ДОДАТОК А

Тексти програм

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 11 18:23:07 2024

@author: Admin
"""
import time
import networkx as nx
#import matplotlib.pyplot as plt
import numpy as np
from numba import jit


@jit(nopython=True, cache=True)
def EM02(E, theta, n, m, ncom, *args):
    thetanew = np.zeros_like(theta)
    q = np.zeros((m, ncom), dtype=float)
    for ij,(i,j) in enumerate(E):
        d = np.sum(theta[i] * theta[j])
        q[ij] = theta[i] * theta[j] / d
    s = np.sqrt(2*np.sum(q, axis=0))
    for i in range(n):
        idx = np.nonzero(E==i)
        thetanew[i] = np.sum(q[idx[0]], axis=0) / s
    return thetanew

@jit(nopython=True, cache=True)
def EM0(E, theta, n, m, ncom, *args):
    thetanew = np.zeros_like(theta)
    q = np.zeros((m, ncom), dtype=float)
    for ij,(i,j) in enumerate(E):
        d = sum(theta[i,:] * theta[j,:])
        q[ij,:] = theta[i,:] * theta[j,:] / d
    s = np.sqrt(2*np.sum(q, axis=0))
    for i in range(n):
        idx = np.nonzero(E==i)
        thetanew[i,:] = np.sum(q[idx[0],:], axis=0) / s
    return thetanew

@jit(nopython=True, cache=True)
def EM1(E, k, *args):
    knew = np.zeros_like(k)
    d = np.sum(k, axis=0)
    for ij,(i,j) in enumerate(E):
        q = k[i,:] * k[j,:] / d[:]
        D = np.sum(q)
        q /= D
        knew[i,:] += q
        knew[j,:] += q
    return knew
```

```
@jit(nopython=True, cache=True)
def EM2(E, k, n, m, ncom, tol, mask, *args):
    knew = np.zeros_like(k)
    d = np.sum(k, axis=0)
    s = np.ones(n)
    for ij,(i,j) in enumerate(E):
        if mask[ij]:
            q = k[i,:] * k[j,:] / d[:]
            D = np.sum(q)
            q /= D
            knew[i,:] += q
            knew[j,:] += q
            s[i] = 0
            s[j] = 0
    knew = np.where(knew >= tol, knew, 0)
    ind = np.nonzero(s)
    knew[ind] = k[ind]
    for ij,(i,j) in enumerate(E):
        if mask[ij]:
            s = knew[i,:] + knew[j,:]
            if np.count_nonzero(s) == 1:
                mask[ij] = 0
    return knew
#
#
def LogLike0(E, ncom, theta):
    LL = sum( np.log(sum(theta[i,:]*theta[j,:])) for i,j in E)
    LL -= np.sum(np.dot(theta, theta.transpose()))
    return LL

def LogLike1(E, ncom, k):
    d = np.sum(k, axis=0)
    LL = sum( np.log(sum(k[i,:] * k[j,:] / d[:])) for i,j in
E)
    LL -= np.sum( np.dot(k/d, k.transpose()))
    return LL

def Fin0(n, ncom, theta, tol):
    d = np.sum(theta, axis=0)
    k = theta * d
    k = np.where(k >= tol, k, 0)
    s = np.sum(k, axis=1)
    return (k.transpose()/s).transpose()

def Fin1(n, ncom, k, tol):
    k = np.where(k >= tol, k, 0)
    s = np.sum(k, axis=1)
    return (k.transpose()/s).transpose()

def Fin2(n, ncom, k, tol):
    s = np.sum(k, axis=1)
    return (k.transpose()/s).transpose()
```

```
def test_EM(G, ncom, method, func_LL, fin, samples=100,
tol=1e-4):
    n = G.number_of_nodes()
    rng = np.random.default_rng(1234)
    L_best = -np.inf
    iter_total = 0
    tm = -time.time()
    for run in range(samples):
        param_old = rng.random([n, ncom], dtype=float)
        E = np.array(G.edges())
        m = G.number_of_edges()
        mask = np.ones(m)
        iterations = 0
        eps = 1
        while eps > tol:
            param_new = method(E, param_old, n, m, ncom, tol,
mask)
            iterations += 1
            eps = np.max(np.abs(param_new - param_old))
            param_old = param_new
        iter_total += iterations
        LL = func_LL(E, ncom, param_new)
        if LL > L_best:
            L_best = LL
            param_best = param_new
            iter_best = iterations
    prob = fin(n, ncom, param_best, tol)
    tm += time.time()
    return {'prob': prob, 'LL':LL, 'iter_best':iter_best,
'iter_total':iter_total, 'time':tm}
#
#
A = np.array([[0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0],
[1,  0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]])
#
samples = 100
tol = 0.001
#
G = nx.from_numpy_array(A)
test_EM(G, 2, EM0, LogLike0, Fin0, 1, tol)
test_EM(G, 2, EM1, LogLike1, Fin1, 1, tol)
```

```
test_EM(G, 2, EM2, LogLike1, Fin2, 1, tol)
#
#
rez = []
K = 3
G = nx.from_numpy_array(A)
rez.append(test_EM(G, K, EM0, LogLike0, Fin0, samples, tol))
rez.append(test_EM(G, K, EM1, LogLike1, Fin1, samples, tol))
rez.append(test_EM(G, K, EM2, LogLike1, Fin2, samples, tol))
timeA = [meth['time'] for meth in rez]
LLA = [meth['LL'] for meth in rez]

kar = []
K = 2
G = nx.karate_club_graph()
kar.append(test_EM(G, K, EM0, LogLike0, Fin0, samples, tol))
kar.append(test_EM(G, K, EM1, LogLike1, Fin1, samples, tol))
kar.append(test_EM(G, K, EM2, LogLike1, Fin2, samples, tol))
#
timeK = [meth['time'] for meth in kar]
LLK = [meth['LL'] for meth in kar]

G = nx.read_gml("polblogs.gml")
mapping = dict(zip(G, range(G.number_of_nodes())))
G = nx.relabel_nodes(G, mapping, copy=False)
G = nx.Graph(G)
G.remove_edges_from(nx.selfloop_edges(G))
degs = np.array(G.degree())
idx = np.nonzero(degs[:,1]==0)
G.remove_nodes_from(idx[0])
mapping = dict(zip(G, range(G.number_of_nodes())))
G = nx.relabel_nodes(G, mapping, copy=False)
pol = []
K = 2
pol.append(test_EM(G, K, EM0, LogLike0, Fin0, samples, tol))
pol.append(test_EM(G, K, EM1, LogLike1, Fin1, samples, tol))
pol.append(test_EM(G, K, EM2, LogLike1, Fin2, samples, tol))
timeP = [meth['time'] for meth in pol]
LLP = [meth['LL'] for meth in pol]
#
#
les = []
K = 6
G = nx.les_miserables_graph()
mapping = dict(zip(G, range(G.number_of_nodes())))
nx.relabel_nodes(G, mapping, copy=False)
les.append(test_EM(G, K, EM0, LogLike0, Fin0, samples, tol))
les.append(test_EM(G, K, EM1, LogLike1, Fin1, samples, tol))
les.append(test_EM(G, K, EM2, LogLike1, Fin2, samples, tol))
#
timeL = [meth['time'] for meth in les]
LLL = [meth['LL'] for meth in les]
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	67 с.
	Інші документи	
2.	Презентаційні матеріали	16 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Крамаренко Д.П.			Розробка та дослідження методу кластерізації мереж на спільноти, які перекриваються
Перевірив		Шергін В.Л.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./спец.
ДСзм-23-1	122
ХНУРЕ кафедра ШІ	