SAMRAAT DABOLAY 1BM22CS236

Lab Question: Genetic Algorithm

Code:

```
import numpy as np

# Objective: Calculate the total distance for a given set of routes
def objective(routes, distance_matrix):
    total_distance = 0
    for route in routes:
        # Add the distance from depot to the first location
        total_distance += distance_matrix[0][route[0]]
        # Add the distances between consecutive locations in the route
        for i in range(len(route) - 1):
            total_distance += distance_matrix[route[i]][route[i + 1]]
        # Add the distance from the last location back to depot
        total_distance += distance_matrix[route[-1]][0]
    return total_distance

# Generate an initial population with more diverse routes
def initialize_population(n_pop, n_locations, n_vehicles):
    population = []
    for _ in range(n_pop):
        # Randomly generate routes (random permutations of locations)
        routes = []
        remaining_locations = list(range(1, n_locations))  # Exclude depot (location 0)
        for _ in range(n_vehicles):
            route_size = len(remaining_locations) // (n_vehicles - len(routes))
            route = np.random.choice(remaining_locations, size=route_size, replace=False).tolist()
            for loc in route:
                remaining_locations.remove(loc)
            routes.append(route)
        population.append(routes)
    return population

# Evaluate fitness of the population (lower is better)
def evaluate_fitness(population, distance_matrix):
    return [objective(routes, distance_matrix) for routes in population]

# Tournament selection
def tournament_selection(population, scores, k=3):
    selected = np.random.choice(len(population), k, replace=False)
    best = selected[np.argmin([scores[i] for i in selected])]
    return population[best]
```

SAMRAAT DABOLAY 1BM22CS236

```
# Order Crossover (OX)
def crossover(p1, p2):
    size = len(p1)
    child = [None] * size
    start, end = sorted(np.random.choice(range(size), 2, replace=False))
    child[start:end+1] = p1[start:end+1]

    # Fill remaining positions
    p2_idx = 0
    for i in range(size):
        if child[i] is None:
            while p2[p2_idx] in child:
                p2_idx += 1
            child[i] = p2[p2_idx]
    return child

# Swap mutation (increased mutation rate)
def mutation(routes, n_locations):
    route = routes[np.random.randint(len(routes))]  # Select a random route to mutate
    if len(route) > 1:
        i, j = np.random.choice(len(route), 2, replace=False)
        route[i], route[j] = route[j], route[i]  # Swap two locations in the route
    return routes

# Genetic algorithm for solving VRP
def genetic_algorithm(distance_matrix, n_iter, n_pop, n_vehicles, r_mut, r_cross, elitism=True):
    n_locations = len(distance_matrix)  # Including depot (location 0)
    population = initialize_population(n_pop, n_locations, n_vehicles)
    best, best_eval = population[0], objective(population[0], distance_matrix)

    for gen in range(n_iter):
        scores = evaluate_fitness(population, distance_matrix)

        # Track and print the best fitness score
        for i in range(n_pop):
            if scores[i] < best_eval:
                best, best_eval = population[i], scores[i]
                print(f"Generation {gen}: New best solution with distance {scores[i]:.2f}")

        # Create next generation
        children = []
        for _ in range(n_pop):
            p1 = tournament_selection(population, scores)
            p2 = tournament_selection(population, scores)
```

```
            offspring = crossover(p1, p2) if np.random.rand() < r_cross else p1
            offspring = mutation(offspring, n_locations) if np.random.rand() < r_mut else offspring
            children.append(offspring)

        # Elitism: carry the best solution to the next generation
        if elitism:
            children[np.argmax([objective(child, distance_matrix) for child in children])] = best

        population = children

    return best, best_eval

# Example distance matrix with 10 locations (including the depot at location 0)
distance_matrix = np.array([
    [0, 10, 15, 20, 25, 30, 35, 40, 45, 50],   # Depot (location 0)
    [10, 0, 35, 25, 30, 10, 15, 20, 25, 30],    # Location 1
    [15, 35, 0, 20, 10, 15, 20, 25, 30, 35],    # Location 2
    [20, 25, 20, 0, 15, 20, 25, 30, 35, 40],    # Location 3
    [25, 30, 10, 15, 0, 10, 15, 20, 25, 30],    # Location 4
    [30, 10, 15, 20, 10, 0, 5, 10, 15, 20],     # Location 5
    [35, 15, 20, 25, 15, 5, 0, 5, 10, 15],      # Location 6
    [40, 20, 25, 30, 20, 10, 5, 0, 5, 10],      # Location 7
    [45, 25, 30, 35, 25, 15, 10, 5, 0, 5],      # Location 8
    [50, 30, 35, 40, 30, 20, 15, 10, 5, 0]      # Location 9
])

# Parameters
n_iter = 100  # Number of generations
n_pop = 50  # Population size
n_vehicles = 3  # Number of vehicles
r_mut = 0.2  # Mutation rate
r_cross = 0.7  # Crossover rate

best_routes, best_score = genetic_algorithm(distance_matrix, n_iter, n_pop, n_vehicles, r_mut,
r_cross)
print("Optimal routes:", best_routes)
print(f"Total distance: {best_score:.2f}")
```

Output:

```
Generation 0: New best solution with distance 245.00
Generation 0: New best solution with distance 240.00
Generation 1: New best solution with distance 225.00
Generation 1: New best solution with distance 220.00
Generation 2: New best solution with distance 210.00
Generation 2: New best solution with distance 205.00
Generation 6: New best solution with distance 200.00
Generation 10: New best solution with distance 185.00
Optimal routes: [[4, 5, 2], [3, 5, 1], [2, 3, 4]]
Total distance: 185.00
```