ISOLABILITY
    AUI_G = (2 * AVI_G) / (1 + AVU_G * AVI_G)
    print ("AVERAGE UNIFIABILITY AND ISOLABILITY = ", AUI_G)
    # AVERAGE NORMALISED UNIFIABILITY AND ISOLABILITY
    ANUI_G = AUI_G / 2
    print ("AVERAGE NORMALISED UNIFIABILITY AND ISOLABILITY = ", ANUI_G)

    EQ_G = EQ (GG, cluster)
    print ("EXTENDED MODULARITY = ", EQ_G)

    # DISPLAY GRAPH
    plt.figure (figsize = (15, 15))
    nx.draw_networkx (GG, with_labels = True, node_size = 200, node_color =
list (partition.values()), cmap = plt.get_cmap ('gist_rainbow'))
    plt.show ()

# KARATE CLUB
print ('\nKARATE CLUB')
karate_Graph = nx.karate_club_graph ()
nx.write_gml (karate_Graph, "karate.gml")
readGraph ("karate.gml")

# LFR BENCHMARK
print ('\n\n\nLFR BENCHMARK')
# LFR_Graph = LFR_benchmark_graph (n = 1000, tau1 = 2, tau2 = 1.1, mu = 0.1,
min_degree = 20, max_degree = 50, max_iters = 2500, seed = 10)
# LFR_Graph = LFR_benchmark_graph (n = 250, tau1 = 3, tau2 = 1.5, mu = 0.1,
average_degree = 5, min_community = 20, seed = 10)
LFR_Graph = LFR_benchmark_graph (n = 100, tau1 = 3, tau2 = 1.5, mu = 0.25,
average_degree = 10, min_community = 5, seed = 10)
nx.set_node_attributes (LFR_Graph, {n: ','.join (map (str,
LFR_Graph.nodes[n]['community'])) for n in LFR_Graph.nodes()}, 'community')
nx.write_gml (LFR_Graph, "LFR_Overlapping.gml")
readGraph ("LFR_Overlapping.gml")
```

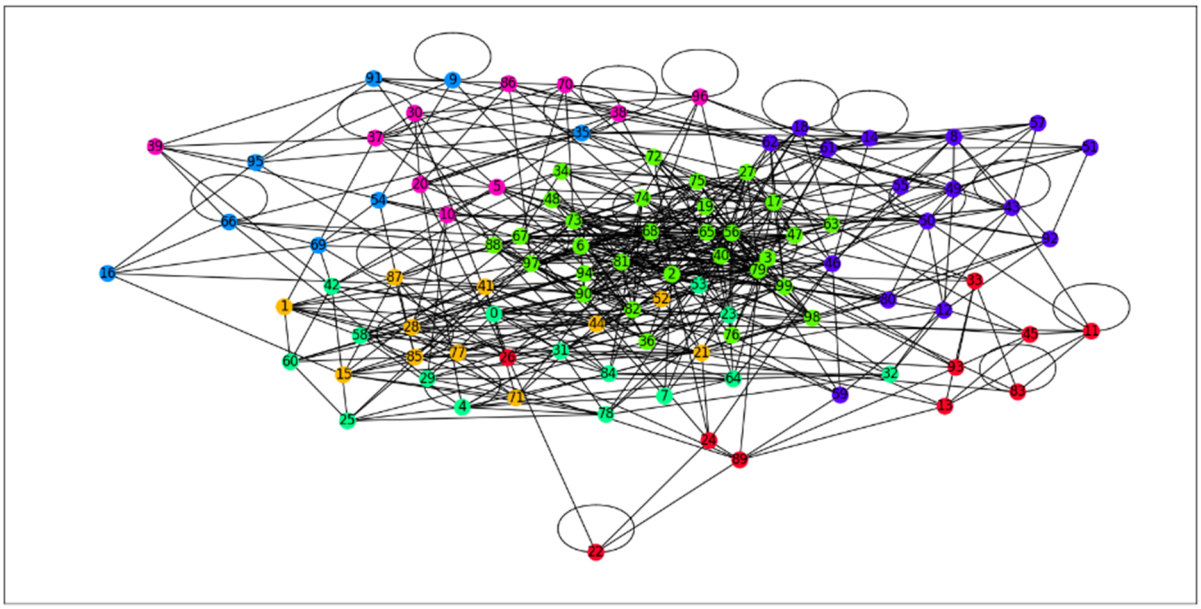
OUTPUT AND OBSERVATIONS:

// KARATE CLUB




```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                  Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab6.py"

KARATE CLUB
CLUSTER AND THEIR NODES
Cluster  1 =  [0, 1, 2, 3, 7, 11, 12, 13, 17, 19, 21]
Cluster  2 =  [4, 5, 6, 10, 16]
Cluster  3 =  [23, 24, 25, 27, 28, 31]
Cluster  4 =  [8, 9, 14, 15, 18, 20, 22, 26, 29, 30, 32, 33]
AVERGAE UNIFIABILITY =  0.14741413916146298
AVERAGE ISOLABILITY =  0.41488154348134487
AVERAGE UNIFIABILITY AND ISOLABILITY =  0.7819400955168019
AVERAGE NORMALISED UNIFIABILITY AND ISOLABILITY =  0.39097004775840094
EXTENDED MODULARITY =  1.9543796988983917
```

// LFR Graph



```
LFR BENCHMARK
CLUSTER AND THEIR NODES
Cluster  1 =  [11, 13, 22, 24, 26, 33, 45, 83, 89, 93]
Cluster  2 =  [1, 15, 21, 28, 41, 44, 52, 71, 77, 85, 87]
Cluster  3 =  [2, 3, 6, 17, 19, 27, 34, 36, 40, 47, 48, 56, 63, 65, 67, 68, 72, 73, 74, 75, 76, 79, 81, 82, 88, 90, 94, 97,
98, 99]
Cluster  4 =  [0, 4, 7, 23, 25, 29, 31, 32, 42, 53, 58, 60, 64, 78, 84]
Cluster  5 =  [9, 16, 35, 54, 66, 69, 91, 95]
Cluster  6 =  [8, 12, 14, 18, 43, 46, 49, 50, 51, 55, 57, 59, 61, 62, 80, 92]
Cluster  7 =  [5, 10, 20, 30, 37, 38, 39, 70, 86, 96]
AVERGAE UNIFIABILITY =  0.18377223440660925
AVERAGE ISOLABILITY =  0.36330886540635243
AVERAGE UNIFIABILITY AND ISOLABILITY =  0.6811406390678844
AVERAGE NORMALISED UNIFIABILITY AND ISOLABILITY =  0.3405703195339422
EXTENDED MODULARITY =  3.3153967509873463
```

AIM VII: TO ANALYSE THE ACCURACY OF COMMUNITIES DETECTED WITH WLC METHOD WITH FOLLOWING EXPERIMENTAL SETUP:

ACCURACY MEASURES: GENERALISED EXTERNAL INDEX

THEORY:

1. **Generalised External Index (GEI):** These generalised external indexes are used as general measures for comparing two partitions of the same data set into overlapping categories. If falls under the concept of Rand Index (RI), which further falls under the concept of ground-truth-based validation measures for overlapping community structure. GEI is calculated as follows:

$$GEI\ (\Psi, C) = \frac{a_G}{a_G + d_G}$$

$$where, \Psi = \{\Psi_1, \Psi_2, \Psi_3, \dots, \Psi_k\}\ is\ the\ set\ of\ detected\ communities$$

$$C = \{c_1, c_2, c_3, \dots, c_j\} is\ the\ set\ of\ ground - truth\ communities.$$

$$a_G\ is\ agreement\ associated\ to\ pair\ (i,j)\ calculated\ as:$$

$$a_G\ (i,j) = \min\ (\alpha_\Psi\ (i,j),\ \alpha_C\ (i,j)\} + \min\ \{\beta_\Psi\ (i),\ \beta_C\ (i)\} + \min\ \{\beta_\Psi\ (j),\ \beta_C\ (j)\}$$

$$d_G\ (i,j) = \text{abs}\ [\alpha_\Psi\ (i,j) - \alpha_C\ (i,j)] + \text{abs}\ [\beta_\Psi\ (i) - \beta_C\ (i)] + \text{abs}\ [\beta_\Psi\ (j) - \beta_C\ (j)]$$

Here, $\alpha_\Psi\ (i,j)$ represents number of communities shared by nodes i and j in partition Ψ
$\alpha_C\ (i,j)$ represents number of communities shared by nodes i and j in partition C
$\beta_\Psi\ (i)$ represents number of communities in Ψ that node i is a part of, minus 1
$\beta_C\ (i)$ represents number of communities in C that node i is a part of, minus 1
$\beta_\Psi\ (j)$ represents number of communities in Ψ that node j is a part of, minus 1
$\beta_C\ (j)$ represents number of communities in C that node j is a part of, minus 1

DATASETS: Zachary's Karate Club, LFR Graphs for Overlapping Communities.

CODE:

```
import networkx as nx
import numpy as np
import time
import matplotlib.pyplot as plt
from collections import defaultdict
from networkx.algorithms.community import LFR_benchmark_graph

def graphPlot (path, title):

    GG = nx.read_gml (path, label = 'id')
    community = {}
    openResult = open ('results.txt', 'r')
    readLine = openResult.readlines ()

    ii = 0
    for line in readLine:
        aa = list (map (int, line.split ()))
        for xx in range(0, len(aa)):
            aa [xx] = aa [xx] + 1
        community [ii] = aa
        ii = ii + 1

    comDict = defaultdict (lambda: 0)
    comColour = dict ()

    for ii, com in community.items ():
        comColour |= {node: ii + 10 for node in com}
        for node in com:
            comDict [node] = comDict [node] + 1

    pos = nx.spring_layout (GG, k = 0.2, seed = 4572321)

    overlappedNodes = {node for node, n_comm in comDict.items() if n_comm > 1}
    nodeColour = [0 if nn in overlappedNodes else comColour [nn] for nn in GG]

    options = {
        "pos" : pos,
        "with_labels" : False,
        "node_color" : nodeColour,
        "node_size" : 250,
    }

    plt.figure (figsize = (15, 15))
    plt.title (title)
    nx.draw_networkx (GG, **options)
    plt.show ()
```

```
def WLC (path):
    # path : the path of txt file containing edges of graph

    tt = []
    tri = []

    GG = nx.read_gml (path, label = 'id')
    nx.write_edgelist (GG, 'tempEdgeList.txt', delimiter = ',')
    GG = nx.read_edgelist ('tempEdgeList.txt', comments = '#', delimiter =
',', nodetype = int, encoding = 'utf-8')
    ns = len (GG.nodes ())
    NN = GG.number_of_edges ()

    tt = []
    den = nx.density (GG)
    re = []
    res = []
    res1 = []
    res2 = []
    rx = []
    ww1 = []

    tps1 = time.time ()
    T11 = list (GG.nodes ())

    ii = 0
    while ii < len (T11):
        cpt1 = 0
        zx = list (GG.neighbors (T11 [ii]))
        aa = len (zx)
        jj = 0
        while jj < aa - 1:
            jj1 = jj + 1
            while jj1 < aa:
                if  GG.has_edge (zx [jj], zx [jj1]):
                   cpt1 = cpt1+1
                jj1 = jj1 + 1
            jj = jj + 1

        if aa > 1:
            ww1.append (2 * cpt1 / (aa * (aa - 1)))
        else:
            ww1.append (0)
        ii = ii + 1

