CSCI - 5352 (Network Analysis and Modeling) - Problem Set 3

Name of Student : Rajarshi Basak

October 10, 2018

[1] In the table provided, e_{rr} is the diagonal and a_r is the row or column sum for each type. For each type (i.e. Black, Hispanic, White and Other), the values of e_{rr} and a_r have been summarized in the table below.

r (Type)	Black	Hispanic	White	Other
e_{rr}	0.258	0.157	0.306	0.016
a_r	0.288	0.203	0.423	0.083

Table 1: Table for e_{rr} and a_r

From Equation (7.76) in *Networks* the the Modularity Q is given by

$$Q = \sum_r (e_{rr} - a_r^2)$$

Hence from the table above, we calculate the value of $(e_{rr} - a_r^2)$ for each type, which has been tabulated below:

r (Type)	Black	Hispanic	White	Other
$e_{rr} - a_r^2$	0.175056	0.115791	0.127071	0.009111

Table 2: Table for $(e_{rr} - a_r^2)$

Summing over the values in the table above, the modularity Q of the network with respect to ethnicity is given by

$$Q = 0.175056 + 0.115791 + 0.127071 + 0.009111$$

$$=> \boxed{Q = 0.427029}$$

About homophily, We conclude that couples tend to be formed more between people of the same ethnicity. To be precise, in decreasing order of magnitude, couples tend to form more between Black people, then White people, then Hispanic, and finally the others, as suggested by the values of $(e_{rr} - a_r^2)$ for each ethnicity.

[2] We are given an undirected "line graph" consisting of n vertices in a single component, with diameter $n-1$, and composed of $n-2$ vertices with degree 2 and 2 vertices with degree 1.

(a) We divide the network into 2 contiguous groups, such that Part A consist of the first r vertices from the left connected by $r-1$ edges, and Part B consist of the next $n-r$ vertices connected by $n-r-1$ edges.

Since there are two ends for each edge between vertices in Part A of the network and one extra end that connects Part A to Part B, the number of edges that attach to vertices in Part A is given by $2(r-1)+1$. Similarly, the number of edges that attach to Part B is $2(n-r-1)+1$

Hence the fraction of edges between vertices in Part A and Part B, e_{rr}^{PartA} and e_{rr}^{PartB} are respectively given by

$$e_{rr}^{PartA} = \frac{r-1}{n-1}$$

and

$$e_{rr}^{PartB} = \frac{n-r-1}{n-1}$$

whereas the fraction of ends of edges that attach to vertices in Part A and Part B, a_r^{PartA} and a_r^{PartB}, are respectively given by

$$a_r^{PartA} = \frac{2(r-1)+1}{2(n-1)}$$

and

$$a_r^{PartB} = \frac{2(n-r-1)+1}{2(n-1)}$$

Now the the modularity Q of the network is given by

$$Q = (e_{rr}^{PartA} - a_r^{PartA^2}) + (e_{rr}^{PartB} - a_r^{PartB^2})$$

$$=> Q = \left[\frac{r-1}{n-1} - \left(\frac{2(r-1)+1}{2(n-1)}\right)^2\right] + \left[\frac{n-r-1}{n-1} - \left(\frac{2(n-r-1)+1}{2(n-1)}\right)^2\right]$$

or

$$=> Q = \left[\frac{r-1}{n-1} + \frac{n-r-1}{n-1}\right] - \left[\frac{(2(r-1)+1)^2}{4(n-1)^2} + \frac{(2(n-r-1)+1)^2}{4(n-1)^2}\right]$$

or

$$=> Q = \frac{n-2}{n-1} - \left[\frac{4(r-1)^2+1+4(r-1)+4(n-r-1)^2+1+4(n-r-1)}{4(n-1)^2}\right]$$

or

$$=> Q = \frac{n-2}{n-1} - \left[\frac{4r^2+4-8r+1+4r-4+4n^2+4r^2+4-8nr-8n+8r+1+4n-4r-4}{4(n-1)^2}\right]$$

