In [58]:

```
import networkx as nx
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import squareform
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.pairwise import cosine_similarity
```

In [59]:

```
! pip install yellowbrick --user
```

```
Requirement already satisfied: yellowbrick in c:\users\user\appdata\roamin
g\python\python38\site-packages (1.3.post1)
Requirement already satisfied: scikit-learn>=0.20 in c:\users\user\anacond
a3\lib\site-packages (from yellowbrick) (0.23.1)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\users\user
\anaconda3\lib\site-packages (from yellowbrick) (3.2.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\user\anaconda3\lib
\site-packages (from yellowbrick) (1.5.0)
Requirement already satisfied: numpy<1.20,>=1.16.0 in c:\users\user\anacon
da3\lib\site-packages (from yellowbrick) (1.19.5)
Requirement already satisfied: cycler>=0.10.0 in c:\users\user\anaconda3\l
ib\site-packages (from yellowbrick) (0.10.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\anaco
nda3\lib\site-packages (from scikit-learn>=0.20->yellowbrick) (2.1.0)
Requirement already satisfied: joblib>=0.11 in c:\users\user\anaconda3\lib
\site-packages (from scikit-learn>=0.20->yellowbrick) (0.16.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\user\anaconda
3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\user\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\user\anaco
nda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.
1)
Requirement already satisfied: six in c:\users\user\anaconda3\lib\site-pac
kages (from cycler>=0.10.0->yellowbrick) (1.15.0)
```

Network Summary

In [136]:

```
vk_df = pd.read_csv("Ivanova_Untitled.csv", sep=';')
vk_df = vk_df.set_index('Unnamed: 0')
```

In [137]:

```
vk_df.head()
```

Out[137]:

Unnamed: 0	14911	16517	17131	23697	31694	32834	70642	72615	77312	81527	...	229
14911	0	1	1	0	0	0	0	0	0	0	...	
16517	1	0	0	0	0	0	0	0	0	0	...	
17131	1	0	0	0	0	0	0	0	0	0	...	
23697	0	0	0	0	0	0	0	1	1	0	...	
31694	0	0	0	0	0	0	0	0	0	0	...	

5 rows × 205 columns

In [138]:

```
attr_df = pd.read_csv("Ivanova_attributes.csv", sep=',')
attr_df = attr_df.set_index('Id')
attr_df = attr_df.drop('timeset', 1)
attr_df.rename(columns={'Label': 'Label', '0': 'Name', '1': 'Surname', '2': 'Nickname',
                        '3': 'Screen Name', '4': 'Sex', '5': 'Photo', '6': 'Relation',
'9': 'BDate'}, inplace=True)
```

In [139]:

```
attr_df.head()
```

Out[139]:

Id	Label	Name	Surname	Nickname	Screen Name	Sex	
14911	Андрей Маркин	Андрей	Маркин	NaN	andrey.markinn	2	84.userapi.cor
16517	Катерина Дуденкова	Катерина	Дуденкова	NaN	pathofmath	1	95.userapi.con
17131	Maksim Sudakov	Maksim	Sudakov	NaN	maksim.sudakov	2	19.userapi.cor
23697	Юлия Московская	Юлия	Московская	NaN	chertacoffe	1	https://vk.com/i
31694	DELETED	DELETED	NaN	NaN	NaN	2	https://vk.com/imag

Node attributes

In [64]:

```
list(attr_df.columns)
```

Out[64]:

```
['Label',
 'Name',
 'Surname',
 'Nickname',
 'Screen Name',
 'Sex',
 'Photo',
 'Relation',
 'BDate']
```

Size, Order, Diameter, Radius

In [65]:

```
vk_df_array = np.array(vk_df)
```

In [66]:

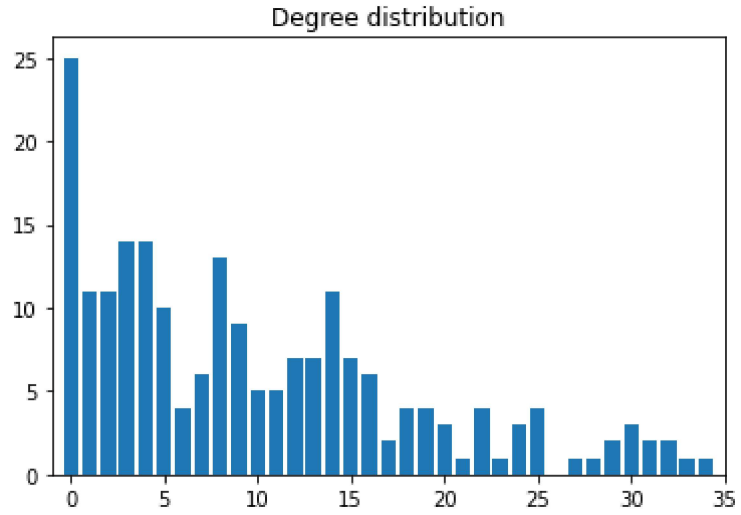
```
G = nx.from_numpy_array(vk_df_array)
nodes = len(G)
edges = len(G.edges)
node = max(nx.connected_components(G), key=len)
g = G.subgraph(node).copy()
diameter = nx.diameter(g)
radius = nx.radius(g)
print('Size =', nodes)
print('Order =', edges)
print('Diameter =', diameter)
print('Radius =', radius)
```

```
Size = 205
Order = 1028
Diameter = 6
Radius = 4
```

Degree distribution

In [67]:

```
degree_seq = [degree for (node, degree) in G.degree]
bins, freq = np.unique(degree_seq, return_counts=True)
plt.bar(bins, freq)
plt.xlim((-1, 35))
plt.title('Degree distribution')
plt.show()
```



In [68]:

```
def power_law_cdf(x, alpha=3.5, x_min=1):
    return 1 - (x / x_min) ** (1 - alpha)
```

In [69]:

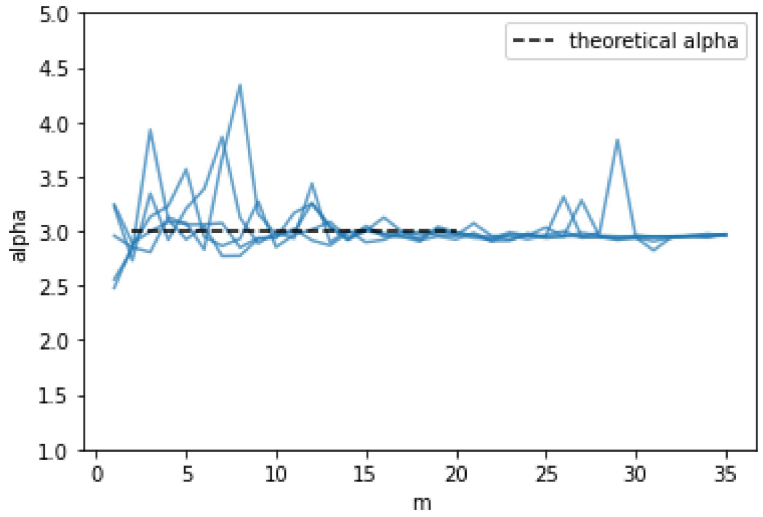
```
def mle_power_law_params(degree_sequence):
    from scipy.stats import kstest
    best_alpha, best_x_min = None, None
    min_ks_dist = np.inf
    for x_min in np.arange(degree_sequence.min(), degree_sequence.max()):
        x = degree_sequence[degree_sequence >= x_min]
        alpha = 1 + len(x) / np.log(x / x_min).sum()
        ks_dist = kstest(x, power_law_cdf, args=(alpha, x_min)).statistic
        if ks_dist < min_ks_dist:
            min_ks_dist, best_alpha, best_x_min = ks_dist, alpha, x_min
    return best_alpha, best_x_min, min_ks_dist
```

In [70]:

```
def estimate_power_law(n, m_min, m_max):
    alphas = []
    k_min = []
    min_ks_dists = []
    for m in range (m_min, m_max + 1):
        G = nx.barabasi_albert_graph(n, m)
        deg = np.array(list(dict(G.degree).values()))
        alpha, x_min, min_ks = mle_power_law_params(deg)
        alphas.append(alpha)
        k_min.append(x_min)
        min_ks_dists.append(min_ks)
    return (np.array(alphas), np.array(k_min), np.array(min_ks_dists))
```

In [71]:

```
n, m_min, m_max = 500, 1, 35
m_space = np.arange(m_min, m_max + 1)
for _ in range(5):
    alpha, k_min, ks_dist = estimate_power_law(n, m_min, m_max)
    plt.plot(m_space, alpha, alpha=0.7, c='tab:blue')
plt.plot([2, 20], [3, 3], 'k--', label='theoretical alpha')
plt.ylim((1, 5))
plt.xlabel('m')
plt.ylabel('alpha')
plt.legend()
plt.show()
```



Average path length

In [72]:

```
path = nx.average_shortest_path_length(g)
print('Average shortest path length =', '{:.4f}'.format(path))
```

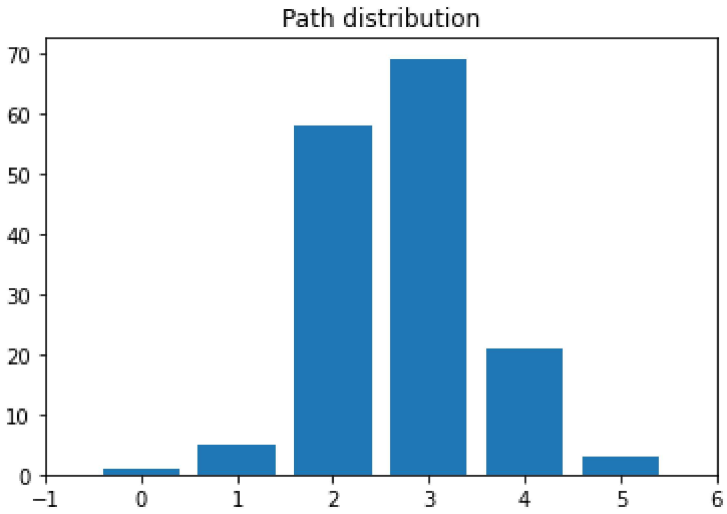
```
Average shortest path length = 2.7832
```

In [73]:

```
p = nx.shortest_path_length(g, source=0)
p = list(p.values())
```

In [74]:

```
bins, freq = np.unique(p, return_counts=True)
plt.bar(bins, freq)
plt.xlim((-1, 6))
plt.title('Path distribution')
plt.show()
```



Clustering Coefficient

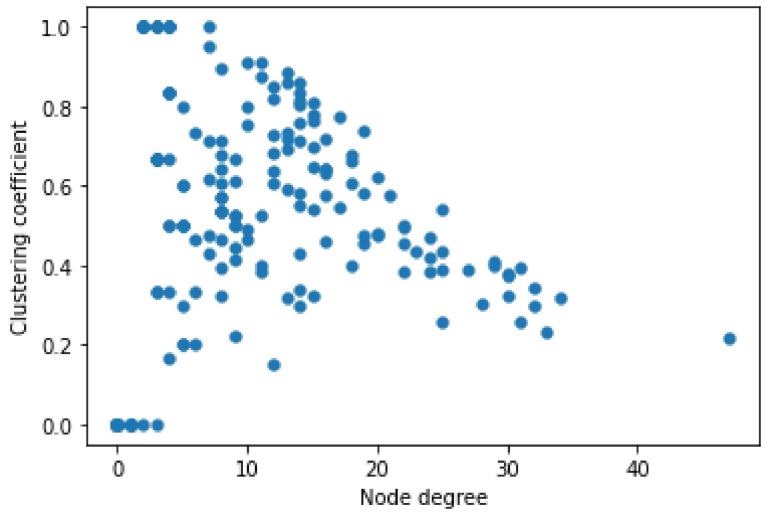
In [75]:

```
coef = np.array(list(nx.clustering(G).values()))
degrees = np.array(list(dict(G.degree).values()))
average_local_coef = nx.average_clustering(G)
global_coef = nx.transitivity(G)
print('Average local coef =', '{:.4f}'.format(average_local_coef))
print('Global coef =', '{:.4f}'.format(global_coef))
```

```
Average local coef = 0.4936
Global coef = 0.4609
```

