Sanjana Shetty
5E

Ant Colony Optimization

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

# 1. Define the Problem: Taking custom 2D city coordinates as input
def input_city_coordinates():
    """
    Function to input custom city coordinates.
    The user is prompted to input coordinates for each city.
    """
    n_cities = int(input("Enter the number of cities: "))
    cities = []

    for i in range(n_cities):
        # Taking x and y coordinates as input
        x, y = map(float, input(f"Enter coordinates for city {i + 1} (x,
y): ").split())
        cities.append([x, y])

    return np.array(cities)  # Return as a NumPy array for convenience

# 2. Distance Function: Calculate Euclidean distance between two cities
def distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2)**2))

# 3. Construct Solutions: Build a solution for each ant
def construct_solution(n_points, pheromone, points, alpha, beta):
    visited = [False] * n_points
    current_point = np.random.randint(n_points)  # Start from a random
city
    visited[current_point] = True
    path = [current_point]
    path_length = 0
```

```
    while False in visited:
        unvisited = np.where(np.logical_not(visited))[0]
        probabilities = np.zeros(len(unvisited))

        # Calculate the probabilities for the unvisited cities
        for i, unvisited_point in enumerate(unvisited):
            pheromone_value = pheromone[current_point, unvisited_point] **
alpha
            distance_value = distance(points[current_point],
points[unvisited_point]) ** beta
            probabilities[i] = pheromone_value / distance_value

        # Normalize the probabilities
        probabilities /= np.sum(probabilities)

        # Choose the next city based on probabilities
        next_point = np.random.choice(unvisited, p=probabilities)
        path.append(next_point)
        path_length += distance(points[current_point], points[next_point])
        visited[next_point] = True
        current_point = next_point

    return path, path_length

# 4. Update Pheromones: Update pheromone levels based on the ants'
solutions
def update_pheromones(pheromone, paths, path_lengths, evaporation_rate,
Q):
    pheromone *= evaporation_rate  # Evaporate all pheromones
    for path, path_length in zip(paths, path_lengths):
        for i in range(len(path) - 1):
            pheromone[path[i], path[i + 1]] += Q / path_length
        pheromone[path[-1], path[0]] += Q / path_length  # Close the loop
    return pheromone

# 5. Main ACO Algorithm: Main function to run the ACO
def ant_colony_optimization(points, n_ants, n_iterations, alpha, beta,
evaporation_rate, Q):
    n_points = len(points)
    pheromone = np.ones((n_points, n_points))  # Initial pheromone levels
```

```
    best_path = None
    best_path_length = np.inf

    # 6. Iterate: Run the ACO for a set number of iterations
    for iteration in range(n_iterations):
        paths = []
        path_lengths = []

        # Construct solutions for each ant
        for _ in range(n_ants):
            path, path_length = construct_solution(n_points, pheromone,
points, alpha, beta)
            paths.append(path)
            path_lengths.append(path_length)

            # Update the best solution found
            if path_length < best_path_length:
                best_path = path
                best_path_length = path_length

        # Update pheromones based on all ants' paths
        pheromone = update_pheromones(pheromone, paths, path_lengths,
evaporation_rate, Q)

        # Optional: Print or log progress
        print(f"Iteration {iteration + 1}/{n_iterations}, Best Path
Length: {best_path_length:.2f}")

    # Return the best path found
    return best_path, best_path_length

# 7. Output the Best Solution: Plotting the best path in 2D space
def plot_best_path(points, best_path):
    fig, ax = plt.subplots(figsize=(8, 6))
    ax.scatter(points[:, 0], points[:, 1], c='red', marker='o',
label='Cities')

    # Draw the best path found by the ants
    path_points = points[best_path]
```

```python
    path_points = np.vstack([path_points, path_points[0]])  # Close the loop
    ax.plot(path_points[:, 0], path_points[:, 1], c='green', linewidth=2, marker='o', label='Best Path')

    # Display the cities' indices
    for i, point in enumerate(points):
        ax.text(point[0], point[1], str(i), fontsize=12, ha='right')

    ax.set_xlabel('X Coordinate')
    ax.set_ylabel('Y Coordinate')
    ax.set_title('Best TSP Solution Found by ACO')
    ax.legend()
    plt.show()

