МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ПРЕПОДАВАТЕЛЬ

профессор, д-р техн. наук, профессор		Ю.А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3

ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

по дисциплине: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.		
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

1. Задание

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

2. Сравнить найденное решение с представленным в условии задачи оптимальным решением.

3. Представить графически найденное решение.

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

2. Индивидуальное задание по варианту

Вариант 10

DJ89 – Представление порядка

3. Теоретические сведения

Генетический алгоритм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как мутации, отбор и кроссинговер.

В данной работе были реализованы следующие операторы:

1. Репродукция — это процесс, в котором хромосомы копируются в промежуточную популяцию для дальнейшего "размножения" согласно их значениям целевой (фитнес-) функции. При этом хромосомы с лучшими значениями целевой функции имеют большую вероятность попадания одного или более потомков в следующее поколение.

2. Кроссинговер. В данной работе использован min-max кроссинговер.

3. Мутация – данный оператор совершает инверсию случайного гена в

хромосоме. Иногда данный оператор играет вторичную роль, так как

его вероятность слишком мала, примерно 0.01%

Данный алгоритм также пригоден для решения задачи коммивояжера.

В этой случае в роли хромосомы будет выступать тур – маршрут, который прошел

коммивояжер.

Фитнесс функция – расстояние, которое пришлось проехать по всему маршруту.

Туры могут быть реализованы различными представлениями: порядковое, путевое

и соседство. Для каждого из них существуют разнообразные виды кроссовера, которые

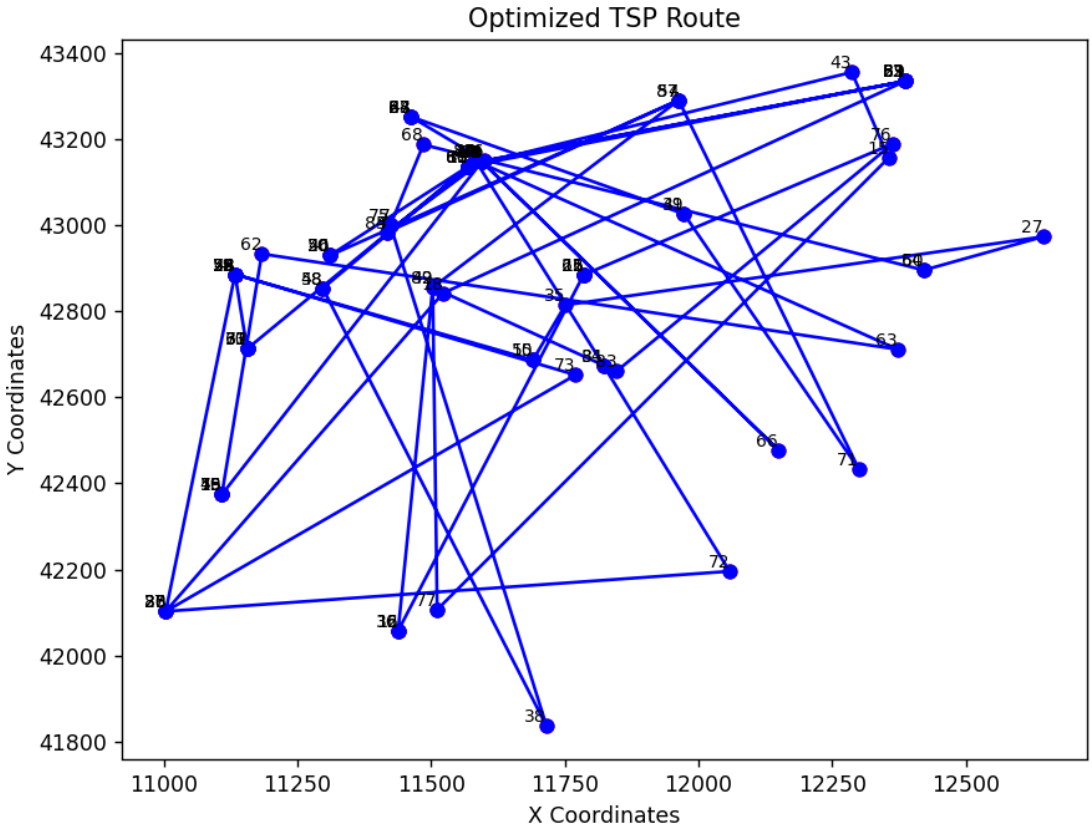
напрямую зависят на точность нахождения оптимального пути.

