ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

д-р техн. наук, 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 курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №		
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Задание

1) Создать программу, использующую МА для решения задачи поиска гамильтонова пути.

2) Представить графически найденное решение. Предусмотреть возможность пошагового просмотра процесса поиска решения.

3) Сравнить найденное решение с представленным в условии задачи оптимальным решением

4) Реализовать с использованием муравьиных алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта

5) Представить графически найденное решение.

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

7) **Вариант 2**

Путь
Dj89.tsp

Теоретические сведения

Муравьиные алгоритмы (МА) основаны на использовании популяции потенциальных решений и разработаны для решения задач комбинаторной оптимизации, прежде всего, поиска различных путей на графах. Кооперация между особями (искусственными муравьями) здесь реализуется на основе моделирования. При этом каждый агент, называемый искусственным муравьем, ищет решение поставленной задачи. Искусственные муравьи последовательно строят решение задачи, передвигаясь по графу, откладывают феромон и при выборе дальнейшего участка пути учитывают концентрацию этого фермента. Чем больше концентрация феромона в последующем участке, тем больше вероятность его выбора.

Описание работы программы.

Используемые операторы в Алгоритме муравьиной оптимизации (АМО):

1. **Инициализация популяции муравьев:**
   - Инициализация начальной популяции муравьев, каждый из которых начинает свой маршрут из случайного города.
2. **Оператор селекции следующего города:**
   - Турнирный отбор из двух городов для определения следующего города в маршруте муравья.
3. **Оператор скрещивания:**
   - Узловой одноточечный кроссовер для формирования маршрута муравья.
4. **Оператор обновления феромонов:**
   - Обновление феромонов на путях в зависимости от качества маршрута муравья.

Параметры АМО:

- **Количество муравьев в популяции:**
  - 50 муравьев.
- **Количество поколений:**
  - 50 поколений.
- **Количество элитных муравьев:**
  - 20 элитных муравьев, используемых для усиления феромонов.
- **Параметры феромонов:**
  - Начальный уровень феромонов: 0.4.
  - Интенсивность феромонов (CONST_Q): 20,000.
- **Параметры феромонного обновления:**
  - Коэффициент испарения (EVAPORATION_AMOUNT): 0.4.

Задача коммивояжера:

- **Города:**
  - 'dj89.txt'.

- **Визуализация:**
  - Визуализация лучшего маршрута на каждой итерации, а также вывод информации о популяции в выбранных поколениях.

- **Вывод результатов:**
  - Отображение лучшего маршрута на каждой итерации, номера поколений, в которых найдены решения, и значения фитнес-функции.

Решение задачи:

- **Лучший маршрут:**
  - Вывод лучшего маршрута и его длины по завершении алгоритма.
- **Лучший маршрут на каждой итерации:**
  - Возможность пользователя просматривать лучшие маршруты на различных итерациях.

Демонстрация работы программы:

Число поколений: 150

Число популяции: 150

Количество элитных муравьев: 20

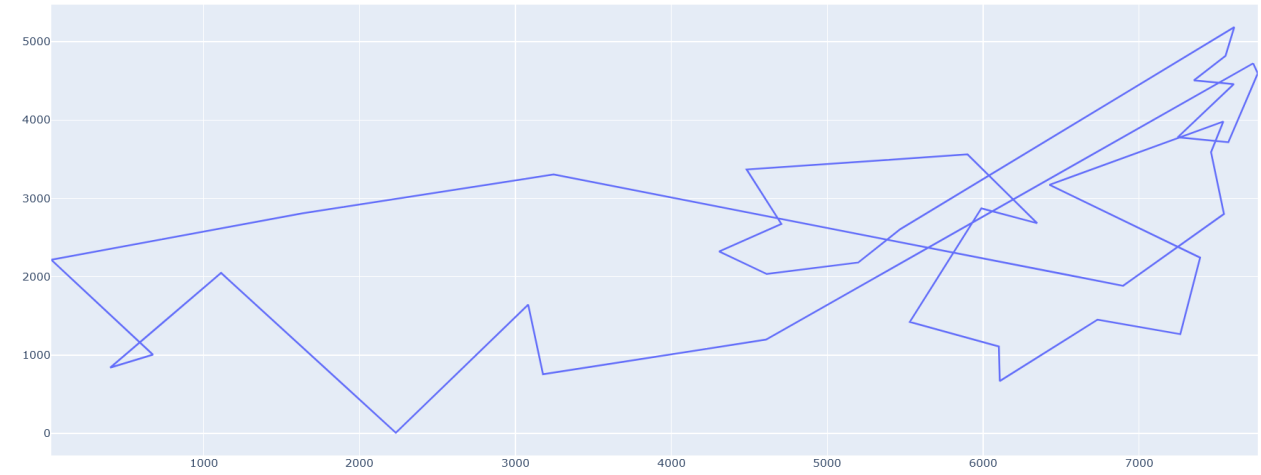

Рис. 1 Начальное поколение

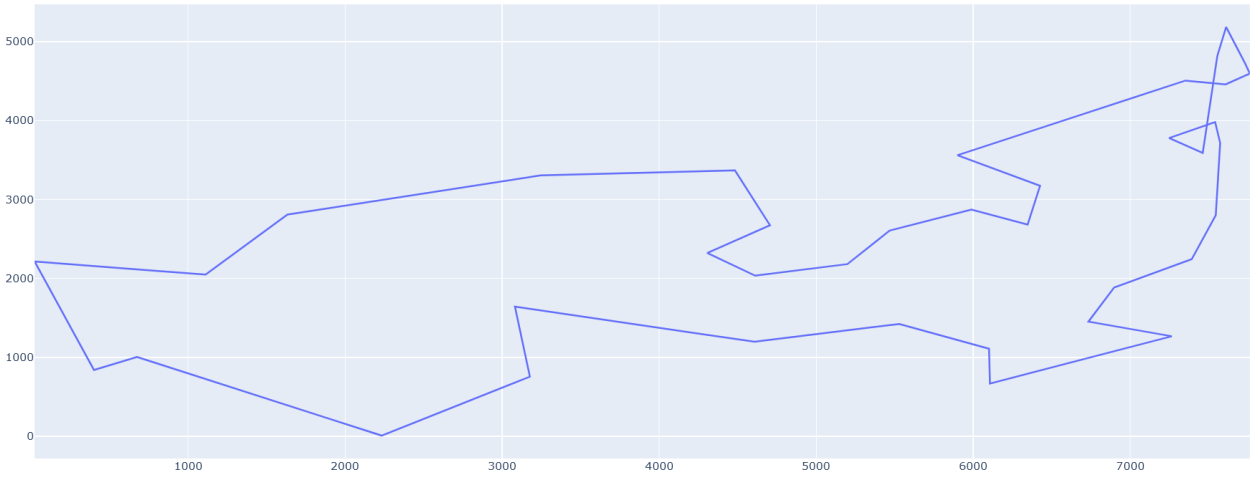

Рис. 2 Конечное поколение

```
Лучший путь:
[19, 37, 6, 28, 36, 18, 7, 31, 38, 9, 1, 8, 16, 22, 3, 34, 5, 29, 2, 26, 4, 35, 10, 24, 32, 21, 13, 25, 14, 23, 11, 12, 15, 33, 20, 30, 27, 17]
Номер поколения, в котором было найдено решение 81
Значение фитнес функции  28213.02925593975
```

Рис. 3 Результат выполнения

Число поколений: 50

Число популяции: 50

Количество элитных муравьев: 10

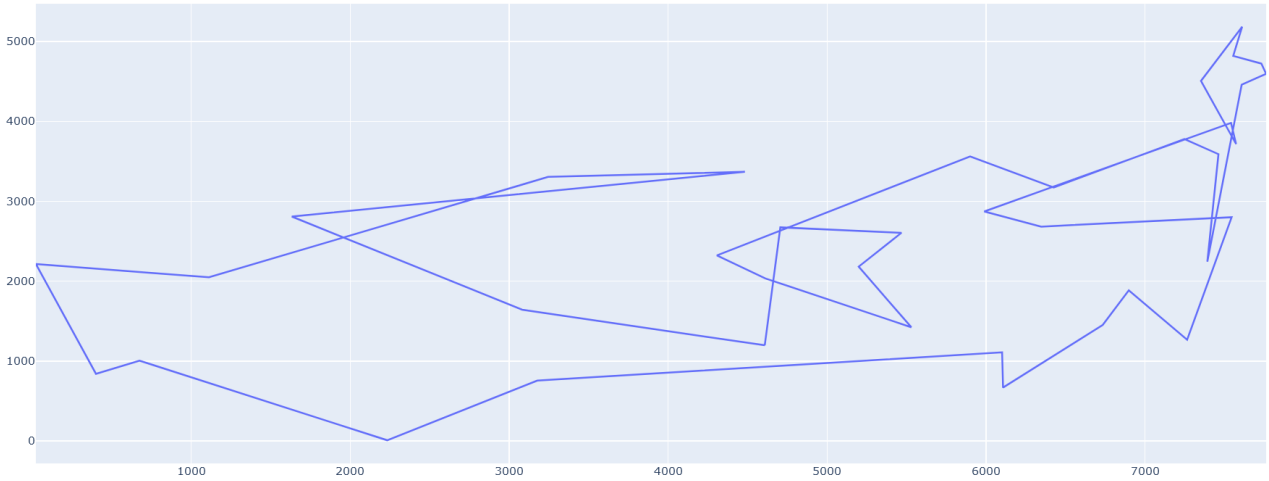

Рис. 4 Начальное поколение

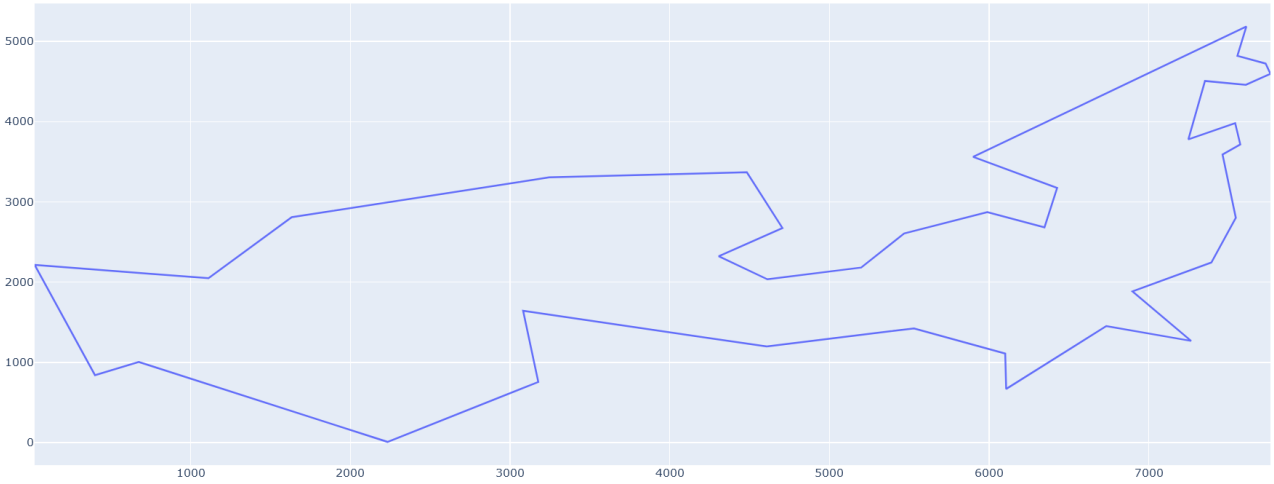

Рис. 5 Конечное поколение

```
Лучший путь:
[34, 3, 22, 16, 1, 8, 9, 38, 31, 18, 7, 28, 6, 37, 19, 27, 17, 30, 36, 20, 33, 15, 12, 11, 23, 14, 25, 13, 21, 32, 24, 10, 35, 4, 26, 2, 29, 5]
Номер поколения, в котором было найдено решение 43
Значение фитнес функции  28117.37847769546
```

Рис. 6 Результат выполнения

Число поколений: 250

Число популяции: 50

Количество элитных муравьев: 10

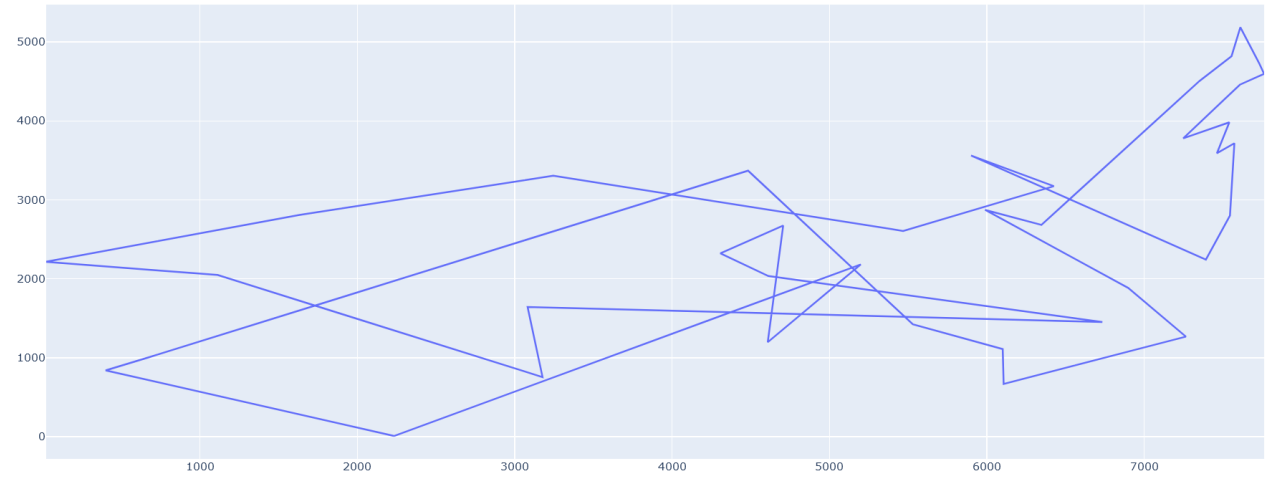

Рис. 7 Начальное поколение

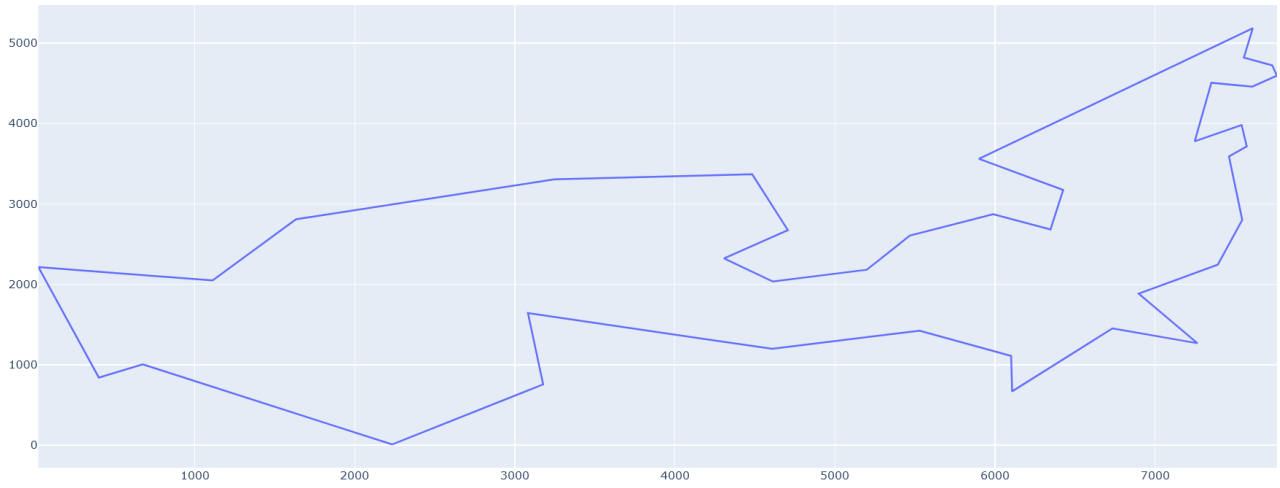

Рис. 8 Конечное поколение

```
Лучший путь:
[2, 29, 5, 34, 3, 22, 16, 1, 8, 9, 38, 31, 18, 7, 28, 6, 37, 19, 17, 27, 30, 36, 20, 33, 15, 12, 11, 23, 14, 25, 13, 21, 32, 24, 10, 35, 4, 26]
Номер поколения, в котором было найдено решение 153
Значение фитнес функции  28039.915632888155
```

Рис. 9 Результат выполнения

Число поколений: 150

Число популяции: 150

Количество элитных муравьев: 20

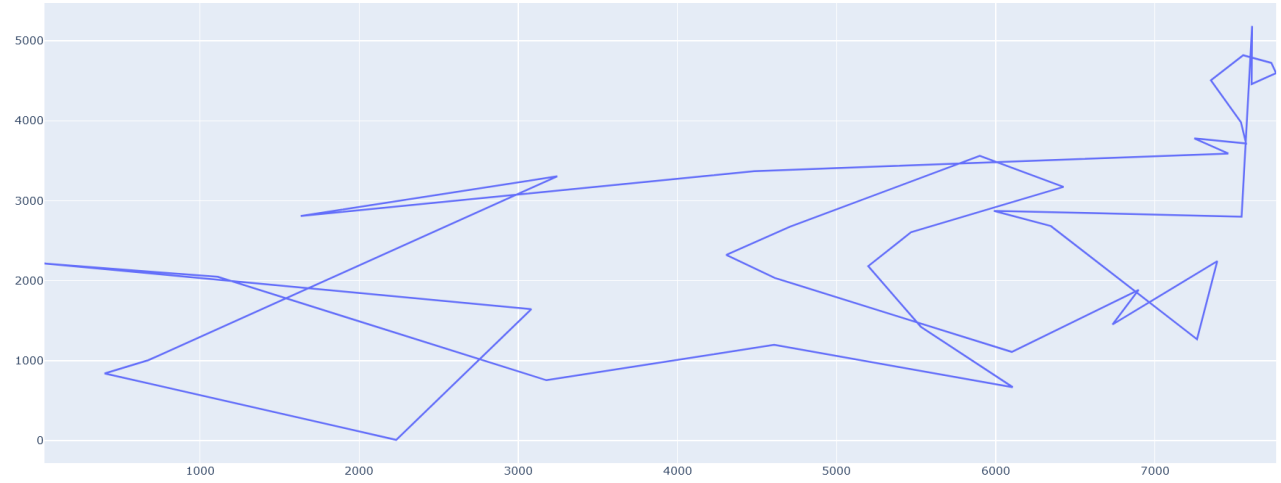

Рис. 10 Начальное поколение

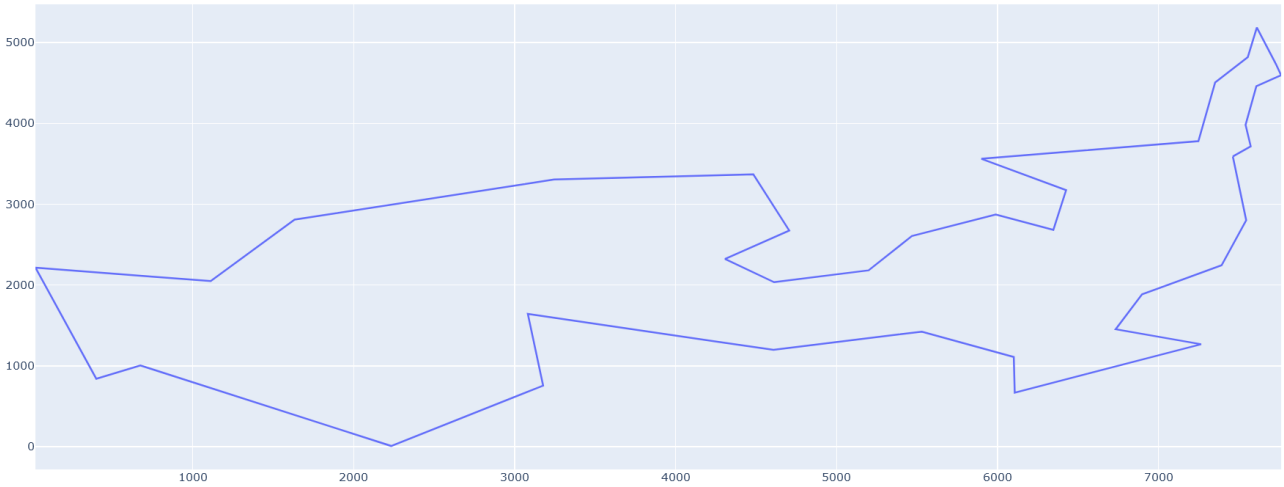

Рис. 11 Конечное поколение

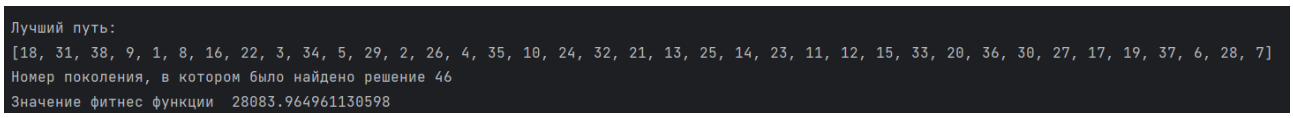

Рис. 12 Результат выполнения

Выводы

Сравнивая полученные значения фитнес-функции полученные во время выполнения 3 лабораторной работы, зависимость времени выполнения от изменения различных входных параметров, можем сделать вывод, что невооруженным глазом видна превосходность МА над ГА в задаче коммивояжера. Так же можем сказать, что время напрямую коррелирует с количеством популяций. А точность решения увеличивается при увеличении количества элитных муравьев.

Листинг программы

```
import numpy as np
import random
import os
from pathlib import Path
import plotly.graph_objects as go

SIZE = 0
TOWNS = list()
PHEROMONES = list()

CONST_Q = 20000
CONST_NEAR = 200
CONST_A = 1.0
CONST_B = 2.0
START_WEIGHT = 0.4

ELITE_ANTS = 20  # элитные муравьи

EVAPORATION_AMOUNT = 0.4

GENERATIONS = 150  # число поколений
POPULATION = 150  # число популяции


class Town(object):
    def __init__(self, number=0, x=0, y=0, phen=0):
        self.number = number  # Fixed: start from 1
        self.x = x
        self.y = y
        self.phen = phen

    def distance(self, town):
        return ((town.x - self.x) ** 2 + (town.y - self.y) ** 2) ** 0.5

    @classmethod
    def clone(cls, town):
        return cls(town.number, town.x, town.y, town.phen)

    def print(self):
        print("Town", self.number, ':', self.x, self.y)


class Ant(object):

    def __init__(self, mode="Spawn", start=None, ant=None):
        self.start = None
        self.fit = 0
        self.tour = list()
        if mode == "Spawn":
            self.start = Town.clone(start)
            self.tour.append(start)
        elif mode == "Clone":
            for i in ant.tour:
                self.tour.append(Town.clone(i))
            self.fit = ant.fit
            self.start = Town.clone(ant.start)

    def findPath(self):
        # Нахождение претендентов
        while len(self.tour) < SIZE:
            aspirants, prob = list(), list()
```

```
            for town in TOWNS:
                if self.findTown(town) == -1:
                    aspirants.append(town)

            if not aspirants:
                break  # Terminate the loop if all towns are visited

            for town in aspirants:
                distance = self.tour[-1].distance(town)
                pheromone = PHEROMONES[self.tour[-1].number - 1][town.number -
1]
                prob.append(pheromone * CONST_A * (CONST_NEAR / distance) **
CONST_B)

            st_mat = sum(prob)
            for k in range(len(prob)):
                prob[k] = prob[k] / st_mat

            r = random.random()
            end = 0
            for j in range(len(prob)):
                end += prob[j]
                if r <= end:
                    self.tour.append(Town.clone(aspirants[j]))
                    break
            print(f"tour[-1].number: {self.tour[-1].number}, town.number:
{town.number}")

    def print(self):
        print("[", end="")
        for i in range(0, len(self.tour)):
            if i == (len(self.tour) - 1):
                print(self.tour[i].number, end="]\n")
            else:
                print(self.tour[i].number, end=", ")

    def fitness(self):
        sum_ = 0
        for i in range(0, len(self.tour)):
            if i == (len(self.tour) - 1):
                sum_ += self.tour[i].distance(self.tour[0])
            else:
                sum_ += self.tour[i].distance(self.tour[i + 1])
        self.fit = sum_
        return sum_

    def clear(self):
        del (self.tour)
        self.tour = list()
        self.tour.append(self.start)

    def findTown(self, town):
        counter = 0
        for i in self.tour:
            if i.number == town.number:
                return counter
            else:
                counter += 1
        return -1


def computePHEROMONES(ants):
    for ant in ants:
        for k in range(0, len(ant.tour) - 1):
```

```
            current_town = ant.tour[k].number - 1  # Adjust for zero-indexing
            next_town = ant.tour[k + 1].number - 1  # Adjust for zero-indexing
            try:
                if 0 <= current_town < SIZE and 0 <= next_town < SIZE:
                    PHEROMONES[current_town][next_town] += CONST_Q / ant.fit
                    PHEROMONES[next_town][current_town] += CONST_Q / ant.fit
                else:
                    print(f"Invalid towns: {current_town + 1}, {next_town + 1}")
                    print(f"Ant tour: {[town.number for town in ant.tour]}")
            except IndexError:
                print(f"IndexError in iteration {k} with towns {current_town +
1} and {next_town + 1}")
                print(f"Ant tour: {[town.number for town in ant.tour]}")
                print(f"Length of PHEROMONES: {len(PHEROMONES)}, Size: {SIZE}")
                raise

        last_town = ant.tour[-1].number - 1  # Adjust for zero-indexing
        first_town = ant.tour[0].number - 1  # Adjust for zero-indexing
        try:
            if 0 <= last_town < SIZE and 0 <= first_town < SIZE:
                PHEROMONES[last_town][first_town] += CONST_Q / ant.fit
                PHEROMONES[first_town][last_town] += CONST_Q / ant.fit
            else:
                print(f"Invalid last and first towns: {last_town + 1},
{first_town + 1}")
                print(f"Ant tour: {[town.number for town in ant.tour]}")
        except IndexError:
            print(f"IndexError for the last and first towns {last_town + 1} and
{first_town + 1}")
            print(f"Ant tour: {[town.number for town in ant.tour]}")
            print(f"Length of PHEROMONES: {len(PHEROMONES)}, Size: {SIZE}")
            raise


def downgrade(PHEROMONES=PHEROMONES):
    for i in range(0, len(PHEROMONES)):
        for j in range(0, len(PHEROMONES[i])):
            PHEROMONES[i][j] *= 1 - EVAPORATION_AMOUNT
            if PHEROMONES[i][j] < START_WEIGHT:
                PHEROMONES[i][j] = START_WEIGHT


def findBestPath(population):
    best = population[0]
    for i in population:
        if best.fit > i.fit:
            best = i
    return best


# MAIN
print(Path.cwd())
path_file = Path(Path.cwd(), 'dj38.txt')
with open(path_file, 'r') as file:
    lines = file.readlines()

# Ищем строку, содержащую DIMENSION
dimension_line = [line for line in lines if line.startswith("DIMENSION")][0]
SIZE = int(dimension_line.split(":")[1].strip())

TOWNS = []

# Ищем начало секции с координатами
coord_section_start = lines.index("NODE_COORD_SECTION\n") + 1
```

```
for line in lines[coord_section_start:]:
    if line.startswith("EOF"):
        break

    parts = line.split()
    number = int(parts[0])
    x = float(parts[1])
    y = float(parts[2])

    TOWNS.append(Town(number, x, y))

for i in range(0, SIZE):
    PHEROMONES.append(list())
    for j in range(SIZE):
        PHEROMONES[-1].append(START_WEIGHT)

start_town = TOWNS[random.randint(0, SIZE - 1)]

ants = [Ant(mode="Spawn", start=start_town) for i in range(0, POPULATION)]
bests = list()
elite, numE = ants[0], 0

for i in range(0, GENERATIONS):
    for ant in ants:
        ant.findPath()
        ant.fitness()

    downgrade(PHEROMONES)
    computePHEROMONES(ants)

    os.system('')
    print(i)
    bests.append(Ant(mode="Clone", ant=findBestPath(ants)))
    print(bests[-1].fit)
    print()

    if bests[-1].fit < elite.fit:
        numE = i
        elite = bests[-1]

    for i in range(0, ELITE_ANTS):
        computePHEROMONES([elite])

    for ant in ants:
        ant.clear()

os.system('')
print("Лучший путь:", end="\n")
elite.print()
print("Номер поколения, в котором было найдено решение", numE)
print("Значение фитнес функции ", elite.fit, end="\n\n")

layout1 = go.Layout(title=go.layout.Title(x=0.5))

print("Введите номер нужного поколения ( 0 :", GENERATIONS - 1, ") или слово
break для выхода из цикла:")
text = input()
while str(text) != "break":
    print()
    if 0 <= int(text) < GENERATIONS:
        print("\nЛучший тур в поколении:")
        path = bests[int(text)]
        path.print()
```

```
        print("Значение фитнес функции:", path.fit)
        x, y = list(), list()
        for town in path.tour:
            x.append(town.x)
            y.append(town.y)
        x.append(path.tour[0].x)
        y.append(path.tour[0].y)
        fig = go.Figure()
        fig = go.Scatter(x=x, y=y, name="V0", legendgroup="V0")
        go.Figure(data=fig, layout=layout1).show()
    print("Введите номер нужного поколения ( 0 -", GENERATIONS - 1, ") или слово
break для выхода из цикла:")
    text = input()
```