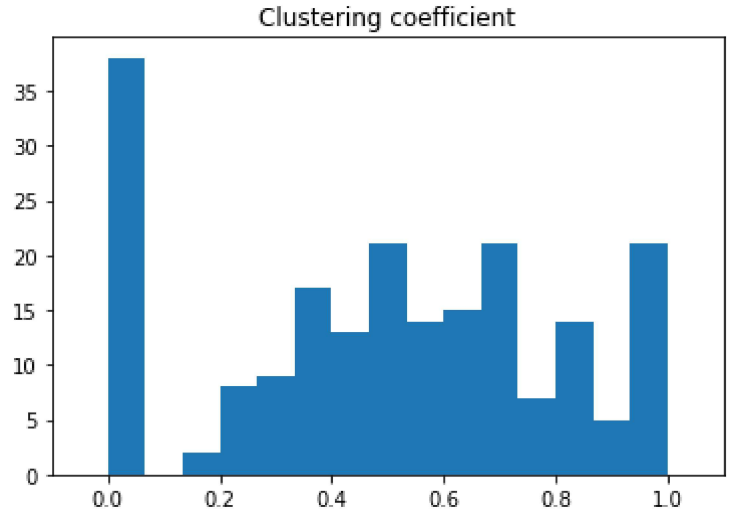
In [76]:

```
plt.scatter(degrees, coef, s=35, linewidths=0.3)
plt.xlabel('Node degree')
plt.ylabel('Clustering coefficient')
plt.show()
```



In [77]:

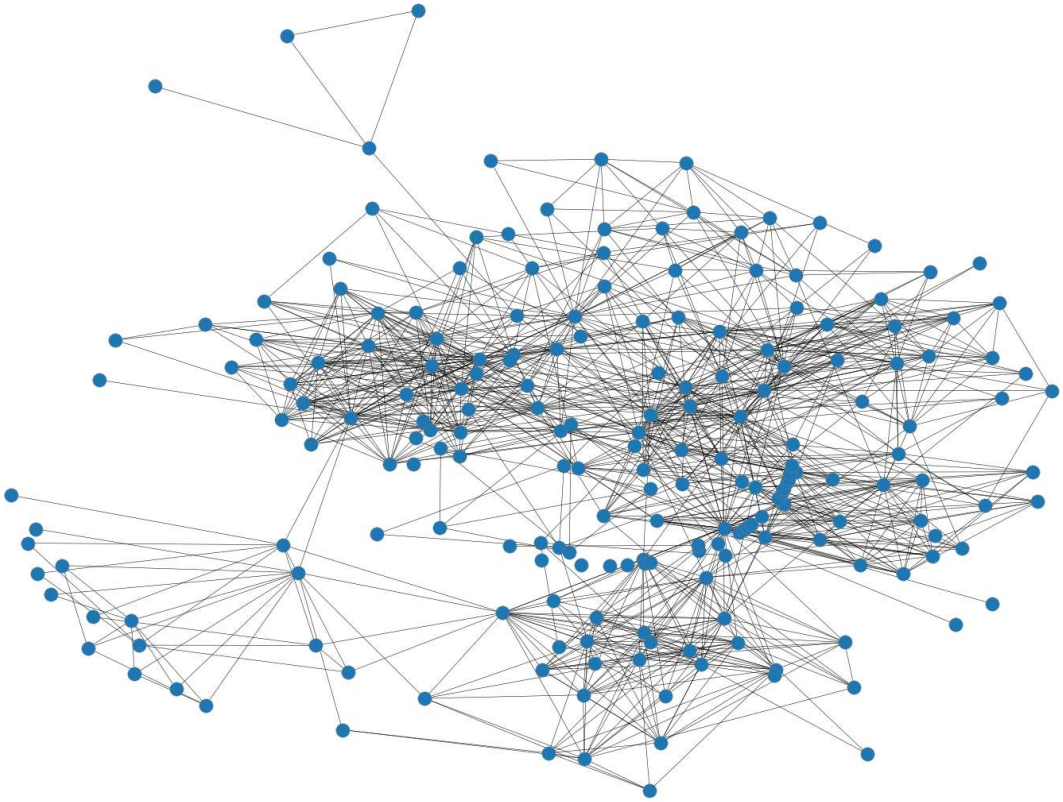
```
plt.hist(coef, 15)
plt.xlim((-0.1, 1.1))
plt.title('Clustering coefficient')
plt.show()
```



Network layout

In [78]:

```
plt.figure(figsize=(20, 15))
pos = nx.kamada_kawai_layout(G)
nx.draw(G,
        pos,
        width=0.5,
        linewidths=0.5,
        edgecolors='gray',
        cmap=plt.cm.hot,
        )

plt.show()
```



Structural Analysis

The closest random graph model

In [79]:

```
G_ER = nx.gnm_random_graph(nodes, edges)
d_ER = nx.diameter(G_ER)
average_clust_ER = nx.average_clustering(G_ER)
degree_seq_ER = [degree for (node, degree) in G_ER.degree]
average_degree_ER = np.mean(degree_seq_ER)
```

In [80]:

```
G_BA = nx.barabasi_albert_graph(nodes, 5)
d_BA = nx.diameter(G_BA)
average_clust_BA = nx.average_clustering(G_BA)
degree_seq_BA = [degree for (node, degree) in G_BA.degree]
average_degree_BA = np.mean(degree_seq_BA)
```

In [81]:

```
G_WS = nx.watts_strogatz_graph(nodes, 10, 0.1)
d_WS = nx.diameter(G_WS)
average_clust_WS = nx.average_clustering(G_WS)
degree_seq_WS = [degree for (node, degree) in G_WS.degree]
average_degree_WS = np.mean(degree_seq_WS)
```

In [82]:

```
G_conf = nx.configuration_model(degree_seq)
G_conf = nx.Graph(G_conf)
nodes = max(nx.connected_components(G_conf), key=len)
g = G_conf.subgraph(nodes).copy()
d_conf = nx.diameter(g)
average_clust_conf = nx.average_clustering(G_conf)
degree_seq_conf = [degree for (node, degree) in G_conf.degree]
average_degree_conf = np.mean(degree_seq_conf)
```

In [83]:

```
def printing (name, diameter, clustering_coefficient, average_degree):
    print(name)
    print('diameter =', '{:.0f}'.format(diameter))
    print('clustering coefficient =', '{:.4f}'.format(clustering_coefficient))
    print('average_degree =', '{:.0f}'.format(average_degree))
    print('------------------------------')

printing('My network', diameter, average_local_coef, np.mean(degrees))
printing('Random  network', d_ER, d_ER, average_degree_ER)
printing('Preferential attachment', d_BA, average_clust_BA, average_degree_BA)
printing('Small world', d_WS, average_clust_WS, average_degree_WS)
printing('Configuration model', d_conf, average_clust_conf, average_degree_conf)
```

```
My network
diameter = 6
clustering coefficient = 0.4936
average_degree = 10
------------------------------
Random  network
diameter = 4
clustering coefficient = 4.0000
average_degree = 10
------------------------------
Preferential attachment
diameter = 4
clustering coefficient = 0.1180
average_degree = 10
------------------------------
Small world
diameter = 6
clustering coefficient = 0.4918
average_degree = 10
------------------------------
Configuration model
diameter = 5
clustering coefficient = 0.1047
average_degree = 9
------------------------------
```

The best model is small Small world

Centrality measures

In [84]:

```
def centralities (type):
    list_centrality_dict = list(type(G).items())
    list_centrality_dict.sort(key=lambda i: i[1], reverse = True)
    unzipped_centrality_dict = zip(*list_centrality_dict)
    unzipped_centrality_dict = list(unzipped_centrality_dict)

    centrality = dict(zip(np.array(vk_df.columns), np.array(list(type(G).values()))))
    list_centrality = list(centrality.items())
    list_centrality.sort(key=lambda i: i[1], reverse = True)
    unzipped_centrality = zip(*list_centrality)
    unzipped_centrality = list(unzipped_centrality)
    top10_centrality = unzipped_centrality[0][:10]
    n = 0
    labels_dict = {}
    names = []
    for i in unzipped_centrality_dict[0][:10]:
        labels_dict[i] = attr_df.loc[int(top10_centrality[n])][1]
        names.append(attr_df.loc[int(top10_centrality[n])][0])
        print(attr_df.loc[int(top10_centrality[n])][0], '{:.4f}'.format(unzipped_centra
lity[1][n]))
        n += 1

    return (labels_dict, names)
```

In [85]:

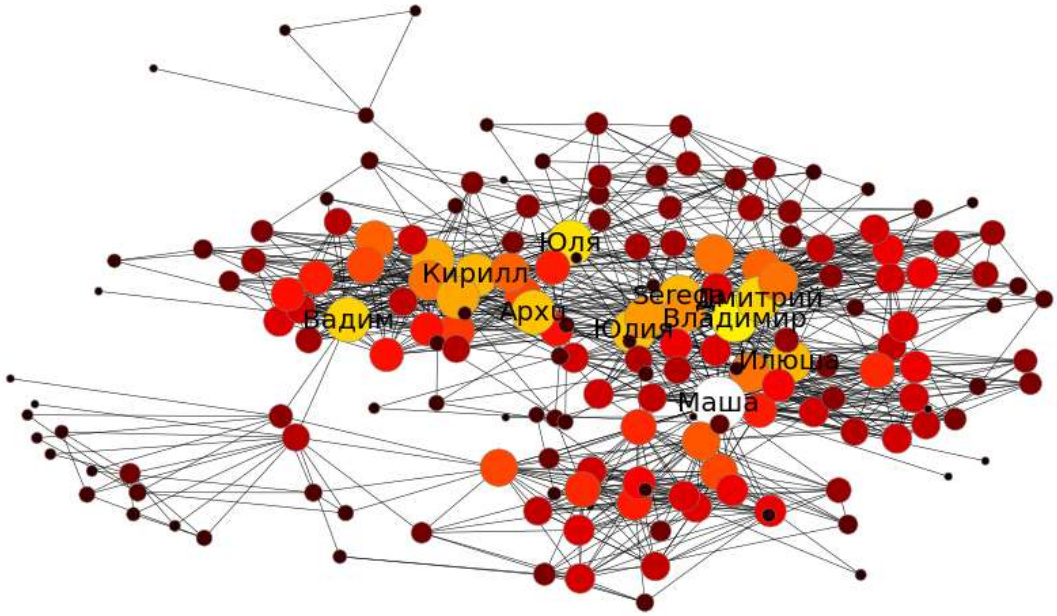
```
print('Degree centrality')
labels_degree_centrality, names_deg = centralities(nx.degree_centrality)
print('------------------------------')
print('Closeness centrality')
labels_closeness_centrality, names_clos = centralities(nx.closeness_centrality)
print('------------------------------')
print('Betweenness centrality')
labels_betweenness_centrality, names_betw = centralities(nx.betweenness_centrality)
print('------------------------------')
```

```
Degree centrality
Маша Чанина 0.2304
Владимир Елизаров 0.1667
Юля Лесковец 0.1618
Вадим Луговой 0.1569
Дмитрий Корнеев 0.1569
Юлия Московская 0.1520
Архи Ваня 0.1520
Serega Frolov 0.1471
Кирилл Батуров 0.1471
Илюша Кузьмин 0.1471
------------------------------
Closeness centrality
Маша Чанина 0.3861
Владимир Елизаров 0.3604
Андрей Дементеев 0.3604
Юлия Московская 0.3572
Юля Лесковец 0.3550
Дмитрий Корнеев 0.3509
Архи Ваня 0.3498
Кирилл Батуров 0.3478
Даша Щербакова 0.3468
Вадим Луговой 0.3458
------------------------------
Betweenness centrality
Маша Чанина 0.1142
Вадим Луговой 0.0939
Вика Иванова 0.0684
Юлия Гудкова 0.0553
Юля Лесковец 0.0548
Ольга Лобанова 0.0399
Дарья Сафонова 0.0390
Анна Луговая 0.0331
Юлия Московская 0.0330
Дмитрий Корнеев 0.0269
------------------------------
```

