|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 3](#_Toc41018233)

[2 Выбор наилучшего решения 4](#_Toc41018234)

[3 Описание решения для наилучшего набора контейнеров 4](#_Toc41018235)

[3.1 Последовательный алгоритм компоновки 4](#_Toc41018236)

[3.2 Итерационный алгоритм компоновки 6](#_Toc41018237)

[Приложение 1. 10](#_Toc41018238)

# 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 на подграфы . Критерием оптимальности в данной задаче является число внешних связей. Его можно записать в виде формулы

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 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|  | 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.

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

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

Рассмотрим вершины {}, имеющие общие связи с вершинами из подмножества – вершиной . Приращение связей при включении вершины в подмножество является коэффициентом внешней связанности вершины и считается по формуле

где – количество связей с вершинами из уже сформированного подмножества .  
А сам 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 для вершин-кандидатов {} представлены в таблице 3.1.3.

Таблица 3.1.3. Приращение связей для вершин-кандидатов.

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

Из них необходимо выбрать ту, при включении которой в подмножество будет минимально приращение количество связей. Включаем в подмножество вершину , таким образом {.

К вершинам-кандидатам на добавление, имеющим общие связи с вершинами из подмножества , добавились вершины {}, имеющие общую связь с добавленной на предыдущем этапе вершиной . Приращения связей для вершин-кандидатов представлены в таблице 3.1.4.

Таблица 3.1.4. Приращение связей для вершин-кандидатов.

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

После добавления вершины в подмножество оно становится сформированным, потому как его размер изначально полагался равным 3, таким образом {.

Исключаем уже скомпонованные вершины из дальнейшего рассмотрения.

Находим вершину с наименьшим количеством внешних связей среди оставшихся вершин – . Добавляем данную вершину в новое подмножество .

Перечень вершин, имеющих общие связи с уже размещенной вершиной : {}. Значение для этих вершин представлено в таблице 3.1.5.

Таблица 3.1.5. Приращение связей для вершин-кандидатов.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3 | 6 | 15 | 17 | 19 | 22 | 25 |
|  | -25 | -30 | -22 | -15 | -25 | -21 | -28 |

Добавляем вершину с наименьшим по модулю приращением , таким образом .

Повторяем процедуру до тех пор, пока не будет достигнут необходимый размер подмножества . Вершины в будут добавлены в следующем порядке: .  
Таким образом .

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

* {;
* ;
* ;
* ;
* .

Количество внешних связей для такого разбиения

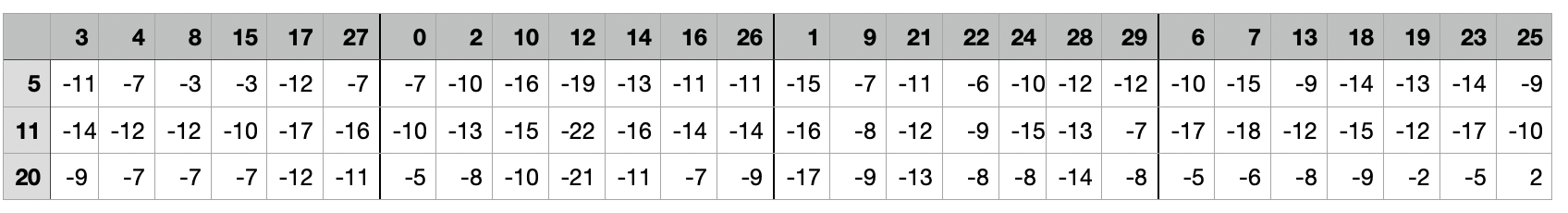
## 3.2 Итерационный алгоритм компоновки

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:

Если , 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 .