    TT = GG.nodes ()
    while len (TT) > 0:
        nst = []
```

```
        SS = GG.subgraph (TT)
        for kk in TT:
            nst.append ([SS.degree (kk), kk])

        nst.sort (reverse = True)
        ll = nst [0][1]
        print ('PROCESSING NODE ', ll)
        ini = list (set (SS.neighbors (ll)))
        ini.append (ll)
        nn = len (ini)
        nn1 = len (ini)
        bb = True

        while bb == True:
            mm1 = []
            temp = -1
            for rr in ini:
                aa = ww1 [T11.index (rr)]
                xx = list (SS.neighbors (rr))
                wx1 = 0
                wx2 = 0
                if len (xx) > 0:
                    for rx1 in xx:
                        dd1 = ww1 [T11.index (rx1)]
                        dd = (dd1 + len (sorted (nx.common_neighbors (GG, rr,
rx1))))
                        wx1 = wx1 + dd
                        if rx1 in ini:
                            wx2 = wx2 + dd
                    if wx1 > 0:
                        bl = wx2 / wx1
                        if bl < 0.5:
                            ini.remove (rr)

            nn1 = len (ini)
            if  nn1 < nn:
                nn = nn1
                bb = True
            else:
                bb = False

        bb = 1
        while bb == 1:
            xx = []
            for kk in ini:
                xx.extend (GG.neighbors (kk))
                xx = list (set (xx) - set (ini))
```

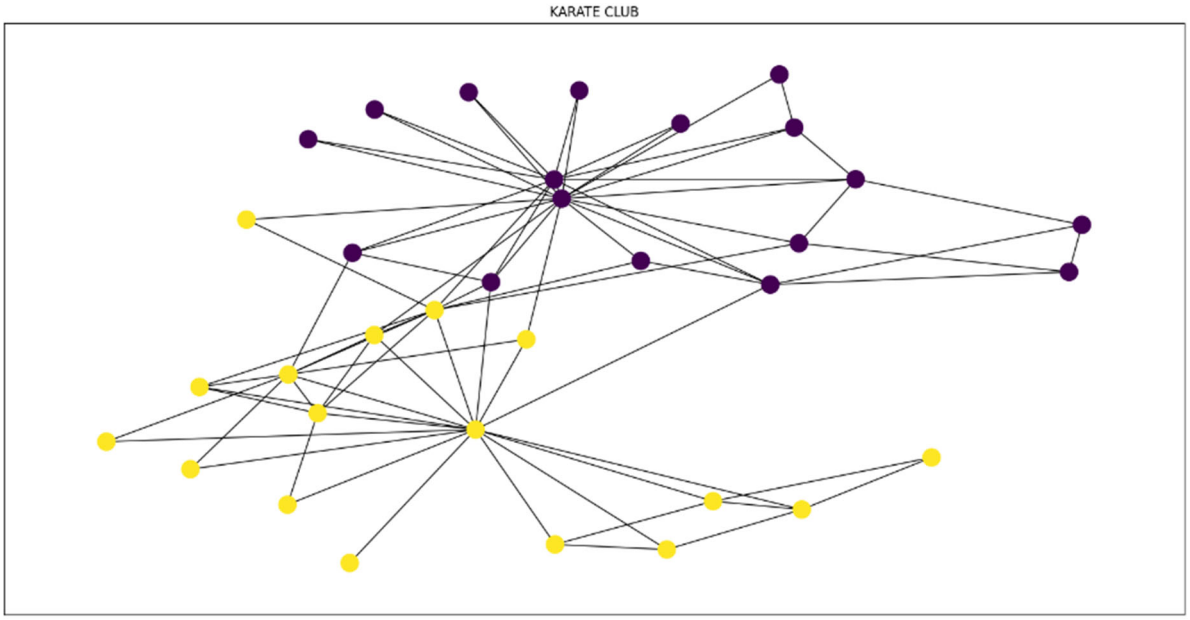
```
            nn = len (ini)
            mm1 = []

for rr in xx:
                xx1 = list (GG.neighbors (rr))
                wx1 = 0
                wx2 = 0
                if len (xx1) > 0:
                    for rx1 in xx1:
                        dd1 = ww1 [T11.index (rx1)]
                        dd = (dd1 + len (sorted (nx.common_neighbors (GG, rr,
rx1))))
                        wx1 = wx1 + dd
                        if rx1 in ini:
                            wx2 = wx2 + dd
                    if wx1 > 0:
                        bl = wx2 / wx1
                        if bl >= 0.4:
                            mm1.append (rr)

            ini.extend (mm1)
            nn1 = len (ini)
            if nn1 > nn:
                bb = 1
            else:
                bb = 0
                break

        res.append (ini)
        rx.extend (ini)
        TT = list (set (TT) - set (ini))
        if (len (ini) == 0):
            TT.remove (ll)
    tps2 = time.time ()

    newFile = open ("results.txt", "w")
    for res1 in res:
        for kk in res1:
            newFile.write (str (kk-1))
            newFile.write (' ')
        newFile.write ('\n')
    newFile.close ()
```

```
def GEI (graph, attr):
    nn = graph.number_of_nodes ()
    gt_membership = [graph.nodes [v][attr] for v in graph.nodes ()]
    gt_communities = {}
    for ii in range (0, nn):
        if gt_membership [ii] not in gt_communities:
            gt_communities [gt_membership [ii]] = []
        gt_communities [gt_membership [ii]].append (ii)
    communities = {}
    ii = 0
    file = open ('results.txt', 'r')
    Lines = file.readlines ()
    for line in Lines:
        aa = list (map (int, line.split ()))
        if len (aa):
            communities [ii] = aa
            ii = ii + 1
    num = len (communities)

    alpha_psi = []
    for ii in range (0, nn + 1):
        alpha_psi.append (list (np.zeros (nn + 1, dtype = int)))

    for ii in range (0, nn):
        for jj in range (ii + 1, nn):
            for cc in range (0, num):
                if ii in communities [cc] and jj in communities [cc]:
                    alpha_psi [ii][jj] += 1
    alpha_gt = []
    for ii in range (0, nn + 1):
        alpha_gt.append (list (np.zeros (nn + 1, dtype = int)))

    for ii in range (0, nn):
        for jj in range (ii + 1, nn):
            for cc in gt_communities.keys ():
                if ii in gt_communities [cc] and jj in gt_communities [cc]:
                    alpha_gt [ii][jj] += 1
    beta_psi = [0 for ii in range (0, nn)]
    for ii in range (0, nn):
        for cc in range (0, num):
            if ii in communities [cc]:
                beta_psi [ii] += 1
        beta_psi [ii] -= 1
    beta_gt = [0 for ii in range (0, nn)]
    for ii in range (0, nn):
        for cc in gt_communities.keys ():
            if ii in gt_communities [cc]:
                beta_gt [ii] += 1
```

```
        beta_gt [ii] -= 1
    aa_GG = []
    for ii in range(0, nn + 1):
        aa_GG.append (list (np.zeros (nn + 1, dtype = int)))
    for ii in range (0, nn):
        for jj in range (ii + 1, nn):
            aa_GG [ii][jj] = min (alpha_psi [ii][jj], alpha_gt [ii][jj]) + min
(beta_psi [ii], beta_gt [ii]) + min (beta_psi [jj], beta_gt [jj])
    dd_GG = []
    for ii in range (0, nn + 1):
        dd_GG.append (list (np.zeros (nn + 1, dtype = int)))
    for ii in range (0, nn):
        for jj in range (ii + 1, nn):
            dd_GG [ii][jj] = abs (alpha_psi [ii][jj] - alpha_gt [ii][jj]) +
abs (beta_psi [ii] - beta_gt [ii]) + abs (beta_psi [jj] - beta_gt [jj])
    aa = np.sum (aa_GG)
    dd = np.sum (dd_GG)
    GEI = aa / (aa + dd)
    return GEI

# KARATE CLUB
print ('\nKARATE CLUB')
karate_Graph = nx.karate_club_graph ()
nx.write_edgelist (karate_Graph, "karateedgelist.txt", delimiter = ',')
nx.write_gml (karate_Graph, "karate.gml")
WLC ('karate.gml')
print ("GEI Value = ", GEI (karate_Graph, 'club'))
graphPlot ('karate.gml', 'KARATE CLUB')