Мутация в данном алгоритме реализована путем переставления случайного отрезка

маршрута в случайное место

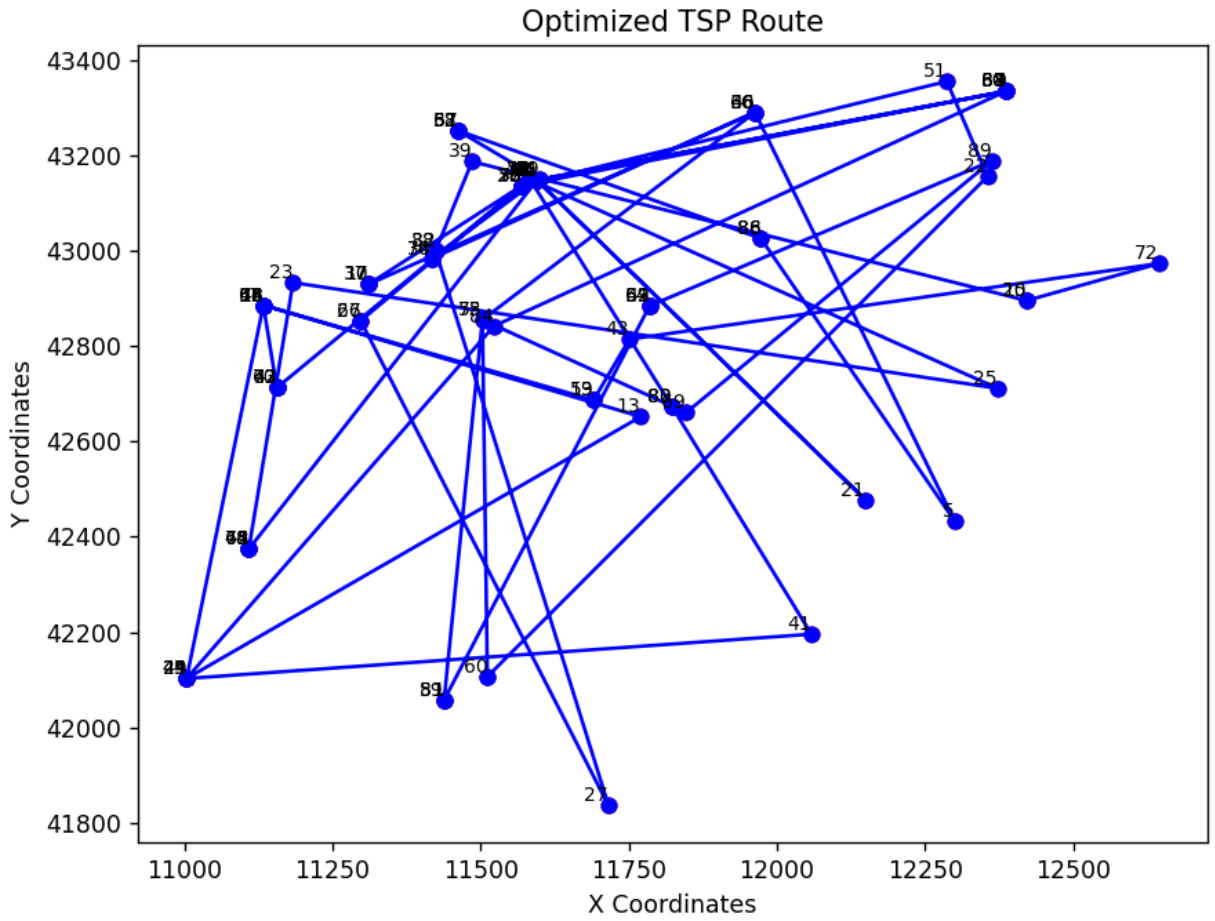
4. Результаты выполнения программы

```
crossover_rate = 0.5
mutation_rate = 0.01
num_generations = 500
population_size = 50
```