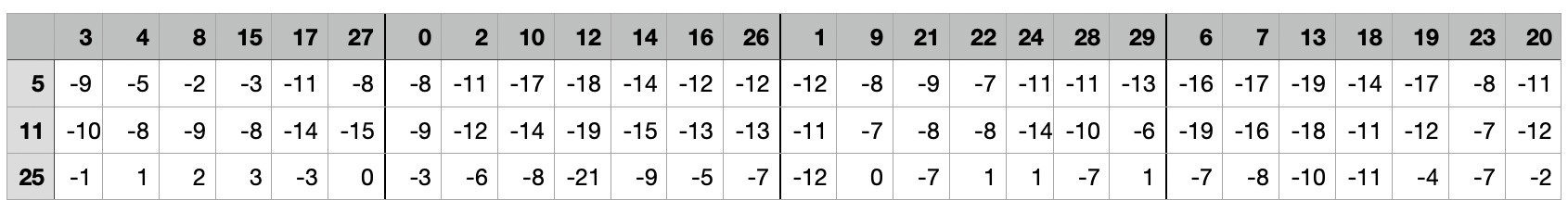
Матрица перестановок подмножества с подмножествами , и представлена в таблице 3.2.1. Находим в матрице перестановок максимальный элемент и, если он больше нуля, выполняем перестановку соответствующих ему вершин, после чего пересчитываем матрицу перестановок и повторяем операцию. Перестановка производится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

Таблица 3.2.1. Матрица перестановок. Подмножество , итерация 1.



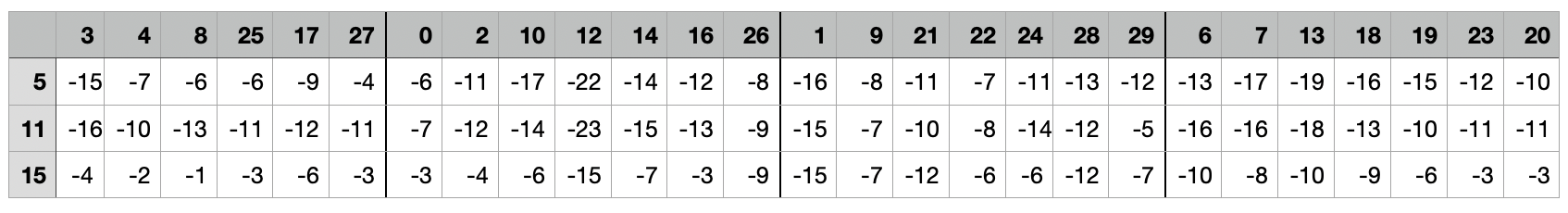
Меняем элементы 20 и 25 местами.

Таблица 3.2.2. Матрица перестановок. Подмножество , итерация 2.



Меняем элементы 25 и 15 местами.

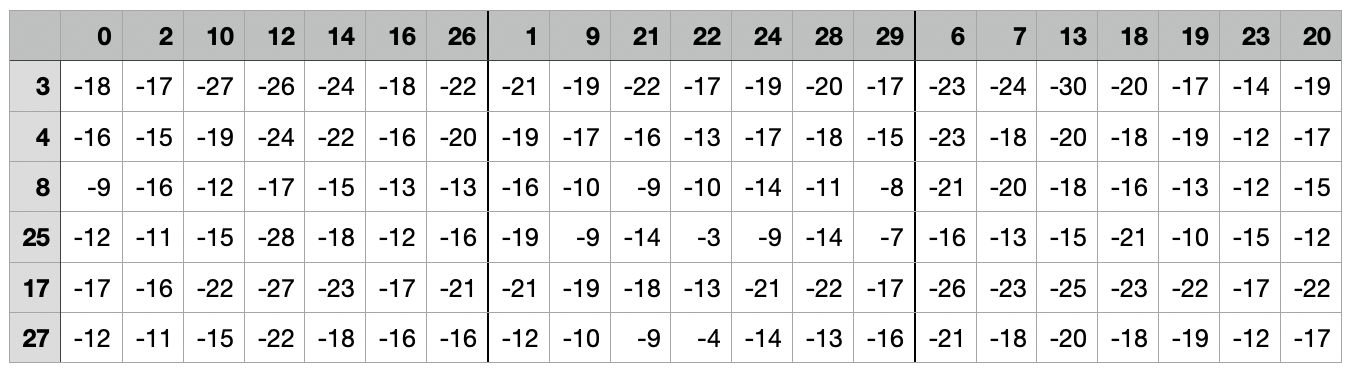
Таблица 3.2.3. Матрица перестановок. Подмножество , итерация 3.