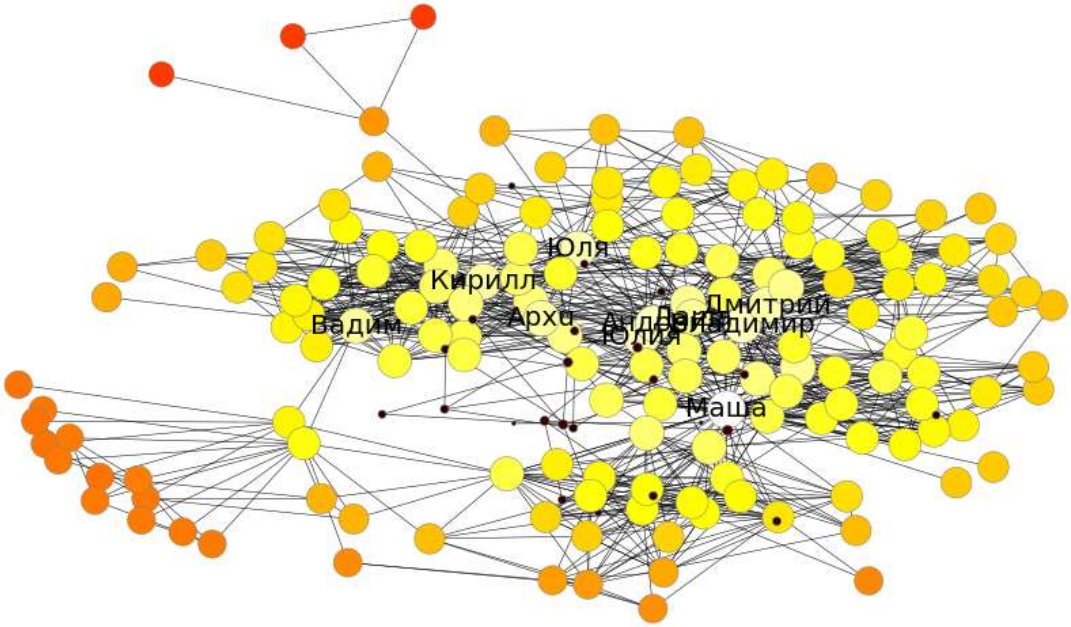
In [86]:

```
plt.figure(figsize=(15, 30))
pos = nx.kamada_kawai_layout(G)

cases = [[1, nx.degree_centrality, labels_degree_centrality, 7e3, 'Degree centrality'],
         [2, nx.closeness_centrality, labels_closeness_centrality, 2e3, 'Closeness cent
rality'],
         [3, nx.betweenness_centrality, labels_betweenness_centrality, 3e4, 'Betweennes
s centrality']]

for i, type_, label, k, t in cases:
    size = np.array(list(type_(G).values()))
    plt.subplot(3, 1, i)

    nx.draw(G,
            pos,
            width=0.5,
            linewidths=0.5,
            edgecolors='gray',
            cmap=plt.cm.hot,
            node_size=size*k,
            node_color=size)
    nx.draw_networkx_labels(G,
            pos,
            label,
            20)
    plt.title(t)

plt.show()
```

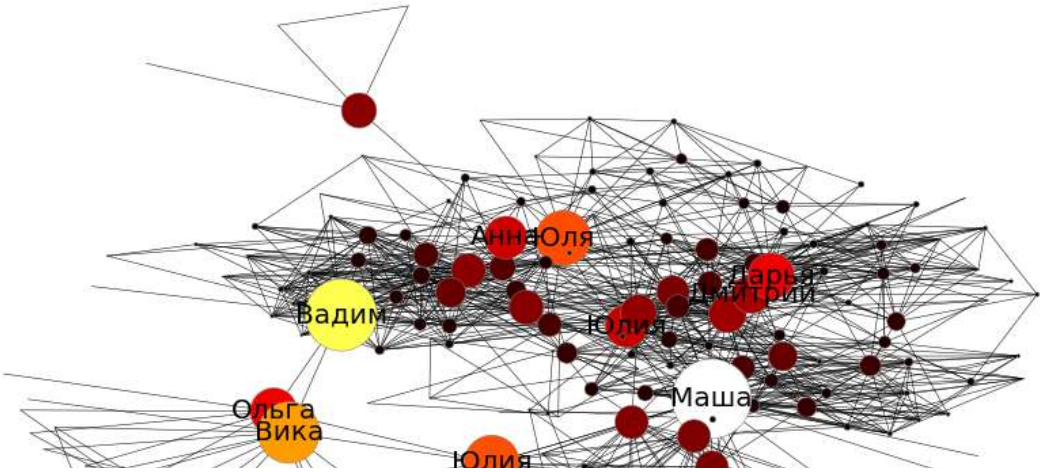
Degree centrality



Closeness centrality



Betweenness centrality



Page-Rank

In [87]:

```
PR_dict = nx.pagerank(G)

PR = dict(zip(np.array(vk_df.columns), np.array(list(PR_dict.values()))))
list_PR = list(PR.items())
list_PR.sort(key=lambda i: i[1], reverse = True)
unzipped_PR = zip(*list_PR)
unzipped_PR = list(unzipped_PR)
top10_PR = unzipped_PR[0][:10]
n = 0
names = []
for i in top10_PR:
    names.append(attr_df.loc[int(top10_PR[n])][0])
    print(attr_df.loc[int(i)][0], '{:.4f}'.format(unzipped_PR[1][n]))
    n += 1
```

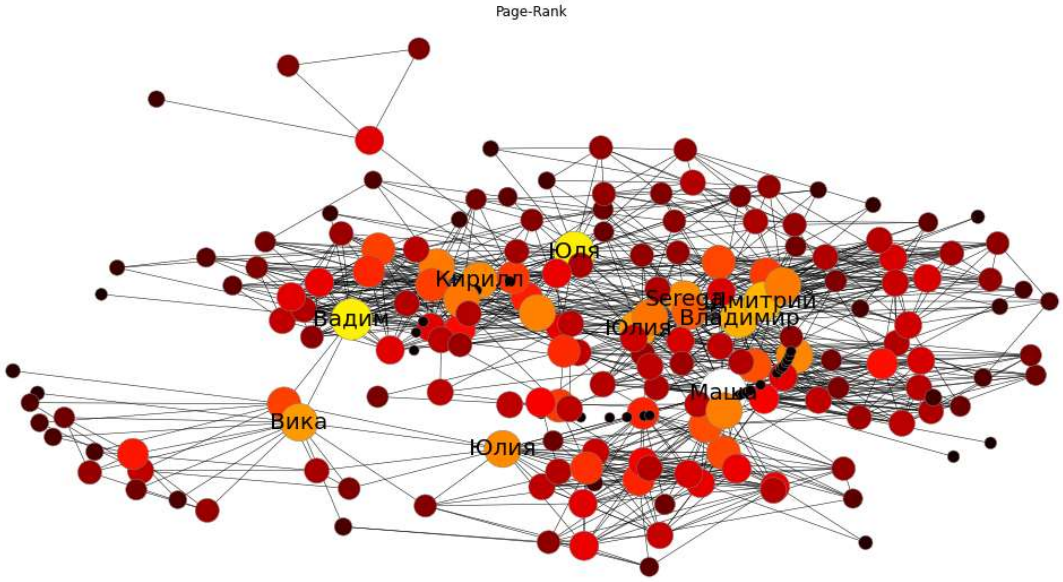
```
Маша Чанина 0.0180
Вадим Луговой 0.0133
Юля Лесковец 0.0132
Дмитрий Корнеев 0.0118
Владимир Елизаров 0.0116
Вика Иванова 0.0111
Юлия Московская 0.0111
Serega Frolov 0.0107
Юлия Гудкова 0.0107
Кирилл Батуров 0.0106
```

In [88]:

```
list_PR_dict = list(PR_dict.items())
list_PR_dict.sort(key=lambda i: i[1], reverse = True)
unzipped_PR_dict = zip(*list_PR_dict)
unzipped_PR_dict = list(unzipped_PR_dict)
n = 0
labels_dict = {}

for i in unzipped_PR_dict[0][:10]:

    labels_dict[i] = attr_df.loc[int(top10_PR[n])][1]
    n += 1
```

In [89]:

```
plt.figure(figsize=(14, 7))
pos = nx.kamada_kawai_layout(G)
PR = np.array(list(PR_dict.values()))

nx.draw(G,
        pos,
        width=0.5,
        linewidths=0.5,
        edgecolors='gray',
        cmap=plt.cm.hot,
        node_size=PR*1e5,
        node_color=PR)
nx.draw_networkx_labels(G,
        pos,
        labels_dict,
        20)
plt.title('Page-Rank')
plt.show()
```



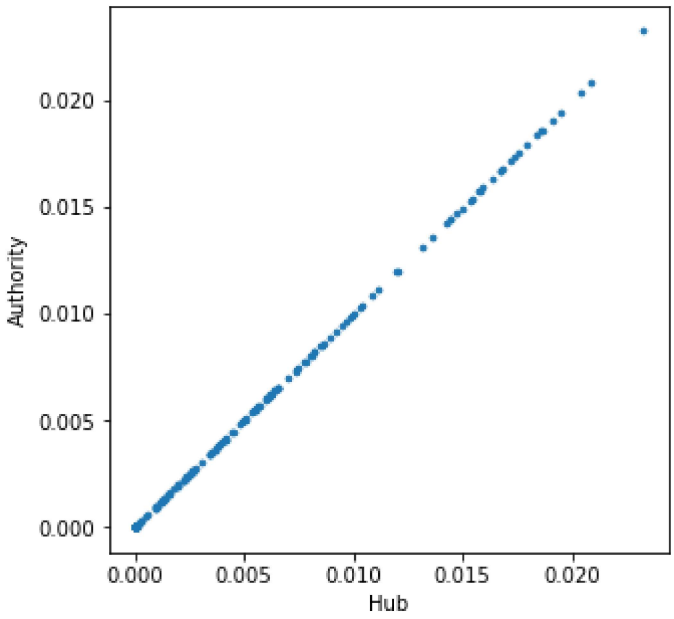
HITS

In [90]:

```
hubs, auth = nx.hits(G)
```

In [91]:

```
plt.figure(figsize=(5, 5))
plt.scatter(hubs.values(), auth.values(), s=7)
plt.xlabel('Hub')
plt.ylabel('Authority')
plt.show()
```



The shape of the graph shows that all of subscriptions are mutual.

Correlation comparison

In [92]:

```
deg_cent = list(nx.degree_centrality(G).values())
clos_cent = list(nx.closeness_centrality(G).values())
betw_cent = list(nx.betweenness_centrality(G).values())
PR = list(PR_dict.values())
centralities = np.array(deg_cent + clos_cent + betw_cent + PR).reshape(4, len(G))
correlation = pd.DataFrame(np.corrcoef(centralities))

correlation.rename({0: 'degree', 1: 'closeness', 2: 'betweenness', 3: 'page rank'}, axi
s=1, inplace = True)
correlation.rename({0: 'degree', 1: 'closeness', 2: 'betweenness', 3: 'page rank'}, axi
s=0)
```

Out[92]:

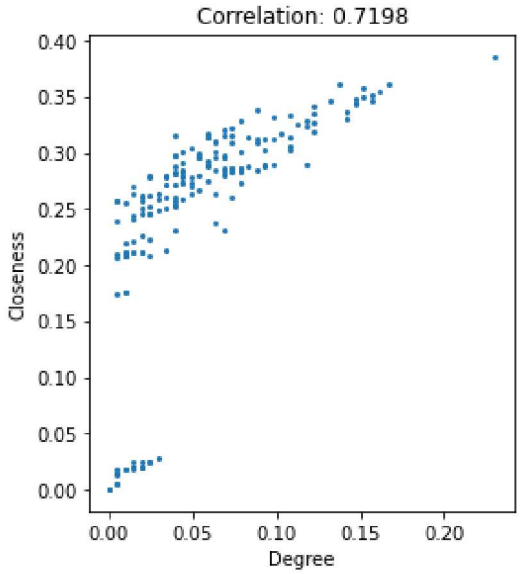
	degree	closeness	betweenness	page rank
degree	1.000000	0.719803	0.608984	0.883192
closeness	0.719803	1.000000	0.339194	0.523375
betweenness	0.608984	0.339194	1.000000	0.686964
page rank	0.883192	0.523375	0.686964	1.000000

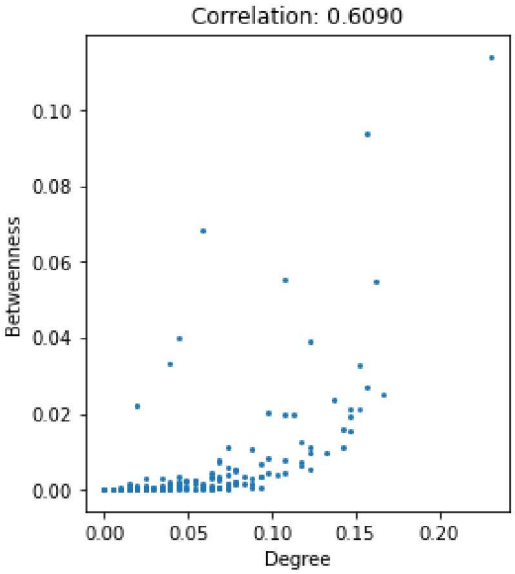
In [93]:

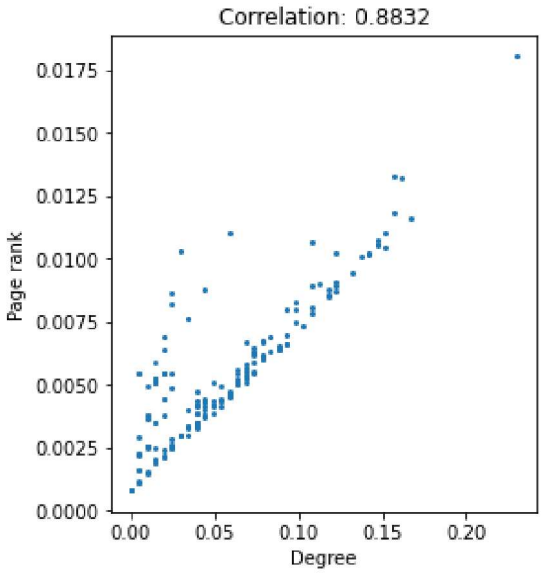
```
def pearson_correlation(i_vals: np.array, j_vals: np.array) -> float:
    return abs(np.corrcoef(i_vals, j_vals)[0][1])
```