# LFR BENCHMARK
print ('\n\n\nLFR BENCHMARK')
LFR_Graph = LFR_benchmark_graph (n = 1000, tau1 = 2, tau2 = 1.1, mu = 0.1,
min_degree = 20, max_degree = 50, max_iters = 2500, seed = 10)
# LFR_Graph = LFR_benchmark_graph (n = 250, tau1 = 3, tau2 = 1.5, mu = 0.1,
average_degree = 5, min_community = 20, seed = 10)
# LFR_Graph = LFR_benchmark_graph (n = 100, tau1 = 3, tau2 = 1.5, mu = 0.25,
average_degree = 10, min_community = 5, seed = 10)
nx.set_node_attributes (LFR_Graph, {n: ','.join (map (str,
LFR_Graph.nodes[n]['community'])) for n in LFR_Graph.nodes()}, 'community')
nx.write_edgelist (LFR_Graph, "lfredgelist.txt", delimiter = ',')
nx.write_gml (LFR_Graph, "LFR_Overlapping.gml")
WLC ('LFR_Overlapping.gml')
print ("GEI Value = ", GEI (LFR_Graph, 'community'))
graphPlot ('LFR_Overlapping.gml', 'LFR BENCHMARK')
```

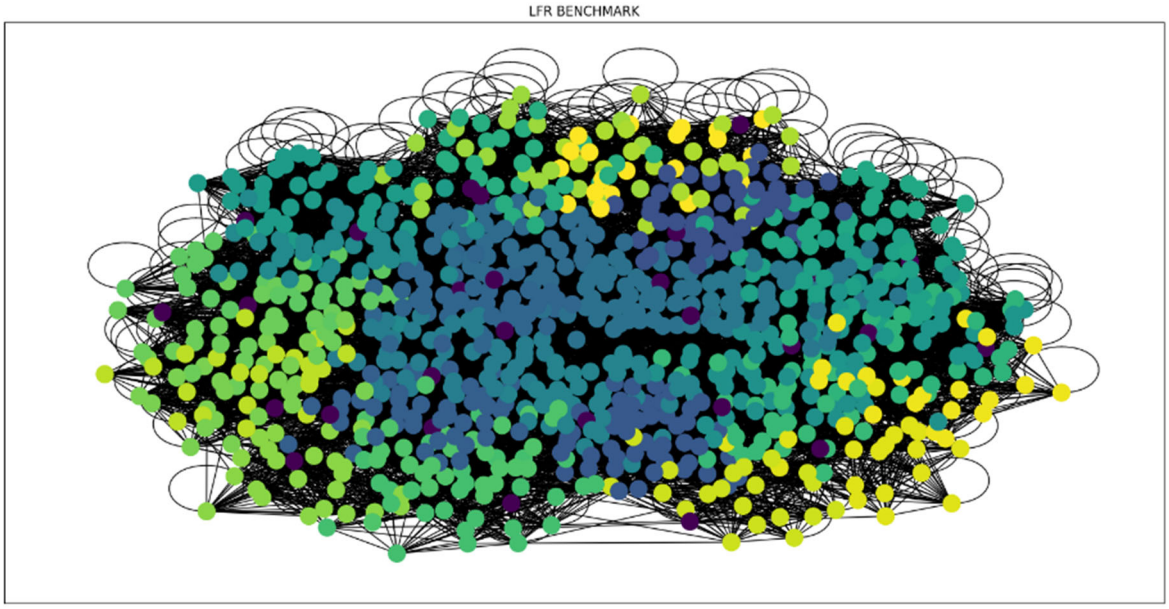
OUTPUT AND OBSERVATIONS:

// KARATE CLUB



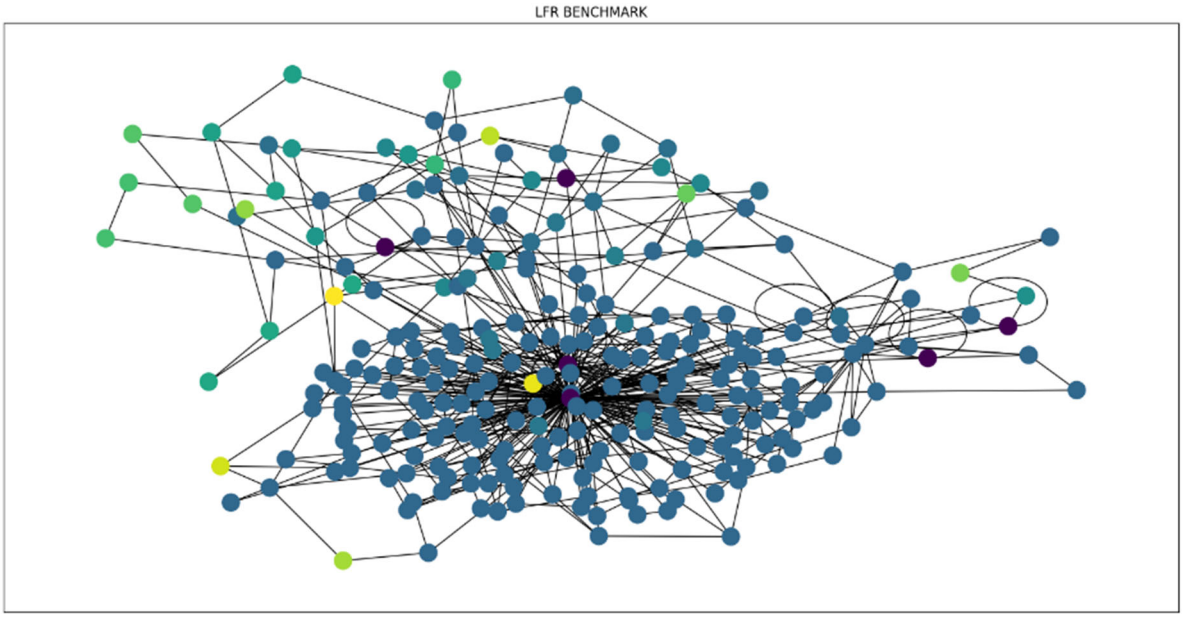

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab7WLC.py"

KARATE CLUB
PROCESSING NODE  33
PROCESSING NODE  0
GEI Value =  0.3026315789473684
```

// LFR BENCHMARK GRAPH FOR n = 1000




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  Code

LFR BENCHMARK
PROCESSING NODE  577
PROCESSING NODE  655
PROCESSING NODE  346
PROCESSING NODE  541
PROCESSING NODE  872
PROCESSING NODE  621
PROCESSING NODE  879
PROCESSING NODE  904
PROCESSING NODE  451
PROCESSING NODE  819
PROCESSING NODE  945
PROCESSING NODE  790
PROCESSING NODE  618
PROCESSING NODE  669
PROCESSING NODE  32
PROCESSING NODE  909
PROCESSING NODE  774
PROCESSING NODE  800
PROCESSING NODE  899
PROCESSING NODE  180
PROCESSING NODE  274
PROCESSING NODE  38
PROCESSING NODE  954
PROCESSING NODE  970
PROCESSING NODE  782
PROCESSING NODE  409
PROCESSING NODE  286
PROCESSING NODE  918
PROCESSING NODE  974
PROCESSING NODE  299
PROCESSING NODE  783
GEI Value =  -0.013394877364467372

⊗ 0 ⚠ 0   Ln 264, Col 37  Spaces: 4  UTF-8  CRLF  Python  3.10.2 64-bit
```

// LFR BENCHMARK GRAPH FOR n = 250




```
LFR BENCHMARK
PROCESSING NODE  73
PROCESSING NODE  93
PROCESSING NODE  4
PROCESSING NODE  169
PROCESSING NODE  158
PROCESSING NODE  18
PROCESSING NODE  175
PROCESSING NODE  106
PROCESSING NODE  40
PROCESSING NODE  16
PROCESSING NODE  193
PROCESSING NODE  129
PROCESSING NODE  69
PROCESSING NODE  248
PROCESSING NODE  247
PROCESSING NODE  239
PROCESSING NODE  180
PROCESSING NODE  153
PROCESSING NODE  136
PROCESSING NODE  127
PROCESSING NODE  76
GEI Value =  0.3631352351828753
PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA>
```

AIM VIII: TO SHOW THE NUMBER OF PREDICTED LINKS WITH FOLLOWING LINK PREDICTION METHODS:

PREDICTION METHODS: COSINE SIMILARITY OR SALTON INDEX, ADAMIC-ADAR INDEX, JACCARD COEFFICIENT, PREFERENTIAL ATTACHMENT, RESOURCE ALLOCATION INDEX

THEORY:

Proximity-based methods for link prediction are the vertex proximity or the measures of similarity. These similarity measures give similarity score between two vertices. Some measures are symmetrical by definition, some have to be modified to achieve symmetry. Similarity measures can be divided into local, global and quasi-local measures. Local methods focus only on the properties of neighbourhoods of a given pair of vertices. Global methods benefit from information contained in the whole graph. Quasi-local methods make use of more information than local methods, but still do not need information about the whole graph.

In this experiment, we'll be using local methods of similarity measure. These measures are mainly variations on common neighbours and are listed as follows:

For a graph *G* defined by a pair (*V*, *E*), where vertex set *V* = {1, 2, …, x, y, …, N} and edge set *E = {(x, y) : x, y ∈ V}*; *x* and *y* are some generic vertices of interest.

1. **Salton Index (Cosine Similarity):** It measures the cosine of the angle between columns of the adjacency matrix, corresponding to the given vertices. The measure is commonly used in information retrieval. This measure is given by:
$$S_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{k_x \times k_y}}$$
2. **Adamic-Adar Index:** This measure develops simple counting of common neighbours by assigning weights to nodes inversely proportional to their degrees. That means that a common neighbour, which is unique for a few nodes only, is more important than a hub. This measure is given by:
$$S_{xy} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log\ k_z}$$
3. **Jaccard Index:** This measure measures the proportion of common neighbours in the total number of neighbours. It reaches its maximum if $\Gamma(x) = \Gamma(y)$, namely, all neighbours are common to both vertices. This measure is given by:
$$S_{xy} = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$
4. **Preferential Attachment:** This measure was developed as a model of the growth of network in a sense of new nodes emerging. This measure is given by:
$$S_{xy} = k_x \cdot k_y$$
5. **Resource Allocation Index:** This measure is motivated by a resource allocation process. It measures how much resource is transmitted between *x* and *y*. This measure is given by:
$$S_{xy} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{k_z}$$

In the above formulas, $S_{xy} : G, x, y \rightarrow \Re$ is the proximity measure.
$\Gamma(x) = \{y : (x, y) \in E\}$ is a set of neighbours of vertex *x*.
$k(x) = |\Gamma(x)|$ is the degree of vertex *x* in graph G.

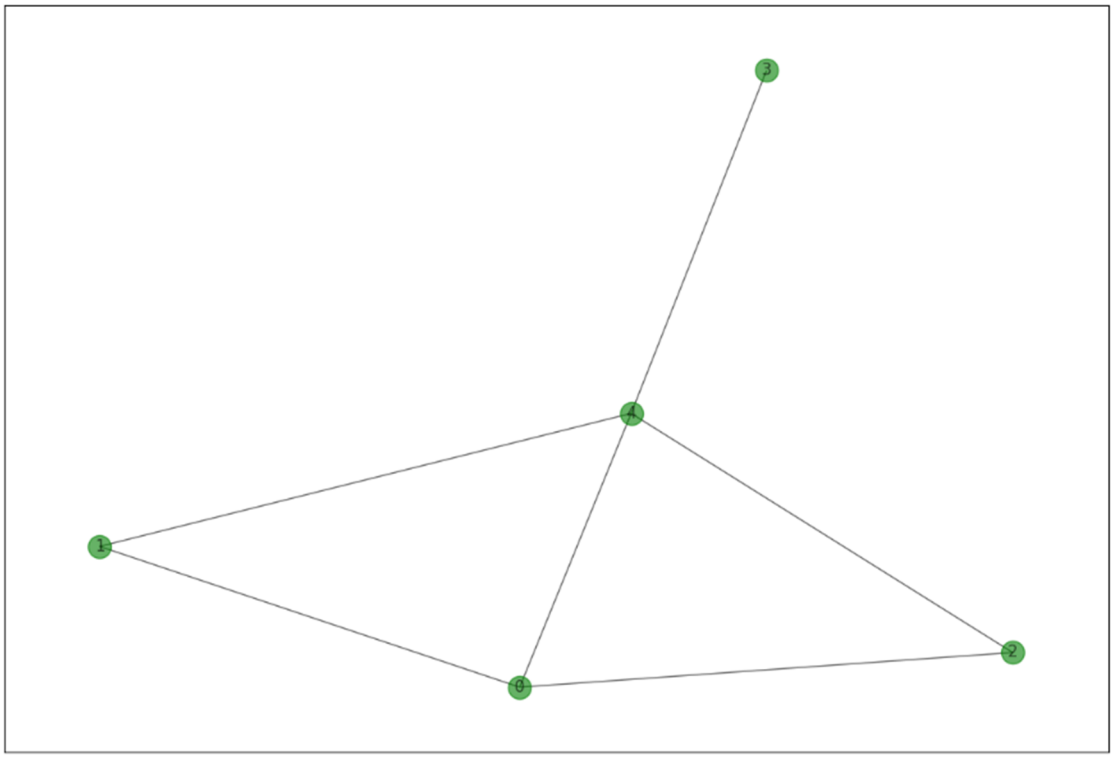
CODE:

```
from turtle import color
import networkx as nx
import matplotlib.pyplot as plt
from pyparsing import alphas
import numpy as np
from scipy import spatial

def printPreds (index, preds):
    print ("\n", index)
    for ii in preds:
        print ("(", ii [0], ", ", ii [1], ") = ", round (ii [2], 3))

def findUnconnected (list, numOfNodes):
    unconnectedList = [[]]
    var = 0
    for ii in range (numOfNodes):
        for jj in range (ii+1, numOfNodes):
            if jj not in list[ii]:
                unconnectedList[var].append (ii)
                unconnectedList[var].append (jj)
                var += 1
                unconnectedList += [[]]
    unconnectedList.remove ([])
    return (unconnectedList)

def saltonList (GG):
    edges = nx.edges (GG)
    numOfNodes = GG.number_of_nodes()
    list = [[]]
    for ii in range (numOfNodes-1):
        list += [[]]
    for ii in range (numOfNodes):
        for jj in edges:
            if (jj[0] == ii):
                list [ii].append (jj[1])
            if (jj[1] == ii):
                list [ii].append (jj[0])
    unconnectedNodes = findUnconnected (list, numOfNodes)

    adjMatrix = [[]]
    var = 0
    for ii in range (numOfNodes):
        for jj in range (numOfNodes):
            if jj not in list [ii]:
                adjMatrix[var].append (0)
            else:
                adjMatrix[var].append (1)
```

