ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |
|  |
| 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.**



# Задание:

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

Сравнить найденное решение с представленным в условии задачи оптимальным решением.

Представить графически найденное решение.

Проанализировать время выполнения и точность нахождения результата в зависимости от вероятности различных видов кроссовера, мутации.

# Программа

import numpy as np

import matplotlib.pyplot as plt

from deap import base, creator, tools, algorithms

import math

import time

import sys

# Параметры, которые можно изменить

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

POPULATION\_SIZE = 500       # Размер популяции

N\_GENERATIONS = 1000         # Максимальное количество поколений

CX\_PROB = 0.9                # Вероятность кроссовера

MUT\_PROB = 0.1               # Вероятность мутации

TOURNAMENT\_SIZE = 5          # Размер турнира

MUTATION\_INDPB = 0.2         # Вероятность мутации каждого гена

ELITE\_SIZE = 1               # Количество лучших решений, сохраняемых между поколениями

PATIENCE = 100               # Порог терпения (макс. кол-во поколений без улучшений)

# Чтение координат городов из файла

def read\_cities(filename):

    cities = []

    try:

        with open(filename, 'r') as f:

            for line in f:

                parts = line.strip().split()

                if len(parts) == 3:  # Ожидаем: номер, x, y

                    cities.append((float(parts[1]), float(parts[2])))

    except FileNotFoundError:

        print(f"Файл '{filename}' не найден.")

        sys.exit(1)

    except Exception as e:

        print(f"Ошибка при чтении файла: {e}")

        sys.exit(1)

    return np.array(cities)

# Функция для вычисления евклидова расстояния между двумя городами

def euclidean\_distance(city1, city2):

    return math.sqrt((city1[0] - city2[0]) \*\* 2 + (city1[1] - city2[1]) \*\* 2)

# Функция для вычисления общей длины маршрута

def total\_distance(route, cities):

    dist = 0

    for i in range(len(route) - 1):

        dist += euclidean\_distance(cities[route[i]], cities[route[i - 1]])

    dist += euclidean\_distance(cities[route[-1]], cities[route[0]])  # Замыкание тура

    return dist

# Настройка DEAP: создание индивидуумов и популяции

def setup\_ga(cities):

    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))  # Минимизация расстояния

    creator.create("Individual", list, fitness=creator.FitnessMin)

    toolbox = base.Toolbox()

    toolbox.register("indices", np.random.permutation, len(cities))

    toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indices)

    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    toolbox.register("mate", tools.cxPartialyMatched)  # PMX-кроссовер

    toolbox.register("mutate", tools.mutShuffleIndexes, indpb=MUTATION\_INDPB)  # Мутация

    toolbox.register("select", tools.selTournament, tournsize=TOURNAMENT\_SIZE)

    toolbox.register("evaluate", lambda ind: (total\_distance(ind, cities),))

    return toolbox

# Алгоритм с проверкой на отсутствие улучшений

def run\_ga(toolbox):

    pop = toolbox.population(n=POPULATION\_SIZE)

    hof = tools.HallOfFame(ELITE\_SIZE)

    stats = tools.Statistics(lambda ind: ind.fitness.values[0])

    stats.register("min", np.min)

    stats.register("avg", np.mean)

    start\_time = time.time()

    # Инициализация переменных для отслеживания улучшений

    best\_fitness = float("inf")

    generations\_without\_improvement = 0

    for gen in range(N\_GENERATIONS):

        pop, logbook = algorithms.eaSimple(pop, toolbox, cxpb=CX\_PROB, mutpb=MUT\_PROB,

                                           ngen=1, stats=stats, halloffame=hof, verbose=False)

        current\_best = hof[0].fitness.values[0]

        print(f"Поколение {gen + 1}, лучшая длина маршрута: {current\_best:.2f}")

        if current\_best < best\_fitness:

            best\_fitness = current\_best

            generations\_without\_improvement = 0  # Сбросить счётчик

        else:

            generations\_without\_improvement += 1

        # Если нет улучшений в течение PATIENCE поколений, остановить алгоритм

        if generations\_without\_improvement >= PATIENCE:

            print(f"Остановка на {gen + 1} поколении из-за отсутствия улучшений.")

            break

    end\_time = time.time()

    print(f"Время выполнения: {end\_time - start\_time:.2f} секунд")

    return hof[0]

# Функция для визуализации маршрута

def plot\_route(route, cities):

    route\_cities = cities[route]

    plt.figure(figsize=(10, 6))

    plt.plot(route\_cities[:, 0], route\_cities[:, 1], 'o-', markersize=8)

    plt.plot([route\_cities[-1, 0], route\_cities[0, 0]],

             [route\_cities[-1, 1], route\_cities[0, 1]], 'o-')  # Замыкание маршрута

    plt.title("Найденный маршрут коммивояжера")

    plt.xlabel("X")

    plt.ylabel("Y")

    plt.grid(True)

    plt.show()