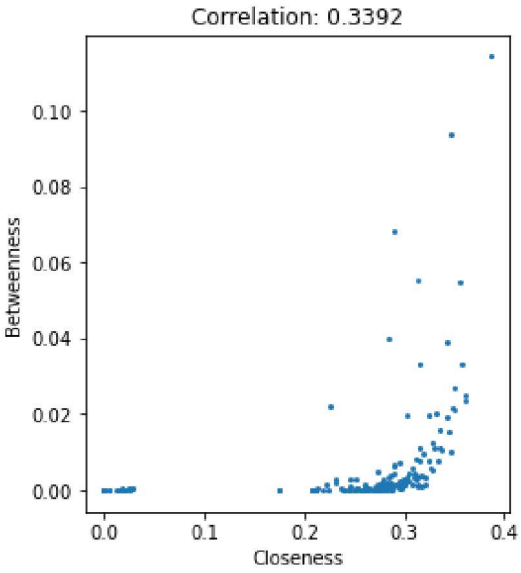
In [94]:

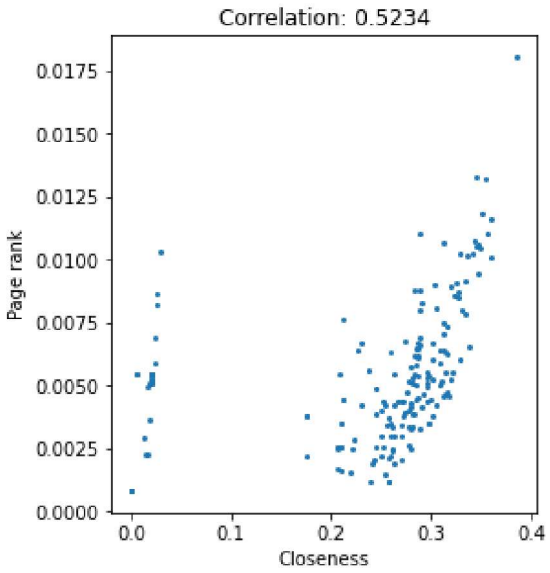
```
centvals = [
    ('Degree', deg_cent),
    ('Closeness', clos_cent),
    ('Betweenness', betw_cent),
    ('Page rank', PR)]
plt.figure(figsize=(2*5, 5*5))
plt.subplots_adjust(hspace=0.4, wspace=0.4)
k = 1

for i in range(len(centvals)):
    for j in range(i + 1, len(centvals)):
        i_label, i_vals = centvals[i]
        j_label, j_vals = centvals[j]
        correlation = pearson_correlation(i_vals, j_vals)
        plt.subplot(4, 2, k)
        plt.scatter(i_vals, j_vals, s=4)
        plt.title('Correlation: {:.4f}'.format(correlation))
        plt.xlabel(i_label)
        plt.ylabel(j_label)
        k += 1
```

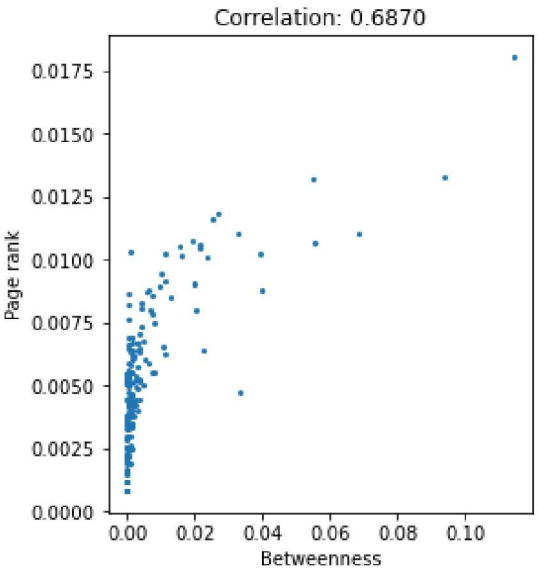

Correlation: 0.7198
Closeness
Degree


Correlation: 0.6090
Betweenness
Degree


Correlation: 0.8832
Page rank
Degree


Correlation: 0.3392
Betweenness
Closeness


Correlation: 0.5234
Page rank
Closeness


Correlation: 0.6870
Page rank
Betweenness

In [95]:

```
top_names = np.array(names_deg + names_clos + names_betw + names).reshape(4, 10)
top_names = pd.DataFrame(top_names)
top_names.rename({0: 'degree', 1: 'closeness', 2: 'betweenness', 3: 'page rank'}, axis=
0, inplace = True)
```

In [96]:

```
top_names
```

Out[96]:

	0	1	2	3	4	5	6
degree	Маша Чанина	Владимир Елизаров	Юля Лесковец	Вадим Луговой	Дмитрий Корнеев	Юлия Московская	Архи Ван
closeness	Маша Чанина	Владимир Елизаров	Андрей Демендеев	Юлия Московская	Юля Лесковец	Дмитрий Корнеев	Архи Ван
betweenness	Маша Чанина	Вадим Луговой	Вика Иванова	Юлия Гудкова	Юля Лесковец	Ольга Лобанова	Дарь Сафонов
page rank	Маша Чанина	Вадим Луговой	Юля Лесковец	Дмитрий Корнеев	Владимир Елизаров	Вика Иванова	Юли Московск

Assortative Mixing

In [97]:

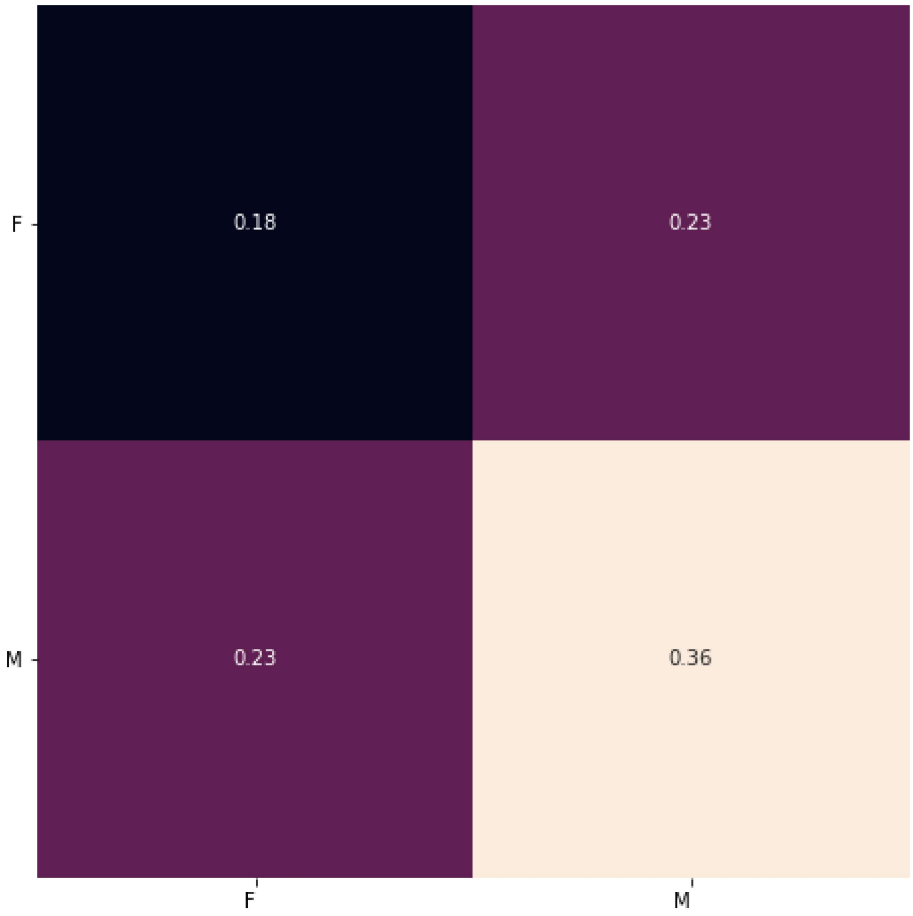
```
attr_sex = dict(zip(list(G.nodes), list(attr_df['Sex'])))

nx.set_node_attributes(G, attr_sex, "Sex")
```

In [98]:

```
mixing, mapping = nx.attribute_mixing_matrix(G, 'Sex'), nx.attribute_mixing_dict(G, 'Se
x')

mapping['F'] = mapping.pop(1)
mapping['M'] = mapping.pop(2)
mapping['F']['F'] = mapping['F'].pop(1)
mapping['F']['M'] = mapping['F'].pop(2)
mapping['M']['F'] = mapping['M'].pop(1)
mapping['M']['M'] = mapping['M'].pop(2)
```

In [99]:

```
fig = plt.figure(figsize=(8, 8))
hmap = sns.heatmap(
    mixing,
    cbar=False,
    annot=True,
    square=True)

hmap.set_xticklabels(
    labels=[m for m in mapping],
    rotation=0,
    horizontalalignment='right')

hmap.set_yticklabels(
    labels=[m for m in mapping],
    rotation=0)

plt.show()
```



Only sex was available for this analysis as country, city and education are not mandatory.

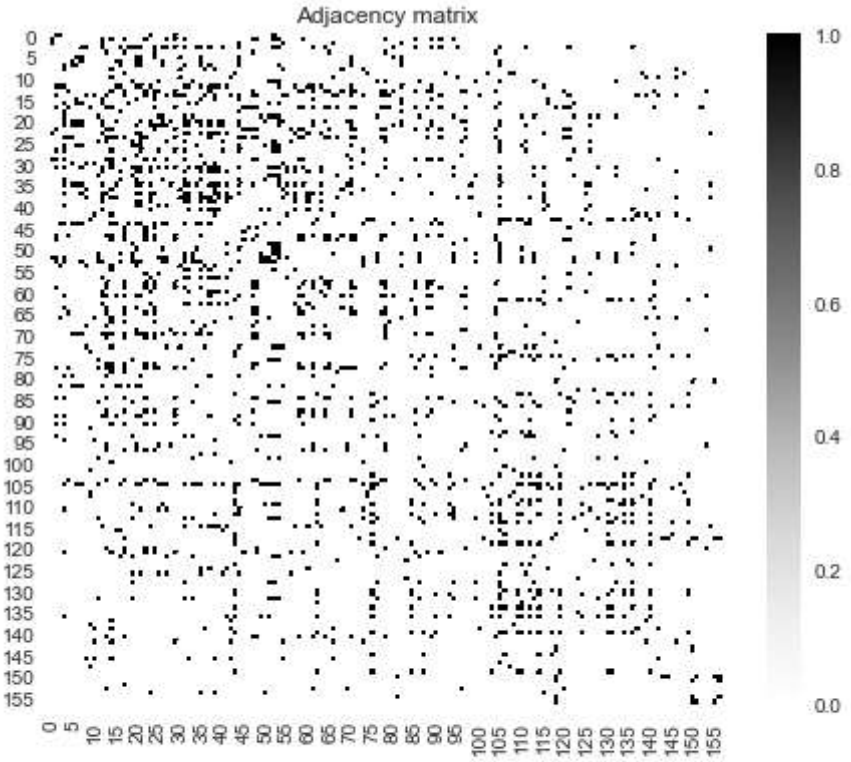
Node structural equivalence/similarity

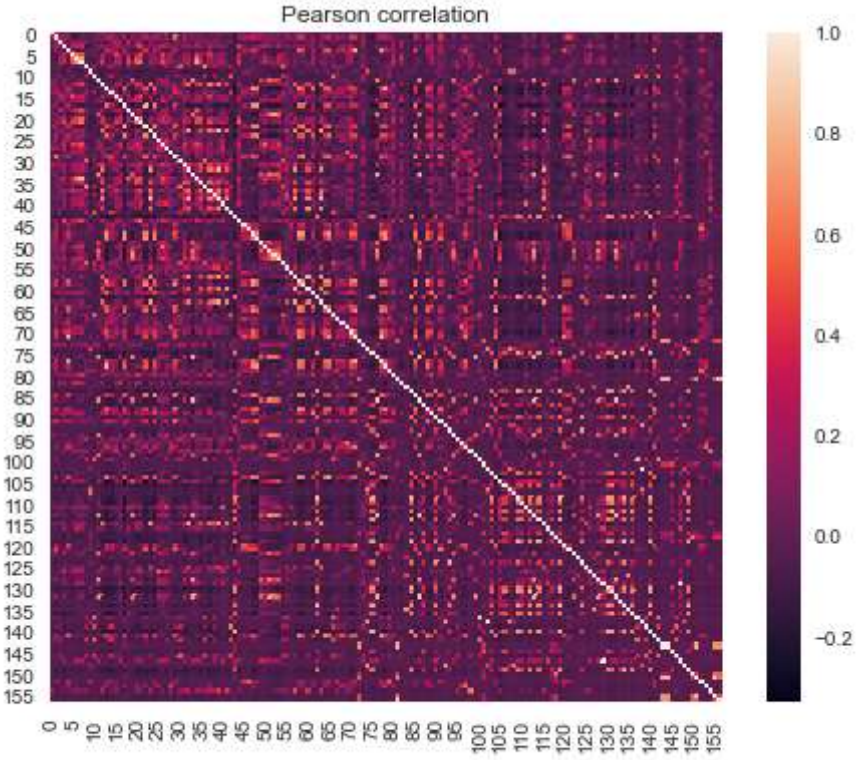
In [100]:

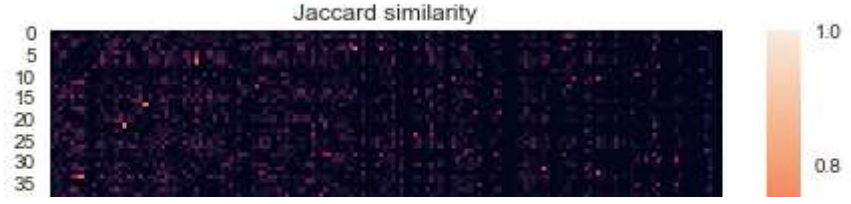
```
def sim_matrices(G):
    A = nx.to_numpy_array(G)
    corr = np.corrcoef(A)
    J = np.zeros(A.shape)
    for i, j, c in nx.jaccard_coefficient(nx.from_numpy_array(A)):
        J[i, j] = c
        J[j, i] = c
    cos = cosine_similarity(A)
    return A, corr, J, cos
```

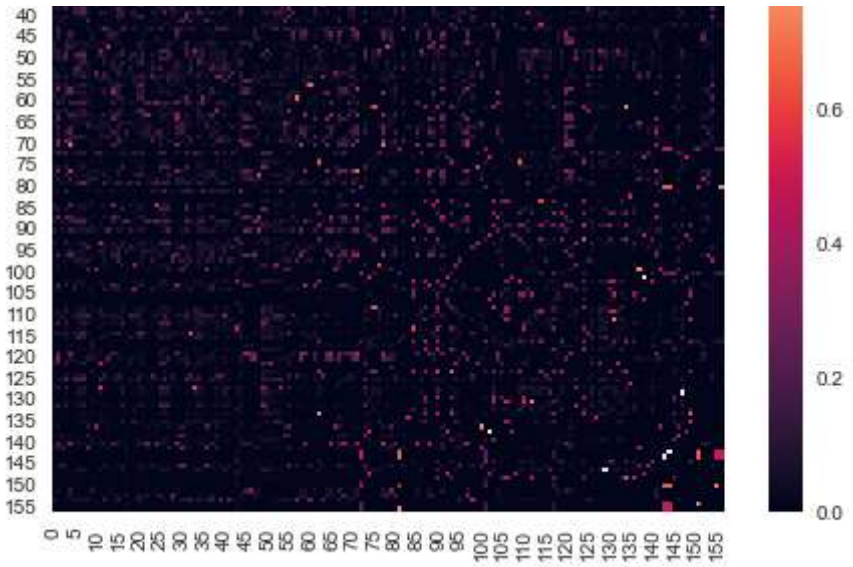
In [135]:

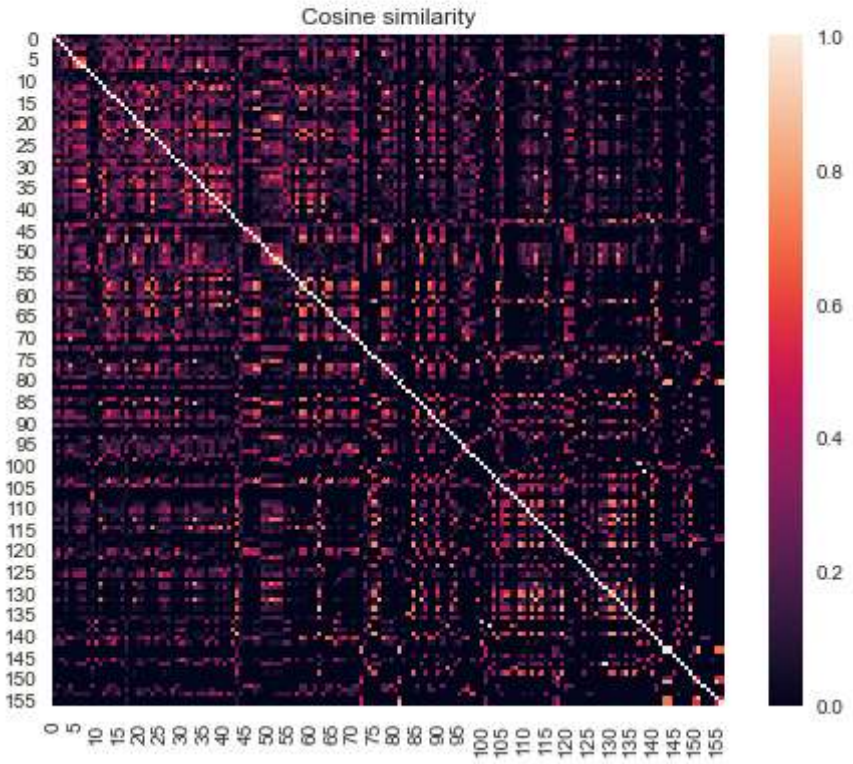
```
node = max(nx.connected_components(G), key=len)
g = G.subgraph(node).copy()
A, corr, J, cos = sim_matrices(g)
fig = plt.figure(figsize=(8, 8*4))
plt.subplots_adjust(hspace=0.4, wspace=0.4)
cases = [[1, A, plt.cm.Greys, 'Adjacency matrix'],
         [2, corr, None, 'Pearson correlation'],
         [3, J, None, 'Jaccard similarity'],
         [4, cos, None, 'Cosine similarity'], ]

for i, matrix, cmap, t in cases:
    plt.subplot(4, 1, i)
    hmap = sns.heatmap(
        matrix,
        cmap=cmap,
        square=True)
    plt.title(t)
```


Adjacency matrix
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155
1.0 0.8 0.6 0.4 0.2 0.0


Pearson correlation
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155
1.0 0.8 0.6 0.4 0.2 0.0 −0.2


Jaccard similarity
0 5 10 15 20 25 30 35
1.0 0.8





Community Detection

Clique search

In [50]:

```
print('Number of cliques', len(list(nx.find_cliques(G))))
```

```
Number of cliques 445
```

In [108]:

```
def largest_cliques(G):
    cliques = list(nx.find_cliques(G))
    i_max = 0
    for i in range(len(cliques)):
        if len(cliques[i]) > i_max:
            i_max = len(cliques[i])
    max_cliques = []
    for i in range(len(cliques)):
        if len(cliques[i]) == i_max:
            max_cliques.append(cliques[i])
    rgb = np.ones((len(max_cliques), len(G), 3))
    width = np.ones((len(max_cliques), len(G.edges)))
    blue = np.array([127, 127, 254])/255.0
    for j in range(rgb.shape[0]):
        sub_G = nx.Graph()
        sub_G.add_nodes_from(max_cliques[j])
        sub_G = nx.complete_graph(sub_G)
        edges = list(sub_G.edges)
        for edge in edges:
            for k in range(width.shape[1]):
                  if set(np.array(G.edges)[k]) == set(edge):
                    width[j,k] = 2
        for i in range(i_max):
            rgb[j,np.where(np.array(G.nodes) == max_cliques[j][i])[0][0]] = blue
    return (rgb, width)
```

In [109]:

```
colors, widths = largest_cliques(G)
```

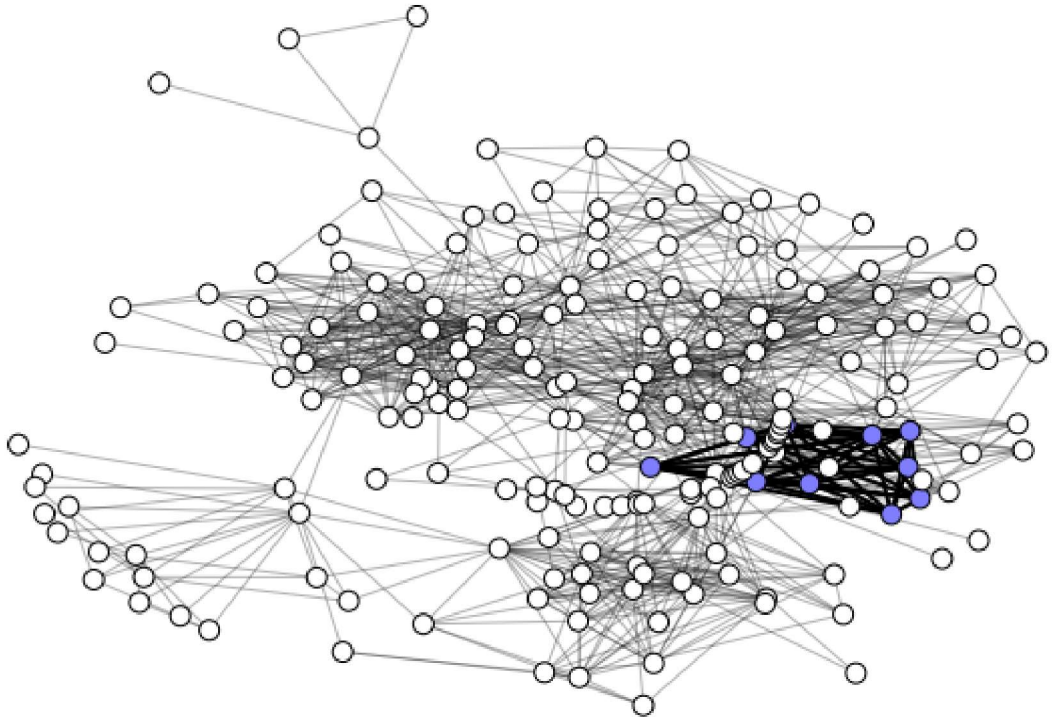
In [110]:

```
size = np.unique(colors[0], axis=0, return_counts=True)[1][0]
```

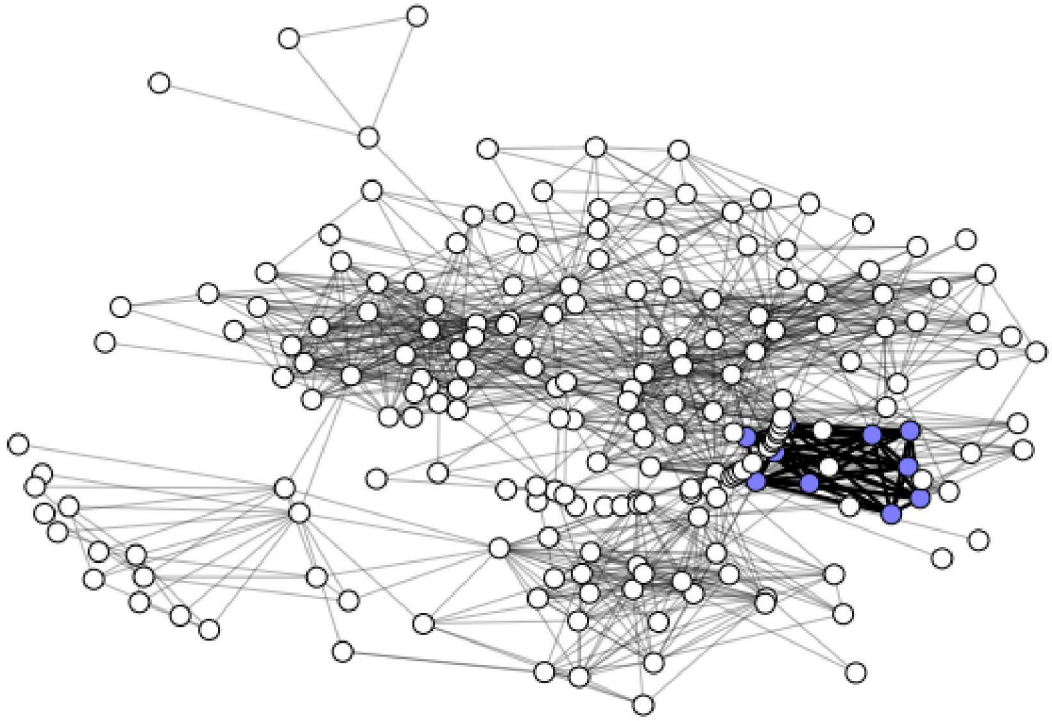
In [113]:

```
plt.figure(figsize=(10, 32))
i = 0
for i in range(colors.shape[0]):
    b_edges = np.array(list(G.edges))[widths[i] == widths[i].max()]
    b_edges = [tuple(l) for l in b_edges]
    plt.subplot(4, 1, i+1)
    nodes = nx.draw_networkx_nodes(
        G,
        pos,
        node_color=colors[i],
        node_size=100,
        linewidths=1,
        edgecolors='black'
    )
    nx.draw_networkx_edges(
        G,
        pos,
        alpha=0.3,
        width=widths[i].min()
    )
    nx.draw_networkx_edges(
        G,
        pos,
        width=widths[i].max(),
        edgelist=b_edges
    )

    plt.title('Clique of the size {}'.format(size))
    plt.axis('off')
```