```
        var += 1
        adjMatrix += [[]]
    adjMatrix.remove ([])

    var = 0
    for ii in unconnectedNodes:
        cosineSimilarity = 1 - spatial.distance.cosine (adjMatrix [ii[0]],
adjMatrix [ii[1]])
        unconnectedNodes[var].append (round (cosineSimilarity, 3))
        var += 1
    return unconnectedNodes

# MAKING A RANDOM GRAPH OF FEW NODES
numOfNodes = 5
GG = nx.erdos_renyi_graph (numOfNodes, p = 0.6)
# GG = nx.karate_club_graph ()
edges = nx.edges (GG)
print ("GRAPH EDGE SET:\n", edges)

# SALTON INDEX (COSINE SIMILARITY)
preds = saltonList (GG)
printPreds ("SALTON INDEX", preds)

# ADAMIC-ADAR INDEX
preds = nx.adamic_adar_index (GG)
printPreds ("ADAMIC-ADAR INDEX", preds)
# with open ('adamicadar.txt', 'w') as file:
#     for ii in preds:
#         file.write (str (ii))
#         file.write ('\n')

# JACCARD COEFFICIENT
preds = nx.jaccard_coefficient (GG)
printPreds ("JACCARD COEFFICIENT", preds)

# PREFERENTIAL ATTACHMENT
preds = nx.preferential_attachment (GG)
printPreds ("PREFERENTIAL ATTACHMENT", preds)

# RESOURCE ALLOCATION INDEX
preds = nx.resource_allocation_index (GG)
printPreds ("RESOURCE ALLOCATION INDEX", preds)

# GRAPH PLOTTING
plt.figure (figsize = (15, 15))
nx.draw_networkx (GG, with_labels = 'True', node_color = 'green', alpha = 0.6)
plt.show ()
```

OUTPUT AND OBSERVATIONS:

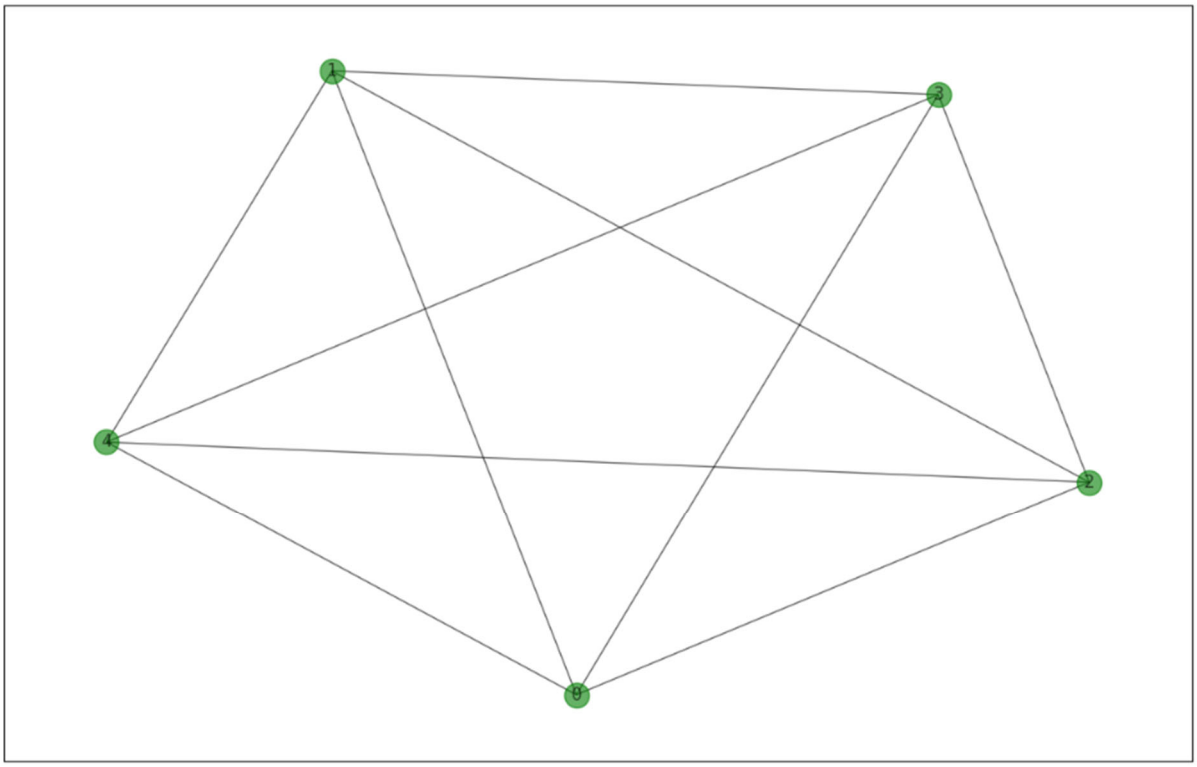
For n = 5, p = 0.6




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER                    Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab8.py"
GRAPH EDGE SET:
 [(0, 1), (0, 2), (0, 4), (1, 4), (2, 4), (3, 4)]

 SALTON INDEX
( 0 ,  3 ) =  0.882
( 1 ,  2 ) =  1
( 1 ,  3 ) =  0.707
( 2 ,  3 ) =  0.707

 ADAMIC-ADAR INDEX
( 0 ,  3 ) =  0.721
( 1 ,  2 ) =  1.632
( 1 ,  3 ) =  0.721
( 2 ,  3 ) =  0.721

 JACCARD COEFFICIENT
( 0 ,  3 ) =  0.333
( 1 ,  2 ) =  1.0
( 1 ,  3 ) =  0.5
( 2 ,  3 ) =  0.5

 PREFERENTIAL ATTACHMENT
( 0 ,  3 ) =  3
( 1 ,  2 ) =  4
( 1 ,  3 ) =  2
( 2 ,  3 ) =  2

 RESOURCE ALLOCATION INDEX
( 0 ,  3 ) =  0.25
( 1 ,  2 ) =  0.583
( 1 ,  3 ) =  0.25
( 2 ,  3 ) =  0.25
PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA>

⊗ 0 ⚠ 0   Ln 22, Col 33  Spaces: 4  UTF-8  CRLF  Python  3.10.2 64-bit
```

For n = 5.

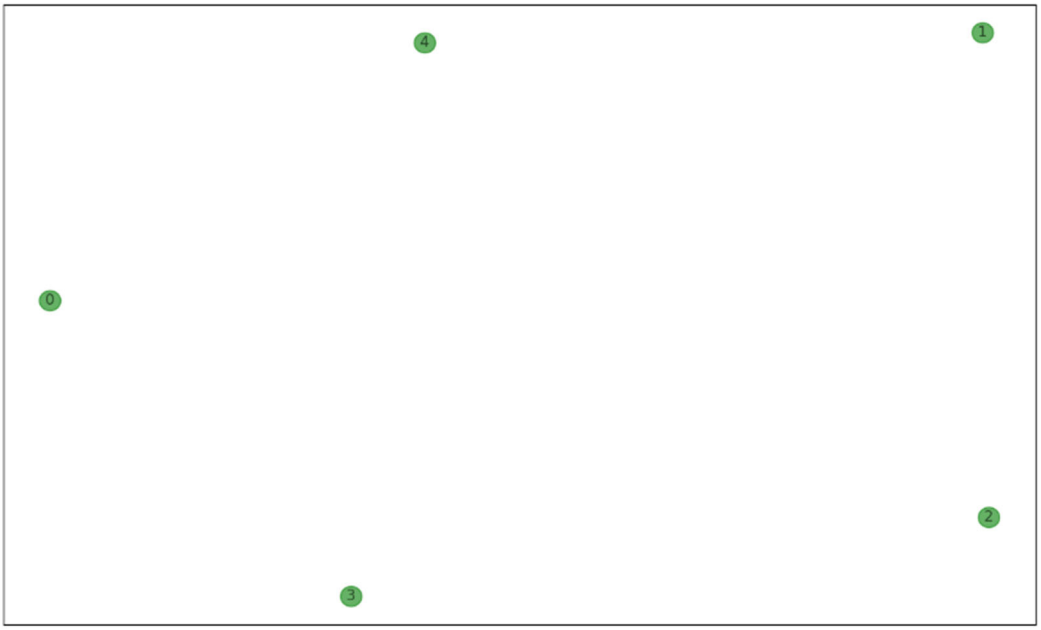
When all the nodes are connected to one another.




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER                    Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab8.py"
GRAPH EDGE SET:
 [(0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]

 SALTON INDEX

 ADAMIC-ADAR INDEX

 JACCARD COEFFICIENT

 PREFERENTIAL ATTACHMENT

 RESOURCE ALLOCATION INDEX
```

For n = 5,

When none of the nodes are connected to any of the other nodes.




```
 SALTON INDEX
( 0 ,  1 ) =  1
( 0 ,  2 ) =  1
( 0 ,  3 ) =  1
( 0 ,  4 ) =  1
( 1 ,  2 ) =  1
( 1 ,  3 ) =  1
( 1 ,  4 ) =  1
( 2 ,  3 ) =  1
( 2 ,  4 ) =  1
( 3 ,  4 ) =  1

 ADAMIC-ADAR INDEX
( 0 ,  1 ) =  0
( 0 ,  2 ) =  0
( 0 ,  3 ) =  0
( 0 ,  4 ) =  0
( 1 ,  2 ) =  0
( 1 ,  3 ) =  0
( 1 ,  4 ) =  0
( 2 ,  3 ) =  0
( 2 ,  4 ) =  0
( 3 ,  4 ) =  0

 JACCARD COEFFICIENT
( 0 ,  1 ) =  0
( 0 ,  2 ) =  0
( 0 ,  3 ) =  0
( 0 ,  4 ) =  0
( 1 ,  2 ) =  0
( 1 ,  3 ) =  0
( 1 ,  4 ) =  0
( 2 ,  3 ) =  0
( 2 ,  4 ) =  0
( 3 ,  4 ) =  0

 PREFERENTIAL ATTACHMENT
( 0 ,  1 ) =  0
( 0 ,  2 ) =  0
( 0 ,  3 ) =  0
( 0 ,  4 ) =  0
( 1 ,  2 ) =  0
( 1 ,  3 ) =  0
( 1 ,  4 ) =  0
( 2 ,  3 ) =  0
( 2 ,  4 ) =  0
( 3 ,  4 ) =  0