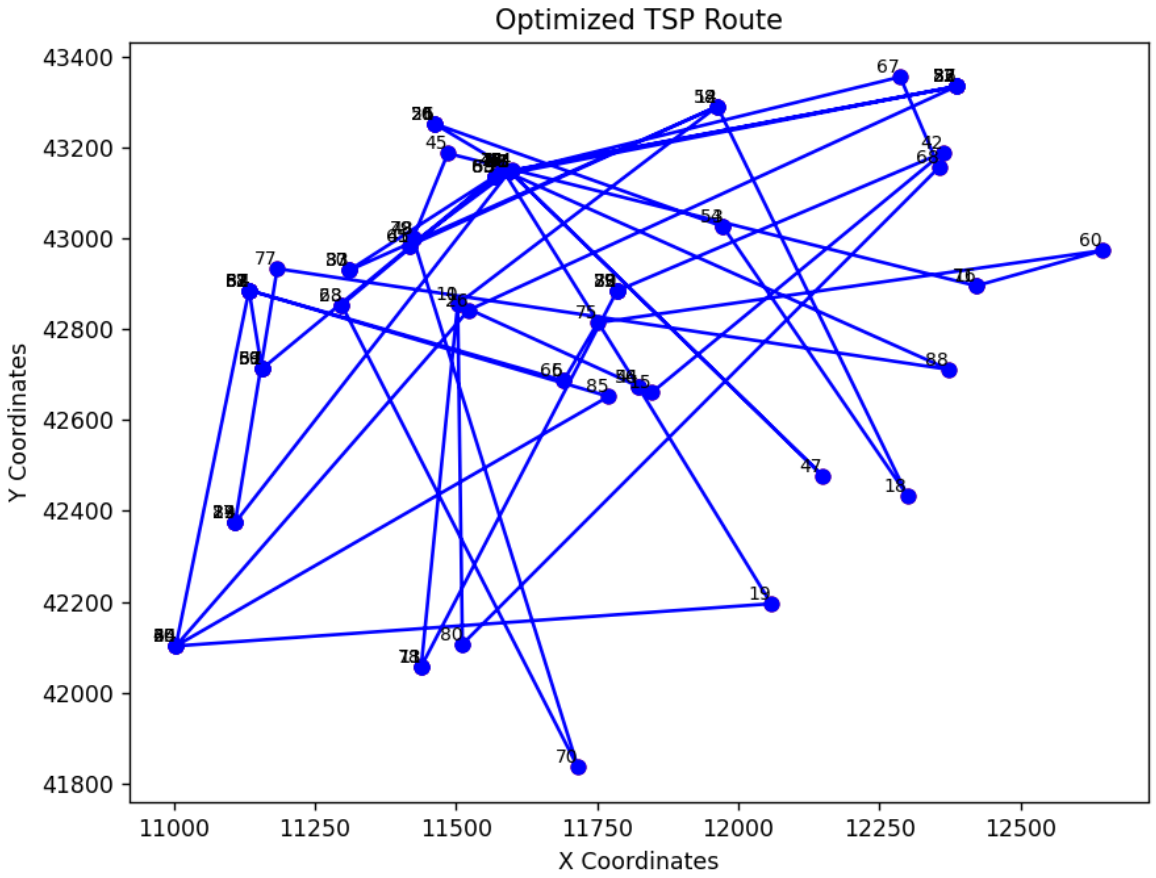

```
Best solution order representation: [77, 49, 16, 36, 12, 11, 22, 65, 6, 76, 83,
82, 87, 57, 71, 41, 39, 47,
64, 88, 23, 46, 69, 89, 3, 29, 80, 26, 40, 50, 4, 84, 5, 9, 66, 74, 51, 79, 56,
13, 45, 62, 63, 2,
14, 86, 42, 24, 81, 34, 72, 20, 28, 37, 18, 32, 53, 21, 17, 61, 30, 70, 67, 33,
31, 8, 25, 44, 59, 52, 85,
73, 78, 55, 10, 35, 27, 60, 54, 68, 7, 75, 38, 58, 48, 1, 19, 43, 15]
Best fitness: 1.807933397206845e-05
```

```
crossover_rate = 0.1
mutation_rate = 0.01
num_generations = 500
population_size = 50
```



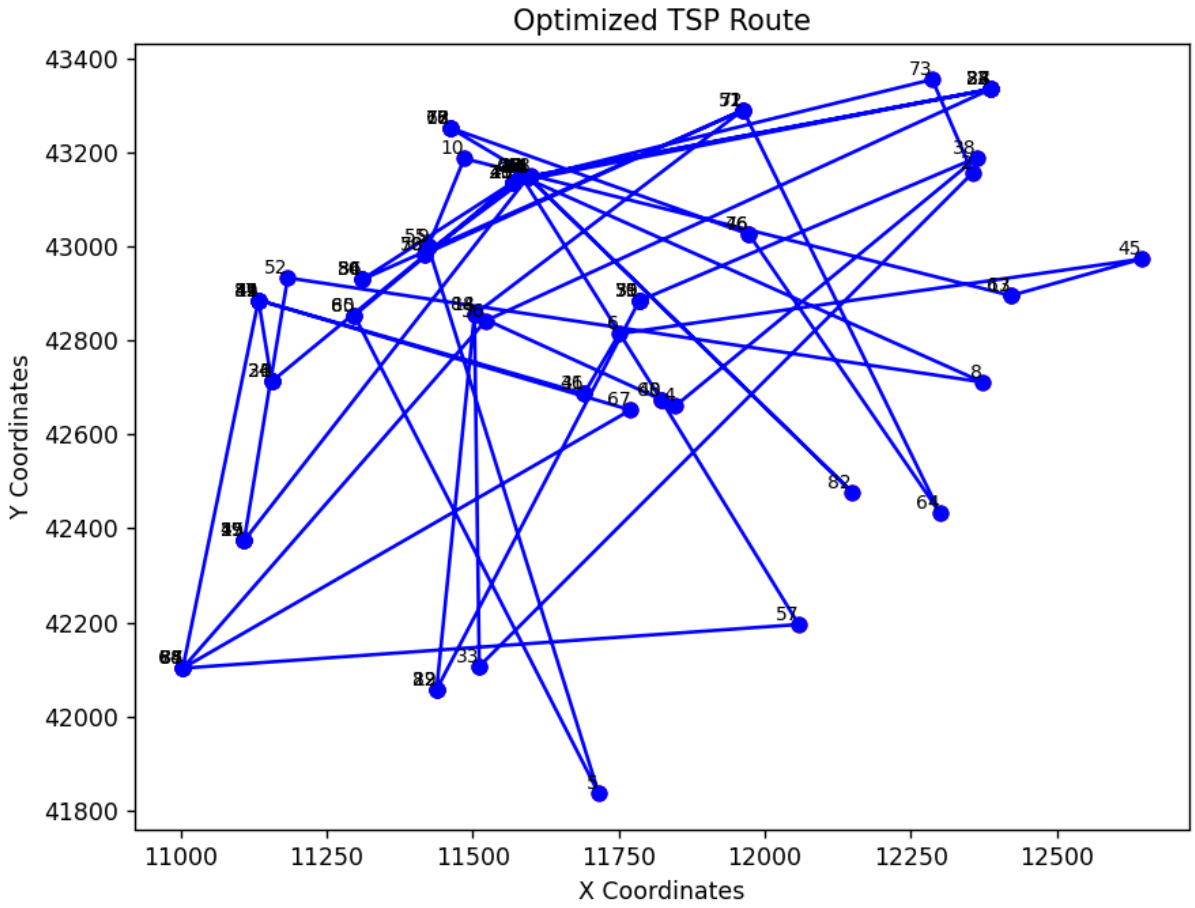

```
Best solution order representation:
[60, 55, 1, 81, 59, 3, 62, 49, 54, 89, 69, 73, 36, 40, 5, 88, 86, 7, 58, 32,
61, 79, 87, 9, 71, 35, 78, 37, 10, 17, 66, 76, 34, 85, 21, 52, 16, 74, 48, 65,
2, 23, 25, 14, 31, 50, 11, 12, 83, 80, 41, 24, 45, 29, 84, 4, 68, 30, 33, 28, 56,
42, 70, 47, 63, 18, 57, 75, 46, 8, 44, 13, 64, 53, 19, 43, 72, 15, 20, 39, 82,
38,
27, 67, 26, 6, 77, 51, 22]
Best fitness: 1.8507432549958912e-05
```

```
crossover_rate = 0.9
mutation_rate = 0.01
num_generations = 500
population_size = 50
```



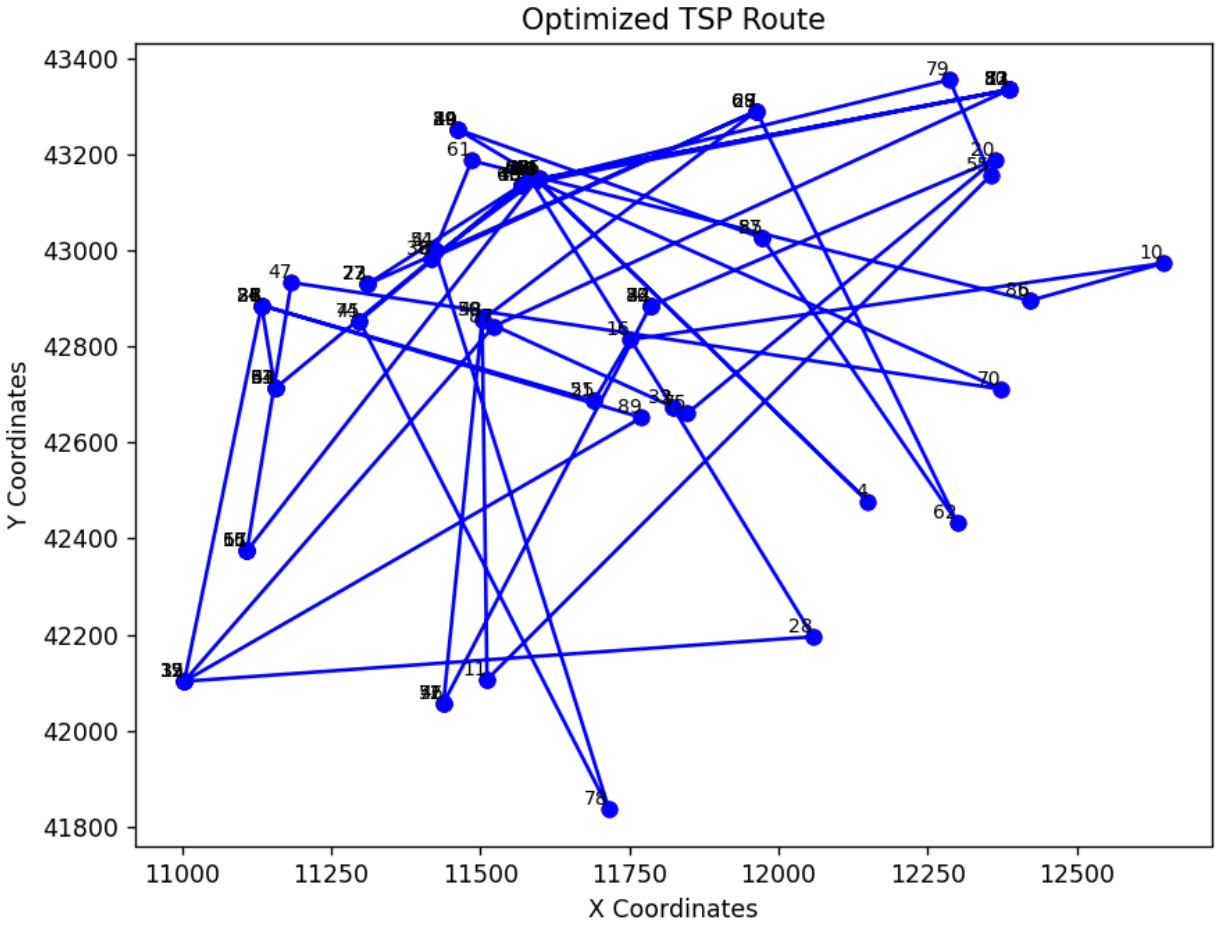

```
Best solution order representation:
 [80, 10, 13, 11, 78, 73, 20, 82, 39, 42, 15, 1, 58, 14, 18,
 3, 54, 6, 50, 21, 25, 34, 27, 86, 43, 31, 69, 30, 83, 37, 2,
  65, 41, 53, 47, 81, 57, 84, 29, 17, 23, 77, 88, 72, 76, 12, 46,
  74, 44, 56, 19, 35, 40, 24, 26, 22, 52, 33, 89, 9, 55, 59, 36, 61,
   7, 38, 32, 87, 62, 51, 64, 85, 4, 5, 66, 75, 60, 16, 71, 45, 48, 79, 70, 63,
28, 49, 8, 67, 68]
Best fitness: 1.888744814589149e-05
```

```
crossover_rate = 0.5
mutation_rate = 0.15
num_generations = 500
population_size = 50
```




```
Best solution order representation:
[33, 88, 89, 12, 22, 50, 39, 31, 75, 38, 4, 14, 72, 71, 64, 76, 46,
62, 16, 77, 78, 18, 7, 24, 84, 69, 11, 54, 30, 86, 51, 70, 58, 35, 82,
21, 61, 59, 47, 19, 25, 52, 8, 23, 66, 20, 53, 79, 48, 60, 57, 85, 87, 68,
 56, 32, 28, 83, 43, 40, 29, 34, 26, 1, 3, 42, 81, 15, 44, 37, 74, 67, 49,
  36, 41, 6, 45, 63, 17, 10, 9, 55, 5, 80, 65, 27, 13, 73, 2]
Best fitness: 1.9379608447367338e-05
```

```
crossover_rate = 0.5
mutation_rate = 0.3
num_generations = 500
population_size = 50
```




```
Best solution order representation:
[11, 48, 71, 52, 46, 42, 84, 30, 73, 20, 75, 59, 27, 68, 62, 57,
 85, 29, 14, 49, 80, 65, 50, 72, 24, 18, 43, 23, 22, 77, 69, 6, 38,
  60, 4, 36, 9, 15, 56, 1, 67, 47, 70, 82, 76, 37, 63, 58, 2, 33, 28,
   17, 35, 32, 87, 12, 81, 83, 39, 13, 40, 53, 64, 44, 31, 21, 34, 8,
    7, 88, 19, 89, 26, 51, 25, 16, 10, 5, 86, 61, 41, 54, 78, 45, 74, 3, 66, 79,
55]
Best fitness: 1.795981732319523e-05
```

Вывод

В ходе выполнения лабораторной работы была решена задача аппроксимации функции с помощью генетического программирования, графически отображены найденные решения. Из схем хорошо видно, что при увеличении вероятности мутации сильно увеличивается время поиска решения и падает точность. Также можем сделать вывод, что при увеличении кроссинговера точность решения сильно падает.

Код программы

```
import random
import numpy as np
import matplotlib.pyplot as plt

# Считать данные из файла
def read_tsp_file(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
        coordinates = []
        for line in lines:
            parts = line.split()
            if len(parts) > 1 and parts[0].isdigit():  # Исключаем строки
'EOF' и другие, не являющиеся координатами
                city_id = int(parts[0])  # Идентификатор города
                x, y = map(float, parts[1:3])  # Используем второе и третье
значение как координаты
                coordinates.append((city_id, x, y))
    return coordinates




# Рассчитать расстояние между двумя точками
def distance(point1, point2):
    x1, y1 = point1
    x2, y2 = point2
    return np.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)


# Инициализация популяции случайными путями
def initialize_population(num_individuals, num_cities):
    population = []
    for _ in range(num_individuals):
        individual = list(range(num_cities))
        random.shuffle(individual)
        population.append(individual)
    return population


# Рассчитать длину пути для каждой особи в популяции
def calculate_fitness(population, distances):
    fitness_values = []
    for individual in population:
        length = sum(distances[individual[i]][individual[i + 1]] for i in
range(len(individual) - 1))
        length += distances[individual[-1]][individual[0]]  # Замыкаем путь
        fitness_values.append(1 / length)  # Используем обратную величину
длины пути
    return fitness_values


# Выбор родителей с использованием рулеточной селекции
def select_parents(population, fitness_values):
    selected_parents = random.choices(population, weights=fitness_values,
k=2)
    return selected_parents


# Оператор кроссовера (например, PMX)
# Оператор кроссовера (одноточечный кроссовер)
def crossover(parent1, parent2):
```

```
    n = len(parent1)
    crossover_point = random.randint(0, n - 1)

    child1 = parent1[:crossover_point] + [gene for gene in parent2 if gene
not in parent1[:crossover_point]]
    child2 = parent2[:crossover_point] + [gene for gene in parent1 if gene
not in parent2[:crossover_point]]

    return child1, child2


# Оператор мутации (например, инвертирование)
def mutate(individual, mutation_rate):
    # Реализуйте оператор мутации, например, инвертирование
    # Используйте mutation_rate для управления частотой мутаций
    pass


# Главная функция генетического алгоритма
def genetic_algorithm(num_generations, population_size, crossover_rate,
mutation_rate, tsp_data):
    num_cities = len(tsp_data)
    distances = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(i, num_cities):
            distances[i][j] = distances[j][i] = distance(tsp_data[i][1:],
tsp_data[j][1:])

    population = initialize_population(population_size, num_cities)
    best_solution = None
    best_fitness = 0

    for generation in range(num_generations):
        fitness_values = calculate_fitness(population, distances)

        for i in range(0, population_size, 2):
            parent1, parent2 = select_parents(population, fitness_values)

            if random.random() < crossover_rate:
                child1, child2 = crossover(parent1, parent2)
            else:
                child1, child2 = parent1[:], parent2[:]

            if random.random() < mutation_rate:
                mutate(child1, mutation_rate)
            if random.random() < mutation_rate:
                mutate(child2, mutation_rate)

            population[i] = child1
            population[i + 1] = child2

            if fitness_values[i] > best_fitness:
                best_solution = child1
                best_fitness = fitness_values[i]

    return best solution, best fitness


def plot_route(coordinates, order_representation):
    x = [point[1] for point in coordinates]
    y = [point[2] for point in coordinates]

    plt.figure(figsize=(8, 6))
    plt.plot(x + [x[0]], y + [y[0]], marker='o', linestyle='-', color='b')
```

```
    plt.scatter(x, y, color='r')

    for i, city_id in enumerate(order_representation):
        plt.text(x[i], y[i], str(city_id), fontsize=8, ha='right',
va='bottom')

    plt.title("Optimized TSP Route")
    plt.xlabel("X Coordinates")
    plt.ylabel("Y Coordinates")
    plt.show()


if __name__ == "__main__":
    tsp_data = read_tsp_file("Dj89.txt")
    crossover_rate = 0.5
    mutation_rate = 0.3
    num_generations = 500
    population_size = 50
    best_solution, best_fitness = genetic_algorithm(num_generations,
population_size, crossover_rate, mutation_rate,
                                                    tsp_data)

    order_representation = [tsp_data[i][0] for i in best_solution]

    print("Best solution order representation:", order_representation)
    print("Best fitness:", best_fitness)

    plot_route(tsp_data, order_representation)
```