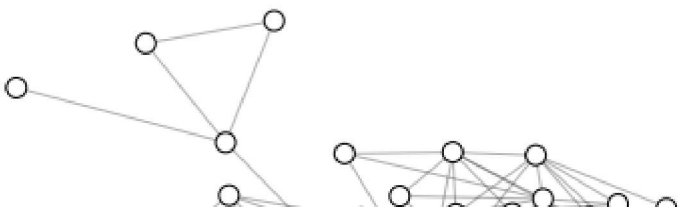
Clique of the size 10

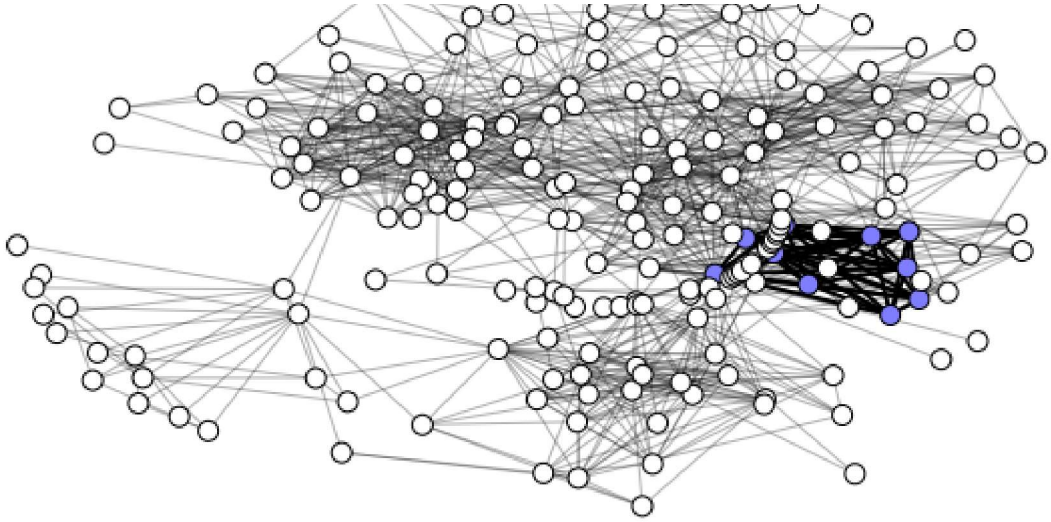


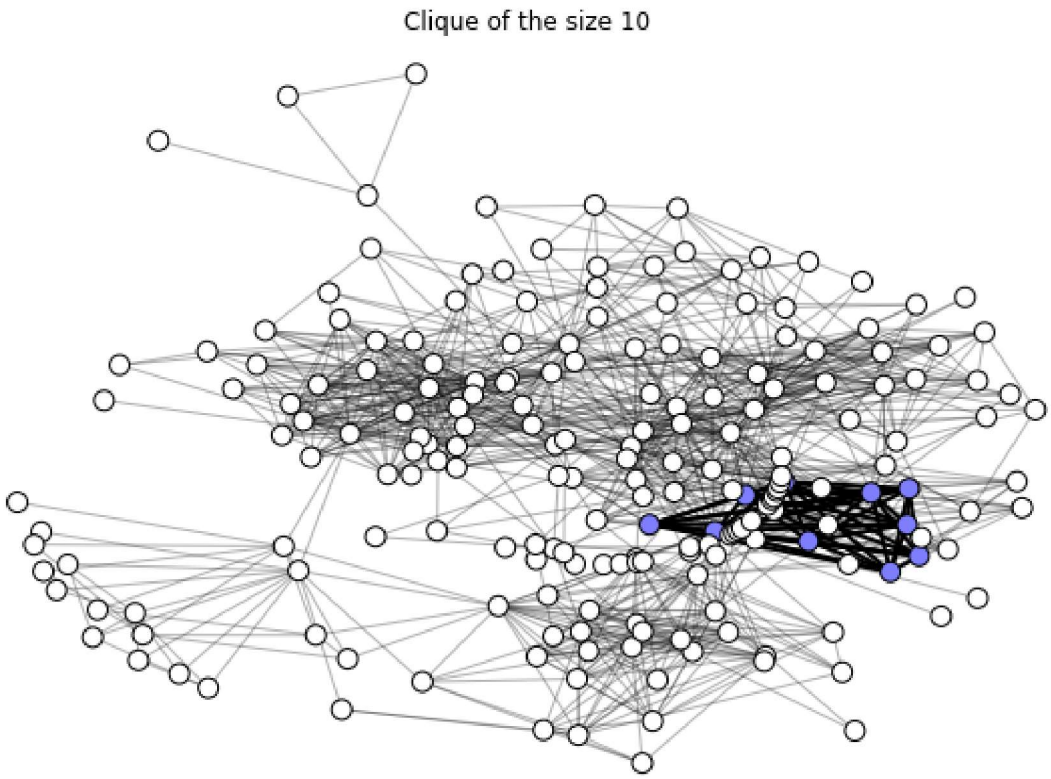
Clique of the size 10



Clique of the size 10







k-cores visualization

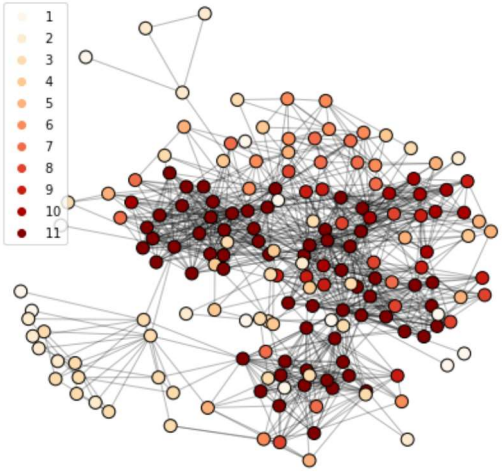
In [114]:

```
def k_core_decompose(G):
    return np.fromiter(nx.core_number(G).values(), int)
```

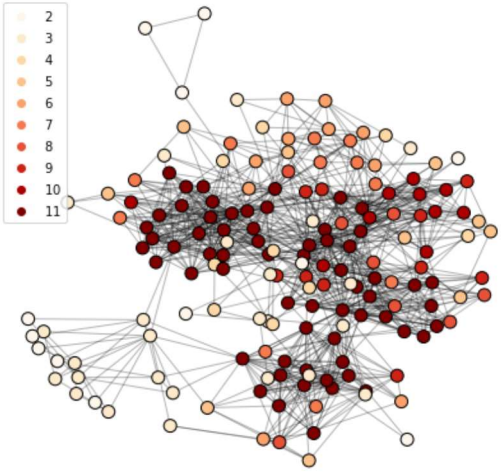
In [115]:

```
plt.figure(figsize=(8*2, 8*4))
x_max, y_max = np.array(list(pos.values())).max(axis=0)
x_min, y_min = np.array(list(pos.values())).min(axis=0)
for i in range(8):
    plt.subplot(4, 2, i+1)
    subG = nx.k_core(G, i+1)
    nodes = nx.draw_networkx_nodes(
        subG,
        pos,
        cmap=plt.cm.OrRd,
        node_color=k_core_decompose(subG),
        node_size=100,
        edgecolors='black'
    )
    nx.draw_networkx_edges(
        subG,
        pos,
        alpha=0.3,
        width=1,
        edge_color='black'
    )

    eps = (x_max - x_min) * 0.05
    plt.xlim(x_min-eps, x_max+eps)
    plt.ylim(y_min-eps, y_max+eps)
    plt.legend(*nodes.legend_elements())
    plt.axis('off')
    plt.title('k-shells on {}-core'.format(i+1))
```

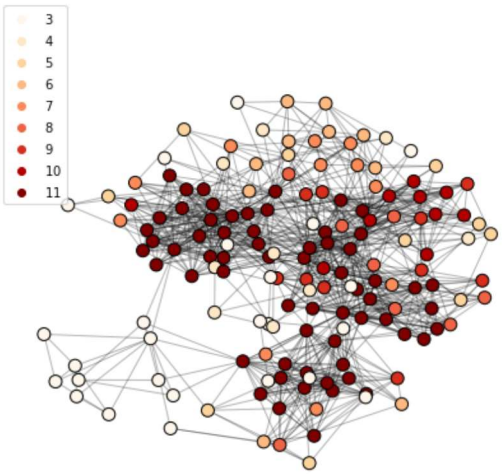
k-shells on 1-core



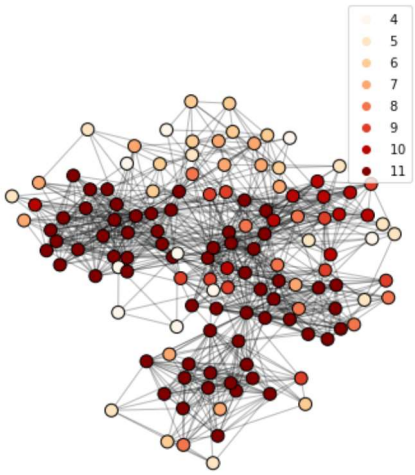
k-shells on 2-core



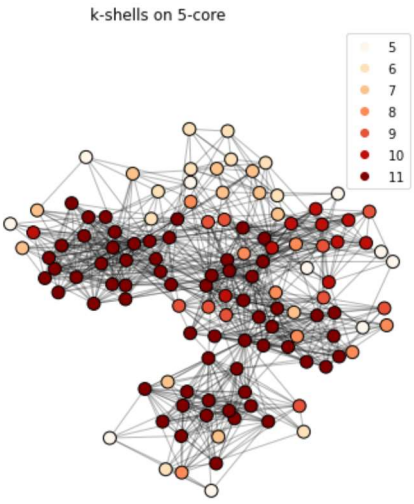
k-shells on 3-core



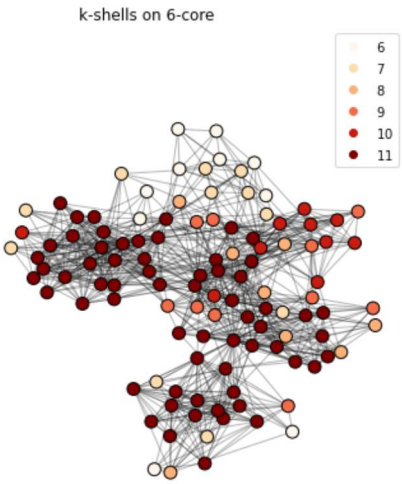
k-shells on 4-core



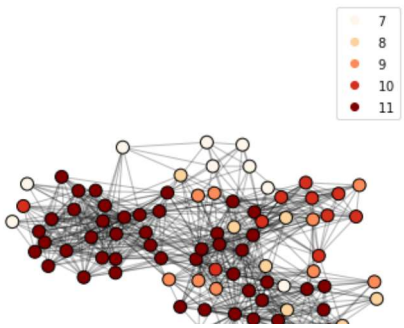
k-shells on 5-core



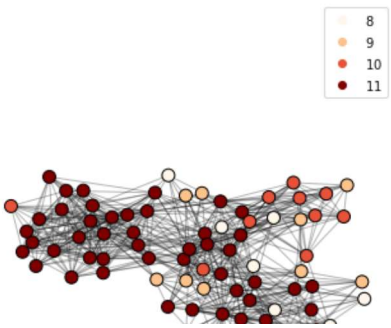
k-shells on 6-core

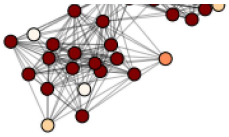


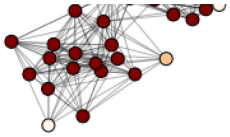
k-shells on 7-core



k-shells on 8-core







Community detection algorithms

Girvan Newman algorithm

In [116]:

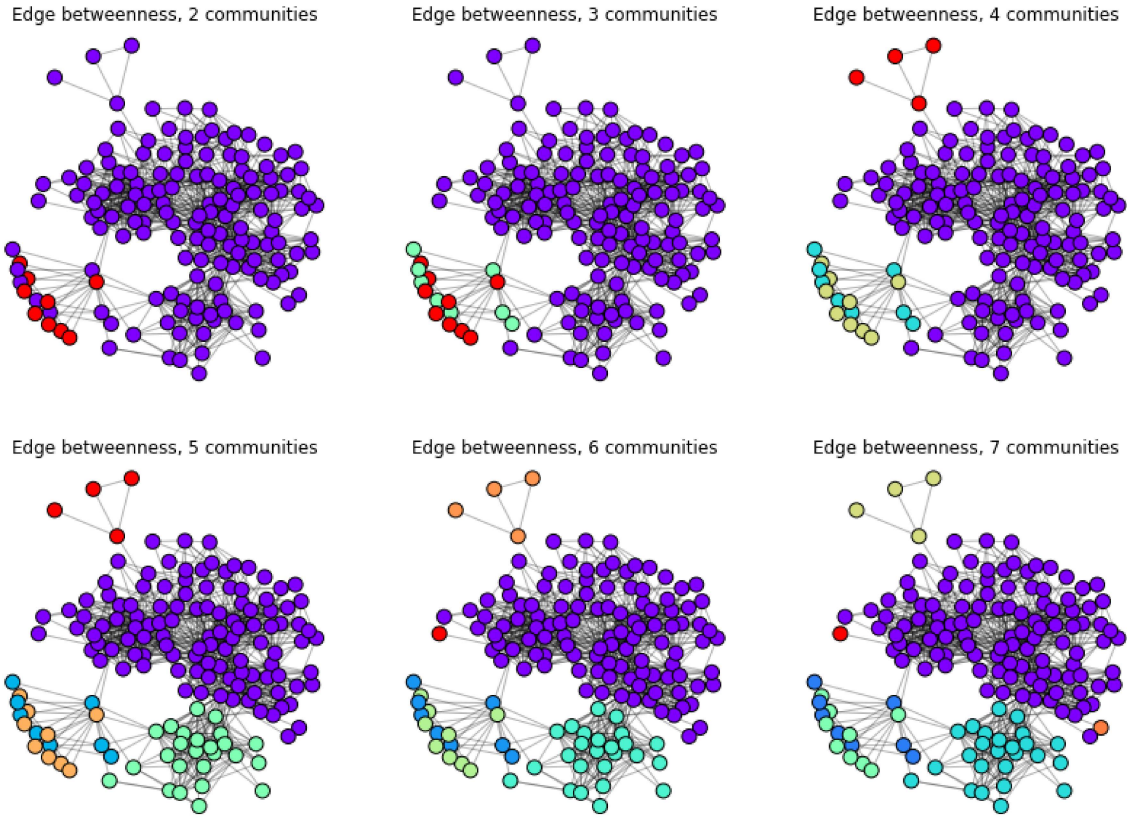
```
node = max(nx.connected_components(G), key=len)
g = G.subgraph(node).copy()
```

In [117]:

```
def edge_betweenness(G, n):
    com_gen = nx.algorithms.community.girvan_newman(G)
    labels = []
    for i in range(n):
        communities = next(com_gen)
        c_com = []
        for node in G.nodes:
            for i, c in enumerate(communities):
                if node in c:
                    c_com.append(i)
        labels.append(c_com)
    return np.array(labels)
```

In [118]:

```
plt.figure(figsize=(7*2, 7*3))
colors = edge_betweenness(g, 6)
for i in range(colors.shape[0]):
    plt.subplot(4, 3, i+1)
    nx.draw_networkx_nodes(
        g,
        pos,
        cmap=plt.cm.rainbow,
        node_color=colors[i],
        node_size=100,
        edgecolors='black'
    )
    nx.draw_networkx_edges(g, pos, alpha=0.3)
    plt.title('Edge betweenness, {} communities'.format(i+2))
    plt.axis('off')
```

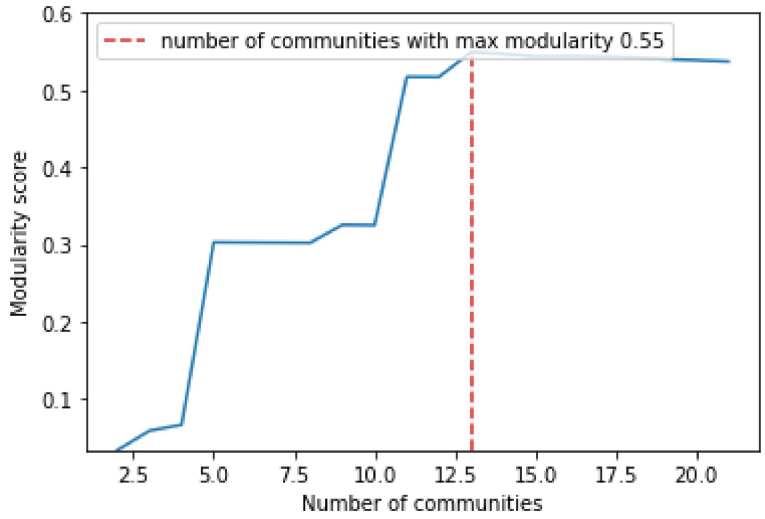


In [119]:

```
def edge_betw_modularity(G, n):

    return np.array([nx.algorithms.community.modularity(G, x) for _, x in zip(range(n),
nx.algorithms.community.girvan_newman(G))])
```

In [120]:

```
n_iterations = 20
modularity = edge_betw_modularity(g, n_iterations)
plt.figure(figsize=(6, 4))
plt.plot(np.arange(n_iterations)+2, modularity)
best_n = np.argmax(modularity) + 2
label = 'number of communities with max modularity {:.2f}'.format(max(modularity))
plt.plot(
    [best_n, best_n], [min(modularity), max(modularity)],
    'k--', c='tab:red',
    label=label
)
plt.ylabel('Modularity score')
plt.xlabel('Number of communities')
plt.legend(loc='upper left')
plt.ylim((modularity.min(), 0.6))
plt.show()
```



Laplacian Eigenmaps

In [121]:

```
from scipy.linalg import fractional_matrix_power
from sklearn.cluster import KMeans

def norm_laplacian(A):
    D = np.diag(np.sum(A, axis=1))
    D_powered = fractional_matrix_power(D, -0.5)
    L = D - A
    L_norm = D_powered@L@D_powered
    return (L_norm, np.sum(A, axis=1))
```

In [122]:

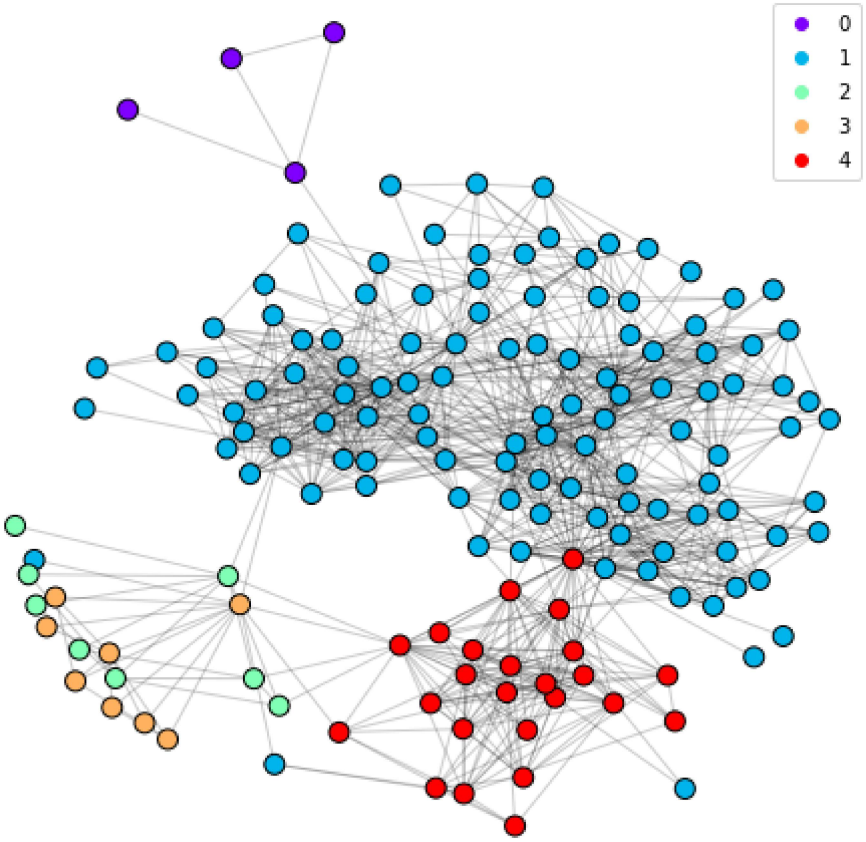
```
def spectral_embedding(L, degree_seq, n_components):
    vals, vecs = np.linalg.eigh(L)
    for i in range(len(degree_seq)):
        vecs[:,i] = vecs[:,i] /(np.sqrt(degree_seq[i])+10**(-10))
    return (vecs[:,1:n_components+1])
```

In [123]:

```
def spectral_clustering(G, n_clusters, n_components):
    A = nx.to_numpy_array(G)
    L, degree_seq = norm_laplacian(A)
    embedding = spectral_embedding(L, degree_seq, n_components)
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(embedding)
    return kmeans.labels_, embedding
```

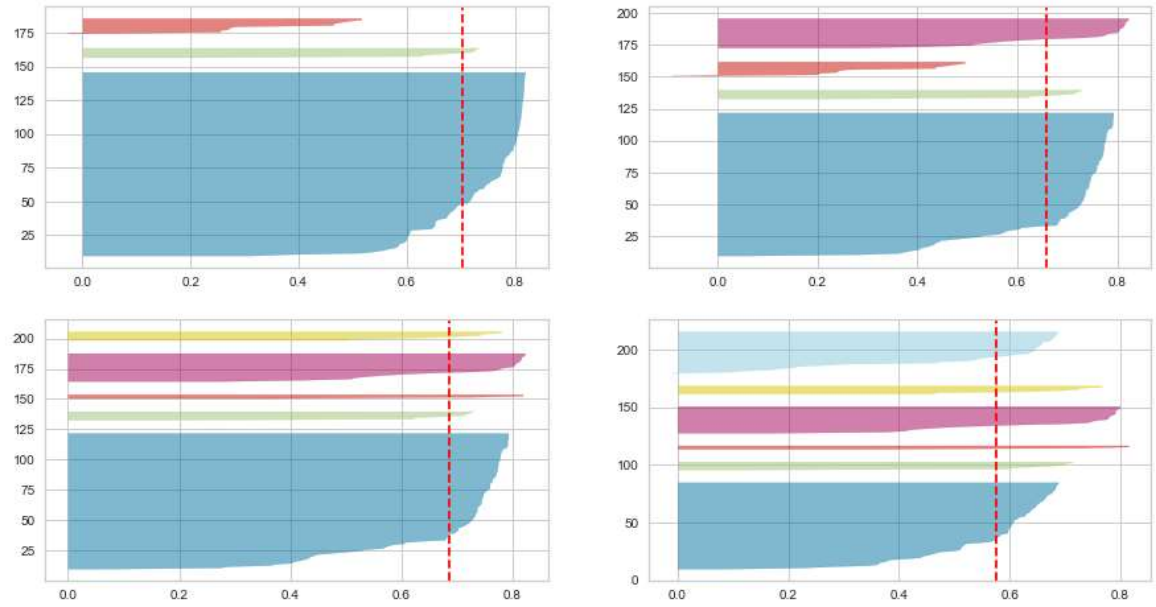
In [124]:

```
plt.figure(figsize=(8, 8))
node_color, embedding =spectral_clustering(g, 5, 4)
nodes = nx.draw_networkx_nodes(
    g,
    pos,
    cmap=plt.cm.rainbow,
    node_color = node_color,
    node_size=100,
    linewidths=1,
    edgecolors='black'
)

nx.draw_networkx_edges(
    g,
    pos,
    alpha=0.2,
    edge_color='black'
)

plt.axis('off')
plt.legend(*nodes.legend_elements())
plt.show()
```


0
1
2
3
4

In [125]:

```
from yellowbrick.cluster import SilhouetteVisualizer

fig, ax = plt.subplots(2, 2, figsize=(15,8))
for i in [3, 4, 5, 6]:
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_state=4
2)
    q, mod = divmod(i-1, 2)

    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(embedding)
```