 RESOURCE ALLOCATION INDEX
( 0 ,  1 ) =  0
( 0 ,  2 ) =  0
( 0 ,  3 ) =  0
( 0 ,  4 ) =  0
( 1 ,  2 ) =  0
( 1 ,  3 ) =  0
( 1 ,  4 ) =  0
( 2 ,  3 ) =  0
( 2 ,  4 ) =  0
( 3 ,  4 ) =  0
PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA>
```

AIM IX: TO ANALYSE THE PERFORMANCE OF LINK PREDICTION METHODS MENTIONED IN THE PREVIOUS EXPERIMENT (EXP-8) IN TERMS OF PRECISION AND AREA UNDER THE RECEIVER OPERATING CHARACTERISTICS CURVE (AUC ROC).

THEORY:

1. **AUC-ROC Curve:** AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease. The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

$$\frac{n' + 0.5n"}{n}$$

where,

n is the number of times that we randomly pick a pair of links from missing links set and unconnected link set;

n' is the number of times that the missing link got a higher score than unconnected link

n" is the number of time when missing link equals unconnected link

The AUC value will be 0.5 if the score is generated from an independent and identical distribution. Thus, the degree to which the AUC exceeds 0.5 indicates how much better the predictions are when compared to prediction by chance.

2. **Precision-Recall:** A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve. A no-skill classifier is one that cannot discriminate between the classes and would predict a random class or a constant class in all cases. Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class. Precision is referred to as the positive predictive value. Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as sensitivity.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} = \frac{True\ Positives}{Total\ Predicted\ Positives}$$

$$True\ Positive\ Rate\ (TPR)/Recall/Sensitivity\ = \frac{True\ Positives\ (TP)}{TP + False\ Negatives\ (FN)}$$

$$Specificity = \frac{True\ Negative\ (TN)}{True\ Negatives\ (TN) + False\ Postitives\ (FP)}$$

$$False\ Positive\ Rate\ (FPR) = 1 - Specificity = \frac{FP}{TN + FP}$$

CODE:

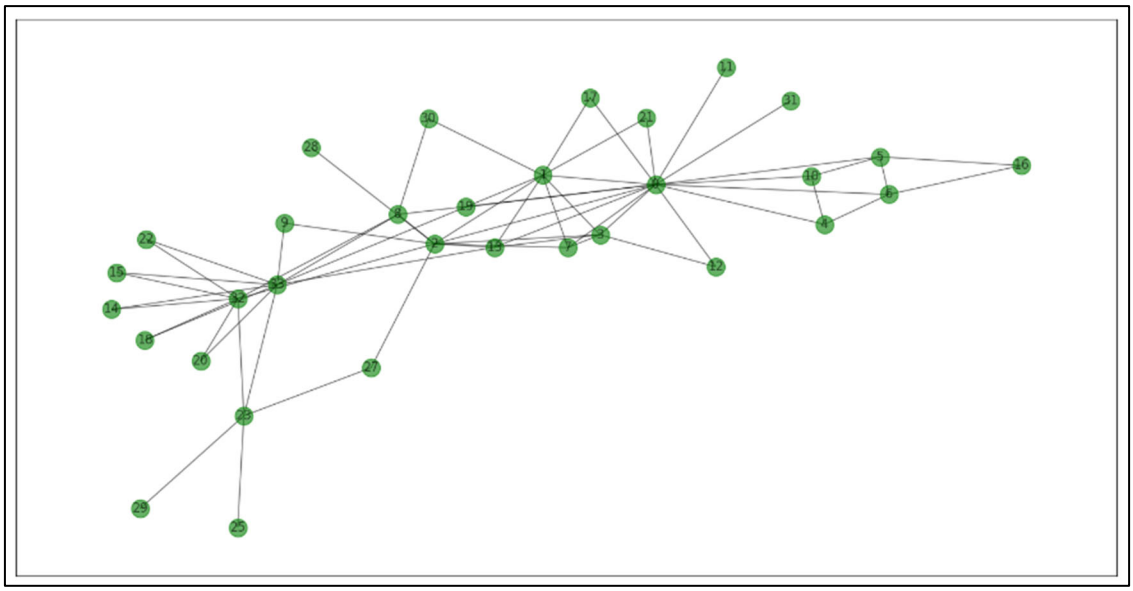
```
import networkx as nx
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import spatial

def findUnconnected (list, numOfNodes):
    unconnectedList = [[]]
    var = 0
    for ii in range (numOfNodes):
        for jj in range (ii+1, numOfNodes):
            if jj not in list[ii]:
                unconnectedList[var].append (ii)
                unconnectedList[var].append (jj)
                var += 1
                unconnectedList += [[]]
    unconnectedList.remove ([])
    return (unconnectedList)

def saltonList (GG):
    edges = nx.edges (GG)
    numOfNodes = GG.number_of_nodes()
    list = [[]]
    for ii in range (numOfNodes-1):
        list += [[]]
    for ii in range (numOfNodes):
        for jj in edges:
            if (jj[0] == ii):
                list [ii].append (jj[1])
            if (jj[1] == ii):
                list [ii].append (jj[0])
    unconnectedNodes = findUnconnected (list, numOfNodes)
    adjMatrix = [[]]
    var = 0
    for ii in range (numOfNodes):
        for jj in range (numOfNodes):
            if jj not in list [ii]:
                adjMatrix[var].append (0)
            else:
                adjMatrix[var].append (1)
        var += 1
        adjMatrix += [[]]
    adjMatrix.remove ([])

    var = 0
    for ii in unconnectedNodes:
```

```
        cosineSimilarity = 1 - spatial.distance.cosine (adjMatrix [ii[0]],
adjMatrix [ii[1]])
        unconnectedNodes[var].append (round (cosineSimilarity, 3))
        var += 1
    return unconnectedNodes

def precisionAUC (GG, df, preds, measure, t1, t2, train, test):

    # plt.figure ()
    # plt.title (measure)
    # nx.draw_networkx (GG, with_labels = True, alpha = 0.6)

    # PREDS LIST SORTED WITH HIGHEST SCORE AT THE TOP
    preds = sorted (preds, reverse = True, key = lambda x: x [2])

    # TOP 10000 LINKS SELECTED FROM THE PREDICTED LINKS
    preds = preds [:10000]

    # LINKS EXTRACTED INTO NEW LIST AND THE SDCORES ARE LEFT BEHIND
    predictedLinks = []
    for ii, jj, kk in preds:
        predictedLinks.append ([ii, jj])

    # LINKS EXTRACTED FROM TrAIN AND TEST DATAFRAMES
    l1, l2 = [], []
    x1, x2 = len (train), len (test)
    for ii in range (0, x1 - 1):
        l1.append ([train ['n1'][ii], train ['n2'][ii]])

    for ii in range (x1, x1 + x2):
        l2.append ([test ['n1'][ii], test ['n2'][ii]])

    # LINKS PUT INTO SETS FOR COMPUTATION PURPOSE
    trainSet = set (tuple (ii) for ii in l1)
    testSet = set (tuple (ii) for ii in l2)
    predictedLinksSet = set (tuple (ii) for ii in predictedLinks)

    truePositive = predictedLinksSet.intersection (testSet)
    falsePositive = predictedLinksSet.difference (truePositive)
    falseNegative = testSet.difference (truePositive)

    tpLen = len (truePositive)
    fpLen = len (falsePositive)
    fnLen = len (falseNegative)

    precision = tpLen / (tpLen + fpLen)
    print ("PRECISION = ", round (precision, 3))
```

```
    recall = tpLen / (tpLen + fnLen)
    # print ("RECALL = ", recall)

    nn, n1, n2 = 0., 0., 0.
    for ii in range (0, 10000):
        nn += 1
        tt, ff = np.random.randint (0, tpLen), np.random.randint (0, fpLen)
        if preds [tt][2] > preds [ff][2]:
            n1 += 1
        if preds [tt][2] == preds[ff][2]:
            n2 += 1
    AUC = (n1 + 0.5 * n2) / nn
    print ("AREA UNDER THE CURVE = ", round (AUC, 3))

# MAKING A RANDOM GRAPH OF FEW NODES
# GG = nx.erdos_renyi_graph (5, p = 0.8)
GG = nx.karate_club_graph ()
edges = nx.edges (GG)
print ("GRAPH EDGE SET:\n", edges)
newFile = open ('edgelist.txt', 'w')
for ii in edges:
    newFile.write (str (ii [0]))
    newFile.write (' ')
    newFile.write (str (ii [1]))
    newFile.write ('\n')
newFile.close ()
df = pd.read_csv("edgelist.txt",delimiter = ' ',header = None)
df.columns = ['n1','n2']
print (df)

# MAKING TRAINING AND TEST SETS
t1 = int (len (df) * 0.8)
t2 = len (df) - t1
train = df [:][:t1]
test = df [:][t1:t2+t1]

GG = nx.Graph ()
GG = nx.from_pandas_edgelist (train, 'n1', 'n2')

# SALTON INDEX (COSINE SIMILARITY)
preds = saltonList (GG)
print ("\nSALTON INDEX")
precisionAUC (GG, df, preds, "SALTON INDEX", t1, t2, train, test)

# ADAMIC-ADAR INDEX
preds = nx.adamic_adar_index (GG)
print ("\nADAMIC-ADAR INDEX")
precisionAUC (GG, df, preds, "ADAMIC-ADAR INDEX", t1, t2, train, test)
```

```
# JACCARD COEFFICIENT
preds = nx.jaccard_coefficient (GG)
print ("\nJACCARD COEFFICIENT")
precisionAUC (GG, df, preds, "JACCARD COEFFICIENT", t1, t2, train, test)

# PREFERENTIAL ATTACHMENT
preds = nx.preferential_attachment (GG)
print ("\nPREFERENTIAL ATTACHMENT")
precisionAUC (GG, df, preds, "PREFERENTIAL ATTACHMENT", t1, t2, train, test)

# RESOURCE ALLOCATION INDEX
preds = nx.resource_allocation_index (GG)
print ("\nRESOURCE ALLOCATION INDEX")
precisionAUC (GG, df, preds, "RESOURCE ALLOCATION INDEX", t1, t2, train, test)

# GRAPH PLOTTING
plt.figure ()
plt.title ("ORIGINAL GRAPH")
nx.draw_networkx (GG, with_labels = True, node_color = 'green', alpha = 0.6)
plt.show ()
```

OUTPUT AND OBSERVATIONS:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER                                   Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Documents\NITS\Semester VI\(LAB) CS321 SNA> python -u "d:\Documents\NITS\Semester VI\(LAB) CS321 SNA\lab9.py"
GRAPH EDGE SET:
 [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 10), (0, 11), (0, 12), (0, 13), (0, 17), (0, 19), (0,
21), (0, 31), (1, 2), (1, 3), (1, 7), (1, 13), (1, 17), (1, 19), (1, 21), (1, 30), (2, 3), (2, 7), (2, 8), (2, 9), (2, 13),
(2, 27), (2, 28), (2, 32), (3, 7), (3, 12), (3, 13), (4, 6), (4, 10), (5, 6), (5, 10), (5, 16), (6, 16), (8, 30), (8, 32), (
8, 33), (9, 33), (13, 33), (14, 32), (14, 33), (15, 32), (15, 33), (18, 32), (18, 33), (19, 33), (20, 32), (20, 33), (22, 32
), (22, 33), (23, 25), (23, 27), (23, 29), (23, 32), (23, 33), (24, 25), (24, 27), (24, 31), (25, 31), (26, 29), (26, 33), (
27, 33), (28, 31), (28, 33), (29, 32), (29, 33), (30, 32), (30, 33), (31, 32), (31, 33), (32, 33)]
    n1  n2
0    0   1
1    0   2
2    0   3
3    0   4
4    0   5
..  ..  ..
73  30  32
74  30  33
75  31  32
76  31  33
77  32  33

[78 rows x 2 columns]
C:\Users\subho\AppData\Local\Programs\Python\Python310\lib\site-packages\scipy\spatial\distance.py:630: RuntimeWarning: inva
lid value encountered in double_scalars
  dist = 1.0 - uv / np.sqrt(uu * vv)

SALTON INDEX
PRECISION =  0.013
AREA UNDER THE CURVE =  0.769

ADAMIC-ADAR INDEX
PRECISION =  0.025
AREA UNDER THE CURVE =  0.988

JACCARD COEFFICIENT
PRECISION =  0.025
AREA UNDER THE CURVE =  0.985

PREFERENTIAL ATTACHMENT
PRECISION =  0.025
AREA UNDER THE CURVE =  0.985

RESOURCE ALLOCATION INDEX
PRECISION =  0.025
AREA UNDER THE CURVE =  0.988

⊗ 0 ⚠ 0   Ln 160, Col 31  Spaces: 4  UTF-8  CRLF  Python  3.10.2 64-bit
```

