МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №  43

ОТЧЁТ

ЗАЩИЩЁН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

### профессор                                   Скобцов Ю.А.

должность, уч. Степень, звание   подпись, дата           инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №6.

Оптимизация путей на графах с помощью муравьиных алгоритмов.

по курсу: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Задание:**

Вариант 3:  
Тип данных: координаты городов Оптимальное решение att48

1 6734 1453 2 2233 10 3 5530 1424 4 401 841 5 3082 1644 6 7608 4458 7 7573 3716 8 7265 1268 9 6898 1885 10 1112 2049 11 5468 2606 12 5989 2873 13 4706 2674 14 4612 2035 15 6347 2683 16 6107 669 17 7611 5184 18 7462 3590 19 7732 4723 20 5900 3561 21 4483 3369 22 6101 1110 23 5199 2182 24 1633 2809 25 4307 2322 26 675 1006 27 7555 4819 28 7541 3981 29 3177 756 30 7352 4506 31 7545 2801 32 3245 3305 33 6426 3173 34 4608 1198 35 23 2216 36 7248 3779 37 7762 4595 38 7392 2244 39 3484 2829 40 6271 2135 41 4985 140 42 1916 1569 43 7280 4899 44 7509 3239 45 10 2676 46 6807 2993 47 5185 3258 48 3023 1942 EOF 1 8 38 31 44 18 7 28 6 37 19 27 17 43 30 36 46 33 20 47 21 32 39 48 5 42 24 10 45 35 4 26 2 29 34 41 16 22 3 23 14 25 13 11 12 15 40 9 -1 EOF

Часть 1.

1. Создать программу, использующую МА для решение задачи поиска гамильтонова пути. Индивидуальное заданию выбирается по таблице В.1 в приложении В согласно номеру варианта.
2. Представить графически найденное решение. Предусмотреть возможность пошагового просмотра процесса поиска решения.
3. Сравнить найденное решение с представленным в условии задачи оптимальным решением.

Часть 2.

Реализовать с использованием муравьиных алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта (см. таблицу 3.1. и приложение Б.).

Представить графически найденное решение.

Сравнить найденное решение с представленным в условии задачи оптимальным решением и результатами, полученными в лабораторной работе №3. . Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.

1. **Краткие теоретические сведения:**

Муравьиные алгоритмы (МА) основаны на использовании популяции потенциальных решений и разработаны для решения задач комбинаторной оптимизации, прежде всего, поиска различных путей на графах. Кооперация между особями (искусственными муравьями) здесь реализуется на основе моделирования. При этом каждый агент, называемый искусственным муравьем, ищет решение поставленной задачи. Искусственные муравьи последовательно строят решение задачи, передвигаясь по графу, откладывают феромон и при выборе дальнейшего участка пути учитывают концентрацию этого фермента. Чем больше концентрация феромона в последующем участке, тем больше вероятность его выбора.

1. **Программа и результаты выполнения индивидуального задания с комментариями и выводами:**

**Код программы на ЯП Python:**

import random

import matplotlib.pyplot as plt

from itertools import permutations

import timeit

class Ant:

def \_\_init\_\_(self, graph):

self.graph = graph

self.path = []

self.current\_city = None

def start(self, start\_city):

self.current\_city = start\_city

self.path.append(start\_city)

def move\_to\_next\_city(self):

next\_cities = list(set(range(len(self.graph))) - set(self.path))

probabilities = [self.graph[self.current\_city][city]['pheromone'] for city in next\_cities]

total\_pheromone = sum(probabilities)

if total\_pheromone == 0:

return random.choice(next\_cities)

probabilities = [p / total\_pheromone for p in probabilities]

next\_city = random.choices(next\_cities, weights=probabilities)[0]

self.path.append(next\_city)

self.current\_city = next\_city

return next\_city

def is\_finished(self):

return len(self.path) == len(self.graph)

def path\_length(self):

distance = 0

for i in range(len(self.path) - 1):

distance += self.graph[self.path[i]][self.path[i+1]]['distance']

return distance

def update\_pheromones(graph, ants, decay\_rate, Q):

for i in range(len(graph)):

for j in range(i + 1, len(graph)):

pheromone = graph[i][j]['pheromone']

delta\_pheromone = sum([Q / ant.path\_length() for ant in ants if (i, j) in zip(ant.path[:-1], ant.path[1:])])

graph[i][j]['pheromone'] = (1 - decay\_rate) \* pheromone + delta\_pheromone

graph[j][i]['pheromone'] = graph[i][j]['pheromone']

def create\_graph(cities):

n = len(cities)

graph = [[{'distance': abs(cities[i][0] - cities[j][0]) + abs(cities[i][1] - cities[j][1]), 'pheromone': 1} for j in range(n)] for i in range(n)]

return graph

def find\_hamiltonian\_path(graph, num\_ants, max\_iterations, decay\_rate, Q):

best\_ant = None

best\_path\_length = float('inf')

for \_ in range(max\_iterations):

ants = [Ant(graph) for \_ in range(num\_ants)]

for ant in ants:

ant.start(random.randint(0, len(graph) - 1))

while not ant.is\_finished():

ant.move\_to\_next\_city()

update\_pheromones(graph, ants, decay\_rate, Q)

for ant in ants:

path\_length = ant.path\_length()

if path\_length < best\_path\_length:

best\_path\_length = path\_length

best\_ant = ant

return best\_ant

def find\_travelling\_salesman\_path(graph, num\_ants, max\_iterations, decay\_rate, Q):

best\_ant = None

best\_path\_length = float('inf')

for \_ in range(max\_iterations):

ants = [Ant(graph) for \_ in range(num\_ants)]

for ant in ants:

ant.start(random.randint(0, len(graph) - 1))

while not ant.is\_finished():

ant.move\_to\_next\_city()

