Gene Expression

import random

# -----------------------------

# 1. Define the Problem

# -----------------------------

# Example items

values = [10, 40, 30, 50] # values of the 4 items

weights = [5, 4, 6, 3] # weights of the 4 items

capacity = 10 # maximum knapsack weight

num\_items = len(values)

# -----------------------------

# 2. Initialize Parameters

# -----------------------------

POP\_SIZE = 10 # population size

GENERATIONS = 30 # number of generations

CROSSOVER\_RATE = 0.8

MUTATION\_RATE = 0.1

# -----------------------------

# 3. Initialize Population

# -----------------------------

def create\_chromosome():

"""Generate a random binary chromosome."""

return [random.randint(0, 1) for \_ in range(num\_items)]

def initialize\_population():

"""Create an initial population."""

return [create\_chromosome() for \_ in range(POP\_SIZE)]

# -----------------------------

# 4. Evaluate Fitness

# -----------------------------

def fitness(chromosome):

total\_value = 0

total\_weight = 0

for gene, v, w in zip(chromosome, values, weights):

if gene == 1:

total\_value += v

total\_weight += w

if total\_weight > capacity:

return 0 # penalty for exceeding capacity

return total\_value

# -----------------------------

# 5. Selection

# -----------------------------

def selection(population):

"""Tournament selection."""

tournament\_size = 3

selected = random.sample(population, tournament\_size)

selected = sorted(selected, key=fitness, reverse=True)

return selected[0]

# -----------------------------

# 6. Crossover

# -----------------------------

def crossover(parent1, parent2):

if random.random() < CROSSOVER\_RATE:

point = random.randint(1, num\_items - 1)

child1 = parent1[:point] + parent2[point:]

child2 = parent2[:point] + parent1[point:]

return child1, child2

return parent1[:], parent2[:]

# -----------------------------

# 7. Mutation

# -----------------------------

def mutate(chromosome):

for i in range(len(chromosome)):

if random.random() < MUTATION\_RATE:

chromosome[i] = 1 - chromosome[i] # flip bit

return chromosome

# -----------------------------

# 8. Gene Expression (Phenotype Mapping)

# -----------------------------

def decode(chromosome):

"""Translate chromosome into selected items, total value, and total weight."""

chosen\_items = []

total\_value = 0

total\_weight = 0

for i, gene in enumerate(chromosome):

if gene == 1:

chosen\_items.append(i + 1)

total\_value += values[i]

total\_weight += weights[i]

return chosen\_items, total\_value, total\_weight

# -----------------------------

# 9. Iterate Through Generations

# -----------------------------

def gene\_expression\_algorithm():

population = initialize\_population()

best\_solution = None

best\_fitness = 0

for generation in range(GENERATIONS):

new\_population = []

# Evaluate all chromosomes

population = sorted(population, key=fitness, reverse=True)

if fitness(population[0]) > best\_fitness:

best\_solution = population[0]

best\_fitness = fitness(population[0])

# Print progress

print(f"Generation {generation+1}: Best Fitness = {best\_fitness}")

# Elitism: carry the best two

new\_population.extend(population[:2])

# Create new population

while len(new\_population) < POP\_SIZE:

parent1 = selection(population)

parent2 = selection(population)

child1, child2 = crossover(parent1, parent2)

new\_population.append(mutate(child1))

if len(new\_population) < POP\_SIZE:

new\_population.append(mutate(child2))

population = new\_population

return best\_solution, best\_fitness

# -----------------------------

# 10. Output Best Solution

# -----------------------------

best\_chromosome, best\_value = gene\_expression\_algorithm()

items, total\_val, total\_wt = decode(best\_chromosome)

print("\nBest Chromosome:", best\_chromosome)

print("Selected Items:", items)

print("Total Value:", total\_val)

print("Total Weight:", total\_wt)

Output:

