**Національний технічний університет України**

**«Київський політехнічний університет імені Ігоря Сікорського»**

**Інститут прикладного системного аналізу**

**Кафедра Системного проектування**

Лабораторна рoбота № 1

з предмету «Штучний інтелект»

на тему: **“Рішення задачі комівояжера за допомогою класичного генетичного алгоритму”**

Виконала:

Студентка 4-го курсу

групи ДА-81

Желєзнова В.С.

Варіант № 10

Київ – 2021

***Мета роботи:***

Розглянути ідею, основні принципи та етапи реалізації генетичного алгоритму на прикладі рішення класичної задачі комівояжера, та оцінити його ефективність.

***Завдання:***

1. Реалізувати генетичний алгоритм для пошуку найкоротшого шляху між містами згідно варіанту.
2. Проаналізувати ефективність алгоритму та його збіжність на основі виконаної роботи.

**Хід роботи:**

1. Напишемо програму для реалізації генетичного алгоритму

В якості мови оберемо python.

Лістинг ключових фрагментів програми з коментарями:

* **Ініціалізація програми та основні етапи**

# введення основних данних

print('Введіть кількість міст:')

num\_cities = int(input())

print('Введіть кількість ітерацій:')

num\_iter = int(input())

print('Введіть відсоток мутації:')

percent\_mutation = int(input())

population\_size = 28

positions = coordinates\_for\_cities(num\_cities)

print('Координати міст =', positions)

population = generate\_individuals(population\_size, num\_cities)

print('Згенерована популяція = ', population)

fig, ax = plt.subplots(figsize=(6, 6))

ax.scatter(np.array(positions[0]), np.array(positions[1]), c='black')

# прорахуємо дистанцію для кожної особини популяції

distances = []

for i in population:

    distances.append(round(path\_length(positions, i), 3))

print('\nДовжина шляху для кожного варіанту', distances)

print('Мінімальна довжина шляху =', min(distances))

print()

iter = 1

while num\_iter > 0:

    print('Ітерація = ', iter)

    iter = iter + 1

    num\_iter = num\_iter - 1

    # зробимо відбір половини популції

    population = delete\_population(population\_size, distances, population)

    population = population + make\_new\_population(population)

    population = mutations(population, percent\_mutation)

    distances = []

    for i in population:

        distances.append(round(path\_length(positions, i), 3))

    print('Довжина шляху для кожного варіанту після кросинговеру та мутації\n', distances)

    print('Мінімальна довжина шляху =', min(distances))

    idx = distances.index(max(distances))

    x, y = [], []

    for i in range(int(num\_cities)):

        x.append(positions[0][population[idx][i]])

        y.append(positions[1][population[idx][i]])

    x.append(x[0])

    y.append(y[0])

    if num\_iter != 0:

        ax.plot(x, y, color=np.random.rand(3, ), alpha=0.75)

    else:

        ax.plot(x, y, 'r', linewidth=4)

    print('------------------------------------------------------------------------------------------\n')

for i in range(0, int(num\_cities), 1):

    ax.annotate(str(i), (positions[0][i] + 0.2, positions[1][i] + 0.2))

plt.show();

* **Селекція**

def choose\_parents(population\_size):

    parents = []

    temp = []

    for i in range(population\_size \* 2):

        temp.append(i)

    for i in range(population\_size \* 2):

        if i not in parents:

            parents.append(i)

            temp.remove(i)

            if len(temp) != 1:

                k = randint(0, len(temp) - 1)

                parents.append(temp[k])

                temp.remove(temp[k])

            else:

                parents.append(temp[0])

    return parents

В результаті процесу селекції створюється батьківська популяція.

* **Кросовер**

На першому етапі схрещування вибираються пари особин з батьківської популяції.

Можна виділити кілька операторів вибору з батьківської популяції:

* Панміксія - двоє батьків вибираються випадково, кожна особина популяції має рівні шанси бути обраною
* Інбридинг – перша батьківська особина обирається випадково, а друга обирається така, котра найбільш схожа на першу
* Аутбридинг - перша батьківська особина обирається випадково, а друга обирається така, котра найменш схожа на першу

def make\_new\_population(population):  # список родителей, список с путями

    new\_population = []

    num\_pop = len(population[0])

    parents = choose\_parents(len(population))

    for i in range(0, len(parents) // 2 - 1, 2):

        slice\_point1 = randint(1, num\_pop - 3)

        slice\_point2 = randint(slice\_point1 + 1, num\_pop - 2)

        ch1 = population[i][:slice\_point1]

        ch2 = population[i + 1][:slice\_point1]

        t1, t2 = population[i][slice\_point2:], population[i + 1][slice\_point2:]

        for j in range(num\_pop):

            if (population[i][(j + slice\_point1) % num\_pop] not in ch2):

                if (population[i][(j + slice\_point1) % num\_pop] not in t2):

                    ch2.append(population[i][(j + slice\_point1) % num\_pop])

            if (population[i + 1][(j + slice\_point1) % num\_pop] not in ch1):

                if (population[i + 1][(j + slice\_point1) % num\_pop] not in t1):

                    ch1.append(population[i + 1][(j + slice\_point1) % num\_pop])

        ch1 = ch1 + population[i][slice\_point2:]

        ch2 = ch2 + population[i + 1][slice\_point2:]

        new\_population.append(ch1)

        new\_population.append(ch2)

    return new\_population

При схрещуванні двох обраних особин (маршрутів) в результаті обміну хромосомами (ділянками маршруту) формуються два дочірніх маршрути (нащадки). Дані дочірні маршрути формуються за допомогою циклічного кросоверу. Для наглядності продемонструємо діаграму роботи алгортиму даного кросоверу:

Спочатку випадковим чином виявляємо ген(точка з якої почнеться схрещування);

* Мутація

Після того, як закінчиться стадія кросовера, виконуються оператори мутації. Популяція, отримана після мутації записує поверх старої і цим цикл одного покоління завершується.

def mutations(population, percent\_mutation):

    count\_mutation = (int(len(population)) \* int(percent\_mutation)) // 100

    for i in range(count\_mutation):

        mut\_population = randint(0, len(population) - 1)

        point1 = randint(0, len(population[0]) - 1)

        point2 = randint(0, len(population[0]) - 1)

        temp = population[mut\_population][point1]

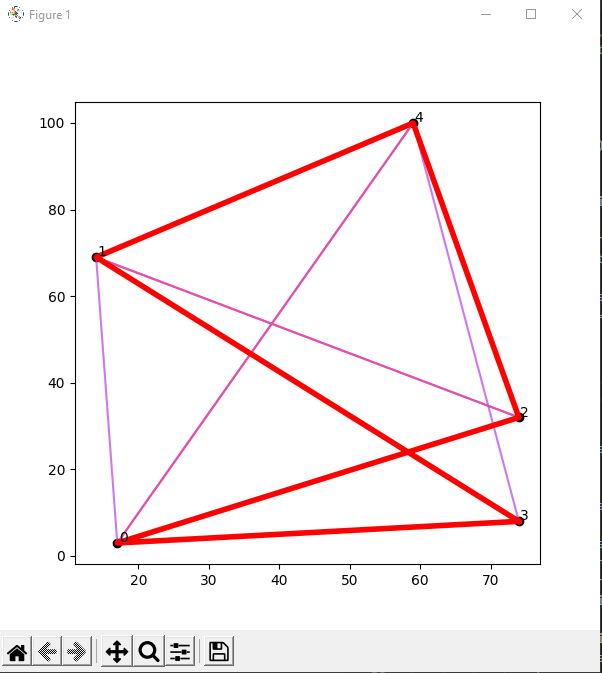
        population[mut\_population][point1] = population[mut\_population][point2]

        population[mut\_population][point2] = temp

    return population

Наступні покоління обробляються таким же чином: відбір, кросовер і мутація.

1. Результати роботи програми



**Висновок:** В ході виконання даної лабораторної роботи було проведено ознайомлення з генетичним алгоритмом на прикладі задачі комівояжера. В результаті була розроблена програма генетичного алгоритму з графічним інтерфейсом.