Положительных элементов среди всех больше нет.

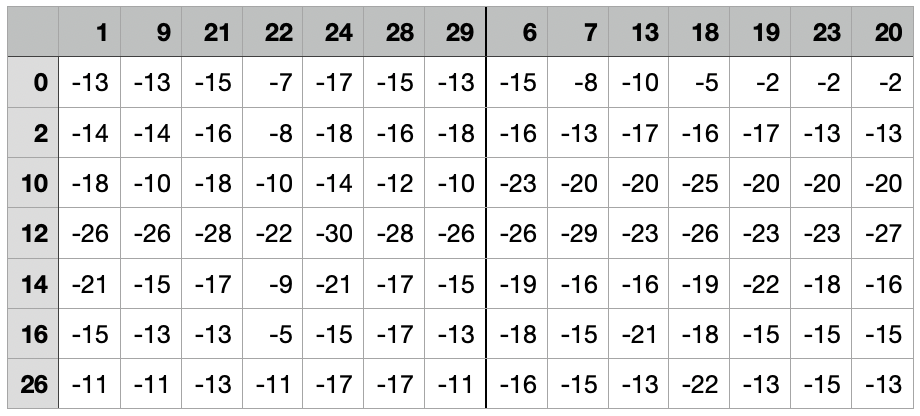
Исключаем скомпонованное подмножество { из рассмотрения и переходим к подмножеству . Матрица перестановок подмножества с подмножествами , и представлена в таблице 3.2.4.

Таблица 3.2.4. Матрица перестановок. Подмножество , итерация 1.



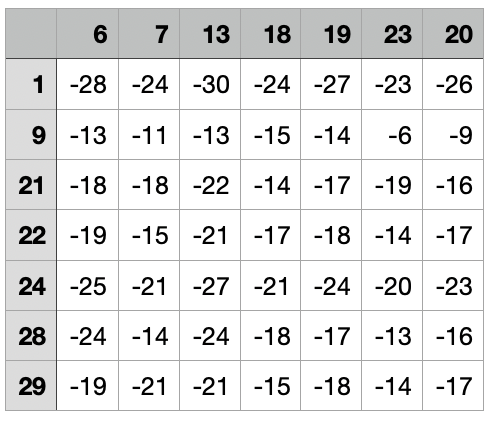
Видно, что среди всех пар элементов из подмножества с элементами из других подмножеств положительных не оказалось, а значит перестановки делать не нужно, а подмножество можно считать скомпонованным. Исключаем вершины подмножества из рассмотрения и переходим к следующему этапу – подмножеству Матрица перестановок подмножества с подмножествами и представлена в таблице 3.2.5.

Таблица 3.2.5. Матрица перестановок. Подмножество , итерация 1.



Положительных элементов в матрице перестановок нет, а значит подмножество мы так же можем считать скомпонованным. Исключаем его из дальнейшего рассмотрения и остается два подмножества: и . Матрица перестановок этих подмножеств представлена в таблице 3.2.6.

Таблица 3.2.5. Матрица перестановок. Подмножество , итерация 1.



Положительный элементов в матрице перестановок так же не оказалось. Это значит, что подмножество можно считать скомпонованным. И остается одно подмножество , которое так же скомпоновано.

Результатом итерационного алгоритма компоновки является решение

* {;
* ;
* ;
* ;
* .

Количество внешних связей для такого разбиения .

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, 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, 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, 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]]