or

$$=> Q = \frac{n-2}{n-1} - \left[\frac{4r^2+1+4n^2+4r^2-8nr-8n+1+4n}{4(n-1)^2}\right]$$

or

$$=> Q = \frac{n-2}{n-1} - \left[\frac{8r^2+4n^2-8nr-4n+2}{4(n-1)^2}\right]$$

or

$$=> Q = \frac{4(n-1)(n-2)}{4(n-1)^2} + \frac{8nr+4n-8r^2-4n^2-2}{4(n-1)^2}$$

or

$$=> Q = \frac{4n^2-12n+8+8nr+4n-8r^2-4n^2-2}{4(n-1)^2}$$

or

$$=> Q = \frac{6-8n+8rn-8r^2}{4(n-1)^2}$$

or

$$=> \boxed{Q = \frac{3-4n+4rn-4r^2}{2(n-1)^2}}$$

(b)

Considering the same graph, when n is even, when we split the network exactly down the middle, i.e. when $r = n/2$, the value of Q is

$$Q_{r=n/2} = \frac{3 - 4n + 4\frac{n}{2}n - 4\frac{n^2}{4}}{2(n-1)^2}$$

or

$$=> \boxed{Q_{r=n/2} = \frac{1}{2(n-1)^2}[3 - 4n + n^2]}$$

For a partition that is one-third of the way from the left, i.e. with $r = n/3$, the value of Q is given by

$$Q_{r=n/3} = \frac{3 - 4n + 4\frac{n}{3}n - 4\frac{n^2}{9}}{2(n-1)^2}$$

or

$$=> Q_{r=n/3} = \frac{1}{2(n-1)^2}\left[3 - 4n + \frac{3n^2}{9}\right]$$

Similarly, when we make a partition one-tenth of the way from the left, i.e. with $r = n/10$, the value of Q is given by

$$Q_{r=n/10} = \frac{3 - 4n + 4\frac{n}{10}n - 4\frac{n^2}{100}}{2(n-1)^2}$$

or

$$=> Q_{r=n/10} = \frac{1}{2(n-1)^2}\left[3 - 4n + \frac{6n^2}{25}\right]$$

Hence as we make the size of one of the partitions smaller, the value of Q decreases, and we observe the maximum value of Q is for $r = n/2$, Therefore, the optimal division in terms of Q is the division that splits the network exactly down the middle, into two parts of equal size.

[2] (b): Alternative Another way to show what is mentioned above is to take a derivative of the expression for Q with respect to r and set it equal to 0. The value of r in terms of n that satisfies this equation should be $r = \frac{n}{2}$. We have

