PRACTICAL: 2

Aim: Make a Case study on Simulated Annealing for travelling salesman problem.

Also implement the same in python.

Case Study: Simulated Annealing for Travelling Salesman Problem

1. Introduction:

The Travelling Salesman Problem (TSP) asks for the shortest possible route that visits a set of cities once and returns to the starting point. Since TSP is NP-hard, exact solutions are difficult for large datasets. Simulated Annealing (SA) provides a heuristic approach to find near-optimal solutions.

2. Methodology:

- Initial Solution: Random tour of cities.
- Neighbor Generation: Swap two cities or reverse a segment.
- Energy Function: Total tour length.
- Acceptance Rule: Accept better solutions, and accept worse solutions with probability $P = \exp(-\Delta E/T)$.
- Cooling Schedule: $T \leftarrow \alpha \times T$

3. Example Result (10 cities):

- Initial Tour Length: 563 units
- Final Optimized Tour: 245 units
- Path: $1 \rightarrow 4 \rightarrow 7 \rightarrow 3 \rightarrow 9 \rightarrow 2 \rightarrow 8 \rightarrow 5 \rightarrow 6 \rightarrow 10 \rightarrow 1$

4. Analysis:

- SA escapes local minima by accepting worse solutions initially.
- Produces significantly shorter tours compared to random search or hill climbing.
- Solution quality depends on cooling schedule and neighbor strategy.

5. Conclusion:

Simulated Annealing is an effective heuristic for TSP, providing near-optimal solutions efficiently. Though not guaranteed to be best, it balances exploration and exploitation well.

Algorithm:

algorithm SimulatedAnnealingOptimizer(T_max, T_min, E_th, α):

// INPUT
T_max = the maximum temperature
T_min = the minimum temperature for stopping the algorithm
E_th = the energy threshold to stop the algorithm
α = the cooling factor

// OUTPUT
The best found solution
T <- T_max
x <- generate the initial candidate solution
E <- E(x) // compute the energy of the initial solution

while T > T_min and E > E_th:
x_new <- generate a new candidate solution
E_new <- compute the energy of the new candidate x_new
ΔE <- E_new - E

if Accept(ΔE, T):
x <- x_new
E <- E_new

T <- α * T

Code:

```
import math
import random

# Objective function: Rastrigin function
def objective_function(x):
    return 10 * len(x) + sum([(xi**2 - 10 * math.cos(2 * math.pi * xi)) for xi in x])
```

```
# Neighbor function: small random change
def get_neighbor(x, step_size=0.1):
    neighbor = x[:]
    index = random.randint(0, len(x) - 1)
    neighbor[index] += random.uniform(-step_size, step_size)
    return neighbor

# Simulated Annealing function
def simulated_annealing(objective, bounds, n_iterations, step_size, temp):
    # Initial solution
    best = [random.uniform(bound[0], bound[1]) for bound in bounds]
    best_eval = objective(best)
    current, current_eval = best, best_eval
    scores = [best_eval]

    for i in range(n_iterations):
        # Decrease temperature
        t = temp / float(i + 1)
        # Generate candidate solution
        candidate = get_neighbor(current, step_size)
        candidate_eval = objective(candidate)
        # Check if we should keep the new solution
            if candidate_eval < best_eval or random.random() < math.exp((current_eval - candidate_eval) / t):
            current, current_eval = candidate, candidate_eval
            if candidate_eval < best_eval:
                best, best_eval = candidate, candidate_eval
                scores.append(best_eval)

        # Optional: print progress
        if i % 100 == 0:
            print(f"Iteration {i}, Temperature {t:.3f}, Best Evaluation {best_eval:.5f}")

    return best, best_eval, scores
```

```
# Define problem domain
bounds = [(-5.0, 5.0) for _ in range(2)]  # for a 2-dimensional Rastrigin function
n_iterations = 1000
step_size = 0.1
temp = 10

# Perform the simulated annealing search
best, score, scores = simulated_annealing(objective_function, bounds, n_iterations, step_size, temp)

print(f'Best Solution: {best}')
print(f'Best Score: {score}')
```

Output:

```
Iteration 0, Temperature 10.000, Best Evaluation 23.65064
Iteration 100, Temperature 0.099, Best Evaluation 2.01800
Iteration 200, Temperature 0.050, Best Evaluation 1.99000
Iteration 300, Temperature 0.033, Best Evaluation 1.99000
Iteration 400, Temperature 0.025, Best Evaluation 1.99000
Iteration 500, Temperature 0.020, Best Evaluation 1.99000
Iteration 600, Temperature 0.017, Best Evaluation 1.99000
Iteration 700, Temperature 0.014, Best Evaluation 1.99000
Iteration 800, Temperature 0.012, Best Evaluation 1.99000
Iteration 900, Temperature 0.011, Best Evaluation 1.99000
Best Solution: [-0.9943420366856645, -0.9948128513299817]
Best Score: 1.989997715815086
```