ДОДАТОК А

Тексти програм

```
import networkx as nx
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
import assort_nc as ass
import numpy as np

GK = nx.karate_club_graph()
n = GK.number_of_nodes()
mapping = dict(zip(GK, "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgh"))
nx.relabel_nodes(GK, mapping, copy=False)
yK = list(ass.nc_part(GK, greedy=False))
ncK = [ass.nc_modularity(GK, comm) for comm in yK]
nc2 = [n * (len(ncK)+1-i) for i,n in enumerate(ncK)]
nc3 = np.array(nc2)
nc10 = [n * np.sqrt((len(ncK)+1-i)) for i,n in enumerate(ncK)]
thr = 1e-3
max_height = max(nc2)
max_index = nc2.index(max_height)
thr2 = 0.1
for i in range(max_index+1, n-1):
    nc2[i] = max_height + thr2
communities = yK

full_comm = list(communities[0])
numbs = {frozenset(node) : 1 for node in full_comm}
idx = {frozenset(node) : i for i,node in enumerate(full_comm)}
prev = set(numbs.keys())
Z = []
for i,comm in enumerate(communities):
    if i!=0:
        setc = {frozenset(c) for c in comm}
        pp = setc.difference(prev).pop()
        child = prev.difference(setc)
        elem1 = child.pop()
        elem2 = child.pop()
        numbs[pp] = numbs[elem1] + numbs[elem2]
        idx[pp] = n+i-1
        Z.append([idx[elem1], idx[elem2], float(nc2[i]),
numbs[pp]])
        prev = setc
last = len(numbs) - 1
Z.append([last-1, last, max_height + thr2, n])
# dendrogram
plt.figure()
labels = [nodes.copy().pop() for nodes in full_comm]
d=dendrogram(Z,labels=labels, color_threshold=max_height+thr,
               above_threshold_color = 'C0', leaf_rotation=0)
plt.savefig("eejet3", dpi=300)
k = [n-i for i in range(n-1)]
kmax = n - max_index
```

```
plt.figure()
plt.plot(k,nc3,'ko', [kmax,kmax],[0,nc3[max_index]],'k-'),
plt.xlabel('k'), plt.ylabel(r'$\mu(G)$'),
plt.gca().invert_xaxis(),
plt.grid(axis='y'), plt.gca().set_xlim(19, 1),
plt.gca().set_ylim(0.4, 0.5)
#plt.savefig("fig4", dpi=300)


import networkx as nx
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
import assort_nc as ass
import numpy as np

GK = nx.read_gml("polbooks.gml")
n = GK.number_of_nodes()
mapping = dict(zip(GK, list(range(n))))
nx.relabel_nodes(GK, mapping, copy=False)

yK = list(ass.nc_part(GK))
ncK = [ass.nc_modularity(GK, comm) for comm in yK]
nc2 = [n * (len(ncK)+1-i) for i,n in enumerate(ncK)]
nc3 = np.array(nc2)
nc10 = [n * np.sqrt((len(ncK)+1-i)) for i,n in enumerate(ncK)]
thr = 1e-3
max_height = max(nc2)
max_index = nc2.index(max_height)
thr2 = 0.2
for i in range(max_index+1, n-1):
    nc2[i] = max_height + thr2
communities = yK

full_comm = list(communities[0])
n = len(full_comm)
numbs = {frozenset(node) : 1 for node in full_comm}
idx = {frozenset(node) : i for i,node in enumerate(full_comm)}
prev = set(numbs.keys())
Z = []
for i,comm in enumerate(communities):
    if i!=0:
        setc = {frozenset(c) for c in comm}
        pp = setc.difference(prev).pop()
        child = prev.difference(setc)
        elem1 = child.pop()
        elem2 = child.pop()
        numbs[pp] = numbs[elem1] + numbs[elem2]
        idx[pp] = n + i - 1
        Z.append([idx[elem1], idx[elem2], float(nc2[i]),
numbs[pp]])
        prev = setc
last = len(numbs) - 1
z2 = list(zip(*Z))
```

```
last1 =
(set(range(last)).difference(set(z2[0]).union(set(z2[1])))).po
p()
Z.append([last1, last, max_height + thr2, n])
# dendrogram
plt.figure()
labels = list(range(n))
d = dendrogram(Z, labels = labels, no_labels = True,
               color_threshold = max_height+thr,
above_threshold_color = 'C0',
               leaf_rotation=0)
#plt.savefig('dendro_fig2.png')
plt.savefig("eejet6", dpi=300)
k = [n-i for i in range(n-1)]
plt.figure()
kmax = n - max_index
plt.plot(k,nc3,'ko', [kmax,kmax],[0,nc3[max_index]],'k-'),
plt.xlabel('k'), plt.ylabel(r'$\mu(G)$'),
plt.gca().invert_xaxis(),
plt.grid(axis='y'), plt.gca().set_xlim(19, 1),
plt.gca().set_ylim(0.4, 0.6)
plt.savefig("fig7", dpi=300)


import pandas as pd
import networkx as nx
from networkx.utils import py_random_state


def nc_modularity(G, communities, weight="weight"):
    r"""Returns the non-correlational modularity of the given
    partition of the graph.

    Parameters
    ----------
    G : NetworkX Graph

    communities : list or iterable of set of nodes
        These node sets must represent a partition of G's
nodes.

    weight : string or None, optional (default="weight")
        The edge attribute that holds the numerical value used
        as a weight. If None or an edge does not have that
attribute,
        then that edge has weight 1.

    Returns
    -------
    Q : float
        The modularity of the partition.
    """
```

```
    if not isinstance(communities, list):
        communities = list(communities)
    nnd = sum(len(comm) for comm in communities)
    m = 2 * G.size(weight="weight")
    sc = list(single_community(G, comm, nnd, weight="weight")
for comm in communities if len(comm) != 0)
    nom = sum(x[0] for x in sc)
    den = sum(x[1] for x in sc)
    mod = nom / den if den != 0 else 0
    return mod / m

def single_community(G, comm, nnd, weight="weight"):
    if len(comm) == 0 :
        return 0, 0
    p = (len(comm) - 1) / (nnd - 1)
    L = sum(wt for u, v, wt in G.edges(comm, data=weight,
