Gene expression algorithm

import random

from tabulate import tabulate

def fitness(path, distances):

"""Calculate total distance of a path"""

total = 0

for i in range(len(path)):

total += distances[path[i]][path[(i + 1) % len(path)]]

return total

def crossover(parent1, parent2):

"""Order crossover"""

start, end = sorted(random.sample(range(len(parent1)), 2))

child = [-1] \* len(parent1)

child[start:end] = parent1[start:end]

fill\_values = [x for x in parent2 if x not in child]

fill\_index = 0

for i in range(len(child)):

if child[i] == -1:

child[i] = fill\_values[fill\_index]

fill\_index += 1

return child, (start, end)

def mutate(chromosome):

"""Swap mutation"""

i, j = random.sample(range(len(chromosome)), 2)

chromosome[i], chromosome[j] = chromosome[j], chromosome[i]

return chromosome, (i, j)

def gene\_expression(sequence):

"""Ensure gene sequence is a valid TSP path"""

# Remove duplicates while preserving order

seen = set()

path = []

for gene in sequence:

if gene not in seen:

path.append(gene)

seen.add(gene)

# Add missing cities

all\_cities = set(range(len(sequence)))

for city in all\_cities - seen:

path.append(city)

return path

def gea\_tsp(distances, pop\_size=4, generations=3):

n\_cities = len(distances)

population = [random.sample(range(n\_cities), n\_cities) for \_ in range(pop\_size)]

best\_path = None

best\_distance = float('inf')

for gen in range(generations):

print(f"\n=== Generation {gen+1} ===")

table = []

new\_population = []

for i in range(0, len(population), 2):

parent1 = population[i]

parent2 = population[(i+1) % len(population)]

# Crossover

child, cross\_points = crossover(parent1, parent2)

expressed\_child = gene\_expression(child)

# Mutation

offspring, mut\_points = mutate(expressed\_child.copy())

offspring = gene\_expression(offspring)

dist = fitness(offspring, distances)

if dist < best\_distance:

best\_distance = dist

best\_path = offspring.copy()

table.append([

parent1, fitness(parent1, distances),

parent2,

cross\_points, expressed\_child,

mut\_points, offspring, dist

])

new\_population.append(offspring)

population = new\_population

headers = ["Parent", "Fitness", "Mate", "Crossover", "After Gene Expression",

"Mutation", "Offspring", "Offspring Fitness"]

print(tabulate(table, headers=headers, tablefmt="grid"))

print("\n✅ Shortest Path Found:", best\_path, "with Distance:", best\_distance)

# ---- User Input ----

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input("Enter number of cities: "))

print("Enter distance matrix row by row (space-separated):")

distances = [list(map(int, input(f"Row {i+1}: ").split())) for i in range(n)]

pop\_size = int(input("Enter population size: "))

generations = int(input("Enter number of generations: "))

gea\_tsp(distances, pop\_size, generations)

