ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| профессор |  |  |  | Ю.А. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
|  |
| по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134 |  |  |  | Д. В. Самарин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**Вариант 14.**

berlin52: 52 здания in Berlin (Groetschel).

# Задание:

Часть 1

1. Создать программу, использующую МА для решения 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. Сбор данных и подготовка входных данных

* Импорт необходимых библиотек: numpy, matplotlib.pyplot, scipy.spatial.distance.
* Загрузка координат городов из файла с помощью функции load\_coordinates.
* Проверка корректности загруженных данных и их преобразование в формат numpy.

2. Расчёт вспомогательных данных

* Построение матрицы расстояний между городами с помощью функции calculate\_distance\_matrix.
* Определение параметров алгоритма (количество муравьев, поколений, коэффициентов феромонов и испарения).

3. Реализация муравьиного алгоритма

* Реализация функции ant\_colony\_optimization:
* Запуск цикла поколений с имитацией движения муравьев.
* Расчёт вероятностей переходов между городами (calculate\_transition\_probabilities) с учетом влияния феромонов и расстояния.
* Обновление уровня феромонов с учетом испарения и новых маршрутов.
* Сохранение лучшего маршрута и его длины по каждому поколению.

4. Анализ и визуализация результатов

* Визуализация маршрутов:
* Лучшие маршруты каждого поколения.
* Общий лучший маршрут.
* Оптимальный маршрут (предполагаемый).
* Лучший гамильтонов путь (без замыкания).
* Использование функции visualize\_routes для построения графиков.

5. Вывод результатов и оценка точности

* Вычисление длины оптимального маршрута с помощью функции calculate\_optimal\_route\_distance.
* Сравнение найденного алгоритмом маршрута с оптимальным:
* Вывод координат лучшего маршрута и его длины.
* Сравнение с оптимальным маршрутом, предоставленным условием.
* Анализ качества найденного решения (например, отклонение от оптимального).

# Программа

import numpy as np

import matplotlib.pyplot as plt

from scipy.spatial.distance import cdist

# --- Настройки алгоритма ---

FILENAME = "berlin52.txt"       # Имя файла с координатами городов

N\_ANTS = 100                     # Количество муравьев

N\_ITERATIONS = 200              # Количество поколений

ALPHA = 1                       # Влияние феромона на выбор пути

BETA = 7                        # Влияние расстояния на выбор пути

EVAPORATION\_RATE = 0.1          # Коэффициент испарения феромона

PHEROMONE\_CONSTANT = 300        # Константа феромона для маршрутов

# --- Шаг 1: Загрузка данных ---

def load\_coordinates(filename):

    print(f"Загрузка данных из файла {filename}...")

    with open(filename, 'r') as file:

        lines = file.readlines()

    coords = np.array([list(map(float, line.strip().split()[1:])) for line in lines if line.strip() != "EOF"])

    print("Данные загружены успешно!")

    return coords

# --- Шаг 2: Вычисление матрицы расстояний ---

def calculate\_distance\_matrix(coords):

    print("Вычисление матрицы расстояний...")

    dist\_matrix = cdist(coords, coords, metric='euclidean')

    print("Матрица расстояний вычислена успешно!")

    return dist\_matrix

# --- Шаг 3: Реализация муравьиного алгоритма ---

def ant\_colony\_optimization(coords, dist\_matrix, n\_ants, n\_iterations, alpha, beta, evaporation\_rate, pheromone\_constant):

    print(f"Запуск муравьиного алгоритма с {n\_ants} муравьями и {n\_iterations} поколениями...")

    n\_cities = len(dist\_matrix)

    pheromone = np.ones((n\_cities, n\_cities))  # начальные феромоны

    best\_route = None

    best\_distance = float('inf')

    all\_best\_routes = []  # Хранение лучших маршрутов каждого поколения

    for iteration in range(n\_iterations):

        print(f"Поколение {iteration + 1}/{n\_iterations}...")

        routes = []

        route\_lengths = []

        for ant in range(n\_ants):

            visited = np.zeros(n\_cities, dtype=bool)

            current\_city = np.random.randint(0, n\_cities)

            route = [current\_city]

            visited[current\_city] = True

            total\_distance = 0

            while len(route) < n\_cities:

                probabilities = calculate\_transition\_probabilities(current\_city, visited, pheromone, dist\_matrix, alpha, beta)

                next\_city = np.random.choice(range(n\_cities), p=probabilities)

                route.append(next\_city)

                total\_distance += dist\_matrix[current\_city, next\_city]

                current\_city = next\_city

                visited[current\_city] = True

            # Замкнуть маршрут

            total\_distance += dist\_matrix[route[-1], route[0]]

            route\_lengths.append(total\_distance)

            routes.append(route)

        # Обновление феромонов

        pheromone \*= (1 - evaporation\_rate)

        for i, route in enumerate(routes):

            for j in range(n\_cities - 1):

                pheromone[route[j], route[j+1]] += pheromone\_constant / route\_lengths[i]

