МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**АДАПТИВНЫЙ АЛГОРИТМ НАХОЖДЕНИЯ МАКСИМАЛЬНОГО ПАРОСОЧЕТАНИЯ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д. И. Кожухарь

(подпись)

Направление 02.03.02 — «Фундаментальные информатика и информационные технологии»

Направленность (профиль) «Вычислительные технологии»

Научный руководитель,

канд. техн. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.Е. Полупанова

(подпись)

Нормоконтролер,

канд. техн. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.Е. Полупанова

(подпись)

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc532555212)

[1 Постановка задачи нахождения максимального паросочетания 4](#_Toc532555213)

[2 Описание алгоритма 5](#_Toc532555214)

[2.1 Теоретическая основа 5](#_Toc532555215)

[2.2 Адаптивные механизмы формирования η – областей 8](#_Toc532555216)

[3 Программная реализация алгоритма 11](#_Toc532555217)

[3.1 Основные сведения о программе 11](#_Toc532555218)

[3.2 Особенности реализации 11](#_Toc532555219)

[4 Тестирование алгоритма 13](#_Toc532555220)

[Заключение 18](#_Toc532555221)

[Список использованных источников 19](#_Toc532555222)

[Приложение А. Файл основной программы main.py 20](#_Toc532555223)

[Приложение Б. Файл алгоритма matching.py 22](#_Toc532555224)

# **Введение**

Среди набора комбинаторно-логических задач на графах важное место занимает проблема определения паросочетаний. Эвристические алгоритмы решения данных задач применяются при проектировании инженерных сетей, коммуникаций, построения систем поддержки принятия решений в неопределенных условиях, проектировании СБИС и т.п. Задачи такого типа относятся к переборным задачам с экспоненциальной временной сложностью.

 Процедура нахождения максимального паросочетания в графе входит в состав большого числа алгоритмов, решающих различные задачи, в частности в транспортной задаче Хичкока, изоморфизме поддеревьев и проблеме выпускного [1]. Часто эта процедура используется в итерационных структурах. Это предъявляет повышенные требования к качеству и особенно времени решения задачи нахождения максимального паросочетания. Существующее в настоящее время большее количество алгоритмов нахождения максимального паросочетания обеспечивают приемлемые результаты при решении задач малой и средней сложности. Возникшие потребности в решении задач большой и очень большой размерности является побудительным мотивом исследований и разработок новых эффективных алгоритмов.

В ходе данной работы будет изложена методика представления решения на базе матрицы смежности графа, адаптивные механизмы видоизменения матрицы смежности для решения задачи нахождения максимального паросочетания в графе.

Таким образом, в рамках курсовой работы будут выполнены:

* рассмотрение принципов работы адаптивного алгоритма;
* анализ параметров, от которых зависит работа алгоритма;
* реализация данного алгоритма на языке Python.

# **Постановка задачи нахождения максимального паросочетания**

Одной из широко востребованных задач целочисленного программирования является задача о паросочетании максимальной мощности, рассматриваемой в комбинаторном направлении теории графов. Паросочетанием графа *G=(X,U)* называет подмножество таких рёбер *U’∈U*, что любые два ребра *uk,ul∈U’* не имеют общих вершин, т.е. не смежны. Паросочетание максимальной мощности (рисункок 1) определяется как паросочетание, состоящее из максимального числа рёбер.



Рисунок 1 - Пример максимального паросочетания в неориентированном графе

Обычно задача нахождения паросочетания рассматривается в контексте двудольных графов, однако она применима и для произвольных неориентированных графов.

# **Описание алгоритма**

## **Теоретическая основа**

Рассмотрим задачу нахождения максимального паросочетания на примере. Пусть дан граф G = (X,U) (рисунок 2).U = {ui| i = 1, 2, …, 9} – множество ребер графа G. Паросочетание такого графа определяется как множество рёбер, не имеющих общих вершин.

Рисунок 2 – Граф G

Построим граф Gd=(U,V)- двойственный для графа G(рисунок 3)*.* Вершины графаGd - соответствуют рёбрам графа G. Пара вершин (ui, uj) в графе Gd связаны ребром в том и только в том случае, если в графе G пара рёбер (ui, uj) смежны, т.е. инциденты одной вершине.

Рисунок 3 – Граф Gd

Множество X0 вершин графа G=(X,U) называется внутренне устойчивым, если любые две вершины xi∈X0 и xj∈X0 не являются смежными.

Максимальное число вершин во внутренне устойчивом множестве графа G называется числом внутренней устойчивости и обозначается как α(G). Иногда число внутренней устойчивости называют также числом независимости графа.

Подмножество вершин P = {u1, u4, u7, u9} множества вершин двойственного графа Gd является внутренне устойчивым, т.к. любые две вершины подмножества P не смежны.

Таким образом, паросочетанию графа G соответствует внутренне устойчивое подмножество двойственного графаGd.

Максимальному по мощности паросочетанию графа G соответствует предельное внутренне устойчивое подмножество (содержащее наибольшее число вершин) двойственного графа Gd.

Построим для двойственного графа Gd матрицу смежности R (рисунок 4).

**

Рисунок 4 – Исходная матрица смежностиR графа Gd

Переставим все столбцы и строки помеченные элементами некоторого внутренне-устойчивого подмножества P таким образом, чтобы они располагались рядом друг с другом начиная с левой (верхней) стороны матрицы. Для нашего примера модифицированная матрица R имеет вид, представленный на рисунке 5.

**

Рисунок 5 – Модифицированная матрица смежности R графа Gd

Анализ состояния матрицы смежности показывает что если столбцы матрицы с номерами от l до l+m помечены элементами, образующими внутренне-устойчивое подмножество, то симметрично относительно главной диагонали матрицы R на пересечении столбцов и строк матрицы с номерами от l до (l+m-1) формируется область Pi квадратной формы размером m \* m, элементы которой имеют нулевое значение. Назовем такую область η-областью [2].

В модифицированной матрице из нашего примера η-область – это область, образованная при пересечении 1-4 столбца с 1-4 строкой(рисунок 5). Столбцы и строки помечены элементами 1,4,7,9. Pi=Pi(l,m)**,** где l – номер первого столбца (и первой строки), с которого начинается η-область Pi,m - число столбцов и строк, на пересечении которых образована η-область Pi. Для нашего примера P1=P1(1,4).

Таким образом, если в результате некоторой перестановки строк и столбцов матрицы смежности образуется η - область Pi(l, m), то это значит, что элементы, которыми помечены столбцы и строки с номерами от l до (l + m - 1), образуют внутренне устойчивое подмножество Ui. И если операция производилась на матрице смежности графа Gd, двойственном к G, то подмножеству Ui соответствует паросочетание в G.

Отсюда схема нахождения в графе G паросочетания максимальной мощности выглядит так:

1. строится граф Gd, двойственный к графу G;
2. строиться матрица смежности R графа Gd;
3. путём перестановок строк и столбцов матрицы смежности R графа Gd формируется 1 Pi(l, m) с максимально возможным значением параметра m;
4. множество элементов Ui, которыми в матрице смежности R помечены столбцы с номерами от l до (l + m - 1), будут составлять максимальное внутренне устойчивое подмножество двойственного графа Gd;
5. декодируем Ui, т.е. сопоставим каждой вершине из множества Ui ребро из исходного графа, тем самым получим максимальное паросочетание графа G.

Таким образом в основе процедур построения максимального паросочетания лежит процедура формирования η -областей в матрице смежности двойственного графа.

## **Адаптивные механизмы формирования η – областей**

Формирование η - областей в матрице R осуществляется в процессе её эволюционной модификации. Эволюционная модификация матрицы R производится путём выборочных групповых перестановок соседних столбцов и строк, что обеспечивает направленное последовательное перемещение элементов матрицы R с нулевым значениями и объединение их в η-области.

Адаптивный процесс состоит из повторяющихся шагов, каждый из которых представляет собой переход от одного решения (состояния матрицы R) к другому – лучшему [3].

На каждом шаге анализируются пары (i, i + 1) соседних строк матрицы смежности. Анализ осуществляется в два такта. На первом такте анализируются все пары (i, i+1), у которых первый элемент i - нечетное число. На втором такте анализируются пары, у которых первый элемент i - четное число.

Например: пусть размер матрицы смежности n = 9, а строки и столбцы помечены номерами от 1 до n, тогда на первом такте рассматриваются пары строк (1, 2), (3, 4), (5, 6), (7, 8). На втором такте - (2, 3),(4,5),(6,7),(8,9).

