**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”  
Інститут прикладного системного аналізу  
Кафедра системного проектування**

**ЗВІТ**

з виконання лабораторної роботи №1  
з дисципліни “Методи та системи штучного інтелекту”  
на тему: “Рішення задачі комівояжера за допомогою класичного генетичного алгоритму”

Виконав:

Студент 4 курсу групи ДА-82

Муравльов А. Д.

Варіант №18

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.

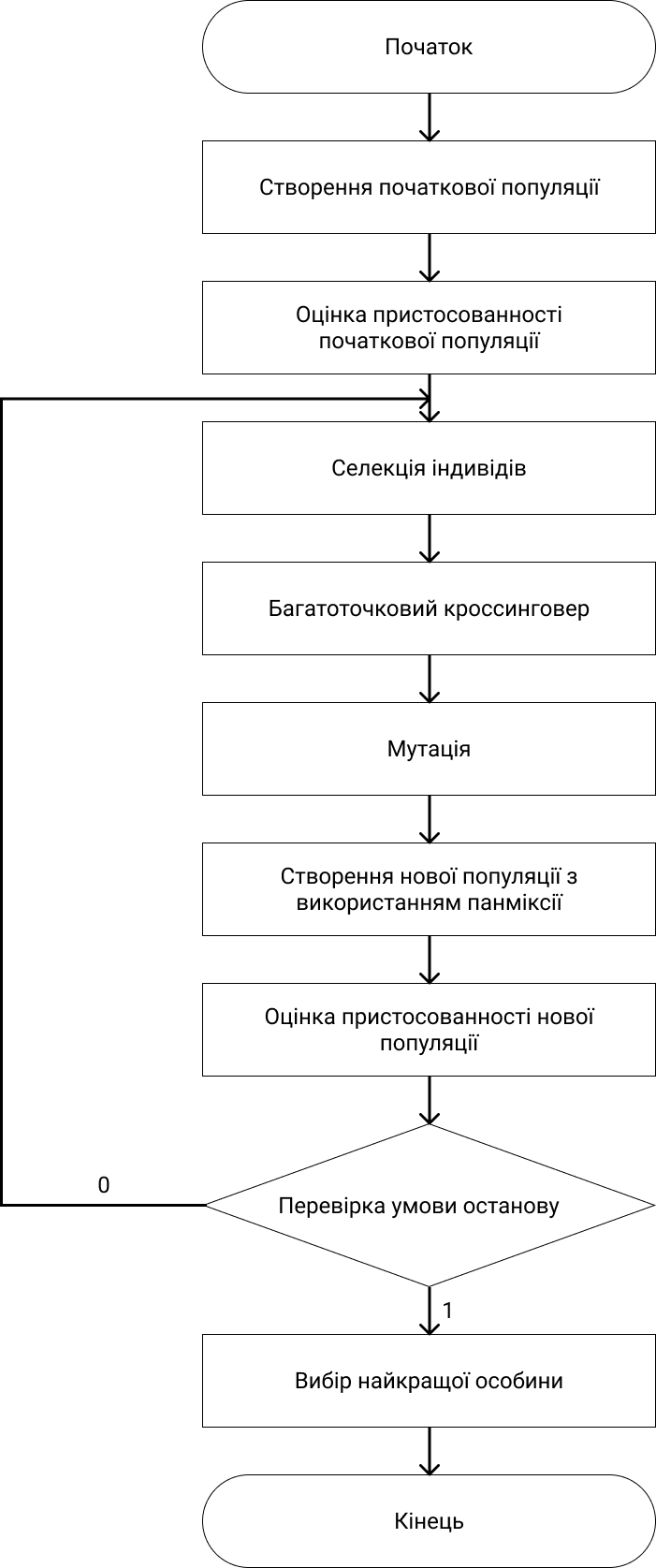
2. Проаналізувати ефективність алгоритму та його збіжність на основі виконаної роботи.

|  |  |  |
| --- | --- | --- |
| **№ Варіанта** | **Тип кросоверу** | **Оператор вибору батьківської пари** |
| 8 | багатоточковий | панміксія |

# **Хід роботи**

***Напишемо програму для реалізації генетичного алгоритму.***  
Програма розроблена у середовищі Visual Studio Code, з використанням плагіну jupyter notebook. У якості мови програмування обрано python 3.10.

***Блок схема генетичного алгоритму з урахуванням типу оператору вибору батьків (згідно варіанту), виду кросоверу та мутації:***



***Хід написання програми та лістинг***

Підхід до написання програми, та генетичний алгоритм в контексті задачі комівояжера:

* Ген: місто, що задається координатами (х,у)
* Індивід (хромосома): один шлях, що задовольняє умови задачі комівояжера
* Популяція: набір можливих шляхів
* Батьки: два шляхи що поєднуються для створення нового
* Mating pool: колекція батьків, що використовуються для створення наступної популяції
* Пристосованість: функція, що визначає наскільки короткий кожен шлях
* Мутація: метод для внесення варіацій до популяції
* Елітизм: метод відбору найкращих індивідів до наступного покоління

Створення початкової популяції:

def create\_starting\_population(size,Number\_of\_city):

    '''Method create starting population

    size= No. of the city

    Number\_of\_city= Total No. of the city

    '''

    population = []

    for i in range(0,size):

        population.append(create\_new\_member(Number\_of\_city))

    return population

Оцінка пристосованості популяції:

def fitness(route,CityList):

    '''Individual fitness of the routes is calculated here

    route= 1d array

    CityList = List of the cities

    '''

*#Calculate the fitness and return it.*

    score=0

*#N\_=len(route)*

    for i in range(1,len(route)):

        k=int(route[i-1])

        l=int(route[i])

        score = score + distance(CityList[k],CityList[l])

    return score

Селекція індивідів:

def selection(popRanked, eliteSize):

    selectionResults=[]

    result=[]

    for i in popRanked:

        result.append(i[0])

    for i in range(0,eliteSize):

        selectionResults.append(result[i])

    return selectionResults

Кроссинговер:

def crossover(a,b):

    '''

    cross over

    a=route1

    b=route2

    return child

    '''

    child1=[]

    child2=[]

    childA=[]

    childB=[]

    childC=[]

    childD=[]

    child1\_2=[]

    child2\_2=[]

    childA2=[]

    childB2=[]

    childC2=[]

    childD2=[]

    geneA=int(random.random()\* len(a))

    geneB=int(random.random()\* len(a))

    start\_gene=min(geneA,geneB)

    end\_gene=max(geneA,geneB)

    for i in range(start\_gene,end\_gene):

        childA.append(a[i])

    for i in range(start\_gene,end\_gene):

        childC.append(b[i])

    childB=[item for item in a if item not in childA]

    childD=[item for item in b if item not in childC]

    child1=childA+childB

    child2=childC+childD

*#second*

    gene1 = int(random.random()\* (len(a)-end\_gene))

    gene2 = int(random.random()\* (len(a)-end\_gene))

    start\_gene2=min(gene1,gene2)+end\_gene

    end\_gene2=max(gene1,gene2)+end\_gene

    for i in range(start\_gene2,end\_gene2):

        childA2.append(child1[i])

    for i in range(start\_gene2,end\_gene2):

        childC2.append(child2[i])

    childB2=[item for item in child1 if item not in childA2]

    childD2=[item for item in child2 if item not in childC2]

    child1\_2=childA2+childB2

    child2\_2=childC2+childD2

    return (child1\_2, child2\_2)

Мутація:

def mutate(route,probablity):

    '''

    mutation

    route= 1d array

    probablity= mutation probablity

    '''

*#for mutating shuffling of the nodes is used*

    route=np.array(route)

    for swaping\_p in range(len(route)):

        if(random.random() < probablity):

            swapedWith = np.random.randint(0,len(route))

            temp1=route[swaping\_p]

            temp2=route[swapedWith]

            route[swapedWith]=temp1

            route[swaping\_p]=temp2

    return route

Створення нової популяції:

def next\_generation(City\_List,current\_population,mutation\_rate,elite\_size):

    population\_rank=rankRoutes(current\_population,City\_List)

*#print(f"population rank : {population\_rank}")*

    selection\_result=selection(population\_rank,elite\_size)

*#print(f"selection results {selection\_result}")*

    mating\_pool=matingPool(current\_population,selection\_result)

*#print(f"mating pool {mating\_pool}")*

    children=breedPopulation(mating\_pool)

*#print(f"childern {children}")*

    next\_generation=mutatePopulation(children,mutation\_rate)

*#print(f"next\_generation {next\_generation}")*

    return next\_generation

Перевірка точки останову:

def genetic\_algorithm(City\_List,size\_population=1000,elite\_size=70,mutation\_Rate=0.01,generation=2000):

    '''size\_population = 1000(default) Size of population

        elite\_size = 75 (default) No. of best route to choose

        mutation\_Rate = 0.05 (default) probablity of Mutation rate [0,1]

        generation = 2000 (default) No. of generation

    '''

    pop=[]

    progress = []

    Number\_of\_cities=len(City\_List)

    population=create\_starting\_population(size\_population,Number\_of\_cities)

    progress.append(rankRoutes(population,City\_List)[0][1])

    print(f"initial route distance {progress[0]}")

    print(f"initial route {population[0]}")

    for i in range(0,generation):

        pop = next\_generation(City\_List,population,mutation\_Rate,elite\_size)

        progress.append(rankRoutes(pop,City\_List)[0][1])

    rank\_=rankRoutes(pop,City\_List)[0]

    print(f"Best Route :{pop[rank\_[0]]} ")

    print(f"best route distance {rank\_[1]}")

    plt.plot(progress)

    plt.ylabel('Distance')

    plt.xlabel('Generation')

    plt.show()

    return rank\_, pop

Вибір найкращої особини:

def rankRoutes(population,City\_List):

    fitnessResults = {}

    for i in range(0,len(population)):

        fitnessResults[i] = fitness(population[i],City\_List)

    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = False)

***Результати роботи програми:***