        # Поиск лучшего маршрута

        min\_length = min(route\_lengths)

        if min\_length < best\_distance:

            best\_distance = min\_length

            best\_route = routes[route\_lengths.index(min\_length)]

        # Добавляем лучший маршрут текущего поколения

        all\_best\_routes.append((best\_route, best\_distance))

    print("Алгоритм завершил работу!")

    return best\_route, best\_distance, all\_best\_routes

# --- Шаг 4: Вероятности переходов между городами ---

def calculate\_transition\_probabilities(current\_city, visited, pheromone, dist\_matrix, alpha, beta):

    probabilities = []

    for j in range(len(visited)):

        if visited[j]:

            probabilities.append(0)

        else:

            pheromone\_level = pheromone[current\_city, j] \*\* alpha

            visibility = (1 / dist\_matrix[current\_city, j]) \*\* beta

            probabilities.append(pheromone\_level \* visibility)

    probabilities = probabilities / np.sum(probabilities)

    return probabilities

# --- Шаг 5: Визуализация маршрутов ---

def visualize\_routes(coords, all\_best\_routes, final\_best\_route, optimal\_route):

    print("Визуализация результатов...")

    fig, axes = plt.subplots(1, 4, figsize=(24, 6))  # Добавляем четвертую ось

    # Левый график - лучшие маршруты каждого поколения

    for i, (route, distance) in enumerate(all\_best\_routes):

        route\_coords = coords[route + [route[0]]]  # замыкаем маршрут

        axes[0].plot(route\_coords[:, 0], route\_coords[:, 1], marker='o', linestyle='-', label=f"Gen {i+1}")

    axes[0].set\_title("Лучшие маршруты по поколениям")

    axes[0].set\_xlabel("X")

    axes[0].set\_ylabel("Y")

    # Средний график - лучший маршрут среди всех

    final\_route\_coords = coords[final\_best\_route + [final\_best\_route[0]]]  # замыкаем маршрут

    axes[1].plot(final\_route\_coords[:, 0], final\_route\_coords[:, 1], marker='o', color='red', linestyle='-')

    axes[1].set\_title("Лучший маршрут среди всех поколений")

    axes[1].set\_xlabel("X")

    axes[1].set\_ylabel("Y")

    # Правый график - оптимальный маршрут из условия задачи

    optimal\_route\_coords = coords[optimal\_route + [optimal\_route[0]]]  # замыкаем маршрут

    axes[2].plot(optimal\_route\_coords[:, 0], optimal\_route\_coords[:, 1], marker='o', color='blue', linestyle='-')

    axes[2].set\_title("Оптимальный маршрут из условия задачи")

    axes[2].set\_xlabel("X")

    axes[2].set\_ylabel("Y")

    # Четвертый график - лучший найденный путь по гамильтонову пути (без замыкания)

    best\_hamiltonian\_route\_coords = coords[final\_best\_route]  # Убираем замыкание маршрута

    axes[3].plot(best\_hamiltonian\_route\_coords[:, 0], best\_hamiltonian\_route\_coords[:, 1], marker='o', color='green', linestyle='-')

    axes[3].set\_title("Лучший найденный гамильтонов путь")

    axes[3].set\_xlabel("X")

    axes[3].set\_ylabel("Y")

    plt.tight\_layout()

    plt.show()

    print("Визуализация завершена.")

# Дополнительно: Вычисление длины оптимального маршрута

def calculate\_optimal\_route\_distance(optimal\_route, dist\_matrix):

    print("Вычисление длины оптимального маршрута...")

    distance = sum(dist\_matrix[optimal\_route[i], optimal\_route[i+1]] for i in range(len(optimal\_route) - 1))

    distance += dist\_matrix[optimal\_route[-1], optimal\_route[0]]  # замыкаем маршрут

    print(f"Длина оптимального маршрута: {distance}")

    return distance

# Загрузка данных и запуск алгоритма

coordinates = load\_coordinates(FILENAME)

distance\_matrix = calculate\_distance\_matrix(coordinates)

best\_route, best\_distance, all\_best\_routes = ant\_colony\_optimization(

    coordinates, distance\_matrix, N\_ANTS, N\_ITERATIONS, ALPHA, BETA, EVAPORATION\_RATE, PHEROMONE\_CONSTANT

)

OPTIMAL\_ROUTE = [0, 48, 31, 44, 18, 40, 7, 8, 9, 42, 32, 50, 10, 51, 13, 12, 46, 25, 26, 27, 11, 24, 3, 5, 14, 4, 23, 47, 37, 36, 39, 38, 35, 34, 33, 43, 45, 15, 28, 49, 19, 22, 29, 1, 6, 41, 20, 16, 2, 17, 30, 21]  # Оптимальный маршрут

