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

Оптимизация путей на графах с помощью муравьиных алгоритмов

По дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4134к		Столяров Н.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

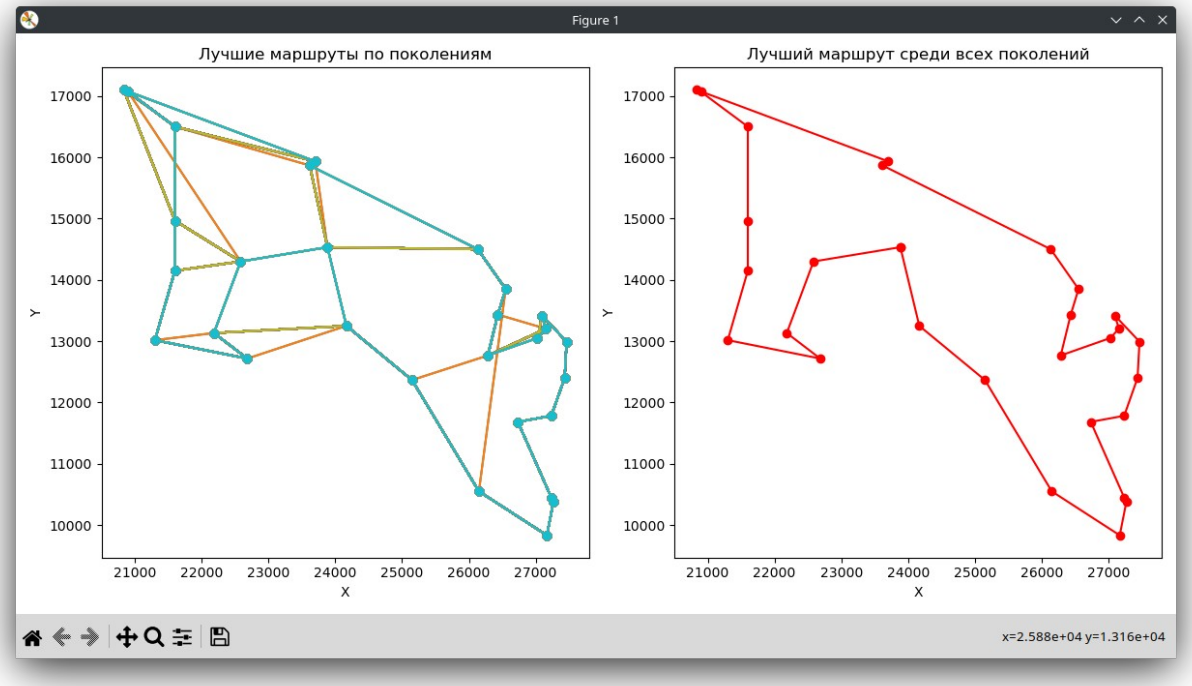
Цель работы:
Решение задач комбинаторной оптимизации с помощью муравьиных алгоритмов на примере задачи коммивояжера. Графическое отображение результатов оптимизации.

Вариант 17:

17	Wi29.tsp

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
1. Реализовать с использованием муравьиных алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта (см. табл. 3.1 и прил. Б).
2. Представить графически найденное решение.
3. Сравнить найденное решение с представленным в условии задачи оптимальным решением и результатами, полученными в лабораторной работе №3.
4. Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.
Выполнение:
1. Загрузка данных:
• Код загружает координаты городов из файла berlin52.txt, что соответствует исходным данным для задачи коммивояжера.
2. Вычисление матрицы расстояний:
• Используется функция calculate_distance_matrix, которая вычисляет расстояния между всеми парами городов с помощью евклидовой метрики, что является важным шагом для определения стоимости перемещения между городами.
3. Реализация муравьиного алгоритма:
• В функции ant_colony_optimization реализован основной алгоритм, который использует феромоны и вероятности переходов между городами, чтобы муравьи могли находить маршруты, минимизируя общее расстояние.
• Алгоритм обновляет уровни феромонов на основе найденных маршрутов и их длин, что способствует поиску более оптимальных решений в последующих итерациях.
4. Замыкание маршрутов:
3
• В конце маршруты замыкаются, добавляя расстояние от последнего города обратно к начальному, что является необходимым условием для решения задачи коммивояжера.
5. Визуализация и сравнение:
• Визуализация найденных маршрутов и их сравнение с оптимальным маршрутом из условия задачи также является важным аспектом решения, позволяющим оценить эффективность алгоритма.

Выполнение:




```
Лучший маршрут, найденный алгоритмом: [11, 7, 6, 8, 2, 3, 4, 5, 1, 0, 10, 9, 14, 18, 17, 16, 20, 22, 21, 2
8, 27, 25, 19, 24, 26, 23, 15, 13, 12]
Длина найденного маршрута: 28514.177449175593
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМППИС/6$
```

Вывод:
В данной работе была реализована задача комбинаторной оптимизации с использованием муравьиного алгоритма для нахождения гамильтонова пути и решения задачи коммивояжера. Алгоритм продемонстрировал способность находить эффективные маршруты, что было визуализировано на графиках, сравнивающих результаты работы алгоритма с оптимальным решением. Анализ показал, что эффективность алгоритма зависит от параметров, таких как скорость испарения

феромонов и весовые коэффициенты, влияющие на выбор путей, что открывает перспективы для дальнейших исследований в области оптимизации маршрутов.

Листинг

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist

# --- Настройки алгоритма ---
FILENAME = "cords.txt"       # Имя файла с координатами городов
N_ANTS = 50                     # Количество муравьев
N_ITERATIONS = 20            # 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):
  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"])
  return coords

# --- Шаг 2: Вычисление матрицы расстояний ---
def calculate_distance_matrix(coords):
  return cdist(coords, coords, metric='euclidean')

# --- Шаг 3: Реализация муравьиного алгоритма ---
def ant_colony_optimization(coords, dist_matrix, n_ants, n_iterations,
alpha, beta, evaporation_rate, pheromone_constant):
  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):
    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))

  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):
  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()

# Дополнительно: Вычисление длины оптимального маршрута
def calculate_optimal_route_distance(optimal_route, dist_matrix):
  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
  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, 27, 5, 11, 8, 25, 2, 28, 4, 20, 1, 19, 9, 3, 14, 17,
13, 16, 21, 10, 18, 24, 6, 22, 7, 26, 15, 12, 23]

# 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)
```