Міністерство освіти і науки України  
Національний технічний університет України

«Київський політехнічний інститут  ім. І. Сікорського»

Кафедра інженерії програмного забезпечення в енергетиці

Практична робота № 5

з курсу: «Проектування кібер-фізичних систем»

**Виконав:**студент 2-го курсу,  
групи ТВ-32  
Щур Роман Ігорович

 Посилання на GitHub репозиторій:

https://github.com/RomanSchur/DCPS\_PR-5

Київ 2025

**Завдання**

**Завдання**: Запрограмувати вирішення обраної задачі за допомогою генетичного алгоритму. Дати можливість обирати кількість та якість початкової популяції чи декількох популяцій. Кількість ітерацій має теж задаватись, включно з опцією “до останнього живого”. Виводити статистику популяцій після кожної ітерації.

Вид генетичного алгоритму, який треба застосувати:

2.1) Елітний генетичний алгоритм (Elitist Genetic Algorithm): Використовує додатковий елітарний механізм, що дозволяє зберігати найкращі особини у кожній популяції.

**Програмна реалізація**

Програма написана мовою програмування Python. Для кращого розуміння і кращої реалізації генетичного алгоритму, кожен крок генетичного алгоритму (**ініціалізація популяції, оцінка придатності, вибір батьків, створення нащадків, вибір для наступної популяції, перевірка на умову зупинки**) опрацьовано в окремій функції. Розроблений генетичний алгоритм шукає найкращий набір генів для хромосоми.

Популяція представляє собою набір хромосом (списків), які складаються з генів (кожен ген - це число, яке генерується випадковим чином в певному діапазоні). За створення початкової популяції відповідає функція create\_initial\_population, вона, використовуючи цикли for \_ in та бібліотеку random, заповнює кожну хромосому випадковим набором генів.

Після створення популяції для кожної хромосоми обчислюється значення придатності (наскільки сума генів хромосоми приближена до ідеального значення). За це завдання відповідає функція calculate\_chromosom\_suitability. Вона приймає популяцію, створює список значень придатності, і за допомогою циклу обчислює суму генів для кожної хромосоми, потім обчислює значення різниці між цією сумою і цільовою сумою, і додає це значення в список придатності.

Функція sort\_population об'єднує список хромосом і список значень придатності в список пар (значення\_придатності, хромосома) за допомогою list, сортує його за зростанням придатності, розділяє список пар знову на два вже відсортованих списки і повертає список хромосом (від найкращої до найгіршої). Це зроблено для того, щоб в наступне покоління переходила певна кількість найкращих хромосом, адже я використав елітний генетичний алгоритм.

Після знаходження найкращих хромосом і додавання їх у нову популяцію починається процес утворення нових хромосом (нащадків). Для цього функція random\_choice вибирає двох випадкових батьків з попередньої популяції, після чого ці батьки передаються у функцію crossover (функція, яка утворює нові хромосоми). Я обрав одноточковий метод схрещування, тобто функція випадково створює точку поділу і ділить дві батьківські хромосоми по цій точці, і обмінює половинки між собою, тим самим утворюючи дві нові хромосоми. Саме такими хромосомами доповнюється нове покоління.

Також є функція mutation, яка відповідає за мутацію хромосоми. Вона приймає хромосому, генерує випадкове число в діапазоні мутації, і якщо число дорівнює числу мутації, функція випадково вибирає ген у хромосомі і замінює його на випадково згенерований. Через цю функцію проходить кожна новостворена хромосома.

Також є функція limit\_check, яка перевіряє, чи не досягнуто ліміт кількості популяцій.

Результатом програми є вивід найкращих хромосом і їх значень придатності для кожного покоління, а також створення списку значень придатності, на основі якого будується графік функцією create\_graf, який зображує, як змінювалися значення кращих хромосом.

Також створено тести для двох функцій, а саме test\_calculate\_chromosom\_suitability і test\_sort\_population, які мають на меті перевірити коректність роботи коду.

Також зазначу,що параметри генетичного алгоритму задаються константо і можуть змінюватись.

import random  
import matplotlib.pyplot as plt  
POPULATION\_SIZE = 10 # розмір популяції  
MAX\_POPULATIONS = 15 # максимальна кількість популяцій  
  
CHROMOSOME\_LENGTH = 6 # довжина хромосоми  
GENE\_MIN = 1 # мінімальне значення гена  
GENE\_MAX = 10 # максимальне значення гена  
ELITE\_SIZE = 2 # розмір еліти  
  
MUTATION\_CHANCE = 3 # шанс мутації  
MUTATION\_MIN = 1#нижній діапазон шансу мутацій  
MUTATION\_MAX = 5#верхній діапазон шансу мутацій  
  
def create\_initial\_population():#функція яка створює початкову популяцію  
 start\_population=[]  
 for \_ in range(POPULATION\_SIZE):  
 chromosom=[]  
 for \_ in range(CHROMOSOME\_LENGTH):  
 generate\_chromosom=random.randint(GENE\_MIN,GENE\_MAX)  
 chromosom.append(generate\_chromosom)  
 start\_population.append(chromosom)  
 #for chromosom in start\_population:  
 #print(chromosom)  
 return start\_population  
  
def calculate\_chromosom\_suitability (start\_population):  
 res\_suitability=[]#список в якому зберігається значення придатності для кожного представника(хромосоми)  
 for chromosom in start\_population:  
 suitability = abs(60-sum(chromosom))  
 #print("Придатність для хромосоми: ",suitability)  
 res\_suitability.append(suitability)  
 return res\_suitability  
  
def random\_choice (population):#функця для рандомного вибору батьків для наступного покоління  
 Сhoice=random.choice(population)  
 return Сhoice  
  
def crossover (chromosom1,chromosom2):#створення нащадків  
 dividing\_point=random.randint(1,CHROMOSOME\_LENGTH)#точка ділення  
 child1=chromosom1[:dividing\_point]+chromosom2[dividing\_point:]  
 child2 = chromosom1[dividing\_point:] + chromosom2[:dividing\_point]  
 return child1, child2  
def mutation(chromosom):  
 chance=random.randint(1,5)  
 if chance == MUTATION\_CHANCE:  
 gene = random.randint(0, CHROMOSOME\_LENGTH - 1) # індекс гена  
 chromosom[gene] = random.randint(GENE\_MIN, GENE\_MAX) # значення гена  
 return chromosom

Табл.№1 – Програмний код

def sort\_population(population, fitness\_values):  
 paired\_data = list(zip(fitness\_values, population))#список пар  
 paired\_data.sort()  
 sorted\_fitness\_values, sorted\_population = zip(\*paired\_data) #ділмо на два масиви  
 return list(sorted\_population)  
def select\_next\_generation(population, fitness\_values):  
 sorted\_population = sort\_population(population, fitness\_values)  
 elites = sorted\_population[:ELITE\_SIZE]  
 next\_generation = elites.copy()  
 while len(next\_generation) < POPULATION\_SIZE:  
 parent1 = random\_choice(population)  
 parent2 = random\_choice(population)  
 child1, child2 = crossover(parent1, parent2)  
 child1 = mutation(child1)  
 child2 = mutation(child2)  
 if len(next\_generation) < POPULATION\_SIZE:  
 next\_generation.append(child1)  
 if len(next\_generation) < POPULATION\_SIZE:  
 next\_generation.append(child2)  
 return next\_generation  
def limit\_check(numpop):  
 if numpop<MAX\_POPULATIONS:  
 return 1  
 else:  
 return 2  
def create\_graf(fit\_values):  
 populations = range(len(fit\_values))  
 plt.plot(fit\_values)  
 plt.xlabel("Популяція")  
 plt.ylabel("Найкраща придатність")  
 plt.title("Зміна найкращої придатності з часом")  
 plt.xticks(populations)  
 plt.show()  
def main():  
 population = create\_initial\_population()  
 num\_populations = 0  
 best\_fitness\_values = []  
 while limit\_check(num\_populations) == 1:  
 fitness\_values = calculate\_chromosom\_suitability(population)  
 best\_index = fitness\_values.index(min(fitness\_values))  
 best\_fitness = fitness\_values[best\_index]  
 best\_fitness\_values.append(best\_fitness)  
 print(f"Популяція {num\_populations}, Придатність: {fitness\_values[best\_index]}, Хромосома: {population[best\_index]}")  
 population = select\_next\_generation(population, fitness\_values)  
  
 num\_populations += 1  
 create\_graf(best\_fitness\_values)  
  
#main()

Табл.№2 – Програмний код тестів

import unittest  
from Pr5 import calculate\_chromosom\_suitability,sort\_population  
class Test(unittest.TestCase):  
 def test\_calculate\_chromosom\_suitability(self):  
 test\_population = [[10, 10, 10, 10, 10, 10],  
 [5, 5, 5, 5, 5, 5],  
 [9, 9, 9, 9, 9, 9],  
 [1, 1, 1, 1, 1, 1]]  
 expected\_results = [0, 30, 6, 54]  
 actual\_results = calculate\_chromosom\_suitability(test\_population)  
 self.assertEqual(actual\_results, expected\_results)  
  
 def test\_sort\_population(self):  
 population = [[1,2], [3,4], [5,6]]  
 fitness = [10, 5, 0]  
 sorted\_pop = sort\_population(population, fitness)  
 self.assertEqual(sorted\_pop[0], [5,6])

**Результати виконання**

Зображення, що містить текст, знімок екрана, дизайн, типографія

Вміст, створений ШІ, може бути неправильним.

Рис №1 – Приклад результат генетичного алгоритму

Зображення, що містить текст, схема, ряд, Графік

Вміст, створений ШІ, може бути неправильним.

Рис №2 – Графік алгоритму