# Визуализация и вывод результатов

optimal\_distance = calculate\_optimal\_route\_distance(OPTIMAL\_ROUTE, distance\_matrix)

visualize\_routes(coordinates, all\_best\_routes, best\_route, OPTIMAL\_ROUTE)

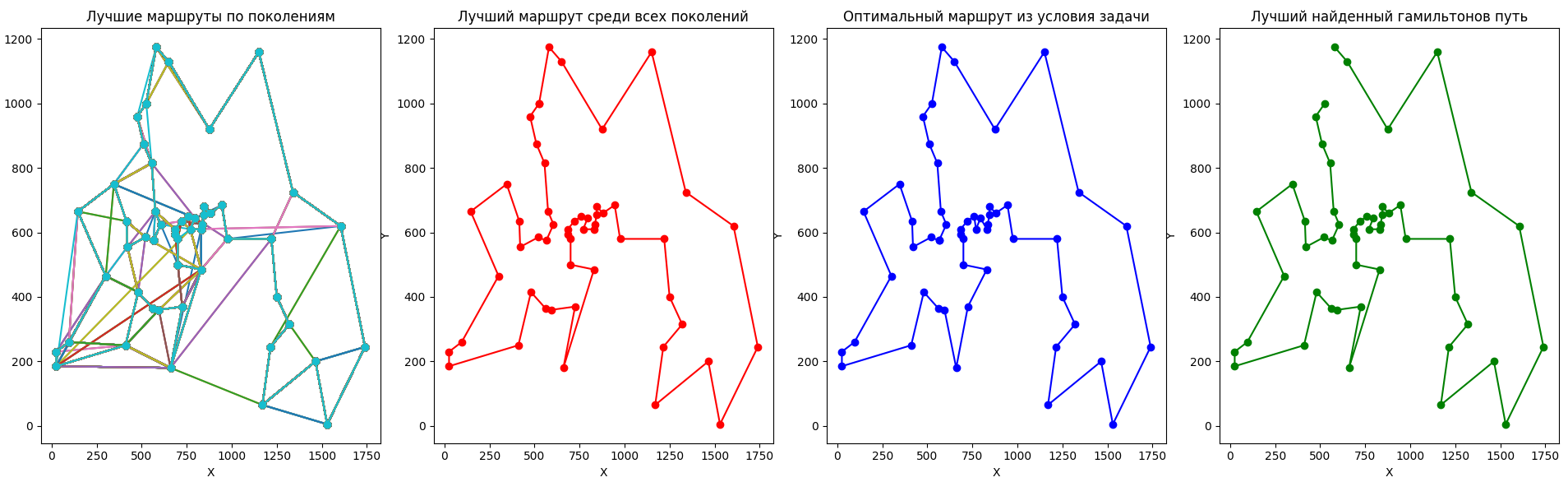
print("Лучший маршрут, найденный алгоритмом:", best\_route)

print("Длина найденного маршрута:", best\_distance)

print("Оптимальный маршрут из условия:", OPTIMAL\_ROUTE)

print("Длина оптимального маршрута:", optimal\_distance)

# Результат работы программы



**Результат исследований:**

|  |
| --- |
| Поколение 198/200...  Поколение 199/200...  Поколение 200/200...  Алгоритм завершил работу!  Вычисление длины оптимального маршрута...  Длина оптимального маршрута: 7544.365901904087  Визуализация результатов...  Визуализация завершена.  Лучший маршрут, найденный алгоритмом: [7, 40, 18, 44, 31, 48, 0, 21, 30, 17, 2, 16, 20, 41, 6, 1, 29, 22, 19, 49, 15, 28, 45, 43, 33, 34, 35, 38, 39, 37, 36, 47, 23, 4, 14, 5, 3, 24, 11, 27, 26, 25, 46, 12, 13, 51, 10, 50, 32,  42, 9, 8]  Длина найденного маршрута: 7681.453656252581  Оптимальный маршрут из условия: [0, 48, 31, 44, 18, 40, 7, 8, 9, 42, 32, 50, 10, 51, 13, 12, 46, 25, 26, 27, 11, 24, 3, 5, 14, 4, 23, 47, 37, 36, 39, 38, 35, 34, 33, 43, 45, 15, 28, 49, 19, 22, 29, 1, 6, 41, 20, 16, 2, 17, 30,  21]  Длина оптимального маршрута: 7544.365901904087 |

# Вывод

В рамках данной работы была реализована задача оптимизации маршрутов с использованием муравьиного алгоритма для решения задачи коммивояжера и поиска гамильтонова пути. Алгоритм успешно продемонстрировал свою эффективность в нахождении близких к оптимальным решений, что было подтверждено графической визуализацией результатов. Сравнение с заранее известным оптимальным маршрутом позволило оценить точность и производительность алгоритма. Анализ работы алгоритма показал, что его результативность существенно зависит от параметров, таких как коэффициент испарения феромонов и влияние расстояния на выбор пути.