Пары строк анализируются независимо друг от друга. По результатам анализа принимается решение о перестановке соседней пары строк. При этом подразумевается также перестановка соответствующих столбцов матрицы смежности, чтобы она продолжала отражать верную структуру графа.

Локальная цель перестановок - перемещение нулевых элементов матрицы снизу-вверх и справа-налево.

Глобальная цель - формирование η - области Pi(l, m) с максимальным значением параметра m, то есть выделение максимального внутренне устойчивого множества вершин двойственного графа.

Пусть для анализа выбрана пара строк (i, i + 1) матрицы R=||rij|| размером n\*n.В строках выделяют две части: 1-я– (j= от 1 до i - 1); 2-я– (j= от i+2 до n).

Суть анализа заключается в определении истинностного значения трёх нижеприведенных условий:

1. $\sum\_{j=1}^{i-1}r\_{i,j}>\sum\_{j=1}^{i-1}r\_{i+1,j}$ - 1-я часть;
2. $\sum\_{j=1}^{i-1}r\_{i,j}= \sum\_{j=1}^{i-1}r\_{i+1,j}$ - 1-я часть, и $\sum\_{j=i+2}^{n}r\_{i,j}>\sum\_{j=i+2}^{n}r\_{i+1,j}$ - 2-я часть;
3. $\sum\_{j=1}^{i-1}r\_{i,j}= \sum\_{j=1}^{i-1}r\_{i+1,j}$ - 1-я часть, и $\sum\_{j=i+2}^{n}r\_{i,j}= \sum\_{j=i+2}^{n}r\_{i+1,j}$ - 2-я часть.

Ответ «да», то есть – переставлять, вырабатывается, если выполняются условия 1 или 2. В случае выполнения условия 3 ответ «да» вырабатывается с некоторой вероятностью, задаваемой априорно – это сделано для возможности преодоления локальных максимумов, которые могут возникнуть в ходе преобразований. В остальных случаях вырабатывается ответ «нет», то есть – переставлять.

Адаптивная поисковая процедура продолжается, пока существуют пары, для которых выполняются условия 1 или 2. В результате будет сформирована η - область P(1,m) и в графе Gd определено максимальное внутренне-устойчивое подмножество, соответствующие максимальному паросочетанию графа G.

# **Программная реализация алгоритма**

##  **Основные сведения о программе**

В качестве интерфейса общения с пользователем программа использует входной файл input.txt, в который вносится необходимая информация для работы алгоритма (матрица смежности исходного графа). По окончанию работы программа выводит результат в выходной файл output.txt

Результатом работы программы является множество ребер, входящих в максимальное паросочетание графа, где ребро представлено в виде пары инцидентных ему вершин (i, j).

Для реализации алгоритма выбран язык программирования Python 3.7.

## **Особенности реализации**

Программная реализация состоит из входного файла input.txt, выходного файла output.txt и двух исполнительных файла main.py и matching.py.

1. matching.pyсодержит:
	1. create\_dual\_graph(graph,c) – вспомогательная функция, которая по матрице смежности исходного графа (входной параметр graph) и количеству ребер в нем (входной параметр c) строит матрицу смежности двойственного графа;
	2. find\_best(dualGraph,count) – вспомогательная функция, которая по модифицированной матрице смежности двойственного графа (входной параметр dualGraph) и количеству ребер в нем (входной параметр count) находит лучшую η – область, т.е. ту, в которой параметр m наибольший;
	3. adaptive\_matching(graph) – основная функция, в которой вызываются функции create\_dual\_graph и find\_best, а так же происходит адаптивное преобразование строк матрицы двойственного графа;
2. main.pyсодержит:
	1. read\_graph() – функция, считывающая исходных граф из входного файла;
	2. generate\_rand\_graph(n) – функция, которая генерирует случайный граф для тестирования на основе генератора из пакета NetworkXи преобразует его матрицу смежности в приемлемый для алгоритма формат;
	3. print\_graph(graph,matching) – процедура, которая рисует граф по его матрице смежности (входной параметр graph) и списку ребер, входящих в максимальное паросочетание (входной параметр count)

# **Тестирование алгоритма**

Пример работы алгоритма на графе, приведенном в качестве примера в пункте 2.1. Исходный граф показан на рисунке 6.



