EASTERN MEDITERRANEAN UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

AING216 Basic Search Strategies

LAB V

Optimization Using Simulated Annealing

1. Objective

By the end of this lab, students will:

- Understand the concept of **Simulated Annealing**
- Implement the algorithm from scratch
- Apply it to solve an optimization problem
- Analyze how parameters affect performance

2. Background

Simulated Annealing is inspired by the physical process of heating and slowly cooling a material to reduce defects.

The algorithm allows worse solutions temporarily to escape local minima.

Acceptance Probability

$$P = e^{-\frac{\Delta E}{T}}$$

Where:

- ΔE: Change in cost (new − current)
- T: Temperature
- P: Probability of accepting worse solution

3. Problem Statement

Task: Traveling Salesman Problem (TSP)

Given a set of cities, find the shortest possible route that:

- Visits each city once
- Returns to the starting point

Part 1: Setup

Step 1: Generate Cities

- Create N random cities (e.g., N = 20–50)
- Each city has coordinates (x, y)

Step 2: Define Cost Function

Total distance of the path:

$$\text{Cost} = \sum_{i=1}^{N} d(cit\,y_i, city_{i+1})$$

Part 2: Simulated Annealing Implementation

Step 1: Initial Solution

- Start with a random route

Step 2: Neighbor Function

- Swap two cities randomly

Step 3: Cooling Schedule

$$T_{new} = \alpha T_{old}$$

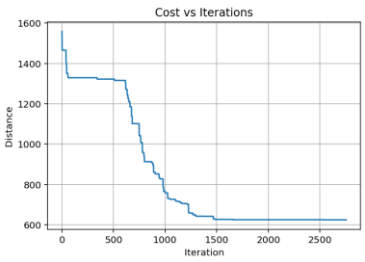
Where:

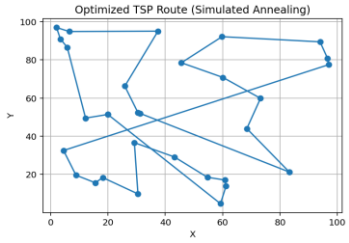
- $\alpha \in (0,1)$, e.g., 0.95

Step 4: Algorithm (Pseudo-code)

```
Initialize temperature T
Initialize current solution S
Best = S

WHILE T > Tmin:
   Generate neighbor S'
   ΔE = cost(S') - cost(S)

   IF ΔE < 0:
      Accept S'
   ELSE:
      Accept with probability exp(-ΔE / T)

   Update Best if needed
   Decrease temperature

RETURN Best
```

4. Implementation Task





Students must:

- Implement SA in Python (or preferred language)
- Plot:
    - Cost vs Iterations
    - Final route visualization

5. Experiments

You should test:

1. Different cooling rates:
    - 0.99, 0.95, 0.90
2. Different initial temperatures
3. Different number of cities

6. Analysis Questions

1. What happens if cooling is too fast?
2. Why does SA accept worse solutions?
3. How does temperature affect exploration vs exploitation?

7. Submission Guidelines

- Python source code (.py) or a single notebook file containing all four questions.
- Output screenshots or console captures for each question.
- Proper comments in code. For randomized questions, state the seed used for the sample trace.

Sample Code:

```
import numpy as np
import matplotlib.pyplot as plt
import random
import math

# ----------------------------
# Generate Cities
# ----------------------------
def generate_cities(n, seed=42):
    random.seed(seed)
    np.random.seed(seed)
    return np.random.rand(n, 2) * 100


# ----------------------------
# Distance Function
# ----------------------------
def distance(city1, city2):
    return np.linalg.norm(city1 - city2)


# ----------------------------
# Total Route Cost
# ----------------------------
def total_distance(route, cities):
    dist = 0
    for i in range(len(route)):
        dist += distance(cities[route[i]], cities[route[(i + 1) % len(route)]])
    return dist
```

```
# ----------------------------
# Neighbor (swap two cities)
# ----------------------------
def get_neighbor(route):
    new_route = route.copy()
    i, j = random.sample(range(len(route)), 2)
    new_route[i], new_route[j] = new_route[j], new_route[i]
    return new_route


# ----------------------------
# Simulated Annealing
# ----------------------------
def simulated_annealing(cities, T=1000, T_min=1e-3, alpha=0.995, max_iter=100000):
    n = len(cities)

    # Initial solution
    current_route = list(range(n))
    random.shuffle(current_route)
    current_cost = total_distance(current_route, cities)

    best_route = current_route.copy()
    best_cost = current_cost

    costs = []

    iteration = 0

    while _________ and iteration < max_iter:
        …..
    return best_route, best_cost, costs
```

```
# ----------------------------
# Plot Route
# ----------------------------
def plot_route(cities, route, title="Route"):
    x = [cities[i][0] for i in route] + [cities[route[0]][0]]
    y = [cities[i][1] for i in route] + [cities[route[0]][1]]

    plt.figure()
    plt.plot(x, y, marker='o')
    plt.title(title)
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid()
    plt.show()


# ----------------------------
# Plot Cost
# ----------------------------
def plot_cost(costs):
    plt.figure()
    plt.plot(costs)
    plt.title("Cost vs Iterations")
    plt.xlabel("Iteration")
    plt.ylabel("Distance")
    plt.grid()
    plt.show()
```

```
# ----------------------------
# Main Execution
# ----------------------------
    N_CITIES = 30

    cities = generate_cities(N_CITIES)

    best_route, best_cost, costs = simulated_annealing(
        cities,
        T=1000,
        T_min=1e-3,
        alpha=0.995,
        max_iter=50000
    )

    print("Best Distance:", best_cost)

    plot_route(cities, best_route, "Optimized TSP Route (Simulated Annealing)")
    plot_cost(costs)
```