$$Q = \frac{3 - 4n + 4rn - 4r^2}{2(n-1)^2}$$

Now,

$$\frac{dQ}{dr} = \frac{1}{2(n-1)^2}\frac{d}{dr}[4rn - 4r^2]$$

$$=> \frac{dQ}{dr} = \frac{1}{2(n-1)^2}[4n - 8r]$$

Setting the above expression equal to 0,

$$\frac{1}{2(n-1)^2}[4n - 8r] = 0$$

implies that

$$4n - 8r = 0$$

since $\frac{1}{2(n-1)^2}$ cannot be equal to 0.
Hence,

$$n - 2r = 0$$

or

$$\boxed{r = \frac{n}{2}}$$

[3] (a) Plot of the modularity score as a function of the number of merges

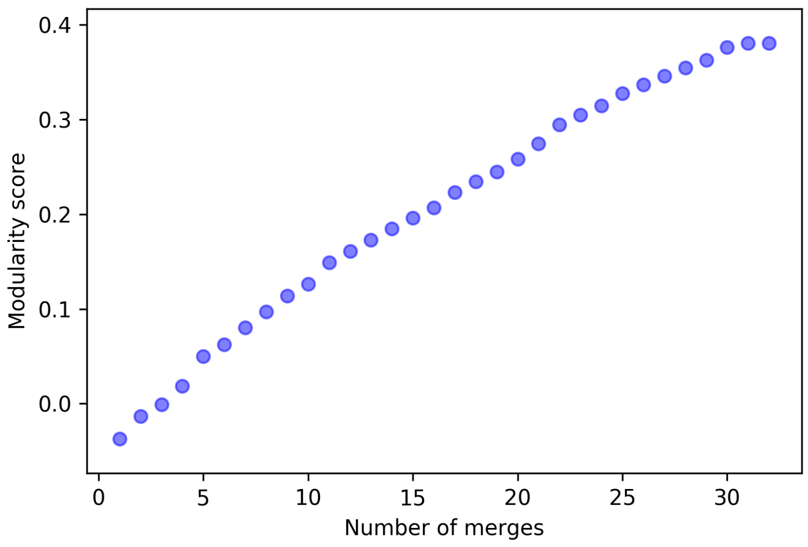

Figure 1: Figure showing Modularity Score vs. Number of merges

[3] (b) Visualization of the network with vertices labeled according to the maximum modularity partition

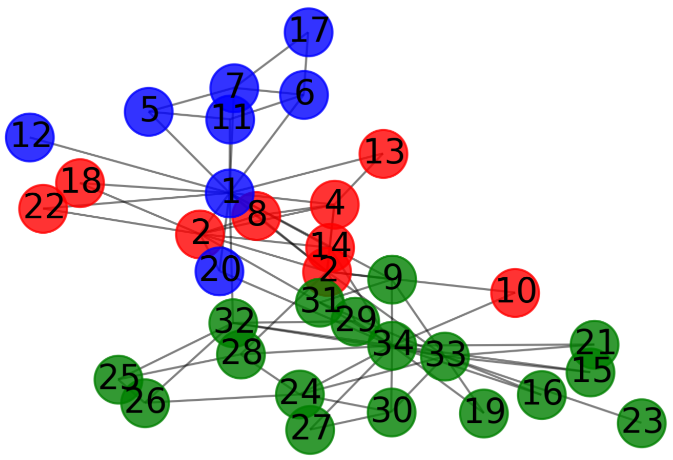

Figure 2: Visualizing the Karate Club Network according to the Maximum Modularity Partition

[3] (c) The normalized mutual information (NMI) between my partition and the social partition is $\boxed{0.5762}$.

[3] (d) The normalized mutual information (NMI) value between two partitions ranges between 0 and 1, with 0 representing the classes members to be completely split across different clusters, and 1 representing perfect labelings which are both homogeneous and complete.

In our case, the value of (NMI) being 0.5762 implies that the two partitions are in agreement to some magnitude, and the fact the value is ¿ 0.5 suggests that the level of agreement is above the mean value.

[4] (a) **Attribute: Student/faculty status**

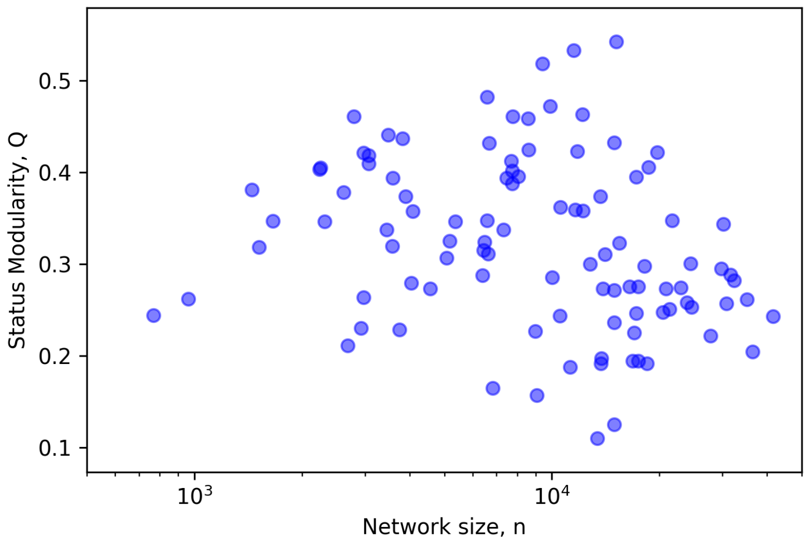

Figure 3: Figure showing Student/faculty status Modularity(Q) vs. Network Size(n)