default=1) if v in comm)
    SL = sum(wt for u, v, wt in G.edges(comm, data=weight,
default=1))
    s1 = 2 * L * (1 - p)
    s2 = (SL - L) * p
    return s1 - s2, 1

@py_random_state("seed")
def nc_part(G, weight="weight", threshold=0.0000001,
seed=None, greedy=True):
    partition = [{u} for u in G.nodes()]
    yield [s.copy() for s in partition]
    #mod = nc_modularity(G, partition, weight=weight)
    mod = -1
    #partition_new = one_iter(G, partition, seed)
    partition_new = greedy_iter(G, partition) if greedy else
one_iter(G, partition, seed)
    new_mod = nc_modularity(G, partition_new, weight="weight")
    #threshold=-0.1
    while new_mod - mod > threshold:
        yield [s.copy() for s in partition_new]
        partition = partition_new
        mod = new_mod
        #partition_new = one_iter(G, partition, seed)
        partition_new = greedy_iter(G, partition) if greedy
else one_iter(G, partition, seed)
        new_mod = nc_modularity(G, partition_new,
weight="weight")

def one_iter(G, community, seed=None):
    nnd = G.number_of_nodes()
    partition = [frozenset(comm) for comm in community]
    part2com = {u : i for i,u in enumerate(partition)}
    mods = [single_community(G, comm, nnd, weight="weight")
            for comm in partition]
    mod_init = tuple([sum(x[0] for x in mods), sum(x[1] for x
in mods)])
```

```
    nbrs = { part : { p for p in partition
                      for u in part if set(G[u]) & p }
             for part in partition }
    #seed.shuffle(partition)
    nb_moves = 1
    best_gain = mod_init[0] / mod_init[1] if mod_init[1] != 0
else 0
    while nb_moves > 0:
        nb_moves = 0
        for part in partition:
            comm_number = part2com[part]
            comm = community[comm_number]
            best_com = comm_number
            mod_iu = mods[comm_number]
            mod_idifu = single_community(G,
comm.difference(part), nnd, weight="weight")
            for p in nbrs[part]:
                com_nbr = part2com[p]
                if com_nbr==comm_number:
                    continue
                cm = community[com_nbr]
                mod_bf = mods[com_nbr]
                mod_af = single_community(G, cm.union(part),
nnd, weight="weight")
                mod_upd = mod_update(mod_init, mod_iu,
mod_idifu, mod_bf, mod_af)
                gain = mod_upd[0] / mod_upd[1] if mod_upd[1]
!= 0 else 0
                if gain > best_gain:
                    best_gain = gain
                    best_mod = mod_upd
                    best_com = com_nbr
            if best_com != comm_number:
                mod_init = best_mod
                community[comm_number].difference_update(part)
                community[best_com].update(part)
                mods[comm_number] = mod_idifu
                mods[best_com] = single_community(G,
community[best_com], nnd, weight="weight")
                part2com[part] = best_com
                nb_moves += 1
        #nb_moves = 0
    community = list(filter(len, community))
    return community

def mod_update(mod_init, mod_iu, mod_idifu, mod_bf, mod_af):
    return (mod_init[0] - mod_iu[0] + mod_idifu[0] - mod_bf[0]
+ mod_af[0],
            mod_init[1] - mod_iu[1] + mod_idifu[1] - mod_bf[1]
+ mod_af[1])
```

```
def greedy_iter(G, community):
    nnd = G.number_of_nodes()
    partition = [frozenset(comm) for comm in community]
    part2com = {u : i for i,u in enumerate(partition)}
    mods = [single_community(G, comm, nnd, weight="weight")
            for comm in partition]
    mod_init = tuple([sum(x[0] for x in mods), sum(x[1] for x
in mods)])
    nbrs = { part : { p for p in partition
                      for u in part if set(G[u]) & p }
            for part in partition }
    best_gain = -1
    best_part = partition[0]
    best_com = 0
    for part in partition:
        comm_number = part2com[part]
        comm = community[comm_number]
        mod_iu = mods[comm_number]
        mod_idifu = single_community(G, comm.difference(part),
nnd, weight="weight")
        for p in nbrs[part]:
            com_nbr = part2com[p]
            if com_nbr == comm_number :  continue
            cm = community[com_nbr]
            mod_bf = mods[com_nbr]
            mod_af = single_community(G, cm.union(part), nnd,
weight="weight")
            mod_upd = mod_update(mod_init, mod_iu, mod_idifu,
mod_bf, mod_af)
            gain = mod_upd[0] / mod_upd[1] if mod_upd[1] != 0
else 0
            if gain > best_gain:
                best_gain = gain
                #best_mod = mod_upd
                best_com = com_nbr
                best_part = part
    numb = part2com[best_part]
    if best_com != numb :
        community[numb].difference_update(best_part)
        community[best_com].update(best_part)
    community = list(filter(len, community))
    return community
```

ДОДАТОК Б

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	59 с.
	Інші документи	
2.	Презентаційні матеріали	плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Мусієнко М.Ю.			Розробка та дослідження методу виявлення спільнот у мережах на основі модулярності	Шифр групи	Код напр./спец.
Перевірив		Кіріченко Л.О.				ДСм-23-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					