Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ РОБОТОТЕХНИКА И КОМПЛЕКСНАЯ АВТОМАТИЗАЦИЯ (РК)

КАФЕДРА РК6 «СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ»

Отчет по лабораторной работе

Спектральный метод балансировки загрузки МВС

Студент ________________ **Абидоков Р. Ш.**
РК6-21М

Преподаватель ________________ **Карпенко А. П.**

2022 г.

Постановка задачи балансировки нагрузки

Пусть (Q, D) – ациклический граф вычислительного процесса, соответствующего некоторой прикладной задаче. Здесь $Q = \{Q_1, Q_2 \dots Q_n\}$ – вершины графа, отождествляемые с соответствующими вычислительными процессами, $D = \{D_{i,j}, i, j \in [1, n], j \neq i\}$ – ребра графа, отождествляемые с информационными связями между процессами.

Положим, что имеется МВС с универсальными процессорами, каждый из которых может выполнить любой из процессов $Q_1, Q_2 \dots Q_n$. Пусть (P, C) – граф данной МВС. Здесь $P = \{P_1, P_2 \dots P_N\}$ – вершины графа, соответствующие процессорам; $C = \{C_{i,j}, i, j \in [1, n], j \neq i\}$ – ребра графа, отождествляемые с коммуникационной сетью.

Задачей оптимального отображения совокупности процессов $Q_1, Q_2 \dots Q_n$ на процессоры $P_1, P_2 \dots P_N$ называется задача оптимального отображения графа (P, C) на граф (Q, D), т.е. задача поиска такого распределения процессов $Q = \{Q_1, Q_2 \dots Q_n\}$ по процессорам $P = \{P_1, P_2 \dots P_N\}$, которое минимизирует некоторый критерий оптимальности (обычно время вычислений).

Одним из распространенных методов приближенного решения задачи оптимального отображения является метод балансировки загрузки. Основная идея метода балансировки загрузки состоит в распределении процессов по процессорам таким образом, чтобы суммарная вычислительная и коммуникационная загрузки процессоров были примерно одинаковы. При этом не учитываются коммуникационные загрузки процессоров, обусловленные транзитными обменами, конфликты при обменах вследствие перегрузки коммуникационной сети, а также времена на организацию обменов.

Вводится в рассмотрение *отображающая матрица*

$$X = \{x_{i,j}, i \in [1, n], j \in [1, N]\},$$

где $x_{i,j} = 1$ – процесс Q_i назначен на на выполнение процессору P_j, $x_{i,j} = 0$ – процесс Q_i не назначен на на выполнение процессору P_j.

Назовем вычислительной загрузкой процессора $P_j, j \in [1:N]$ величину

$$WL_j(X) = p_j \sum_{l=1}^{n} x_{l,j} q_l,$$

где p_j – производительность процессора P_j, q_l – вычислительная сложность процесса Q_l, а его коммуникационной загрузкой – величину

$$CL_j(X) = \sum_{k=1}^{N} \sum_{l=1}^{n} \sum_{m=1}^{n} x_{l,k} x_{m,j} (t_c)_{i,j} d_{m,l},$$

где $(t_c)_{i,j}$ – время, необходимое для передачи байта данных от процессора P_i процессору P_j (без учета времени организации передачи данных). Тогда задачу балансировки загрузки можно записать в виде

$$\mathrm{E}(X) = max_{j \in [1,N]} E_j = max_{j \in [1,N]} \left(WL_j(X) + CL_j(X)\right).$$

Постановка задачи бисекции графа

Поставим задачу разделения графа (Q, D) на два подграфа таким образом, чтобы

- суммарные вычислительные сложности подграфов были равны (т.е. были равны количества процессов в каждом из подграфов) (1)
- количество разрезанных ребер было минимально. (2)

Поставленная задача является двухкритериальной задачей. Поэтому бисекция графа, найденная с помощью данного алгоритма, вообще говоря, не является оптимальной ни по одному из критериев (1), (2). По общему свойству многокритериальных задач, найденная бисекция представляет собой некоторый компромисс между этими критериями оптимальности.

Схема спектрального алгоритма бисекции графа

Определим А $-$ *матрицу смежности* $(n * n)$ графа (Q, D) такую, что $a_{i,j} = 1$, если вершины i и j связаны между собой ребром; $B -$ диагональную *матрицу степеней вершин* $(n * n)$ графа (Q, D) такую, что b_i, равна числу ребер, инцидентных i-й вершине; кроме того, введем в рассмотрение *матрицу Лапласа:*

$$L = B - A, \tag{3}$$

где $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n -$ упорядоченные по возрастанию действительные собственные числа матрицы Лапласа.

Тогда искомая бисекция графа (Q, D) находится следующим образом:

- находим среднее значение $\bar{u}$ компонентов $u_{2,i}$ вектора U_2 - нормализованного собственного вектора, соответствующего второму по величине собственному значению λ_2;
- относим вершины графа(Q, D), соответствующие значениям $u_{2,i} < \bar{u}$ к первому подграфу, а остальные вершины – ко второму подграфу;
- если несколько величин $u_{2,i}$ имеют значение $\bar{u}$, распределяем соответствующие вершины между подграфами равномерно.

Экспериментальная часть

На Рис. 1 приведен вид исходного графа (Q, D).

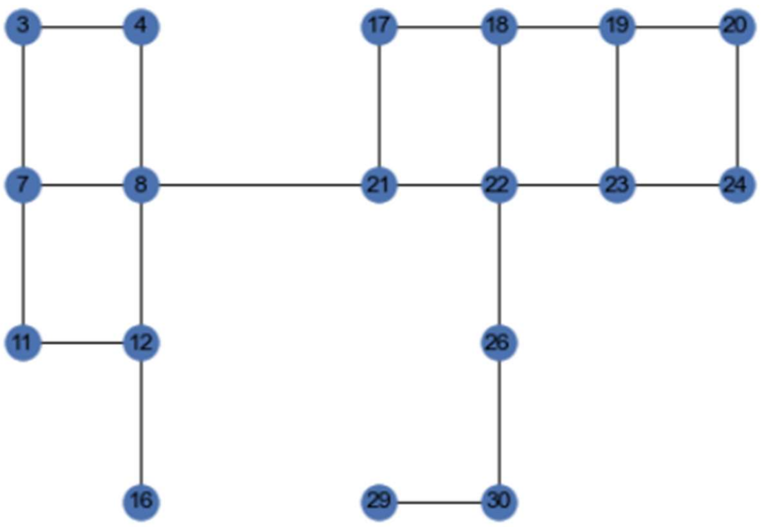

Рис. 1 Исходный вид графа (Q, D)

Составленные матрицы смежности A и степеней вершин B приведены соответственно на Рис. 2, 3.

```
[[0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 1 0 1 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 0 0 1 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0]
 [0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1]
 [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 1 1 0]]
```

Рис. 2 Вид матрицы смежности A

```
[[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 3, 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, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 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, 3, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 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, 2, 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, 2]]
```

Рис. 3 Вид матрицы степеней вершин B

Вычисленная по формуле (3) матрица Лапласа приведена на Рис. 4.

```
[[ 2 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [-1  2  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [-1  0  3 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 -1 -1  4  0 -1  0  0  0  0  0 -1  0  0  0  0  0  0]
 [ 0  0 -1  0  2 -1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 -1 -1  3 -1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  2 -1  0  0 -1  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 -1  3 -1  0  0 -1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 -1  3 -1  0  0 -1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 -1  2  0  0  0 -1  0  0  0]
 [ 0  0  0 -1  0  0  0 -1  0  0  0  3 -1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 -1  0  0 -1  4 -1  0 -1  0  0]
 [ 0  0  0  0  0  0  0  0  0 -1  0  0 -1  3 -1  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 -1  0  0 -1  2  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  2  0 -1]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 -1]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  2]]
```

Рис. 3 Вид матрицы Лапласа L

В приведенных выше матрицах элементы ставятся в соответствие вершинам графа так, как это приведено в Табл. 1.

Табл. 1. Соответствие индексов элементов матриц узлам графа

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Q	3	4	7	8	11	12	16	17	18	19	20	21	22	23	24	26	29	30

С помощью пакета numpy.linalg языка программирования Python были вычислены собственные значения матрицы Лапласа:

(0.0000, 0.1082, 0.2292, 0.6084, 0.7700, 1.3769, 1.3820, 2.0729, 2.2448, 2.3845, 2.7915, 3.1108, 3.5315, 3.6180, 3.7504, 4.3754, 5.6040, 6.0415)

и соответствующие им собственные вектора (из-за громоздкости приведены только отвечающие первым пяти наименьшим собственным числам):

$U_1 =$(0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357, 0.2357)

$U_2 =$(-0.2866, -0.265, -0.2772, -0.2147, -0.3003, -0.2909, -0.3262, 0.0685, 0.1322, 0.175 , 0.1969, -0.0025, 0.1388, 0.1768, 0.1976, 0.2338, 0.3402, 0.3034)

$U_3 =$(-0.0573, -0.0474, -0.0541, -0.0266, -0.0661, -0.0629, -0.0816, 0.1332, 0.1717, 0.2712, 0.3380, 0.0642, 0.0712, 0.2418, 0.3274, -0.2089, -0.5724, -0.4412)

$U_4 =$(0.3974, 0.3678, 0.1852, 0.1145, -0.0689, -0.2811, -0.7179, 0.1210, 0.0521, -0.0392, -0.1036, 0.1164, 0.0428, -0.0424, 0.1049, 0.019, -0.0417, -0.0163)

$U_5 =$(0.2174, 0.1497, 0.1177, -0.0333, 0.0784, -0.0213, -0.0926, -0.5383, -0.3083, 0.0679, 0.359 , -0.3537, -0.2172, 0.1006, .3737, -0.1401, 0.1953, 0.0449)

Среднее значение компонент вектора $U_2 = 0$.

Тогда вершинам, принадлежащим первому классу, соответствуют компо ненты $u_{i,2} < 0$, а вершинам, соответствующим второму классу, $u_{i,2} \geq 0$ Итоговое разбиение показано на Рис. 4.

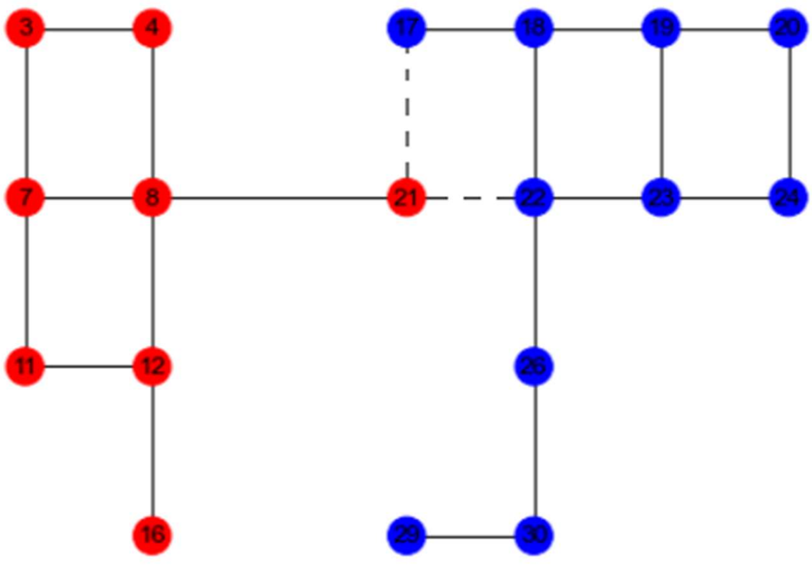

Рис. 4 Итоговое разбиение графа (Q, D) на два класса

Видно, что получившийся результат не является оптимальным ни с точки зрения количества разрезанных ребер (оптимальным решением здесь было бы рассечения по ребрам 8-21 или 22-26), ни с точки зрения равенства количества вершин в двух классах – однако является некоторым компромиссом между двумя этими требованиями.

Ответы на контрольные вопросы

1. Каким требованиям должен удовлетворять граф (Q, D) *для того, чтобы его бисекцию можно было выполнить спектральным алгоритмом?*

Вычислительные сложности всех процессов $Q_i, i \in [1:n]$ одинаковы, количество вершин n четно.

2. Что такое матрица Лапласа графа?

Матрица, равная разности матрицы степеней вершин и матрицы смежности.

3. Перечислите основные этапы спектрального алгоритма бисекции графа.

Нахождение собственных чисел и векторов матрицы Лапласа; Их упорядочивание в соответствии с величиной собственных чисел; Нахождение среднего значения компонент собственного вектора, соответствующего второму собственную числу; Отнесение к одному классу вершин, соответствующих компонентам, меньшим среднего значения, и к другому классу вершин, соответствующих компонентам, большим среднего значения.

4. Покажите правильность бисекции графа, полученной в результате выполнения работы. Если полученная бисекция не является оптимальной по одному или обоим критериям оптимальности, объяснить этот результат.

Задача бисекции графа является задачей многокритериальной оптимизации. Как следствие, полученное решение в общем случае не будет являться оптимальным по каждому из критериев в отдельности, но будет некоторым компромиссным решением между ними, что мы и наблюдаем – количества вершин в подкласах не равны, но близки друг к другу; количество рассеченных ребер не минимально, но достаточно мало.

Исходный код программы

Программа выполнена на языке Python 3.8 с использованием библиотек numpy, networkx, scipy, matplotlib, seaborn

```
import numpy as np
import networkx as nx
from scipy import linalg
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid', {'axes.facecolor': '.9'})
sns.set_palette(palette='deep')
sns_c = sns.color_palette(palette='deep')
np.set_printoptions(precision=4)
%matplotlib inline

def get_default_adjacency_matrix():
    res = np.zeros((32, 32))
    # Горизонтальные ребра внутри квадратов
    for i in range(3):
        for j in range(8):
            res[i+4*j, i+4*j+1] = 1
    # Вертикальные ребра внутри квадратов
    for i in range(4):
        for j in range(3):
            res[i+4*j, i+4*j+4] = 1
        for j in range(4, 7):
            res[i+4*j, i+4*j+4] = 1
    # Горизонтальные соединительные ребра
    for i in range(4):
        res[4*(i+1)-1, 16 + 4*i] = 1
    return res


def form_graph(adj_matr, stop_list_nodes=[], stop_list_edges=[]):
    G = nx.Graph()
    sqr_side = int(np.sqrt(adj_matr.shape[0] / 2))
    for i in range(1, sqr_side+1):
        for j in range(0, sqr_side):
            if (i + sqr_side*j) not in stop_list_nodes:
                G.add_node(i + sqr_side*j, pos=(i, sqr_side - j))
            if (sqr_side**2 + i + sqr_side*j) not in stop_list_nodes:
                G.add_node(sqr_side**2 + i + sqr_side*j, pos=(sqr_side + 1 + i, 
sqr_side - j))
    for i in range(adj_matr.shape[0]):
        for j in range(i, adj_matr.shape[0]):
            if (i+1 not in stop_list_nodes) and (j+1 not in stop_list_nodes) and 
([i+1, j+1] not in stop_list_edges):
                if (adj_matr[i, j] == 1):
                    G.add_edge(i+1, j+1)
    H = nx.Graph()
    H.add_nodes_from(sorted(G.nodes(data=True)))
    H.add_edges_from(G.edges(data=True))
    return H

def print_graph(G, color_map=None):
    pos = nx.get_node_attributes(G, 'pos')
    nx.draw(G, pos, with_labels=True, node_color=color_map)

removed_nodes = [1, 2, 5, 6, 9, 10, 13, 14, 15, 25, 27, 28, 31, 32]
#removed_nodes = []
removed_edges = [[4, 17], [12, 25], [16, 29]]
```

```
#removed_edges = []
G = form_graph(get_default_adjacency_matrix(), removed_nodes, removed_edges)
print_graph(G)

# Матрица смежности
A = nx.adjacency_matrix(G).todense()
node_idxs = np.array(G.nodes())
print('', node_idxs)
print(A)
# Матрица степеней вершин
B = np.zeros(A.shape, dtype=np.int32)
diags = np.sum(A, axis=0).tolist()[0]
np.fill_diagonal(B, diags)
print('     ', node_idxs)
print(B)
# Матрица Лапласа (ака матрица Кирхгофа)
L = B - A
print(node_idxs)
print(L)

def bisection(L):
    eigvals, eigvecs = np.linalg.eig(L)
    sec_eigval, sec_eigvec = sorted(zip(eigvals, eigvecs.transpose()),
key=lambda x: x[0])[1]
    first_cluster = np.sign(sec_eigvec) > 0
    second_cluster = np.sign(sec_eigvec) <= 0
    return first_cluster, second_cluster

first_cluster_idxs, second_cluster_idxs = bisection(L)
first_cluster, second_cluster = node_idxs[first_cluster_idxs.tolist()],
node_idxs[second_cluster_idxs.tolist()]

print(first_cluster_idxs, second_cluster_idxs)
print(first_cluster, second_cluster)

# Закрашиваем
color_map = []
for node in G:
    if node in first_cluster:
        color_map.append('blue')
    else:
        color_map.append('red')
print_graph(G, color_map)
```