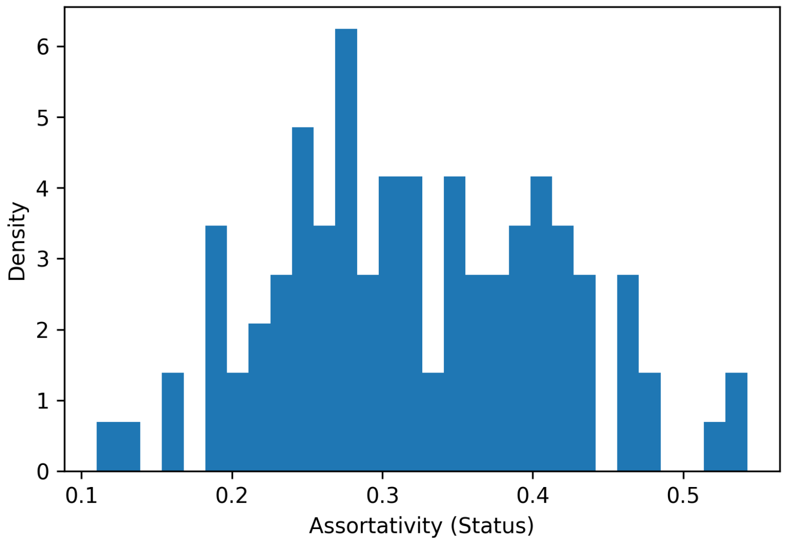

Figure 4: Histogram showing the distribution of assortativity values for student/faculty status

Discussion for Attribute: Student/faculty status

The distribution of points is completely above the line of no assortativity; in other words there are no networks with a negative assortativity. All values are between 10 percent and 60 percent in the positive direction of zero and the average assortativity is 0.3227, thus implying that the status attributes are somewhat assortative. This suggests a fairly strong amount of homophily (like links with like, i.e. students link with students, and faculty link with faculty) by status in the way friendships are formed in Facebook. It seems students like to interact mostly with other students and faculty with other faculty (on Facebook) owing to their age difference or status in general, and rarely do students interact with faculty.

[4] (b) Attribute: Major

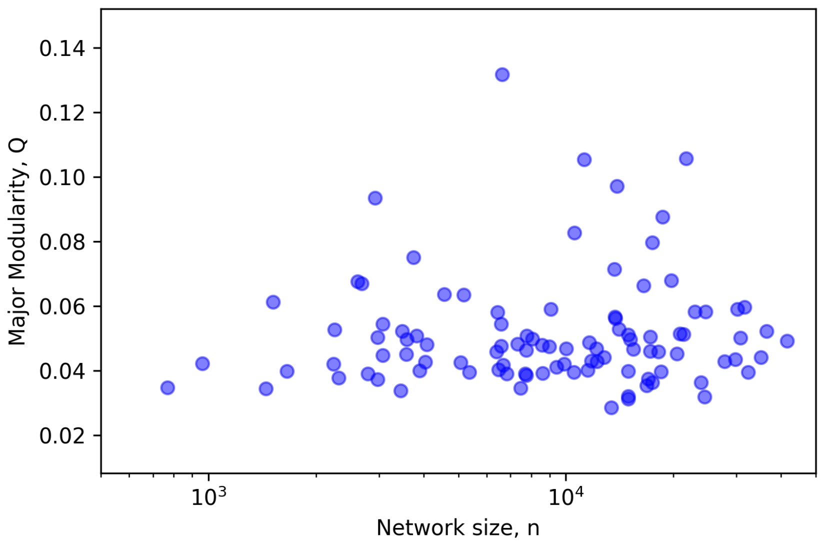

Figure 5: Figure showing Major Modularity(Q) vs. Network Size(n)

