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

# TSP Problem: List of cities as (x, y) coordinates
cities = [(0, 0), (1, 2), (2, 4), (3, 3), (5, 1), (6, 5), (7, 7)]

# Number of ants
num_ants = 50
# Number of iterations
num_iterations = 100
# Alpha and Beta: controls pheromone and distance importance
alpha = 1.0
beta = 5.0
# Evaporation rate
rho = 0.1
# Pheromone intensity
Q = 100

# Distance matrix
def distance(c1, c2):
    return np.sqrt((c1[0] - c2[0])**2 + (c1[1] - c2[1])**2)

# Create distance matrix for the cities
num_cities = len(cities)
dist_matrix = np.zeros((num_cities, num_cities))

for i in range(num_cities):
    for j in range(num_cities):
        if i != j:
            dist_matrix[i][j] = distance(cities[i], cities[j])

# Initialize pheromone levels
pheromone = np.ones((num_cities, num_cities))  # Initially, set
pheromone to 1

# Function to choose the next city using a probability distribution
def choose_next_city(current_city, visited, pheromone, dist_matrix):
    probabilities = []
    cities_to_visit = []
    total_pheromone = 0.0

    # Calculate probabilities for cities not yet visited
    for i in range(num_cities):
        if i not in visited:
            pheromone_ij = pheromone[current_city][i] ** alpha
            distance_ij = dist_matrix[current_city][i] ** beta
            prob = pheromone_ij / distance_ij
```

```
            probabilities.append(prob)
            cities_to_visit.append(i)
            total_pheromone += prob

    # Normalize probabilities
    probabilities = [prob / total_pheromone for prob in probabilities]

    # Select the next city based on the computed probabilities
    next_city = random.choices(cities_to_visit, probabilities)[0]

    return next_city


# Function to update pheromone levels
def update_pheromone(pheromone, paths, dist_matrix):
    global rho, Q
    pheromone_deposit = np.zeros_like(pheromone)

    # Deposit pheromones along the paths taken by ants
    for path in paths:
        path_length = 0
        for i in range(len(path) - 1):
            path_length += dist_matrix[path[i]][path[i + 1]]
        path_length += dist_matrix[path[-1]][path[0]]  # Return to the
start city

        # Add pheromone based on path length (shorter paths get more
pheromone)
        for i in range(len(path) - 1):
            pheromone_deposit[path[i]][path[i + 1]] += Q / path_length
        pheromone_deposit[path[-1]][path[0]] += Q / path_length  #
Return edge

    # Update pheromone matrix by evaporating and adding new pheromone
    pheromone = (1 - rho) * pheromone + pheromone_deposit
    return pheromone

# Main ACO Algorithm
def ant_colony_optimization():
    global pheromone  # Ensure we are modifying the global pheromone
matrix
    best_path = None
    best_path_length = float('inf')
    all_paths = []

    for iteration in range(num_iterations):
        all_ants_paths = []

        # Each ant starts at a random city
```

```
        for ant in range(num_ants):
            visited = [random.randint(0, num_cities - 1)]
            current_city = visited[0]

            # Construct a path for the ant
            for _ in range(num_cities - 1):
                next_city = choose_next_city(current_city, visited, 
pheromone, dist_matrix)
                visited.append(next_city)
                current_city = next_city

            all_ants_paths.append(visited)

        # Update pheromone matrix based on ants' paths
        pheromone = update_pheromone(pheromone, all_ants_paths, 
dist_matrix)

        # Evaluate the best path among all ants
        for path in all_ants_paths:
            path_length = 0
            for i in range(len(path) - 1):
                path_length += dist_matrix[path[i]][path[i + 1]]
            path_length += dist_matrix[path[-1]][path[0]]  # Return to 
the start city

            if path_length < best_path_length:
                best_path_length = path_length
                best_path = path

        # Optional: Print the current best path and its length every 10 
iterations
        if iteration % 10 == 0:
            print(f"Iteration {iteration}, Best Path Length: 
{best_path_length}")

    return best_path, best_path_length

# Run the ACO algorithm
best_path, best_path_length = ant_colony_optimization()

# Display the result
print(f"Best Path: {best_path}")
print(f"Best Path Length: {best_path_length}")

# Plot the best path
x = [cities[i][0] for i in best_path] + [cities[best_path[0]][0]]
y = [cities[i][1] for i in best_path] + [cities[best_path[0]][1]]
```

```
plt.figure(figsize=(8, 6))
plt.plot(x, y, 'b-o', markersize=8)
plt.scatter([city[0] for city in cities], [city[1] for city in cities],
color='red')
plt.title(f"Best Path with Length {best_path_length:.2f}")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.show()
```

Output:

```
Iteration 0, Best Path Length: 23.151543604265996
Iteration 10, Best Path Length: 23.00139688357529
Iteration 20, Best Path Length: 23.00139688357529
Iteration 30, Best Path Length: 23.00139688357529
Iteration 40, Best Path Length: 23.00139688357529
Iteration 50, Best Path Length: 23.00139688357529
Iteration 60, Best Path Length: 23.00139688357529
Iteration 70, Best Path Length: 23.00139688357529
Iteration 80, Best Path Length: 23.00139688357529
Iteration 90, Best Path Length: 23.00139688357529
Best Path: [3, 2, 1, 0, 4, 5, 6]
Best Path Length: 23.00139688357529
```