# Возвращение в начальную точку

ant.path.append(ant.path[0])

update\_pheromones(graph, ants, decay\_rate, Q)

for ant in ants:

path\_length = ant.path\_length()

if path\_length < best\_path\_length:

best\_path\_length = path\_length

best\_ant = ant

return best\_ant

def visualize\_path(cities, path):

plt.figure(figsize=(10, 10))

for i in range(len(path) - 1):

x1, y1 = cities[path[i]]

x2, y2 = cities[path[i+1]]

plt.plot([x1, x2], [y1, y2], color='b', marker='o')

plt.show()

def compare\_with\_optimal(optimal\_path, optimal\_path\_length, found\_path, found\_path\_length):

print("Длина оптимального пути:", optimal\_path\_length)

if found\_path\_length == optimal\_path\_length:

print("Найденное решение совпадает с оптимальным.")

else:

print("Найденное решение отличается от оптимального.")

def run\_experiments(experiments):

results = []

for experiment in experiments:

start\_time = timeit.default\_timer()

best\_ant = find\_travelling\_salesman\_path(graph, \*\*experiment)

end\_time = timeit.default\_timer()

execution\_time = end\_time - start\_time

results.append((best\_ant.path\_length(), execution\_time))

return results

# Данные о городах

cities = [

(6734, 1453), (2233, 10), (5530, 1424), (401, 841), (3082, 1644),

(7608, 4458), (7573, 3716), (7265, 1268), (6898, 1885), (1112, 2049),

(5468, 2606), (5989, 2873), (4706, 2674), (4612, 2035), (6347, 2683),

(6107, 669), (7611, 5184), (7462, 3590), (7732, 4723), (5900, 3561),

(4483, 3369), (6101, 1110), (5199, 2182), (1633, 2809), (4307, 2322),

(675, 1006), (7555, 4819), (7541, 3981), (3177, 756), (7352, 4506),

(7545, 2801), (3245, 3305), (6426, 3173), (4608, 1198), (23, 2216),

(7248, 3779), (7762, 4595), (7392, 2244), (3484, 2829), (6271, 2135),

(4985, 140), (1916, 1569), (7280, 4899), (7509, 3239), (10, 2676),

(6807, 2993), (5185, 3258), (3023, 1942)

]

graph = create\_graph(cities)

# Параметры для эксперимента

experiments = [

{'num\_ants': 10, 'max\_iterations': 100, 'decay\_rate': 0.05, 'Q': 50},

{'num\_ants': 20, 'max\_iterations': 150, 'decay\_rate': 0.07, 'Q': 75},

{'num\_ants': 30, 'max\_iterations': 200, 'decay\_rate': 0.09, 'Q': 100}

]

# Запуск эксперимента

results = run\_experiments(experiments)

# Вывод результатов

for result in results:

print(f'Path length: {result[0]}, Execution time: {result[1]:.2f} seconds')

# Поиск гамильтонова пути

num\_ants = 50

max\_iterations = 200

decay\_rate = 0.1

Q = 100

best\_ant = find\_hamiltonian\_path(graph, num\_ants, max\_iterations, decay\_rate, Q)

print("Лучший путь:", best\_ant.path)

print("Длина лучшего пути:", best\_ant.path\_length())

visualize\_path(cities, best\_ant.path)

# Сравнение с оптимальным решением

optimal\_path = [1, 8, 38, 31, 44, 18, 7, 28, 6, 37, 19, 27, 17, 43, 30, 36, 46, 33, 20, 47, 21, 32, 39, 48, 5, 42, 24, 10, 45, 35, 4, 26, 2, 29, 34, 41, 16, 22, 3, 23, 14, 25, 13, 11, 12, 15, 40, 9]

optimal\_path\_length = sum([graph[optimal\_path[i]][optimal\_path[i+1]]['distance'] for i in range(len(optimal\_path)-1)])

compare\_with\_optimal(optimal\_path, optimal\_path\_length, best\_ant.path, best\_ant.path\_length())

# Поиск пути коммивояжёра

num\_ants = 50

max\_iterations = 200

decay\_rate = 0.1

Q = 100

best\_ant = find\_travelling\_salesman\_path(graph, num\_ants, max\_iterations, decay\_rate, Q)

print("Лучший путь:", best\_ant.path)

print("Длина лучшего пути:", best\_ant.path\_length())

visualize\_path(cities, best\_ant.path)

# Сравнение с оптимальным решением

optimal\_path = [1, 8, 38, 31, 44, 18, 7, 28, 6, 37, 19, 27, 17, 43, 30, 36, 46, 33, 20, 47, 21, 32, 39, 48, 5, 42, 24, 10, 45, 35, 4, 26, 2, 29, 34, 41, 16, 22, 3, 23, 14, 25, 13, 11, 12, 15, 40, 9, 1]

optimal\_path\_length = sum([graph[optimal\_path[i]][optimal\_path[i+1]]['distance'] for i in range(len(optimal\_path)-1)])

compare\_with\_optimal(optimal\_path, optimal\_path\_length, best\_ant.path, best\_ant.path\_length())

**Результат выполнения программы:**

Path length: 156458, Execution time: 2.59 seconds

Path length: 145838, Execution time: 7.76 seconds

Path length: 139948, Execution time: 15.49 seconds

Лучший путь: [1, 46, 42, 36, 30, 18, 19, 43, 6, 32, 13, 2, 47, 41, 38, 8, 37, 7, 15, 39, 14, 35, 29, 17, 16, 21, 40, 10, 12, 44, 25, 20, 33, 9, 31, 22, 5, 0, 28, 27, 45, 26, 11, 24, 4, 3, 34, 23]

Длина лучшего пути: 127287