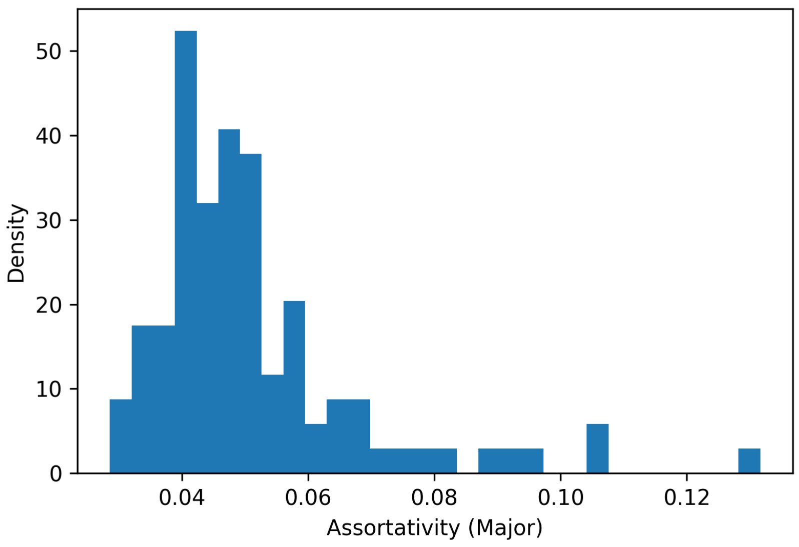

Figure 6: Histogram showing the distribution of assortativity values for major

The distribution of points is entirely above the line of no assortativity; all networks have a positive assortativity with respect to major. All values are within 15 percent in the positive direction of zero and the average assortativity is 0.0511, indicating the fact that the major attributes are slightly assortative. Most of the points are between 0.02 and 0.08, with a few points lying between 0.08 and 0.14. This hints at a slight amount of homophily (like links with like, i.e. students of a given major interact with the same major) by major in the way friendships are formed in Facebook. Since most students interact outside of class and lectures (in addition to having common lectures) for clubs, groups and common interests, they tend to form friendships more with students of other majors. Also, students of different majors sometimes take the same classes, which aids the process of befriending people from other majors.

[4] (c) Attribute: vertex degree

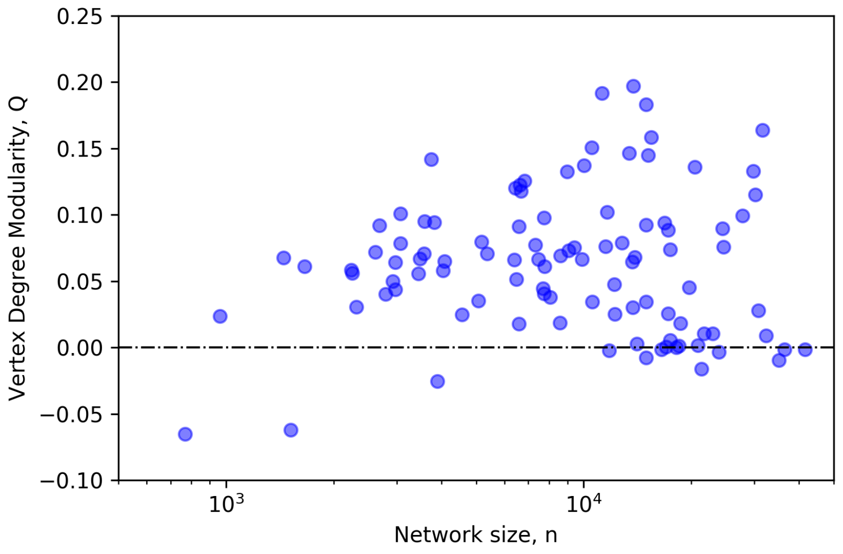

Figure 7: Figure showing Vertex Degree Modularity(Q) vs. Network Size(n)