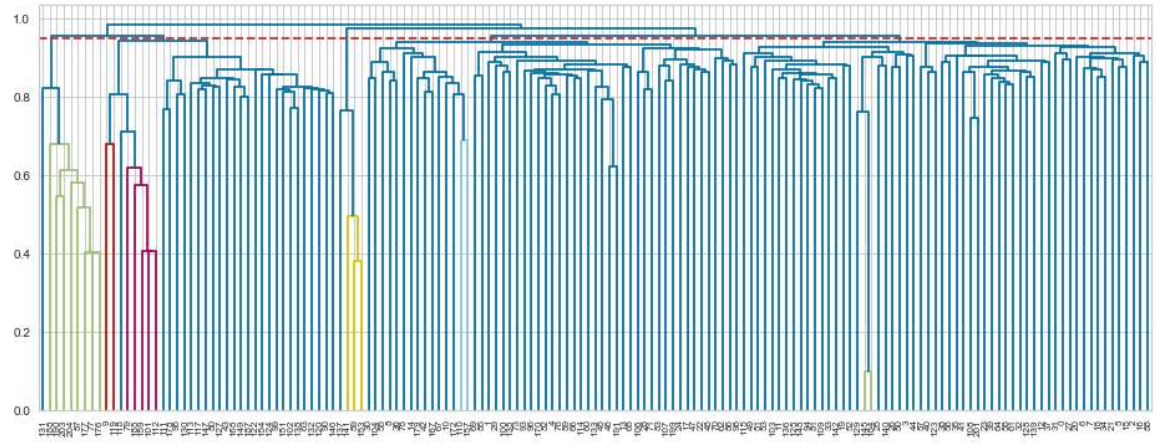
Agglomerative clustering

In [126]:

```
def simrank_distance(G):
    return 1 - nx.simrank_similarity_numpy(G).round(3)
```

In [127]:

```
distance = simrank_distance(g)
plt.figure(figsize=(16, 6))
linked = linkage(squareform(distance), 'complete')
dendrogram(linked, labels=list(g.nodes),
           leaf_font_size=8)
plt.plot([0, 2000], [0.95, 0.95], 'k--', c='tab:red')
plt.show()
```



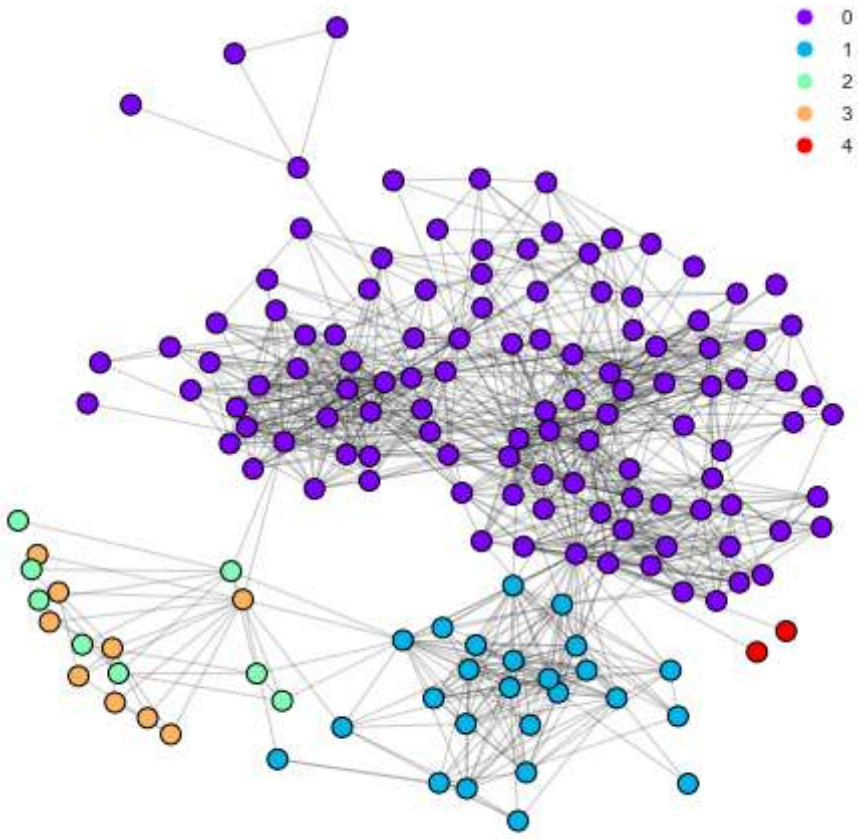
In [128]:

```
def agglomerative_clustering(distance, max_distance):
    linked = linkage(squareform(distance), 'complete')
    n_clusters = sum(linked[:,2] > max_distance) +1
    clustering = AgglomerativeClustering(n_clusters=n_clusters, linkage = 'complete')
    clustering.fit(distance)
    labels = np.array(clustering.labels_).tolist()
    return labels
```

In [129]:

```
labels = agglomerative_clustering(distance, 0.95)
plt.figure(figsize=(8, 8))
nodes = nx.draw_networkx_nodes(
    g,
    pos,
    cmap=plt.cm.rainbow,
    node_color=labels,
    node_size=100,
    linewidths=1,
    edgecolors='black'
)

nx.draw_networkx_edges(
    g,
    pos,
    alpha=0.2,
    edge_color='black'
)

plt.axis('off')
plt.legend(*nodes.legend_elements())
plt.show()
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\cluster\_agglomerative.p
y:492: ClusterWarning: scipy.cluster: The symmetric non-negative hollow ob
servation matrix looks suspiciously like an uncondensed distance matrix
  out = hierarchy.linkage(X, method=linkage, metric=affinity)
```



In [130]:

```
ground_truth_labels = [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 2, 1, 0, 0, 0, 0,
                       0, 3, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0,
                       0, 1, 2, 1, 0, 0, 1, 2, 2, 1, 1, 0, 1, 0, 1, 0, 0, 1, 3, 1,
                       0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 1, 0, 1, 4, 0, 1, 0, 2, 0,
                       1, 0, 0, 3, 3, 1, 0, 4, 2, 1, 3, 3, 0, 0, 0, 3, 3]
```

In [131]:

```
print('ground_truth_score =', '{:.4f}'.format(adjusted_rand_score(ground_truth_labels,
labels)))
```

```
ground_truth_score = 0.9733
```

Louvain method

In [132]:

```
def louvain_method(G):
   # Phase 1
    communities = unfolded_communities(G)
    labels = []
    for node in G.nodes:
        for i, c in enumerate(communities):
            if node in c:
                labels.append(i)

    # Phase 2
    nextG = nx.empty_graph(len(communities), nx.MultiGraph)
    for e in G.edges:
        for i in range(len(communities)):
            for j in range(len(communities)):
                if e[0] in communities[i] and e[1] in communities[j]:
                    nextG.add_edge(i, j)

 # Shuffle colors for better visualization
    palette = np.unique(labels)
    key = np.random.permutation(palette)
    labels = key[np.digitize(labels, palette, right=True)]
    return communities, labels, nextG
```

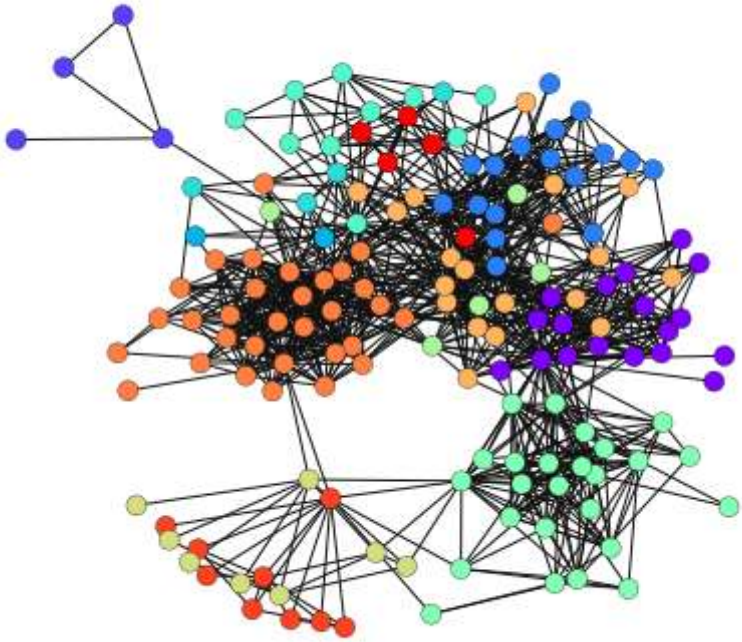
In [133]:

```
def unfolded_communities(G):
    # Proposed template:
    communities = [[n] for n in G.nodes] # initial partition
    prev_max_modularity = -np.inf
    max_modularity = nx.algorithms.community.modularity(G, communities)
    while max_modularity > prev_max_modularity:
        prev_max_modularity = max_modularity
        for node in np.random.permutation(G.nodes):
            for c in communities:
                if node in c:
                    saved_c = c
                    saved_c.remove(node)
                    break
            best_comm = saved_c
            for comm in communities:
                if set(comm).intersection(G.neighbors(node)):
                    comm.append(node)
                    temp_modularity = nx.algorithms.community.modularity(G, communities
)
                    comm.remove(node)
                    if temp_modularity > max_modularity:
                        max_modularity = temp_modularity
                        best_comm = comm
            best_comm.append(node)
    return [c for c in communities if len(c)]
```

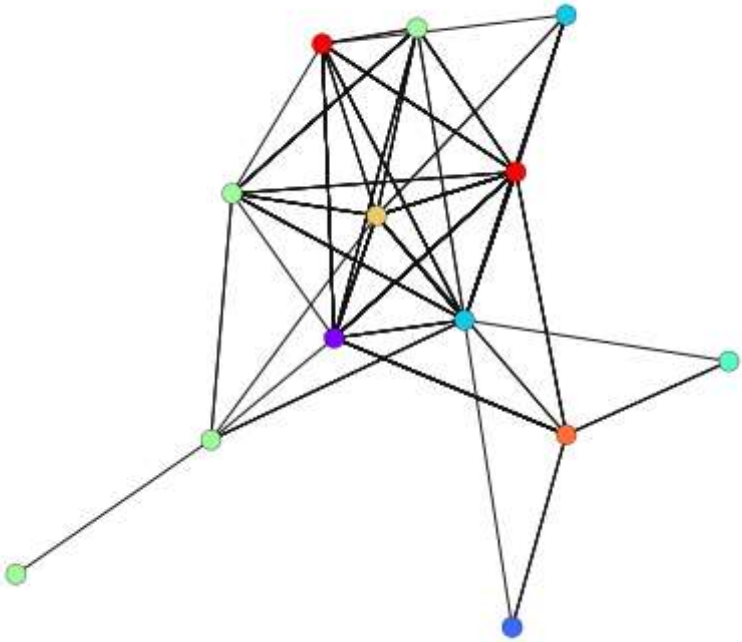
In [134]:

```
n = 4
iterG = g
plt.figure(figsize=(7, 7*3))
for i in range(3):
    plt.subplot(3, 1, i+1)
    communities, labels, nextG = louvain_method(iterG)
    iter_pos = nx.kamada_kawai_layout(iterG)
    nodes = nx.draw_networkx_nodes(
        iterG,
        iter_pos,
        cmap=plt.cm.rainbow,
        node_color=labels,
        edgecolors='black',
        node_size=100)
    nx.draw_networkx_edges(
        iterG,
        iter_pos,
        node_size=100)
    plt.axis('off')
    plt.title(
        '{} nodes, {} communities \nModularity {:.2f}'.format(
            len(iterG), len(communities), nx.community.modularity(iterG, communities)))
    iterG = nextG
```

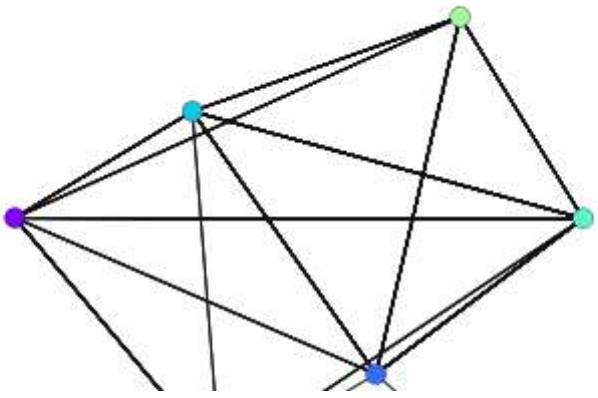
157 nodes, 13 communities
Modularity 0.55

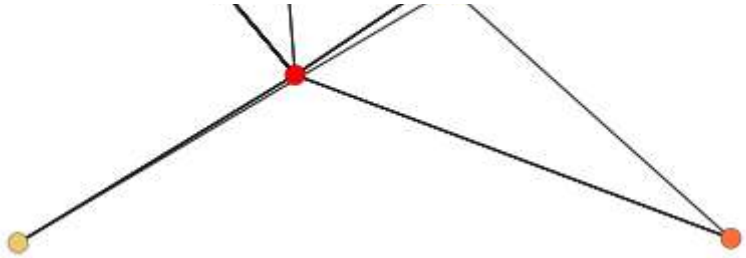


13 nodes, 8 communities
Modularity 0.56



8 nodes, 8 communities
Modularity 0.56





In [ ]: