Министерство образования и науки 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 Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент	Никатов Владислав Алексеевич
Группа	РК6-84б
Тип задания	Лабораторная работа
Тема лабораторной работы	Решение задачи компоновки
Вариант	7

Студент _____ **Никатов В.А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Берчун Ю.В.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2020 г.

Оглавление

1 Задание

Решить задачу компоновки. Компоненты можно группировать в контейнеры из списка определённых размеров (для примера - 3, 4, 5, 6 и 7). При этом сперва нужно решить комбинаторную задачу. Для каждого из возможных вариантов набора контейнеров уже решаем задачу компоновки. В отчёте подробно, по шагам, разбираем решение для наилучшего варианта набора контейнеров.

Таблица 1.1. Матрица смежности для графа из условия.

0	1	0	3	2	0	0	4	1	0	0	3	0	0	0	0	1	5	0	0	0	0	2	1	3	5	5	2	0	2
1	0	0	1	3	0	0	3	0	1	0	2	1	0	3	0	0	3	0	0	2	3	0	3	0	0	1	0	0	0
0	0	0	3	2	2	0	0	4	0	0	0	1	2	1	2	2	0	4	1	0	0	3	1	0	0	0	0	0	0
3	1	3	0	4	0	0	0	0	4	1	2	0	0	0	4	0	4	1	4	2	0	0	0	0	0	2	2	2	4
2	3	2	4	0	4	3	0	3	0	0	0	0	0	2	1	2	0	2	0	0	0	3	2	3	0	3	0	0	0
0	0	2	0	4	0	0	1	0	0	1	3	1	3	0	0	0	1	0	0	0	3	0	0	0	0	2	2	4	1
0	0	0	0	3	0	0	0	0	3	0	1	3	3	2	4	0	0	3	2	0	0	1	1	1	1	0	2	2	0
4	3	0	0	0	1	0	0	0	4	0	2	2	0	0	4	3	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	4	0	3	0	0	0	0	0	2	1	0	0	3	0	0	1	0	2	2	0	0	1	2	3	2	0	0	3
0	1	0	4	0	0	3	4	0	0	0	0	0	2	0	1	0	0	4	0	0	0	4	0	0	0	0	0	2	0
0	0	0	1	0	1	0	0	2	0	0	0	0	0	0	0	2	0	1	0	1	1	0	2	1	3	1	2	0	0
3	2	0	2	0	3	1	2	1	0	0	0	4	0	0	0	3	0	4	0	3	3	0	3	0	0	0	3	0	4
0	1	1	0	0	1	3	2	0	0	0	4	0	0	0	3	0	0	0	2	3	0	2	3	0	0	0	0	4	0
0	0	2	0	0	3	3	0	0	2	0	0	0	0	4	3	1	3	0	0	1	1	0	0	4	0	0	2	3	4
0	3	1	0	2	0	2	0	3	0	0	0	0	4	0	0	1	0	0	0	0	0	0	0	0	0	4	0	0	0
0	0	2	4	1	0	4	4	0	1	0	0	3	3	0	0	1	0	0	2	2	0	0	3	1	0	0	0	2	0
1	0	2	0	2	0	0	3	0	0	2	3	0	1	1	1	0	0	1	0	0	0	1	0	3	1	2	1	0	1
5	3	0	4	0	1	0	0	1	0	0	0	0	3	0	0	0	0	2	0	0	3	0	2	2	0	4	3	0	0
0	0	4	1	2	0	3	0	0	4	1	4	0	0	0	0	1	2	0	0	4	1	1	0	2	0	0	0	0	0
0	0	1	4	0	0	2	0	2	0	0	0	2	0	0	2	0	0	0	0	0	0	0	0	4	1	0	1	0	0
0	2	0	2	0	0	0	0	2	0	1	3	3	1	0	2	0	0	4	0	0	4	0	0	2	0	4	0	0	0
0	3	0	0	0	3	0	0	0	0	1	3	0	1	0	0	0	3	1	0	4	0	0	3	3	4	4	1	0	3
2	0	3	0	3	0	1	0	0	4	0	0	2	0	0	0	1	0	1	0	0	0	0	0	0	0	4	2	4	0
1	3	1	0	2	0	1	0	1	0	2	3	3	0	0	3	0	2	0	0	0	3	0	0	0	0	0	0	2	0
3	0	0	0	3	0	1	0	2	0	1	0	0	4	0	1	3	2	2	4	2	3	0	0	0	1	0	0	2	4
5	0	0	0	0	0	1	0	3	0	3	0	0	0	0	0	1	0	0	1	0	4	0	0	1	0	0	4	1	0
5	1	0	2	3	2	0	0	2	0	1	0	0	0	4	0	2	4	0	0	4	4	4	0	0	0	0	0	1	2
2	0	0	2	0	2	2	0	0	0	2	3	0	2	0	0	1	3	0	1	0	1	2	0	0	4	0	0	0	0
0	0	0	2	0	4	2	1	0	2	0	0	4	3	0	2	0	0	0	0	0	0	4	2	2	1	1	0	0	0
2	0	0	4	0	1	0	1	3	0	0	4	0	4	0	0	1	0	0	0	0	3	0	0	4	0	2	0	0	0

2 Выбор наилучшего решения

В рамках задачи компоновки необходимо разделить граф $G(V, R)$ на подграфы $\{G_i(V_i, R_i)\}_{i=1}^{n}$. Критерием оптимальности в данной задаче является число внешних связей. Его можно записать в виде формулы $Q = \frac{1}{2}\sum_{\substack{i,j=1 \\ i \neq j}}^{n} \left|R_{ij}\right|$.

В ходе решения задачи были получены все возможные сочетания из множества {3, 4, 5, 6, 7}. Для каждого полученного сочетания была решена задача компоновки. Наилучшим сочетанием оказался набор контейнеров {3, 6, 7, 7, 7}. Значением критерия оптимальности Q для такого набора было равно 214.

3 Описание решения для наилучшего набора контейнеров

3.1 Последовательный алгоритм компоновки

Сначала необходимо получить опорное решение с помощью последовательного алгоритма, чтобы потом его улучшить с помощью итерационного.

Первым шагом необходимо вычислить суммарное количество связей для каждой из вершин графа. Эти значения представлены в таблице 3.1.2.

Таблица 3.1.2а. Суммарное количество связей для первой половины вершин.

v_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s_i	20	31	19	31	18	24	32	33	26	18	30	13	32	33	20

Таблица 3.1.2б. Суммарное количество связей для второй половины вершин.

v_i	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
s_i	26	25	23	33	29	14	27	23	20	26	34	25	23	28	24

Выбираем вершину с наименьшим числом внешних связей (v_{11}) и определяем её в первое подмножество V_0. Если для нескольких вершин s_i одинаково, то следует выбирать вершину, которая имеет связи с минимальным числом других вершин. Если и таких вершин несколько – любую из них.

Рассмотрим вершины $\{v_5, v_6, v_{16}, v_{20}, v_{22}, v_{24}\}$, имеющие общие связи с вершинами из подмножества V_0 – вершиной v_{11}. Приращение связей при включении

вершины в подмножество V_0 является коэффициентом внешней связанности вершины и считается по формуле

$$\delta_i = 2 \cdot \mathrm{r}_i - s_i \,,$$

где r_i – количество связей с вершинами из уже сформированного подмножества V_0. А сам δ_i называют коэффициентом внешней связности вершины. Его значение равно разности между числом внешних (с вершинами, не входящими в соответствующее подмножество) связей вершины и внутренних (с вершинами подмножества, в котором находится рассматриваемая вершина). связей вершины.

Значения δ_i для вершин-кандидатов $\{v_5, v_6, v_{16}, v_{20}, v_{22}, v_{24}\}$ представлены в таблице 3.1.3.

Таблица 3.1.3. Приращение связей для вершин-кандидатов.

v_i	5	6	16	20	22	24
δ_i	-16	-28	-23	-12	-21	-18

Из них необходимо выбрать ту, при включении которой в подмножество V_0 будет минимально приращение количество связей. Включаем в подмножество вершину v_{20}, таким образом $V_0 = \{v_{11}, v_{20}\}$.

К вершинам-кандидатам на добавление, имеющим общие связи с вершинами из подмножества V_0, добавились вершины $\{v_{12}, v_{13}, v_{15}, v_{16}, v_{19}, v_{22}\}$, имеющие общую связь с добавленной на предыдущем этапе вершиной v_{20}. Приращения связей для вершин-кандидатов представлены в таблице 3.1.4.

Таблица 3.1.4. Приращение связей для вершин-кандидатов.

v_i	5	6	12	13	15	16	18	19	22	24
δ_i	-16	-24	-28	-25	-24	-23	-27	-27	-21	-18

После добавления вершины v_5 в подмножество V_0 оно становится сформированным, потому как его размер изначально полагался равным 3, таким образом $V_0 = \{v_5, v_{11}, v_{20}\}$.

Исключаем уже скомпонованные вершины из дальнейшего рассмотрения.

Находим вершину с наименьшим количеством внешних связей среди оставшихся вершин – v_4. Добавляем данную вершину в новое подмножество $V_1 = \{v_4\}$.

Перечень вершин, имеющих общие связи с уже размещенной вершиной v_4: $\{v_3, v_6, v_{15}, v_{17}, v_{19}, v_{22}, v_{25}\}$. Значение δ_i для этих вершин представлено в таблице 3.1.5.

Таблица 3.1.5. Приращение связей для вершин-кандидатов.

v_i	3	6	15	17	19	22	25
δ_i	-25	-30	-22	-15	-25	-21	-28

Добавляем вершину с наименьшим по модулю приращением v_{17}, таким образом $V_1 = \{v_4, v_{17}\}$.

Повторяем процедуру до тех пор, пока не будет достигнут необходимый размер подмножества V_1. Вершины в будут добавлены в следующем порядке: v_{15}, v_{27}, v_3, v_8. Таким образом $V_1 = \{v_3, v_4, v_8, v_{15}, v_{17}, v_{27}\}$.

Исключаем скомпонованные вершины из рассмотрения и повторяем процедуру. Результатом алгоритма является следующие опорное решение:

- $V_0 = \{v_5, v_{11}, v_{20}\}$;
- $V_1 = \{v_3, v_4, v_8, v_{15}, v_{17}, v_{27}\}$;
- $V_2 = \{v_0, v_2, v_{10}, v_{12}, v_{14}, v_{16}, v_{26}\}$;
- $V_3 = \{v_1, v_9, v_{21}, v_{22}, v_{24}, v_{28}, v_{29}\}$;
- $V_4 = \{v_6, v_7, v_{13}, v_{18}, v_{19}, v_{20}, v_{23}\}$.

Количество внешних связей для такого разбиения $Q = 219$.

3.2 Итерационный алгоритм компоновки

Для улучшения разбиения необходимо совершить перемещения вершин из одного подмножества в другое так, чтобы значение целевой функции (количество внешних связей) уменьшилось. Для того, чтобы эффект от перемещения для каждой рассматриваемой пары вершин, для них вычисляется значение:

$$\Delta R_{ij} = \delta_i + \delta_j - 2 \cdot R_{ij} .$$

Если $\Delta R_{ij} > 0$, то после перестановки вершин количество связей уменьшится, а значит, такую перестановку следует выполнить. Основной задачей на каждом этапе

является получение матрицы перестановок, строки которой соответствуют одному из подмножеств, а столбцы – второму. Элементами матрицы являются значения ΔR_{ij}.

Матрица перестановок подмножества V_0 с подмножествами V_1, V_2, V_3 и V_4 представлена в таблице 3.2.1. Находим в матрице перестановок максимальный элемент и, если он больше нуля, выполняем перестановку соответствующих ему вершин, после чего пересчитываем матрицу перестановок и повторяем операцию. Перестановка производится ровно до тех пор, пока в матрице не останется положительных элементов.

Таблица 3.2.1. Матрица перестановок. Подмножество V_0, итерация 1.

	3	4	8	15	17	27	0	2	10	12	14	16	26	1	9	21	22	24	28	29	6	7	13	18	19	23	25
5	-11	-7	-3	-3	-12	-7	-7	-10	-16	-19	-13	-11	-11	-15	-7	-11	-6	-10	-12	-12	-10	-15	-9	-14	-13	-14	-9
11	-14	-12	-12	-10	-17	-16	-10	-13	-15	-22	-16	-14	-14	-16	-8	-12	-9	-15	-13	-7	-17	-18	-12	-15	-12	-17	-10
20	-9	-7	-7	-7	-12	-11	-5	-8	-10	-21	-11	-7	-9	-17	-9	-13	-8	-8	-14	-8	-5	-6	-8	-9	-2	-5	2

Меняем элементы 20 и 25 местами.

Таблица 3.2.2. Матрица перестановок. Подмножество V_0, итерация 2.

	3	4	8	15	17	27	0	2	10	12	14	16	26	1	9	21	22	24	28	29	6	7	13	18	19	23	20
5	-9	-5	-2	-3	-11	-8	-8	-11	-17	-18	-14	-12	-12	-12	-8	-9	-7	-11	-11	-13	-16	-17	-19	-14	-17	-8	-11
11	-10	-8	-9	-8	-14	-15	-9	-12	-14	-19	-15	-13	-13	-11	-7	-8	-8	-14	-10	-6	-19	-16	-18	-11	-12	-7	-12
25	-1	1	2	3	-3	0	-3	-6	-8	-21	-9	-5	-7	-12	0	-7	1	1	-7	1	-7	-8	-10	-11	-4	-7	-2

Меняем элементы 25 и 15 местами.

Таблица 3.2.3. Матрица перестановок. Подмножество V_0, итерация 3.

	3	4	8	25	17	27	0	2	10	12	14	16	26	1	9	21	22	24	28	29	6	7	13	18	19	23	20
5	-15	-7	-6	-6	-9	-4	-6	-11	-17	-22	-14	-12	-8	-16	-8	-11	-7	-11	-13	-12	-13	-17	-19	-16	-15	-12	-10
11	-16	-10	-13	-11	-12	-11	-7	-12	-14	-23	-15	-13	-9	-15	-7	-10	-8	-14	-12	-5	-16	-16	-18	-13	-10	-11	-11
15	-4	-2	-1	-3	-6	-3	-3	-4	-6	-15	-7	-3	-9	-15	-7	-12	-6	-6	-12	-7	-10	-8	-10	-9	-6	-3	-3

Положительных элементов среди всех ΔR_{ij} больше нет.

Исключаем скомпонованное подмножество $V_0 = \{v_5, v_{11}, v_{15}\}$ из рассмотрения и переходим к подмножеству V_1. Матрица перестановок подмножества V_1 с подмножествами V_2, V_3 и V_4 представлена в таблице 3.2.4.

Таблица 3.2.4. Матрица перестановок. Подмножество V_1, итерация 1.

	0	2	10	12	14	16	26	1	9	21	22	24	28	29	6	7	13	18	19	23	20
3	-18	-17	-27	-26	-24	-18	-22	-21	-19	-22	-17	-19	-20	-17	-23	-24	-30	-20	-17	-14	-19
4	-16	-15	-19	-24	-22	-16	-20	-19	-17	-16	-13	-17	-18	-15	-23	-18	-20	-18	-19	-12	-17
8	-9	-16	-12	-17	-15	-13	-13	-16	-10	-9	-10	-14	-11	-8	-21	-20	-18	-16	-13	-12	-15
25	-12	-11	-15	-28	-18	-12	-16	-19	-9	-14	-3	-9	-14	-7	-16	-13	-15	-21	-10	-15	-12
17	-17	-16	-22	-27	-23	-17	-21	-21	-19	-18	-13	-21	-22	-17	-26	-23	-25	-23	-22	-17	-22
27	-12	-11	-15	-22	-18	-16	-16	-12	-10	-9	-4	-14	-13	-16	-21	-18	-20	-18	-19	-12	-17

Видно, что среди всех пар элементов из подмножества V_1 с элементами из других подмножеств положительных ΔR_{ij} не оказалось, а значит перестановки делать не нужно, а подмножество $V_1 = \{v_3, v_4, v_8, v_{17}, v_{25}, v_{27}\}$ можно считать скомпонованным. Исключаем вершины подмножества V_1 из рассмотрения и переходим к следующему этапу – подмножеству V_2. Матрица перестановок подмножества V_2 с подмножествами V_3 и V_4 представлена в таблице 3.2.5.

Таблица 3.2.5. Матрица перестановок. Подмножество V_2, итерация 1.

	1	9	21	22	24	28	29	6	7	13	18	19	23	20
0	-13	-13	-15	-7	-17	-15	-13	-15	-8	-10	-5	-2	-2	-2
2	-14	-14	-16	-8	-18	-16	-18	-16	-13	-17	-16	-17	-13	-13
10	-18	-10	-18	-10	-14	-12	-10	-23	-20	-20	-25	-20	-20	-20
12	-26	-26	-28	-22	-30	-28	-26	-26	-29	-23	-26	-23	-23	-27
14	-21	-15	-17	-9	-21	-17	-15	-19	-16	-16	-19	-22	-18	-16
16	-15	-13	-13	-5	-15	-17	-13	-18	-15	-21	-18	-15	-15	-15
26	-11	-11	-13	-11	-17	-17	-11	-16	-15	-13	-22	-13	-15	-13

Положительных элементов в матрице перестановок нет, а значит подмножество $V_2 = \{v_0, v_2, v_{10}, v_{12}, v_{14}, v_{16}, v_{26}\}$ мы так же можем считать скомпонованным. Исключаем его из дальнейшего рассмотрения и остается два подмножества: V_3 и V_4. Матрица перестановок этих подмножеств представлена в таблице 3.2.6.

Таблица 3.2.5. Матрица перестановок. Подмножество V_3, итерация 1.

	6	7	13	18	19	23	20
1	-28	-24	-30	-24	-27	-23	-26
9	-13	-11	-13	-15	-14	-6	-9
21	-18	-18	-22	-14	-17	-19	-16
22	-19	-15	-21	-17	-18	-14	-17
24	-25	-21	-27	-21	-24	-20	-23
28	-24	-14	-24	-18	-17	-13	-16
29	-19	-21	-21	-15	-18	-14	-17

Положительный элементов в матрице перестановок так же не оказалось. Это значит, что подмножество $V_3 = \{v_1, v_9, v_{21}, v_{22}, v_{24}, v_{28}, v_{29}\}$ можно считать скомпонованным. И остается одно подмножество $V_4 = \{v_6, v_7, v_{13}, v_{18}, v_{19}, v_{20}, v_{23}\}$, которое так же скомпоновано.

Результатом итерационного алгоритма компоновки является решение

- $V_0 = \{v_5, v_{11}, v_{20}\}$;
- $V_1 = \{v_3, v_4, v_8, v_{15}, v_{17}, v_{27}\}$;
- $V_2 = \{v_0, v_2, v_{10}, v_{12}, v_{14}, v_{16}, v_{26}\}$;
- $V_3 = \{v_1, v_9, v_{21}, v_{22}, v_{24}, v_{28}, v_{29}\}$;
- $V_4 = \{v_6, v_7, v_{13}, v_{18}, v_{19}, v_{20}, v_{23}\}$.

Количество внешних связей для такого разбиения $Q = 214$.

Малое количество итераций алгоритма было достигнуто хорошей точностью опорного решения, полученного с использованием алгоритма последовательной компоновки.

Код программы на языке Python3.7, реализующий оба алгоритма компоновки для решения поставленной задачи, представлен в приложении 1.

Приложение 1.

main.py

```
from iterative_solver import optimal
from iterative_solver import iterative_algorithm_v2
from sequential_solver import sequential_algorithm
from helper import get_combs
from data import matrix


def main():
    combs = get_combs(min_cont_size=2,
                      max_cont_size=7,
                      element_number=len(matrix))
    group_list_list = []
    new_group_list_list = []
    opt_list = []
    for size_list in combs:
        print(size_list)
        group_list = sequential_algorithm(matrix, size_list)
        group_list_list.append(group_list)

        new_group_list = iterative_algorithm_v2(matrix, group_list)
        new_group_list_list.append(new_group_list)

        opt_list.append(optimal(matrix, new_group_list))

    ind = [i for i, opt in enumerate(opt_list) if opt == min(opt_list)][0]
    print('size_list:', combs[ind])
    print('group_list:', group_list_list[ind])
    print('new_group_list:', new_group_list_list[ind])
    print('opt:', opt_list[ind])


def result():
    size_list = (3, 6, 7, 7, 7)

    old_group_list = sequential_algorithm(matrix, size_list, info=False)
    old_opt = optimal(matrix, old_group_list)

    for i in range(len(old_group_list)):
        old_group_list[i] = sorted(old_group_list[i])

    new_group_list = iterative_algorithm_v2(matrix, old_group_list, info=True)
```

```
    new_opt = optimal(matrix, new_group_list)

    print('old_group_list:', old_group_list)
    print('new_group_list:', new_group_list)
    print('old_opt:', old_opt)
    print('new_Q:', new_opt)


if __name__ == '__main__':
    # main()
    result()
```

sequential_solver.py

```
from print import print_r_matrix


def sequential_algorithm(matrix, size_list, info=False):
    group_list = [[] for _ in size_list]
    use_list = []
    for i, (group, size) in enumerate(zip(group_list, size_list)):
        if info:
            print('Группа', i)
        for _ in range(size):
            element = next_element(group, matrix, use_list, info)
            group.append(element)
            if info:
                print('Добавлен вершина', element)
        if info:
            print('В результате руппа', i, ':', group)
            print()
    return group_list


def first_element(matrix, use_list, info=False):
    # создание словарей нераспределенных вершин:
    # 1. количество связей с ненулевыми неиспользуемыми вершинами
    # 2. сумма     связей с ненулевыми неиспользуемыми вершинами
    count_dict = dict()
    sum_dict = dict()
    # строки матрицы, соответствующие неиспользуемым вершинам
    for elem, line in [(i, line) for i, line in enumerate(matrix) if i not in use_list]:
        # количество ненулевых связей с вершинами, отсутствующими в списке
```

```
    use_list
        count_dict.update([(elem, len([v for el, v in enumerate(line) if v != 0 and el not in use_list]))])
        # сумма связей с вершинами, отсутствующими в списке use_list
        sum_dict.update([(elem, sum([v for el, v in enumerate(line) if el not in use_list]))])
    if info:
        print('Сумма связей для нераспределенных вершин:')
        print_r_matrix([list(sum_dict.values())], [list(sum_dict.keys())])
    # вершины с минимальной суммой
    elem_list = [i for i, v in sum_dict.items() if v == min(sum_dict.values())]
    # оставляем только вершины с минимальной суммой
    for el in [elem for elem in range(len(matrix)) if elem not in elem_list and elem not in use_list]:
        count_dict.pop(el)
    if info:
        print('Вершины с минимальной суммой внешних связей:', list(count_dict.keys()))
    # вершина с минимальным количеством связей
    value = [el for el in elem_list if count_dict[el] == min(count_dict.values())][0]
    if info and len(count_dict) > 1:
        print('Количество внешних связей для них:')
        print_r_matrix([list(count_dict.values())], [list(count_dict.keys())])
        print('Вершина с минимальным количеством внешних связей:', value)
    use_list.append(value)
    return value


def next_element(group, matrix, use_list, info=False):
    adjacent_group = get_adjacent_group(group, matrix, use_list)
    if info:
        print("Перечень неразмещенных вершин, имеющих общие связи с размещенными:", adjacent_group)
    if len(adjacent_group) == 0:
        return first_element(matrix, use_list, info)

    p_list = list(map(sum, matrix))
    delta = dict([(i, 0) for i in adjacent_group])
    for el1 in adjacent_group:
        for el2 in group:
            delta[el1] += matrix[el2][el1] * 2
        delta[el1] -= p_list[el1]
    if info:
        print("Значение дельта для выбранных вершин:")
        print_r_matrix([delta.values()], [delta.keys()])
    value, delta = [(i, v) for i, v in delta.items() if v == max(delta.values())][0]
```

```
    use_list.append(value)
    return value


def get_adjacent_group(group, matrix, use_list):
    adjacent_group = set()
    for el1 in group:
        adjacent_group.update([i for i, el2 in enumerate(matrix[el1]) if el2 != 0 and i not
in use_list])
    return list(adjacent_group)
```

iterative_solver.py

```
import numpy as np
import copy
from print import print_p_matrix
from helper import concat


def iterative_algorithm(matrix, all_group_list, info=False):
    group_list = copy.deepcopy(all_group_list)
    for i in range(len(group_list)):
        for j in range(i, len(group_list)):
            group1 = group_list[i]
            group2 = group_list[j]
            alpha1 = [get_a(matrix, el, group1, group2) for el in group1]
            alpha2 = [get_a(matrix, el, group1, group2) for el in group2]
            p_matrix = get_b(matrix, alpha1, alpha2, group1, group2)
            ind1, ind2 = np.unravel_index(p_matrix.argmax(), p_matrix.shape)
            it = 0
            while p_matrix[ind1][ind2] > 0:
                if info:
                    print('Подмножества:', i, 'и', j, ', итерация:', it)
                    print_p_matrix(p_matrix, group1, group2)
                    print('Меняем элементы', group1[ind1], 'и', group2[ind2], 'местами.')
                    print('group_list', group_list)
                    print()
                group1[ind1], group2[ind2] = group2[ind2], group1[ind1]
                alpha1 = [get_a(matrix, el, group1, group2) for el in group1]
                alpha2 = [get_a(matrix, el, group1, group2) for el in group2]
                p_matrix = get_b(matrix, alpha1, alpha2, group1, group2)
                ind1, ind2 = np.unravel_index(p_matrix.argmax(), p_matrix.shape)
                it += 1
            if info:
```

```
            print('Подмножества:', i, 'и', j, ', итерация:', it)
            print('Положительных элементов следи всех deltaR больше нет.')
            print()
    return group_list


def iterative_algorithm_v2(matrix, all_group_list, info=False):
    all_group_list = copy.deepcopy(all_group_list)
    for i in range(len(all_group_list) - 1):
        group_list = all_group_list[i:]
        p_matrix = get_b_matrix_v2(matrix, group_list)
        el1, el2 = find_elements_to_swap(p_matrix, group_list)
        iter = 1
        while el1 is not None:
            if info:
                print('Подмножество:', i, ', итерация:', iter)
                print_p_matrix(p_matrix, group_list[0], concat(group_list[1:]))
                print('Меняем элементы', el1, 'и', el2, 'местами.')
                print()
            group_list = swap_group(group_list, el1, el2)
            p_matrix = get_b_matrix_v2(matrix, group_list)
            el1, el2 = find_elements_to_swap(p_matrix, group_list)
            iter += 1
        if info:
            print('Подмножество:', i, ', итерация:', iter)
            print_p_matrix(p_matrix, group_list[0], concat(group_list[1:]))
            print('Положительных элементов следи всех deltaR больше нет.')
            print()
    return all_group_list


def get_a(matrix, v_index, group1, group2):
    alpha = 0
    for el in group1:
        alpha -= matrix[v_index][el]
    for el in group2:
        alpha += matrix[v_index][el]
    if v_index in group2:
        alpha *= -1
    return alpha


def get_b(matrix, alpha1, alpha2, group1, group2):
    b = []
    for i in range(0, len(alpha1)):
```

```
        tmp = []
        for j in range(0, len(alpha2)):
            b_value = alpha1[i] + alpha2[j] - 2 * matrix[group1[i]][group2[j]]
            tmp.append(b_value)
        b.append(tmp)
    return np.array(b)


def get_b_v2(elem1, elem2, group_list, matrix):
    val = 0
    for el in group_list[0]:
        val -= matrix[elem1][el]
        val += matrix[elem2][el]
    for group in group_list[1:]:
        if elem2 in group:
            for el in group:
                val += matrix[elem1][el]
                val -= matrix[elem2][el]
    val -= 2 * matrix[elem1][elem2]
    return val


def get_b_matrix_v2(r_matrix, group_list):
    p_matrix = []
    for i, elem in enumerate(group_list[0]):
        p_matrix.append([])
        for j, other_elem in enumerate(concat(group_list[1:])):
            p_matrix[i].append(get_b_v2(elem, other_elem, group_list, r_matrix))
    return np.array(p_matrix)


def find_elements_to_swap(p_matrix, group_list):
    i, j = np.unravel_index(p_matrix.argmax(), p_matrix.shape)
    if p_matrix[i][j] <= 0:
        return None, None
    return group_list[0][i], concat(group_list[1:])[j]


def g_ind(elem, group_list):
    for i, group in enumerate(group_list):
        for j, el in enumerate(group):
            if elem == el:
                return i, j
```

```
def swap_group(group_list, elem1, elem2):
    i1, j1 = g_ind(elem1, group_list)
    i2, j2 = g_ind(elem2, group_list)
    group_list[i1][j1], group_list[i2][j2] = group_list[i2][j2], group_list[i1][j1]
    return group_list


def optimal(r_matrix, group_list):
    result = 0
    for i, group1 in enumerate(group_list):
        for group2 in group_list[i + 1:]:
            for el1 in group1:
                for el2 in group2:
                    result += r_matrix[el1][el2]
    return result
```

helper.py

```
from itertools import combinations_with_replacement


def concat(ll):
    return [el for lst in ll for el in lst]


def get_combs(min_cont_size,
              max_cont_size,
              element_number):
    min_comb_size = (element_number - 1) // max_cont_size + 1
    max_comb_size = (element_number - 1) // min_cont_size + 1
    combs = []
    for comb_size in range(min_comb_size, max_comb_size + 1):
        combs += list(combinations_with_replacement(range(min_cont_size, max_cont_size
+ 1), comb_size))
    return [cont for cont in combs if sum(cont) == element_number]
```

print.py

```
from helper import concat


def print_r_matrix(r_matrix, group_list=None):
    if group_list is None:
        group_list = list([range(len(r_matrix[0]))])
    elem_list = concat(group_list)
    # шапка
    print(end=' R ')
    for el in elem_list:
        if el < 10:
            print(end=' ')
        print(el, end=' ')
    print()
    # столбец + матрица
    for i, line in enumerate(r_matrix):
        # столбец
        if elem_list[i] < 10:
            print(end=' ')
        print(elem_list[i], end='')
        # строка матрицы
        for el in line:
            if el <= -10:
                print(el, end='')
            elif el < 0 or el >= 10:
                print('', el, end='')
            elif el < 10:
                print(' ', el, end='')
        print()

def print_p_matrix(p_matrix, group1, group2):
    # шапка
    print(end='  ')
    for el in group2:
        if el <= -10:
            print('', el, end='')
        elif el < 0 or el >= 10:
            print(' ', el, end='')
        elif el < 10:
            print('  ', el, end='')
    print()
    # столбец + матрица
    for i, line in enumerate(p_matrix):
        # столбец
```

```
    if group1[i] < 10:
        print(end=' ')
    print(group1[i], end='')
    # строка матрицы
    for el in line:
        if el <= -10:
            print('', el, end='')
        elif el < 0 or el >= 10:
            print(' ', el, end='')
        elif el < 10:
            print('  ', el, end='')
    print()
```

data.py

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