# Example usage with custom city inputs:
points = input_city_coordinates()  # Take custom input for city coordinates
best_path, best_path_length = ant_colony_optimization(
    points,
    n_ants=10,             # Number of ants
    n_iterations=100,      # Number of iterations
    alpha=1,               # Pheromone importance
    beta=1,                # Distance importance
    evaporation_rate=0.5,  # Evaporation rate
    Q=1                     # Pheromone deposit
)

print(f"Best Path: {best_path}")
print(f"Best Path Length: {best_path_length:.2f}")

# Plot the best path found
plot_best_path(points, best_path)
```

OUTPUT:

```
Enter the number of cities: 6
Enter coordinates for city 1 (x, y): 0 0
Enter coordinates for city 2 (x, y): 2 4
Enter coordinates for city 3 (x, y): 5 2
Enter coordinates for city 4 (x, y): 6 7
Enter coordinates for city 5 (x, y): 8 3
Enter coordinates for city 6 (x, y): 3 6
Iteration 1/100, Best Path Length: 18.87
Iteration 2/100, Best Path Length: 18.87
Iteration 3/100, Best Path Length: 18.87
Iteration 4/100, Best Path Length: 18.87
Iteration 5/100, Best Path Length: 18.87
Iteration 6/100, Best Path Length: 18.87
Iteration 7/100, Best Path Length: 18.87
Iteration 8/100, Best Path Length: 18.87
Iteration 9/100, Best Path Length: 18.42
Iteration 10/100, Best Path Length: 18.42
Iteration 11/100, Best Path Length: 17.50
Iteration 12/100, Best Path Length: 17.50
Iteration 13/100, Best Path Length: 17.50
Iteration 14/100, Best Path Length: 17.50
Iteration 15/100, Best Path Length: 17.50
Iteration 16/100, Best Path Length: 17.50
Iteration 17/100, Best Path Length: 17.50
Iteration 18/100, Best Path Length: 17.50
Iteration 19/100, Best Path Length: 17.50
Iteration 20/100, Best Path Length: 17.50
Iteration 21/100, Best Path Length: 17.50
Iteration 22/100, Best Path Length: 17.50
Iteration 23/100, Best Path Length: 17.50
Iteration 24/100, Best Path Length: 17.50
Iteration 25/100, Best Path Length: 17.50
Iteration 26/100, Best Path Length: 17.50
Iteration 27/100, Best Path Length: 17.50
Iteration 28/100, Best Path Length: 17.50
Iteration 29/100, Best Path Length: 17.50
Iteration 30/100, Best Path Length: 17.50
Iteration 31/100, Best Path Length: 17.50
Iteration 32/100, Best Path Length: 17.50
Iteration 33/100, Best Path Length: 17.50
Iteration 34/100, Best Path Length: 17.50
Iteration 35/100, Best Path Length: 17.50
Iteration 36/100, Best Path Length: 17.50
Iteration 37/100, Best Path Length: 17.50
Iteration 38/100, Best Path Length: 17.50
Iteration 39/100, Best Path Length: 17.50
Iteration 40/100, Best Path Length: 17.50
Iteration 41/100, Best Path Length: 17.50
Iteration 42/100, Best Path Length: 17.50
```

```
Iteration 85/100, Best Path Length: 17.50
Iteration 86/100, Best Path Length: 17.50
Iteration 87/100, Best Path Length: 17.50
Iteration 88/100, Best Path Length: 17.50
Iteration 89/100, Best Path Length: 17.50
Iteration 90/100, Best Path Length: 17.50
Iteration 91/100, Best Path Length: 17.50
Iteration 92/100, Best Path Length: 17.50
Iteration 93/100, Best Path Length: 17.50
Iteration 94/100, Best Path Length: 17.50
Iteration 95/100, Best Path Length: 17.50
Iteration 96/100, Best Path Length: 17.50
Iteration 97/100, Best Path Length: 17.50
Iteration 98/100, Best Path Length: 17.50
Iteration 99/100, Best Path Length: 17.50
Iteration 100/100, Best Path Length: 17.50
Best Path: [2, 4, 3, 5, 1, 0]
Best Path Length: 17.50
```