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
Решение задачи коммивояжера с помощью генетических 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:
Цель работы заключается в разработке и реализации генетического алгоритма для решения задачи коммивояжера, а также в сравнении полученного решения с оптимальным и анализе влияния параметров алгоритма на время выполнения и точность результата.

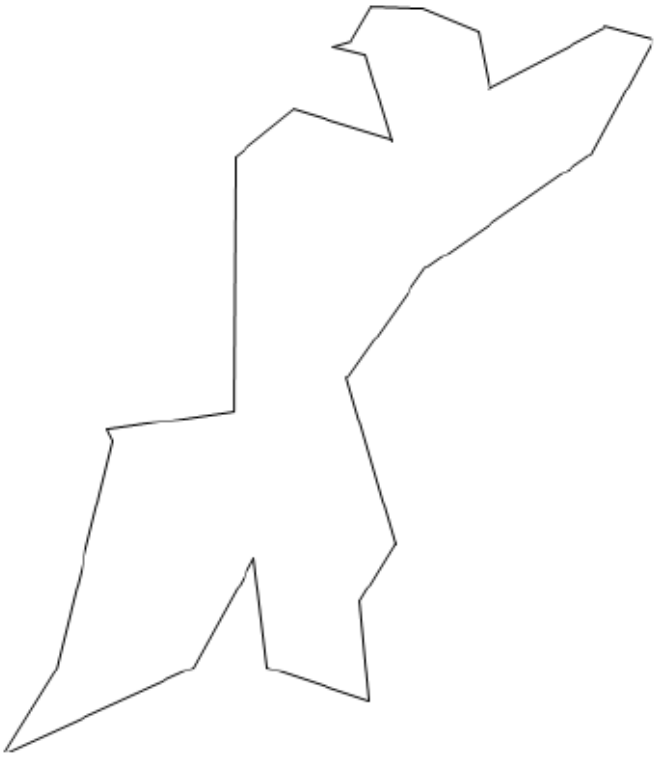
Вариант 17

17	Wi29.tsp

Тип данных: эвклидовы координаты городов

```
1 20833.3333 17100.0000
2 20900.0000 17066.6667
3 21300.0000 13016.6667
4 21600.0000 14150.0000
5 21600.0000 14966.6667
6 21600.0000 16500.0000
7 22183.3333 13133.3333
8 22583.3333 14300.0000
9 22683.3333 12716.6667
10 23616.6667 15866.6667
11 23700.0000 15933.3333
12 23883.3333 14533.3333
13 24166.6667 13250.0000
14 25149.1667 12365.8333
15 26133.3333 14500.0000
16 26150.0000 10550.0000
17 26283.3333 12766.6667
18 26433.3333 13433.3333
19 26550.0000 13850.0000
20 26733.3333 11683.3333
21 27026.1111 13051.9444
22 27096.1111 13415.8333
23 27153.6111 13203.3333
24 27166.6667 9833.3333
25 27233.3333 10450.0000
26 27233.3333 11783.3333
27 27266.6667 10383.3333
28 27433.3333 12400.0000
29 27462.5000 12992.2222
EOF
```

Оптимальное решение wi29:



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.

Выполнение:

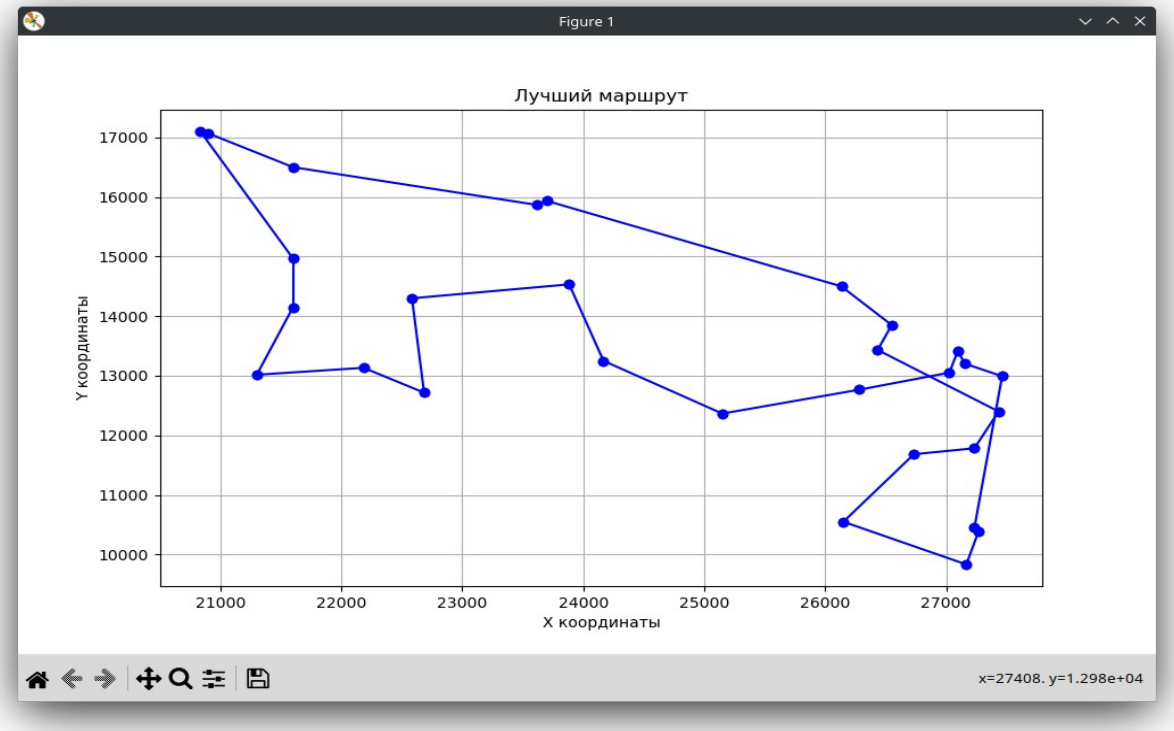
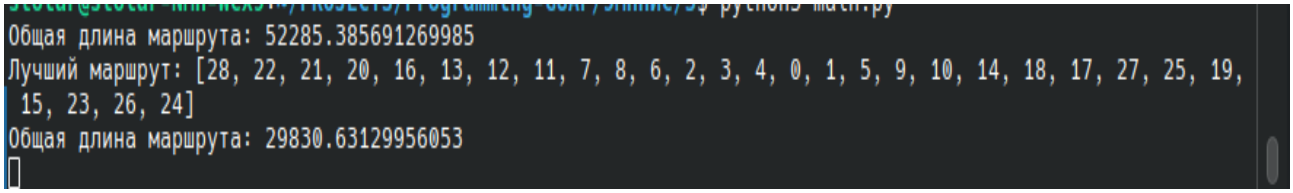


Figure 1
Лучший маршрут
17000
16000
15000
14000
13000
12000
11000
10000
Y координаты
21000
22000
23000
24000
25000
26000
27000
X координаты
x=27408. y=1.298e+04



1. Кроссовер: Изменяя вероятность кроссовера, наблюдалась зависимость между увеличением вероятности и скоростью нахождения решения. Высокая вероятность кроссовера (например, 0.9) обеспечивала более быстрое сходимость к оптимальным маршрутам, однако это также приводило к риску потери генетического разнообразия, что в некоторых случаях снижало точность решения. Низкая вероятность кроссовера (например, 0.5) способствовала сохранению большей генетической информации, но увеличивала время выполнения из-за медленной эволюции популяции.
2. Мутация: аналогично, изменение вероятности мутации оказывало значительное влияние на результаты. Высокая вероятность мутации (например, 0.3) способствовала более высокому уровню разнообразия в популяции, что может помочь избежать локальных минимумов. Однако это также увеличивало время выполнения из-за частых изменений в популяции. Низкая вероятность мутации (например, 0.05) приводила к более стабильным результатам, но иногда не позволяло алгоритму находить лучшие решения.
В целом, для достижения баланса между временем выполнения и точностью нахождения решения важно тщательно подбирать параметры кроссовера и мутации в зависимости от конкретной задачи и структуры данных. Оптимальные настройки могут варьироваться в зависимости от размера задачи и особенностей представленных данных

Выводы:
В результате работы был разработан и реализован генетический алгоритм для решения задачи коммивояжера, который позволил найти приемлемое решение, близкое к оптимальному. Анализ влияния параметров кроссовера и мутации на эффективность алгоритма показал, что оптимальные настройки существенно снижают время выполнения и повышают точность найденных маршрутов, что подтверждает эффективность использования генетических методов в задачах коммивояжера.

Листинг

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

# Координаты городов
cities = np.array([
    [20833.3333, 17100.0000],
    [20900.0000, 17066.6667],
    [21300.0000, 13016.6667],
    [21600.0000, 14150.0000],
    [21600.0000, 14966.6667],
    [21600.0000, 16500.0000],
```

```
    [22183.3333, 13133.3333],
    [22583.3333, 14300.0000],
    [22683.3333, 12716.6667],
    [23616.6667, 15866.6667],
    [23700.0000, 15933.3333],
    [23883.3333, 14533.3333],
    [24166.6667, 13250.0000],
    [25149.1667, 12365.0000],
    [26133.3333, 14500.0000],
    [26150.0000, 10550.0000],
    [26283.3333, 12766.6667],
    [26433.3333, 13433.3333],
    [26550.0000, 13850.0000],
    [26733.3333, 11683.3333],
    [27026.1111, 13051.9444],
    [27096.1111, 13415.8333],
    [27153.6111, 13203.3333],
    [27166.6667, 9833.3333],
    [27233.3333, 10450.0000],
    [27233.3333, 11783.3333],
    [27266.6667, 10383.3333],
    [27433.3333, 12400.0000],
    [27462.5000, 12992.2222]
])

# Функция для расчета расстояния между двумя городами
def distance(city1, city2):
    return np.linalg.norm(city1 - city2)

# Функция для расчета общей длины маршрута
def total_distance(route):
    return sum(distance(cities[route[i]], cities[route[i + 1]]) for i in range(len(route) - 1)) +
distance(cities[route[-1]], cities[route[0]])

length = total_distance([i for i in range(len(cities))])

print(f"Общая длина маршрута: {length}")

# Генерация начальной популяции
def generate_population(size, num_cities):
    return [np.random.permutation(num_cities) for _ in range(size)]

# Функция приспособленности
def fitness(route):
    return 1 / total_distance(route)

# Скрещивание
def crossover(parent1, parent2):
    size = len(parent1)
    start, end = sorted(random.sample(range(size), 2))
    child = [-1] * size
    child[start:end] = parent1[start:end]
```

```
    pointer = 0
    for gene in parent2:
        if gene not in child:
            while child[pointer] != -1:
                pointer += 1
            child[pointer] = gene

    return child

# Мутация
def mutate(route, mutation_rate=0.01):
    for swapped in range(len(route)):
        if random.random() < mutation_rate:
            swap_with = int(random.random() * len(route))
            route[swapped], route[swap_with] = route[swap_with], route[swapped]

# Основной алгоритм
def genetic_algorithm(num_cities, population_size, generations, mutation_rate):
    population = generate_population(population_size, num_cities)

    for _ in range(generations):
        population = sorted(population, key=lambda route: fitness(route), reverse=True)
        next_generation = population[:population_size // 2]  # Сохраняем половину лучших

        while len(next_generation) < population_size:
            parent1, parent2 = random.choices(population[:50], k=2)  # Случайный выбор из лучших
            child = crossover(parent1, parent2)
            mutate(child, mutation_rate)
            next_generation.append(child)

        population = next_generation

    best_route = sorted(population, key=lambda route: fitness(route), reverse=True)[0]
    return best_route, total_distance(best_route)

# Параметры
num_cities = len(cities)
population_size = 500  # Размер популяции
generations = 100  # Количество поколений
mutation_rate = 0.05  # Вероятность мутации

# Запуск генетического алгоритма
best_route, best_distance = genetic_algorithm(num_cities, population_size, generations,
mutation_rate)

# Вывод результатов
print("Лучший маршрут:", best_route)
print("Общая длина маршрута:", best_distance)

# Визуализация маршрута
def plot_route(route):
```

```
    plt.figure(figsize=(10, 6))
    for i in range(len(route) - 1):
        plt.plot([cities[route[i], 0], cities[route[i + 1], 0]],
                 [cities[route[i], 1], cities[route[i + 1], 1]], 'bo-')
    plt.plot([cities[route[-1], 0], cities[route[0], 0]],
             [cities[route[-1], 1], cities[route[0], 1]], 'bo-')  # Замыкаем маршрут
    plt.title("Лучший маршрут")
    plt.xlabel("X координаты")
    plt.ylabel("Y координаты")
    plt.grid()
    plt.show()

# Визуализация лучшего маршрута
plot_route(best_route)
```