міністерство освіти і науки України

**Прикарпатський національний університет**

**імені В.Стефаника**

*Факультет математики та інформатики*

*Кафедра інформаційних технологій*

*Програмування вбудованих систем*

Лабораторна робота № 10

Варіант : **21**

Виконав: ***Романів Ю.І.***

Група ІПЗ-23

Дата: 15 травня 2024р.

Викладач: Лазарович І.М.

Івано-Франківськ - 2023

**Тема:** ‘Застосування генетичних алгоритмів для вирішення задач оптимізації

**Мета роботи:** навчитися розв’язувати задачі оптимізації за допомогою генетичного алгоритму, навчитися програмно реалізовувати генетичний алгоритм.

**Завдання**

Використовуючи будь-яку мову програмування розробити програму для реалізації генетичного алгоритму для розв’язування задачі знаходження максимуму функції. Інтерфейс програми повинен дозволяти вводити такі дані: кількість хромосом, вибір умови зупинки роботи алгоритму – після досягнення заданої точності або після досягнення кількості ітерацій, ймовірність схрещування, ймовірність мутації. Задачу вирішити із точністю 1 знак після коми. Результатом роботи програми повинне бути значення всіх індивідумів у популяції, фенотип, значення функції відповідності для кожного індивідума, середнє значення функції відповідності (див. приклад). Вивести кількість поколінь. При захисті лабораторної роботи студент повинен знати генетичний алгоритм, вміти зробити необхідні зміни у тексті програми, пояснити зміст отриманих результатів.

****

**Code**

import random  
import math  
  
def fitness\_func(x):  
 return x \*\* 3 + math.cos(x) \* math.sqrt(x) - math.sqrt(x \*\* 3 - 7 \* math.sqrt(x))  
  
  
def init\_population(pop\_size, chrom\_len):  
 population = []  
 for \_ in range(pop\_size):  
 chromosome = [random.randint(0, 1) for \_ in range(chrom\_len)]  
 population.append(chromosome)  
 return population  
  
  
def decode(chromosome, lb, ub):  
 chrom\_len = len(chromosome)  
 value = 0  
  
 # print(chromosome, "---")  
 for i, bit in enumerate(chromosome):  
 value += bit \* 2 \*\* (chrom\_len - i - 1)  
 # print(value, " val " , lb + (ub - lb) \* value / (2 \*\* chrom\_len - 1))  
 return lb + (ub - lb) \* value / (2 \*\* chrom\_len - 1)  
  
  
def evaluate(population, lb, ub):  
 fitness = []  
 for chromosome in population:  
 value = decode(chromosome, lb, ub)  
 fitness.append(fitness\_func(value))  
 return fitness  
  
  
def roulette\_selection(population, fitness):  
 total\_fitness = sum(fitness)  
 if len(fitness) > 0:  
 avg = total\_fitness / len(fitness)  
 else:  
 avg = 0  
 # probabilities = [f / total\_fitness for f in fitness]  
 # cumulative\_probs = [sum(probabilities[:i + 1]) for i in range(len(probabilities))]  
 parents = []  
 for i, p in enumerate(fitness):  
 if avg <= p:  
 parents.append(population[i])  
 if len(parents) % 2 == 1:  
 parents.pop()  
  
 return parents  
  
  
def crossover(parents, crossover\_rate):  
 offspring = []  
 for i in range(0, len(parents), 2):  
 if random.random() < crossover\_rate:  
 parent1 = parents[i]  
 parent2 = parents[i + 1]  
 crossover\_point = random.randint(1, len(parent1) - 1)  
 child1 = parent1[:crossover\_point] + parent2[crossover\_point:]  
 child2 = parent2[:crossover\_point] + parent1[crossover\_point:]  
 offspring.append(child1)  
 offspring.append(child2)  
  
 crossover\_point = random.randint(1, len(parent1) - 1)  
 child3 = parent1[:crossover\_point] + parent2[crossover\_point:]  
 child4 = parent2[:crossover\_point] + parent1[crossover\_point:]  
 offspring.append(child3)  
 offspring.append(child4)  
 else:  
 offspring.append(parents[i])  
 offspring.append(parents[i + 1])  
  
 while len(offspring) > pop\_size:  
 offspring.pop()  
  
 return offspring  
  
  
def mutate(population, mutation\_rate):  
 for chromosome in population:  
 for i in range(len(chromosome)):  
 if random.random() < mutation\_rate:  
 chromosome[i] = 1 - chromosome[i]  
 return population  
  
  
def genetic\_algorithm(pop\_size, chrom\_len, lb, ub, crossover\_rate, mutation\_rate, max\_gens, target\_precision):  
 counter = 0  
 population = init\_population(pop\_size, chrom\_len)  
 best\_fitness = float('-inf')  
 best\_chromosome = None  
 best\_value = None  
 generation = 0  
  
 while generation < max\_gens:  
 fitness\_values = evaluate(population, lb, ub)  
 if (len(fitness\_values)):  
 avg\_fitness = sum(fitness\_values) / len(fitness\_values)  
  
 print(f"Generation {generation}: Average fitness = {avg\_fitness:.2f} best\_fit {best\_fitness}")  
  
 for i, chromosome in enumerate(population):  
 value = decode(chromosome, lb, ub)  
 fitness = fitness\_func(value)  
 if fitness > best\_fitness:  
 best\_fitness = fitness  
 best\_chromosome = chromosome  
 best\_value = value  
  
 counter +=1  
 if counter > 20:  
 break  
 else:  
 counter = 0  
  
  
 parents = roulette\_selection(population, fitness\_values)  
 offspring = crossover(parents, crossover\_rate)  
 population = mutate(offspring, mutation\_rate)  
 generation += 1  
  
 print(f"\nbest solution: Chromosome = {best\_chromosome}, Value = {best\_value:.2f}, Fitness = {best\_fitness:.2f}")  
 return best\_value, best\_fitness, generation + 1  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pop\_size = 15  
 chrom\_len = 15  
 lb = 15  
 ub = 24  
 crossover\_rate = 0.8  
 mutation\_rate = 0.1  
 max\_gens = 10000  
 target\_precision = 0.1  
  
 best\_value, best\_fitness, generations = genetic\_algorithm(pop\_size, chrom\_len, lb, ub, crossover\_rate,  
 mutation\_rate, max\_gens, target\_precision)  
 print(f"n of generations: {generations}")

**пояснення стартових змінних**

pop\_size: Розмір популяції, тобто кількість хромосом (індивідів) у кожному поколінні.

chrom\_len: Довжина хромосоми, тобто кількість бітів у бінарному рядку, що представляє хромосому.

lb: Нижня межа діапазону для декодування хромосом у числові значення.

ub: Верхня межа діапазону для декодування хромосом у числові значення.

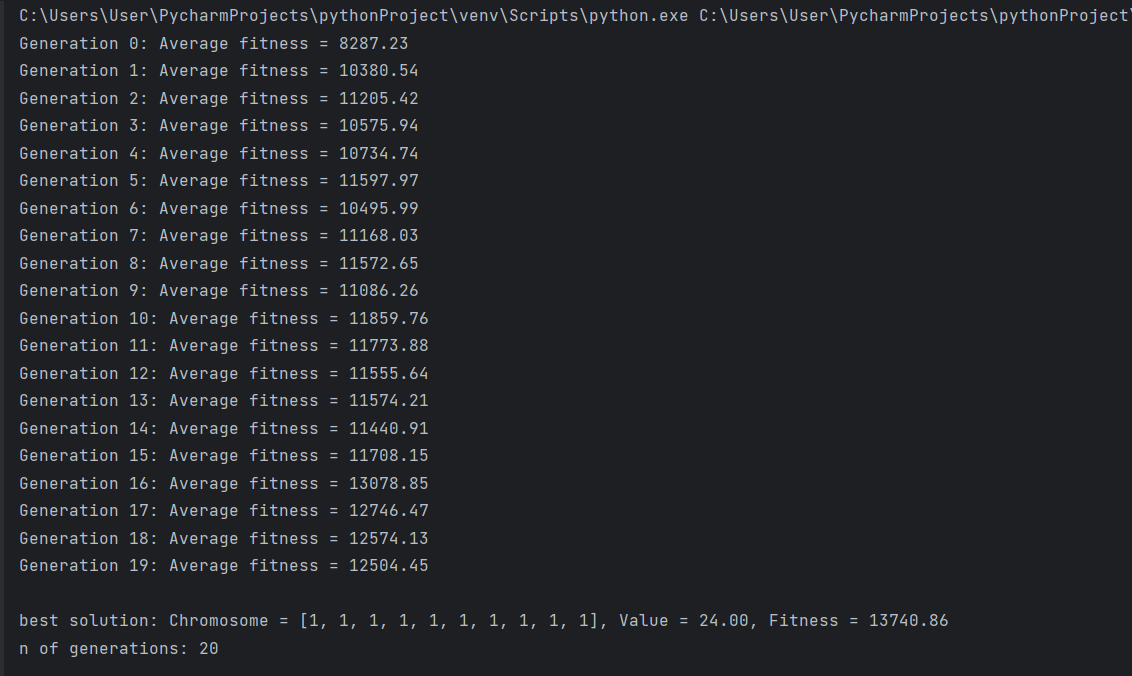
crossover\_rate: Ймовірність схрещування (обміну частинами) між двома хромосомами при створенні нового покоління.

mutation\_rate: Ймовірність мутації (зміни значення біта з 0 на 1 або навпаки) для кожного біта в кожній хромосомі при створенні нового покоління.

max\_gens: Максимальна кількість поколінь, які будуть згенеровані генетичним алгоритмом.

target\_precision: Задана точність для зупинки генетичного алгоритму. Алгоритм зупиниться, коли різниця між найкращим знайденим значенням функції фітнесу та значенням функції фітнесу.

**Логи:**

****

**Висновки**

Під час виконання цієї лабораторної я навчився викрнувати завдання використовуючи генетичний алгоритм.