Рисунок 6 – Исходный граф



Рисунок 7 – Граф, с отмеченными ребрами, входящими в максимальное паросочетание

Далее для тестов сгенерируем случайный граф с помощью функции generate\_rand\_graph.



Рисунок 8 – Случайный граф до нахождения паросочетания



Рисунок 9 – Случайный граф после нахождения паросочетания

Отрисовка графов выполнялась с помощью пакетов NetworkX и MatPlotLib.

Далее приведем сравнительную характеристику адаптивного алгоритма в сравнении с классическим алгоритмом Эдмондса [4] по ряду критериев:

* время работы алгоритма в зависимости от количества вершин,
* время работы алгоритма в зависимости от плотности,
* точность алгоритма (на сколько решение, найденное адаптивным алгоритмом, отличается от эталонного).

На рисунке 10 изображен график времени работы алгоритмов на случайных одинаковых графах разного размера. Как и ожидалось, на графах малого размера адаптивный алгоритм не дает выигрыш во времени. Однако примерно после n = 200 алгоритм Эдмондса начинает работать очень долго, в то время как время работы адаптивного алгоритма растет медленнее.



Рисунок 10 – График зависимости времени работы алгоритмов (в секундах) от количества вершин в графе

На рисунке 11 изображен график времени работы алгоритмов на одинаковых графах (n = 100) при разной плотности графов. Так как в адаптивном алгоритме есть прямая зависимость сложности только от размера графа, то при росте количества ребер сложность практически не меняется. В то же время алгоритм Эдмондса на плотных графах работает дольше, хотя и имеет меньшее время работы на разреженных графах.



Рисунок 11 – График зависимости времени работы алгоритмов (в секундах) от плотности графа

Точность решения алгоритма Эдмондса возьмем за эталонную - 100%. Обозначим a – количество ребер в паросочетании, найденном алгоритмом Эдмондса, b – количество ребер в паросочетании, найденном адаптивным алгоритмом. Так как алгоритм относится к классу эвристических, то найденное решение может быть хуже, чем решение алгоритма Эдмондса. Будем вычислять точность адаптивного алгоритма как b / a \* 100. На рисунке 12 изображен график, показывающий точность найденного решения. Тесты проводились на случайных графах с количеством вершин равным 50. Как видно, иногда решение адаптивного алгоритма действительно отличается от эталонного. Это связано с некой долей случайности поиска решения (3 условие сравнения строк). Однако за время тестов точность не опускалась ниже 93%.



Рисунок 12 – График точности адаптивного алгоритма

# **Заключение**

В ходе курсовой работы была поставлена задача нахождения максимального паросочетания. Были изучены принципы адаптивного поиска и эволюционные механизмы преобразования матрицы смежности для поиска решения данной задачи.

В рамках исследования этого алгоритма было реализовано приложение, в котором можно искать решение задачи максимального паросочетания как на случайно сгенерированном, так и на задаваемом пользователем, графе. Также, был проведен сравнительный анализ созданного алгоритма в сравнении с известным алгоритмом Эдмондса, оценены время работы на разных графах в зависимости от количества вершин и плотности графа.

Эвристические алгоритмы позволяют с достаточной точностью решать важные прикладные комбинаторные задачи, которые входят с состав более глобальных задач. Скорость работы таких алгоритмов зачастую превосходит скорость работы точных полнопереборных аналогов.

# **Список использованных источников**

1. Matching (graphtheory) [Электронный ресурс]. – 2018. – URL: [https://en.wikipedia.org/wiki/Matching\_(graph\_theory)](https://en.wikipedia.org/wiki/Matching_%28graph_theory%29) (дата обращения: 25.11.2018г).
2. Лебедев Б.К., Лебедев О.Б. Эволюционные процедуры решения комбинаторных задач на графах / Лебедев Б.К., Лебедев О.Б. – Таганрогский государственный радиотехнический университет. – 2005. – 10с.
3. Лебедев Б.К., Лебедев О.Б. Курейчик В.М. Поисковая адаптация. Теория и практика. / Лебедев Б.К., Лебедев О.Б. Курейчик В.М. – М.: ФИЗМАТЛИТ. – 2006. – 272 с.
4. Алгоритм Эдмондса нахождения наибольшего паросочетания в произвольных графах. [Электронный ресурс]. – URL: http://e-maxx.ru/algo/matching\_edmonds (дата обращения: 25.11.2018г).
5. Python 3.7.1. [Электронный ресурс]. – URL: https://www.python.org/ (дата обращения: 25.11.2018г).
6. NetworkX 2.2 [Электронный ресурс]. – URL: http://networkx.github.io/documentation/networkx-2.2/ (дата обращения: 25.11.2018г).
7. MatPlotLib 3.0.2 [Электронный ресурс]. – URL: https://matplotlib.org (дата обращения: 25.11.2018г).