# Функция для сравнения найденного маршрута с лучшим маршрутом

def compare\_routes(optimal\_route, found\_route, cities):

    # Вычисление общей длины маршрутов

    best\_distance = total\_distance(optimal\_route, cities)

    found\_distance = total\_distance(found\_route, cities)

    print("Сравнение маршрутов:")

    print(f"Общая длина лучшего маршрута: {best\_distance:.2f}")

    print(f"Общая длина найденного маршрута: {found\_distance:.2f}")

    # Основная функция

def main():

    cities = read\_cities(FILENAME)

    toolbox = setup\_ga(cities)

    best\_route = run\_ga(toolbox)

    # Ваш лучший маршрут, который нужно сравнить (вставьте свои значения, начиная с 0)

    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, 35, 34, 37, 36, 33, 30, 29, 41, 43, 15, 28, 49, 1, 2, 19, 20, 21, 22, 38, 39,

    8]

    # Убедитесь, что оптимальный маршрут корректен и соответствует числу городов

    #if len(optimal\_route) != len(cities):

    print(f"Лучший найденный маршрут: {best\_route}")

    print(f"Общая длина маршрута: {total\_distance(best\_route, cities):.2f}")

    plot\_route(best\_route, cities)

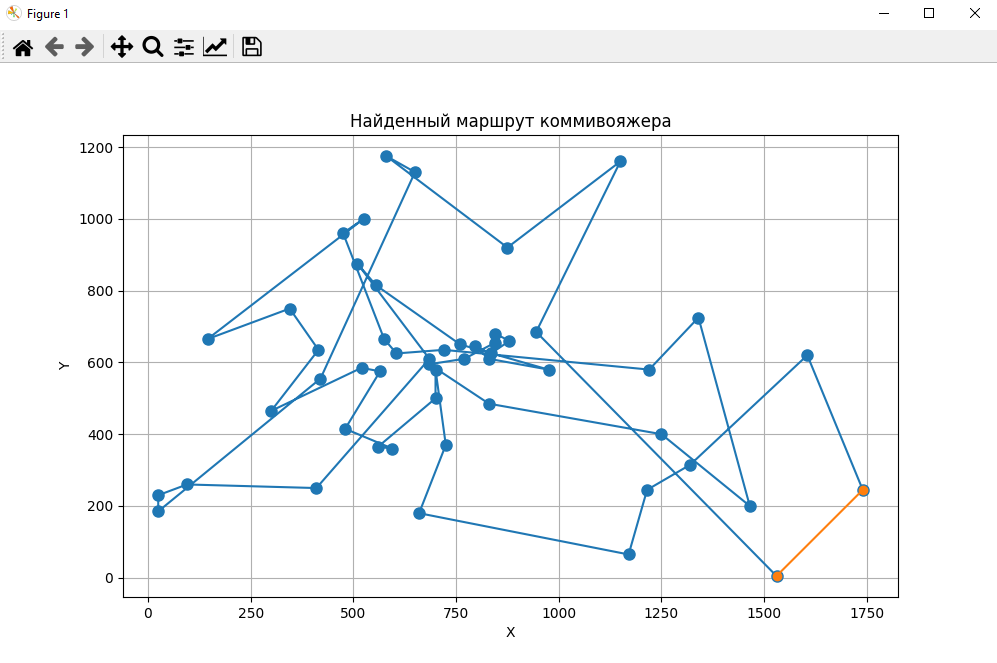
    # Сравнение маршрутов

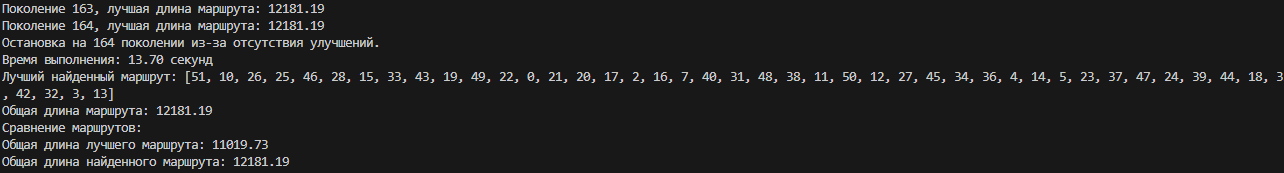
    compare\_routes(optimal\_route, best\_route, cities)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Результат работы программы





**Результат исследований:**

Программа вычислила общую длинную лучшего маршрута, затратив 13 секунд. Разные параметры давали разные результаты. Важно соблюдать баланс и подбирать параметры кроссовер под необходимые задачи тщательней.

# Вывод

В ходе выполнения лабораторной работы был реализован ГА для нахождения оптимального решения поставленной задачи