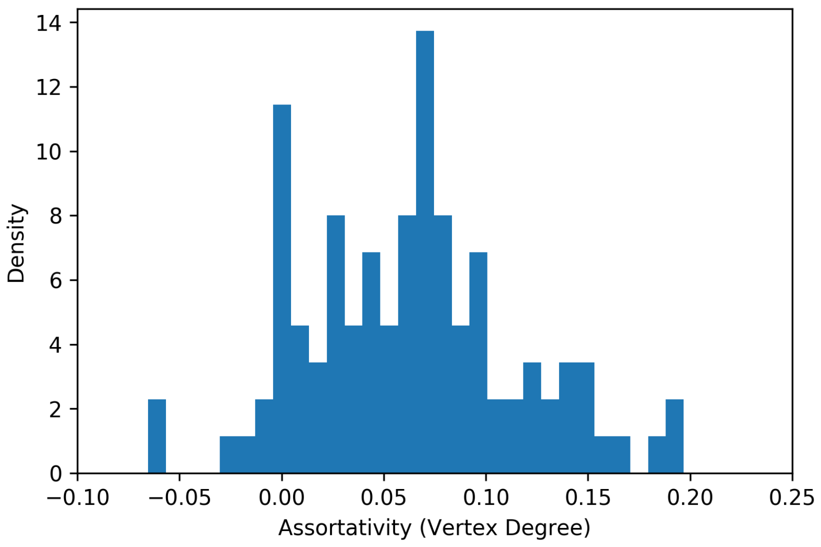

Figure 8: Histogram showing the distribution of assortativity values for vertex degree

Discussion for Attribute: vertex degree
The distribution of points does span the line of no assortativity, and there are several networks with a negative value of assortativity. However, these networks with a negative value of assortativity have a value of assortativity that is bounded by about -0.07. We observe that the networks with positive assortativity have values that are within 20 percent in the positive direction, while those with negative values are within about 7 percent in the negative direction. The average value of assortativity is 0.062, which being slighlty above zero, suggests a small amount of homophily by vertex degree (vertices with more degrees tend to link with vertices with more degree, or popular people link with popular people). In some networks, the small tendency for heterophily (popular people link with unpopular people) might be due to the fact that these networks have a small number of people who are popular, and hence the connections between pairs of popular people is far lesser than the connections between pairs of a popular and an unpopular person.

[5] In a graph, if the ith vertex has k_i stubs, then the total number of stubs in the graph is given by

$$\sum_i k_i = 2m$$

where m is the number of edges in the graph. Hence the total number of stubs in the graph is twice the number of edges in the graph.

Since there are $n!$ ways of arranging n objects, there are $2m!$ ways of arranging $2m$ stubs.

For a single self loop, let $m = 1$. First we consider a single self-loop on a node i which reduces the total number of matchings by a factor of 2 since an edge $(i_1 - > i_2)$ is equivalent to $(i_2 - > i_1)$. If there are multiple self loops, the number of matchings drops by an additional factor of $m!$ since there are m edges in the graph. Hence the number of matchings drops by a factor of $m!2^m$.