# **Приложение А**

Файл основной программы main.py

import networkx as nx

import time

import matplotlib.pyplot as plt

from matching import adaptive\_matching

# Считываем матрицу смежности исходного графа из текстового файла

def read\_graph():

inputMatrix = open('input.txt', 'r')

 graph = []

 for row in inputMatrix:

graph.append(row.replace('\n', '').split(' '))

 for i in range(len(graph)):

 for j in range(len(graph)):

 graph[i][j] = int(graph[i][j])

 return graph

# Генерацияслучайногографа

def generate\_rand\_graph(n):

 graph = nx.fast\_gnp\_random\_graph(n, 0.4)

 m = nx.to\_numpy\_matrix(graph)

 graph = []

 for i in range(len(m)):

 row = []

 for j in range(len(m)):

row.append(int(m[i,j]))

graph.append(row)

returngraph

# Отрисовка графа без отмеченного паросочетания и с ним

def print\_graph(graph,matching):

 g = nx.DiGraph()

 edges = []

 for i in range(len(graph)):

 for j in range(len(graph)):

 if graph[i][j] == 1:

edges.append((i,j))

g.add\_edges\_from(edges)

red\_edges = []

black\_edges = []

 for i in g.edges:

 if i in matching:

red\_edges.append(i)

 else:

black\_edges.append(i)

pos = nx.spring\_layout(g)

nx.draw\_networkx\_nodes(g, pos, cmap=plt.get\_cmap('jet'),

node\_color='Lightgray', node\_size=300)

nx.draw\_networkx\_labels(g, pos)

nx.draw\_networkx\_edges(g, pos, edgelist=black\_edges, edge\_color='black', arrows=False)

nx.draw\_networkx\_edges(g, pos, edgelist=red\_edges, edge\_color='red', arrows=False)

plt.show()

pos = nx.spring\_layout(g)

nx.draw\_networkx\_nodes(g, pos, cmap=plt.get\_cmap('jet'),

node\_color='Lightgray', node\_size=300)

nx.draw\_networkx\_labels(g, pos)

nx.draw\_networkx\_edges(g, pos, edgelist=g.edges, edge\_color='black', arrows=False)

plt.show()

graph = read\_graph()

# matching = adaptive\_matching(graph)

# print\_graph(graph,matching)

f = open('output.txt','w')

f.write(str(adaptive\_matching(graph)))

# Приложение Б

Файл алгоритма matching.py

import random

# Построитьматрицусмежностидвойственногографа
def create\_dual\_graph(graph,c):
 n = len(graph)
dualGraph = [[0] \* c for \_ in range(c)]
 count = 0
 for i in range(n):
 for j in range(i + 1, n):
 if graph[i][j] != 0:
for k in range(n):
 if graph[i][k] != 0 and k != j:
dualGraph[count][graph[i][k] - 1] = 1
dualGraph[graph[i][k] - 1][count] = 1
for k in range(n):
 if graph[j][k] != 0 and k != i:
dualGraph[count][graph[j][k] - 1] = 1
dualGraph[graph[j][k] - 1][count] = 1
 count += 1
 return dualGraph

# Найтилучшуюобласть
def find\_best(dualGraph,count):
 best = (0, 0)
currentCol = 1
 start = 0
 while currentCol< count:
i = start
 while i != currentCol:
 if dualGraph[i][currentCol] != 0:
 if best[1] <currentCol - start:
 best = (start, currentCol - start)
 start = currentCol
currentCol -= 1
i = currentCol - 1
i += 1
currentCol += 1
 if best[1] <currentCol - start:
 best = (start, currentCol - start)
 return best

def adaptive\_matching(init\_graph):

 graph = []
 for i in init\_graph:
graph.append(i.copy())
 # Считаемколичестворебервисходномграфеидляудобстванумеруемих,
 # т. е. заменяемединицывматрицесмежностинапорядковыеномераребер
 n = len(graph)
 count = 0
 memory = {}
 for i in range(n):
 for j in range(i + 1, n):
 if graph[i][j] == 1:
 count += 1
 graph[i][j] = count
 graph[j][i] = count
 memory[count] = (i,j)
 # Строимматрицусмежностидвойственногографа, используяколичествореберв
 # исходномграфеипроставленнуюнумерациювматрицесмежностиисходногографа
 # Создаемсписокномеревребер, дляпоследующейихидентификации
labelDualGraph = [i+1 for i in range(count)]
dualGraph = create\_dual\_graph(graph,count)