AIM X: TO ANALYSE THE INFORMATION DIFFUSION BY INCORPORATING THE FOLLOWING EPIDEMIC MODELS AND DETERMINE THE BEST SUITABLE MODEL FOR REAL-EORLD SCENARIOS: EPIDEMIC MODELS: SUSCEPTIBLE INFECTED (SI), SUSCEPTIBLE INFECTED RECOVERED (SIR), AND SUSCEPTIBLE INFECTED SUSCEPTIBLE (SIS) AND LINEAR CASCADES (LC).

THEORY:

Compartmental models a very general modelling technique. They are often applied to the mathematical modelling of infectious diseases. The population is assigned to compartments with labels – for example, S, I, or R, (Susceptible, Infectious, or Recovered). People may progress between compartments. The order of the labels usually shows the flow patterns between the compartments; for example, SEIS means susceptible, exposed, infectious, then susceptible again.

1. **Susceptible Infected (SI):** The SI model splits the population into two groups, the susceptible individuals who may contract the disease and the infected individuals who may spread the disease to the susceptible.

2. **Susceptible Infected Recovered (SIR):** The SIR model splits the population into three groups, the susceptible individuals who may contract the disease, the infected individuals who may spread the disease to the susceptible and the recovered (or removed) individuals who have either recovered and are immune to the disease or the ones who have deceased (hence, removed instead of recovered).

3. **Susceptible Infected Susceptible (SIS):** The SIS model splits the population into three groups, the susceptible individuals who may contract the disease, the infected individuals who may spread the disease to the susceptible and the susceptible individuals who had previously contracted the disease, recovered and are susceptible again.

$$\boldsymbol{Susceptible\ Equation}, \frac{ds}{dt} = -b\ s(t)\ i(t)$$

$$\boldsymbol{Recovered\ Equation}, \frac{dr}{dt} = k\ i(t)$$

$$\boldsymbol{Infected\ Equation}, \frac{di}{\boldsymbol{dt}} = b\ s(t)\ i(t) - k\ i(t)$$

$$where,$$
$$s(t) = \frac{S\ (t)}{N}\ is\ the\ susceptible\ fraction\ of\ the\ population$$
$$i(t) = \frac{I\ (t)}{N}\ is\ the\ infected\ fraction\ of\ the\ population$$
$$r(t) = \frac{R\ (t)}{N}\ is\ the\ recovered\ fraction\ of\ the\ population$$

DATASETS: Football Club, Karate Club, Dolphin Network, Barabasi-Albert Network.

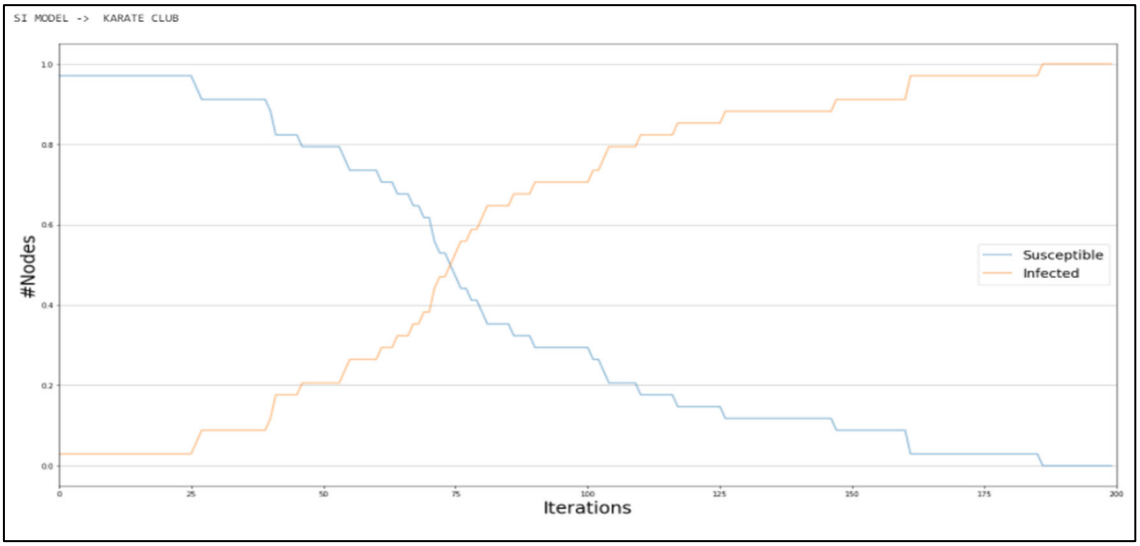
CODE:

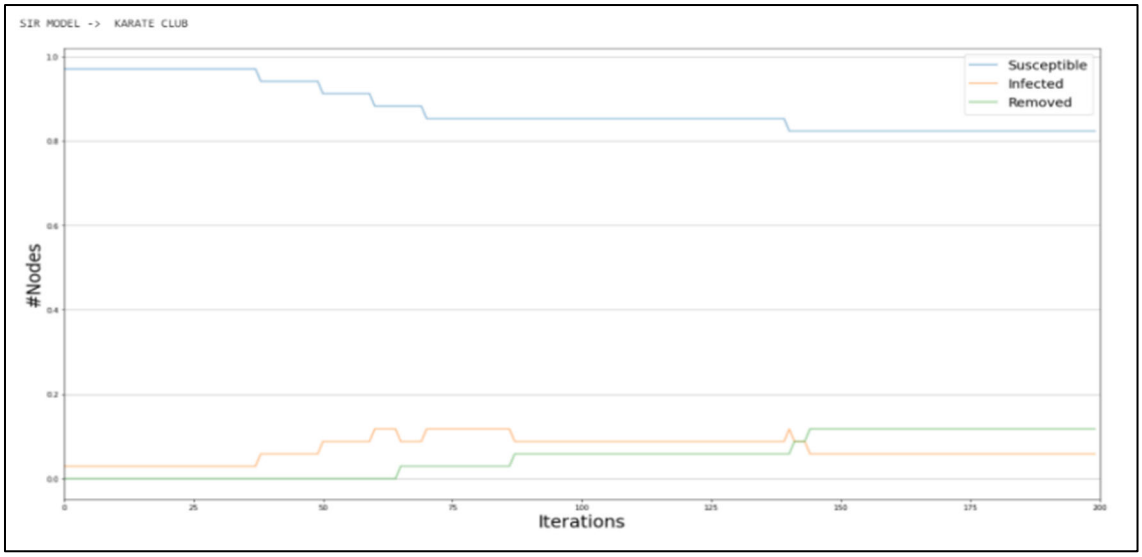
```
import networkx as nx
from bokeh.io import show
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.bokeh.DiffusionTrend import DiffusionTrend
# from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend # FOR JUPYTER
NOTEBOOK

def SImodel (GG, title):
    print ("\nSI MODEL -> ", title)
    SI = ep.SIModel (GG)
    # CONFIGURATION
    cfg = mc.Configuration ()
    cfg.add_model_parameter ('beta', 0.01)
    cfg.add_model_parameter ("percentage_infected", 0.05)
    SI.set_initial_status (cfg)
    # SIMULATION
    iterations = SI.iteration_bunch (200, node_status = True)
    trends = SI.build_trends (iterations)
    # VISUALISATION
    viz = DiffusionTrend (SI, trends)
    # viz.plot() # FOR JUPYTER NOTEBOOK
    show (viz.plot (), new = "tab")

def SIRmodel (GG, title):
    print ("\nSIR MODEL -> ", title)
    SIR = ep.SIRModel (GG)
    # CONFIGURATION
    config = mc.Configuration ()
    config.add_model_parameter ('beta', 0.001)
    config.add_model_parameter ('gamma', 0.01)
    config.add_model_parameter ('percentage_infected', 0.05)
    SIR.set_initial_status (config)
    # SIMULATION
    iterations = SIR.iteration_bunch (200, node_status = True)
    trends = SIR.build_trends (iterations)
    viz = DiffusionTrend (SIR, trends)
    # viz.plot() # FOR JUPYTER NOTEBOOK
    show (viz.plot (), new = "tab")

def SISmodel (GG, title):
    print ("\nSIS MODEL -> ", title)
    SIS = ep.SISModel (GG)
    # CONFIGURATION
    config = mc.Configuration ()
    config.add_model_parameter ('beta', 0.01)
    config.add_model_parameter ('lambda', 0.005)
```