Hence the total number of matchings of this graph given the degree sequence is

$$\boxed{\Omega = \frac{(2m)!}{2^m m!}}$$

The code for Problem 3 (a), (b) and (c) is shown below.

```
import matplotlib.pyplot as plt
import networkx as nx
from networkx.algorithms.community.quality import modularity
from sklearn.metrics.cluster import normalized_mutual_info_score

G = nx.read_adjlist('karate_edges_77.txt')

# Let each node in the graph be in its own community
communities = list()
for i in G.nodes():
    communities.append(set([i]))

# Create a list for keeping track of all merges
tracking_merges = list()

modnew = modularity(G, communities)
print ('The modularity at the beginning is',modnew)
modold = None
comtrial = []
modtrial = 0
num_of_merges = 0
num_merges = []
modularity_scores = []

# Maximizing the modularity to find the best social parition
while (modold is None or modnew > modold):
    comtrial = list(communities)
    modold = modnew
    #print('The current modularity is', modold)
    to_be_merged = None
    for i, x in enumerate(communities):
        for j, y in enumerate(communities):
            if (j <= i or len(x) == 0 or len(y) == 0):
                continue
            comtrial[i] = set([])
            comtrial[j] = x | y
            modtrial = modularity(G, comtrial)
            #dq = modold - modtrial
            #all_dqs.append(dq)
            if (modtrial - modnew >= 0):
                if (modtrial - modnew > 0):
                        modnew = modtrial
                        #print ('Trial Modularity',modnew)
                        to_be_merged = (i, j, modnew - modold)
                    elif (to_be_merged and
                        min(i, j) < min(to_be_merged[0], to_be_merged[1])):
                            modnew = modtrial
                            #print ('Trial Modularity',modnew)
                            to_be_merged = (i, j, modnew - modold)
                comtrial[i] = x
                comtrial[j] = y
    if (to_be_merged is not None):
        tracking_merges.append(to_be_merged)
        i, j, dq = to_be_merged
        x = communities[i]
        y = communities[j]
        communities[i] = set([])
        communities[j] = x | y
    #print('Old Modularity', modold)
    #print('New Modularity', modnew)
    #print(modularity(G,communities))
    num_of_merges += 1
    num_merges.append(num_of_merges)
    modularity_scores.append(modnew)

communities = [c for c in communities if len(c) > 0]

print(communities)

#Plotting the modularity score Q as a function of the number of merges.
x = num_merges
y = modularity_scores

plt.scatter(x,y, alpha=0.5, color = 'b')
plt.xlabel('Number of merges')
plt.ylabel('Modularity score')
plt.savefig('Mod_score_vs_num_merges.png', dpi = 300)
plt.show()

print(len(communities))

pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos,
                       nodelist=['3', '13', '4', '14', '8', '22', '2',
                                 '10', '18'],
                       node_color='r',
                       node_size=500,
                       alpha=0.8)

nx.draw_networkx_nodes(G, pos,
                       nodelist=['17', '6', '12', '20', '1', '7', '11',
                                 '5'],
                       node_color='b',
                       node_size=500,
                       alpha=0.8)

nx.draw_networkx_nodes(G, pos,
                       nodelist=['9', '23', '21', '28', '29', '24', '30',
                                 '31', '34', '33', '32', '26', '19', '27',
                                 '25', '16', '15'],
                       node_color='g',
                       node_size=500,
                         alpha=0.8)

plt.axis('off')

nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.5)

# Creating the labels
labels = {}
for i in range(1,35):
    labels[str(i)] = r'$' + str(i) + '$'
nx.draw_networkx_labels(G, pos, labels, font_size=16)
plt.savefig('networkvis_maxmodpartition', dpi = 300)
plt.show()

# Finding the normalized mutual information (NMI) between my partition
# and the social partition
nmi = normalized_mutual_info_score([1,1,1,1,1,1,1,1,1,2,1,
                                    1,1,1,2,2,1,1,2,1,2,1,
                                    2,2,2,2,2,2,2,2,2,2,2,
                                    2],
                                   [2,1,1,1,2,2,2,1,3,1,2,
                                    2,1,1,3,3,2,1,3,2,3,1,
                                    3,3,3,3,3,3,3,3,3,3,3,
                                    3])

print (nmi)
```

Note : For implementing the greedy modularity maximization, the webpage https://networkx.github.io/dc