 # Покаестьизменения, переставлятьстрокиматрицысмежностидвойственногографа
 # всоответсвиискритериямиперестановки
 possible = True
 while possible:
 possible = False
 for i in range(count):
 if i % 2 != 0:
 sum1 = 0
 sum2 = 0
 for j in range(i - 1):
 sum1 += dualGraph[i - 1][j]
 sum2 += dualGraph[i][j]
 if sum1 > sum2:
for k in range(count):
dualGraph[k][i - 1], dualGraph[k][i] = dualGraph[k][i], dualGraph[k][i - 1]
dualGraph[i - 1], dualGraph[i] = dualGraph[i], dualGraph[i - 1]
labelDualGraph[i - 1], labelDualGraph[i] = labelDualGraph[i], labelDualGraph[i - 1]
 possible = True
elif sum1 == sum2:
 sum1 = 0
 sum2 = 0
 for j in range(i + 2, count):
 sum1 += dualGraph[i - 1][j]
 sum2 += dualGraph[i][j]
 if sum1 > sum2:
for k in range(count):
dualGraph[k][i - 1], dualGraph[k][i] = dualGraph[k][i], dualGraph[k][i - 1]
dualGraph[i - 1], dualGraph[i] = dualGraph[i], dualGraph[i - 1]
labelDualGraph[i - 1], labelDualGraph[i] = labelDualGraph[i], labelDualGraph[i - 1]
 possible = True
elif sum1 == sum2:
 if random.random() < 0.5:
for k in range(count):
dualGraph[k][i - 1], dualGraph[k][i] = dualGraph[k][i], dualGraph[k][i - 1]
dualGraph[i - 1], dualGraph[i] = dualGraph[i], dualGraph[i - 1]
labelDualGraph[i - 1], labelDualGraph[i] = labelDualGraph[i], labelDualGraph[i - 1]
 for i in range(1, count):
 if i % 2 == 0:
 sum1 = 0
 sum2 = 0
 for j in range(i - 1):
 sum1 += dualGraph[i - 1][j]
 sum2 += dualGraph[i][j]
 if sum1 > sum2:
for k in range(count):
dualGraph[k][i - 1], dualGraph[k][i] = dualGraph[k][i], dualGraph[k][i - 1]
dualGraph[i - 1], dualGraph[i] = dualGraph[i], dualGraph[i - 1]
labelDualGraph[i - 1], labelDualGraph[i] = labelDualGraph[i], labelDualGraph[i - 1]
 possible = True
elif sum1 == sum2:
 sum1 = 0
 sum2 = 0
 for j in range(i + 2, count):
 sum1 += dualGraph[i - 1][j]
 sum2 += dualGraph[i][j]
 if sum1 > sum2:
for k in range(count):
dualGraph[k][i - 1], dualGraph[k][i] = dualGraph[k][i], dualGraph[k][i - 1]
dualGraph[i - 1], dualGraph[i] = dualGraph[i], dualGraph[i - 1]
labelDualGraph[i - 1], labelDualGraph[i] = labelDualGraph[i], labelDualGraph[i - 1]
 possible = True
elif sum1 == sum2:
 if random.random() < 0.5:
for k in range(count):
dualGraph[k][i - 1], dualGraph[k][i] = dualGraph[k][i], dualGraph[k][i - 1]
dualGraph[i - 1], dualGraph[i] = dualGraph[i], dualGraph[i - 1]
labelDualGraph[i - 1], labelDualGraph[i] = labelDualGraph[i], labelDualGraph[i - 1]

 # Находимлучшуюобласть
 best = find\_best(dualGraph,count)
 result = set()
 for i in range(best[0],best[0]+best[1]):
result.add(memory[labelDualGraph[i]])
 return result