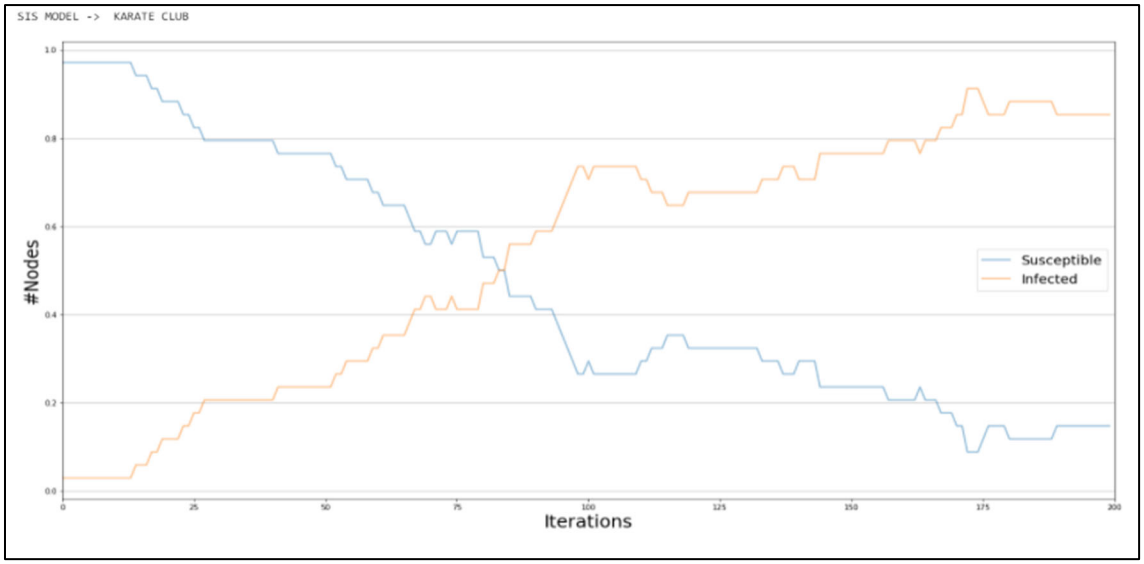
```
    config.add_model_parameter ('fraction_infected', 0.05)
    SIS.set_initial_status (config)
    # SIMULATION
    iterations = SIS.iteration_bunch (200, node_status = True)
    trends = SIS.build_trends (iterations)
    viz = DiffusionTrend (SIS, trends)
    # viz.plot() # FOR JUPYTER NOTEBOOK
    show (viz.plot (), new = "tab")

# LOADING KARATE CLUB GRAPH
GG = nx.read_gml ("karate.gml")
SImodel (GG, "KARATE CLUB")
SIRmodel (GG, "KARATE CLUB")
SISmodel (GG, "KARATE CLUB")

# LOADING FOOTBALL CLUB
GG = nx.read_gml ("football.gml")
SImodel (GG, "FOOTBALL CLUB")
SIRmodel (GG, "FOOTBALL CLUB")
SISmodel (GG, "FOOTBALL CLUB")

# LOADING DOLPHIN NETWORK
GG = nx.read_gml ("dolphins.gml")
SImodel (GG, "DOLPHIN NETWORK")
SISmodel (GG, "DOLPHIN NETWORK")
SIRmodel (GG, "DOLPHIN NETWORK")

# LOADING BARABASI-ALBERT NETWORK
GG = nx.barabasi_albert_graph (100, 15)
SImodel (GG, "BARABASI-ALBERT NETWORK")
SISmodel (GG, "BARABASI-ALBERT NETWORK")
SIRmodel (GG, "BARABASI-ALBERT NETWORK")
```

OUTPUT AND OBSERVATIONS:

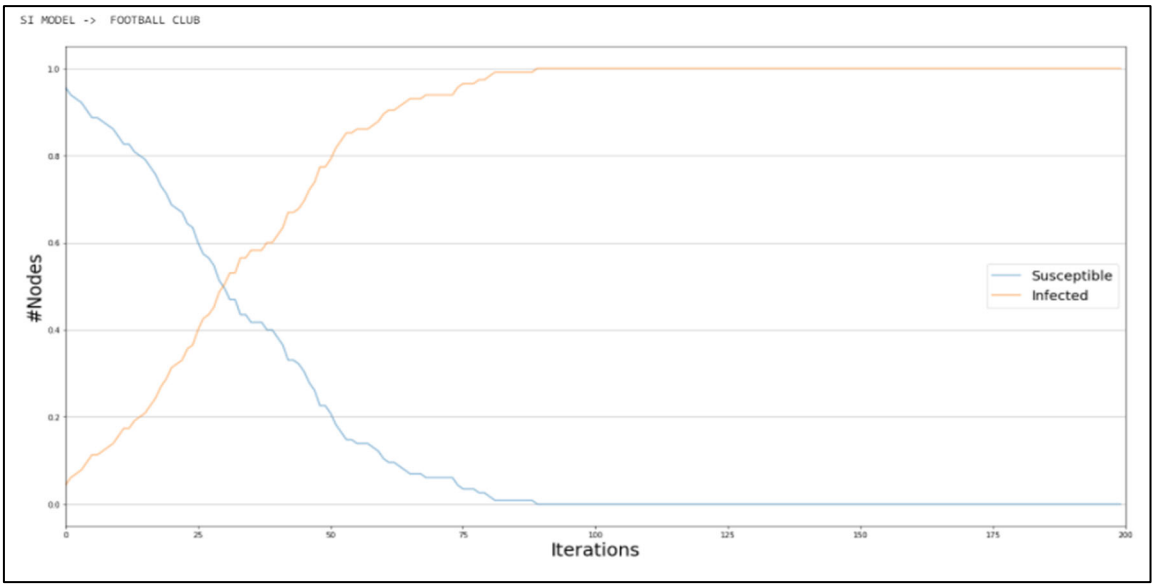
// KARATE CLUB

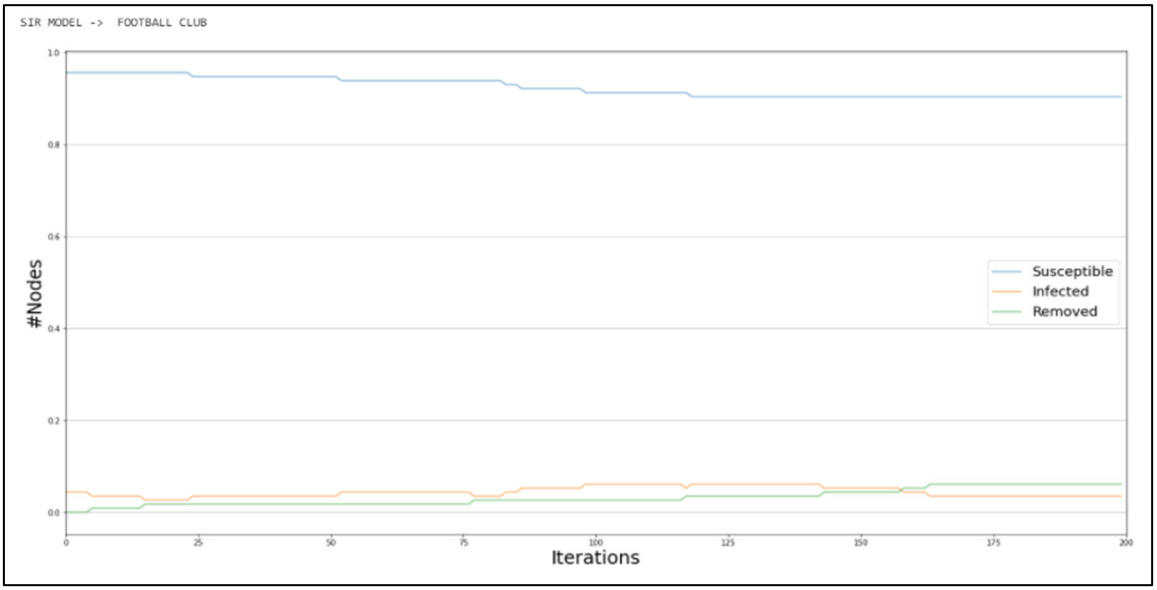


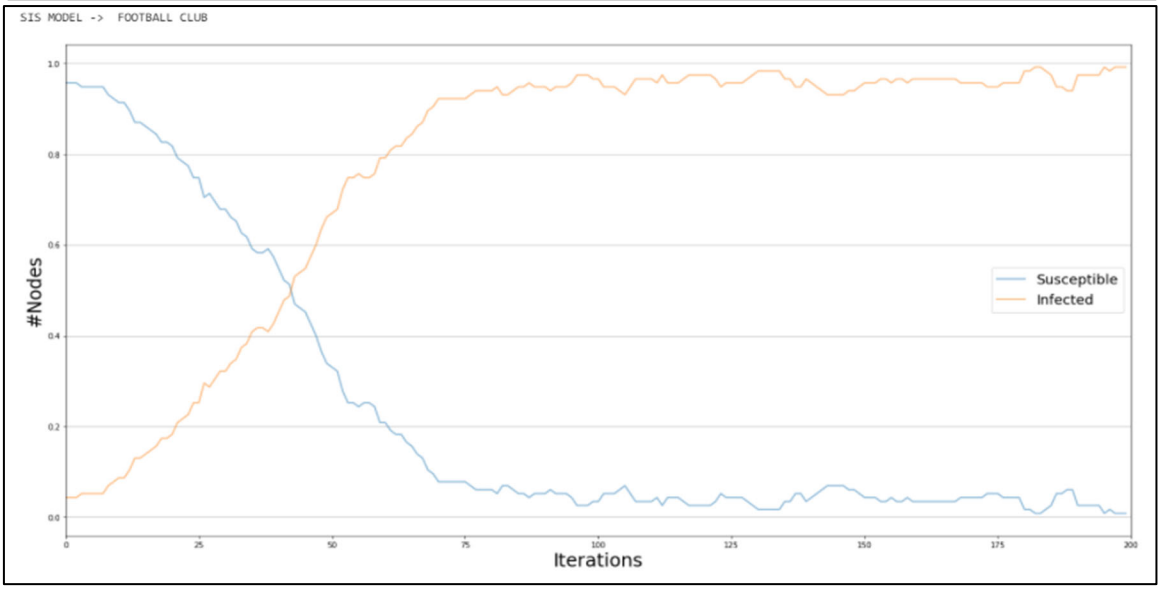




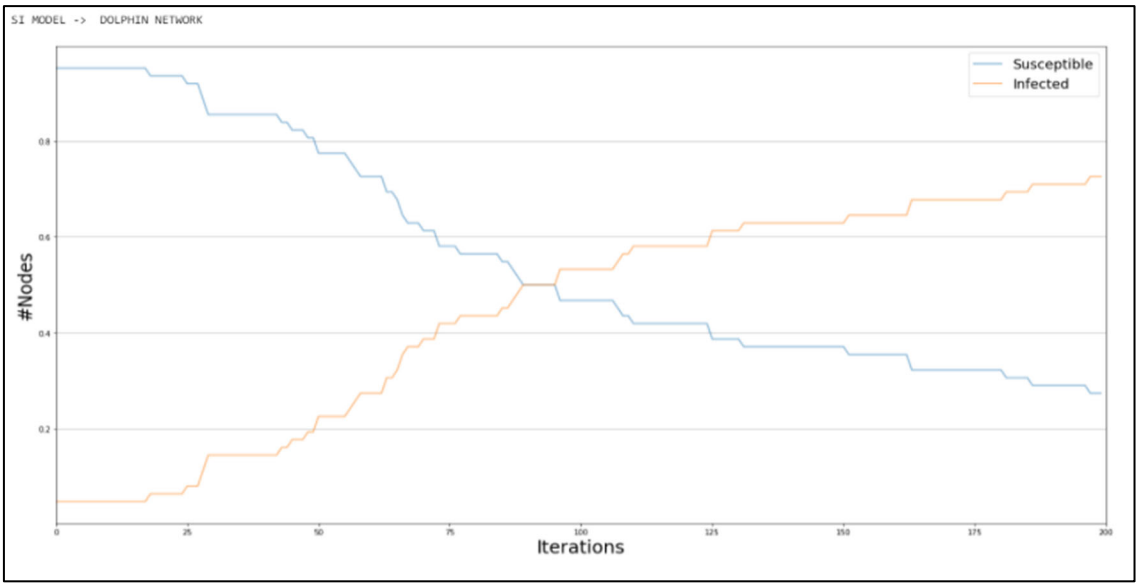
// FOOTBALL CLUB

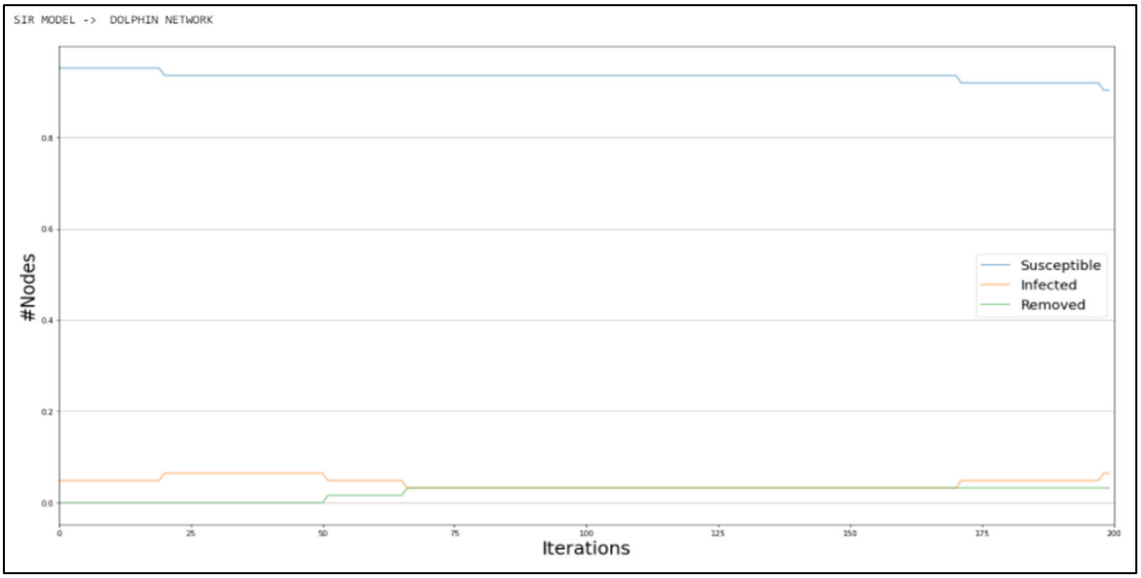


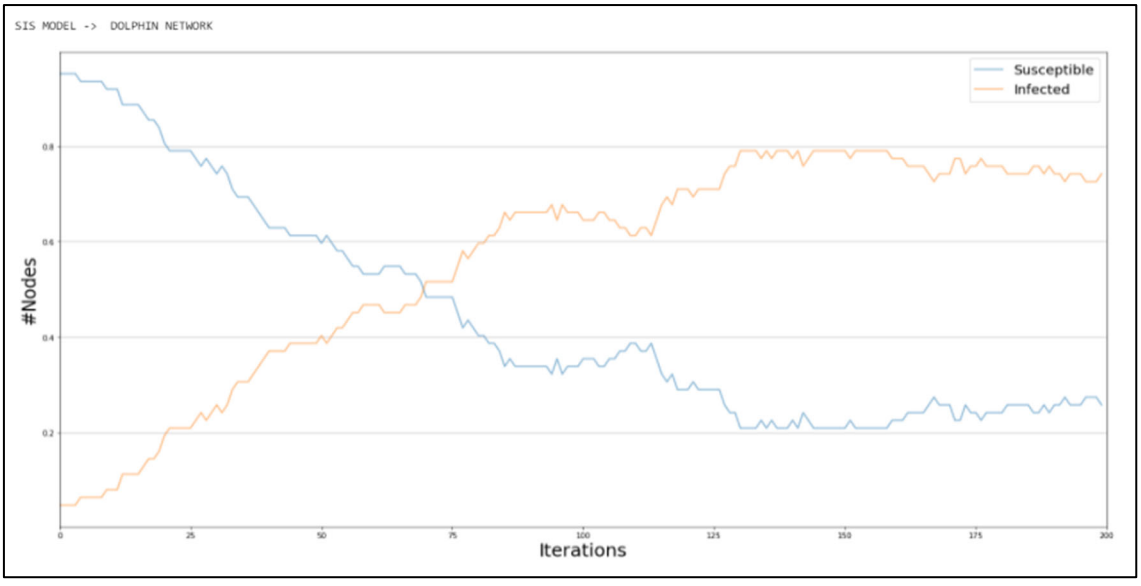




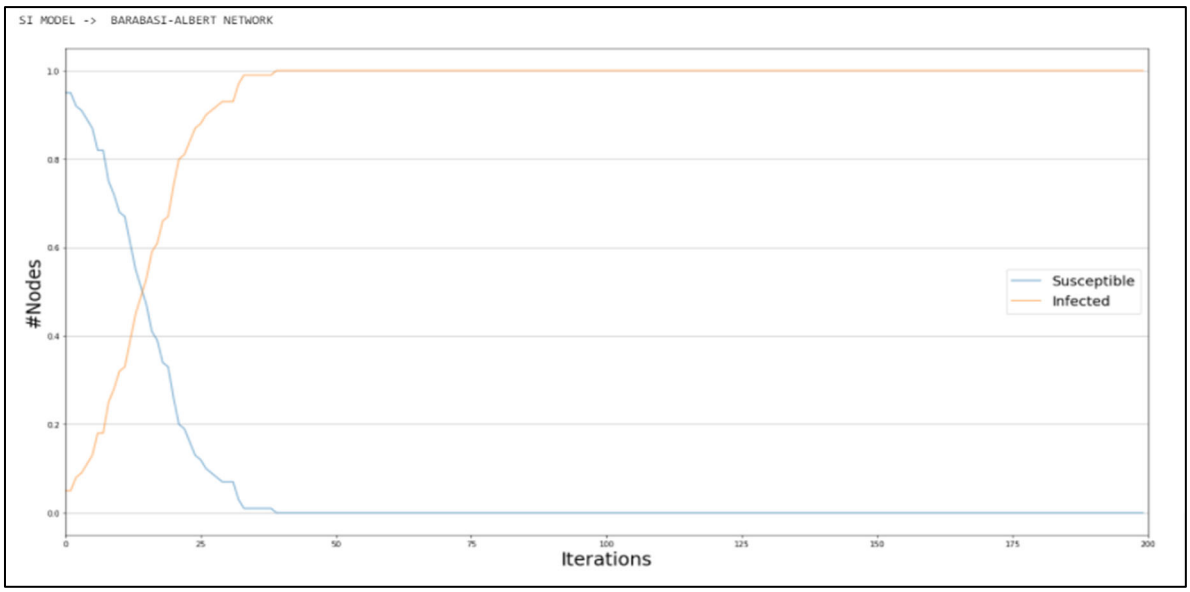
// DOLPHINS NETWORK

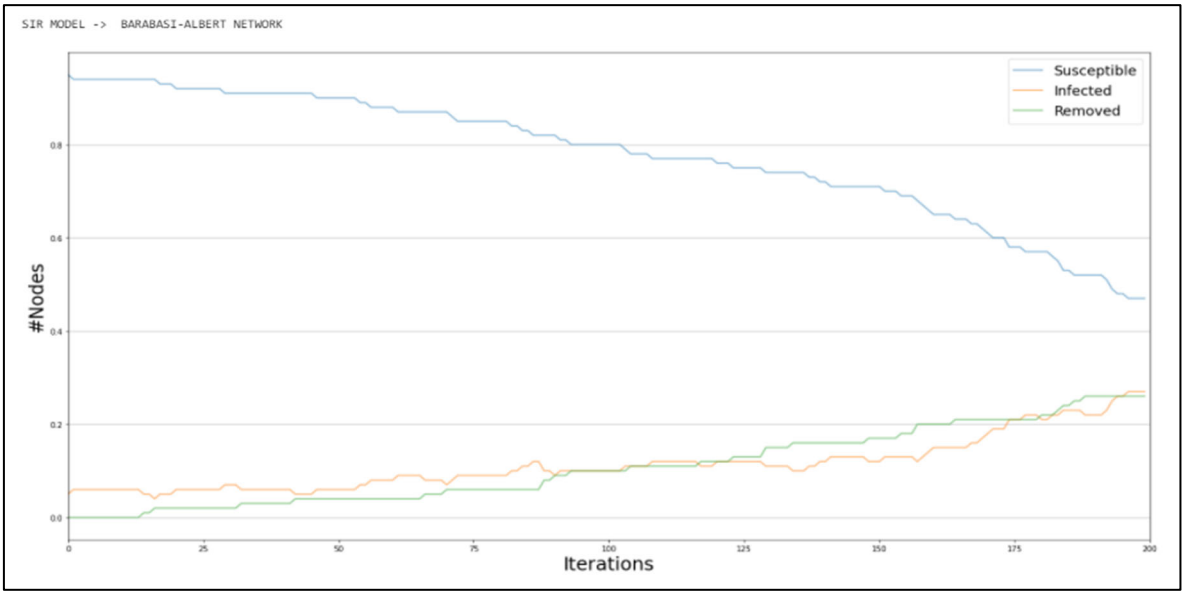


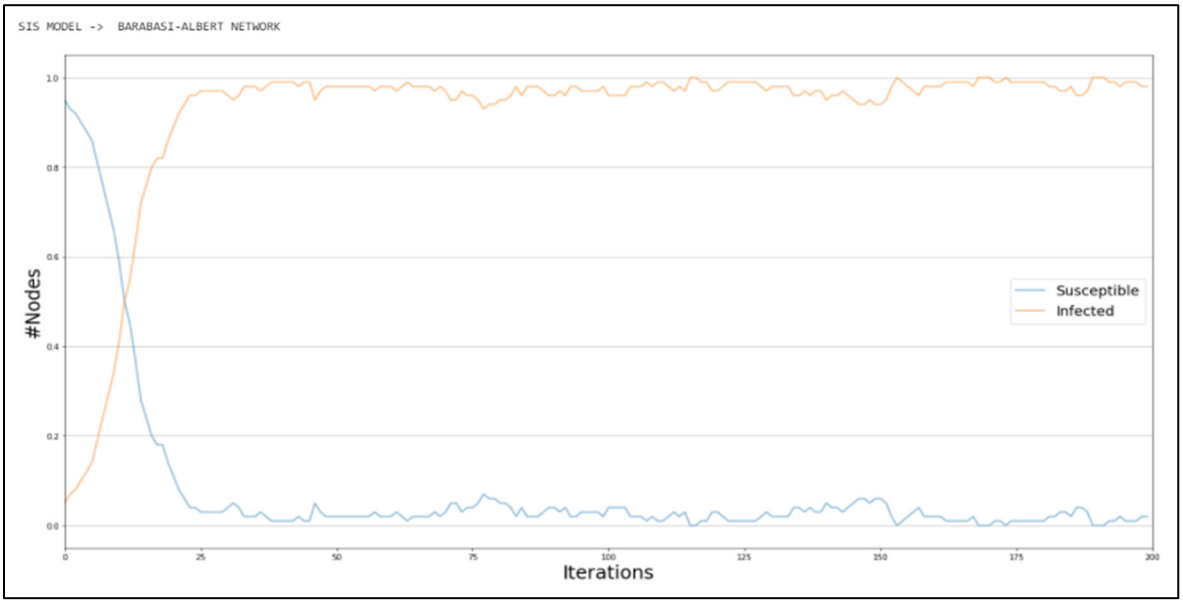




// BARABASI-ALBERT GRAPH


SI MODEL -> BARABASI-ALBERT NETWORK
#Nodes
Iterations
Susceptible
Infected


SIR MODEL -> BARABASI-ALBERT NETWORK
#Nodes
Iterations
Susceptible
Infected
Removed


SIS MODEL -> BARABASI-ALBERT NETWORK
#Nodes
Iterations
Susceptible
Infected