The code for Problem 4 (a) and (b) , i.e. for generating the plots and histograms for student/faculty status and major assortativity is shown below.

```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from os import listdir
from os.path import isfile, join

status_assortativities = {}
major_assortativities = {}
onlyfiles = [f for f in listdir('./') if isfile(join('./', f))]
for i in onlyfiles:
    attr = {}
    dict_status = {}
    dict_major = {}
    if i.endswith('_attr.txt'):
        f = open(i,'r')
        next (f)
        count = 1
        for j in f:
            k = j.strip().split('\t')
            attr[count] = [k[0], k[1], k[2], k[3], k[4]]
            count = count + 1
        for k,v in attr.items():
            dict_status[str(k)] = int(v[0])
            dict_major[str(k)] = int(v[2])
        txt_file = i.replace('_attr.txt','.txt')
        G = nx.read_edgelist(txt_file)
        netsize = len(G.nodes())
        nx.set_node_attributes(G, dict_status,'status')
        stat_assort = nx.attribute_assortativity_coefficient(G, 'status')
        nx.set_node_attributes(G, dict_major,'major')
```

```
            maj_assort = nx.attribute_assortativity_coefficient(G, 'major')
            status_assortativities[txt_file] = (stat_assort, netsize)
            major_assortativities[txt_file] = (maj_assort, netsize)
            print(txt_file, stat_assort, maj_assort, netsize)

    x1 = np.zeros((100))
    y1 = np.zeros((100))
    ind1 = 0
    for j in status_assortativities:
        temp1=(status_assortativities[j])
        x1[ind1] = temp1[1]
        y1[ind1] = temp1[0]
        ind1 = ind1+1

    #Plotting the Figure showing Status Modularity(Q)
    #vs. Network Size(n)
    plt.scatter(x1,y1, alpha=0.5, color = 'b')
    plt.xlabel('Network size, n')
    plt.ylabel('Status Modularity, Q')
    plt.xscale('log')
    plt.xlim((500,50000))
    plt.savefig('status_modularity_1.png', dpi = 300)
    plt.show()

    plt.hist(y1, normed=True, bins=30)
    plt.xlabel('Assortativity (Status)')
    plt.ylabel('Density')
    plt.savefig('status_modularity_histogram', dpi = 300)
    plt.show()

    print('Average Status Modularity:' ,sum(y1)/len(y1))


    x2 = np.zeros((100))
    y2 = np.zeros((100))
    ind2 = 0
    for j in major_assortativities:
        temp2=(major_assortativities[j])
        x2[ind2] = temp2[1]
        y2[ind2] = temp2[0]
        ind2 = ind2+1

    #Plotting the Figure showing Major Modularity(Q)
    #vs. Network Size(n)
    plt.scatter(x2,y2, alpha=0.5, color = 'b')
    plt.xlabel('Network size, n')
    plt.ylabel('Major Modularity, Q')
    plt.xscale('log')
    plt.xlim((500,50000))
    plt.savefig('major_modularity_1.png', dpi = 300)
    plt.show()

    plt.hist(y2, normed=True, bins=30)
    plt.xlabel('Assortativity (Major)')
    plt.ylabel('Density')
    plt.savefig('major_modularity_histogram', dpi = 300)
    plt.show()

    print('Average Major Modularity:' ,sum(y2)/len(y2))
```
The code for Problem 4 (c), i.e. for generating the plot and histogram for vertex degree

assortativity is shown below.

```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
import glob

vertex_assortativities = {}
# Calculating the vertex degree assortativity for each network
for filename in glob.glob('*.txt'):
    graph = nx.read_edgelist(filename, nodetype=int,
                             data=(('weight',float),))
    deg_assort = nx.degree_pearson_correlation_coefficient(graph)
    netsize = len(graph.nodes())
    print (filename, deg_assort, netsize)
    vertex_assortativities[filename] = (deg_assort, netsize)

x = np.zeros((100))
y = np.zeros((100))
ind = 0
for j in vertex_assortativities:
    temp=(vertex_assortativities[j])
    x[ind] = temp[1]
    y[ind] = temp[0]
    ind = ind+1

#Plotting the Figure showing Vertex Degree Modularity(Q)
#vs. Network Size(n)
plt.scatter(x,y, alpha=0.5, color = 'b')
xx = np.linspace(500, 50000, num=100)
yy = np.zeros(np.size(xx))
plt.plot(xx,yy,linestyle='-.', color='k', linewidth=1)
plt.xlabel('Network size, n')
plt.ylabel('Vertex Degree Modularity, Q')
plt.xscale('log')
plt.ylim((-0.1,0.25))
plt.xlim((500,50000))
plt.savefig('vertex_degree_modularity_1.png', dpi = 300)
plt.show()

plt.hist(y, normed=True, bins=100)
plt.xlabel('Assortativity (Vertex Degree)')
plt.ylabel('Density')
plt.savefig('vertex_degree_modularity_histogram', dpi = 300)
plt.show()
```