```
import numpy as np
# Define the problem: Distance matrix between cities
distance_matrix = np.array([
    [0, 2, 2, 5],
    [2, 0, 3, 4],
    [2, 3, 0, 1],
    [5, 4, 1, 0]
])
# Parameters
num_ants = 4
num_iterations = 100
alpha = 1  # Pheromone importance
beta = 2   # Distance importance
evaporation_rate = 0.5
pheromone_constant = 1
# Initialize pheromone levels
num_cities = len(distance_matrix)
pheromone = np.ones((num_cities, num_cities))
# Helper function: Calculate probabilities for next city
def calculate_probabilities(current_city, visited, pheromone,
distance_matrix):
    probabilities = []
```

```
    for city in range(num_cities):
        if city not in visited:
            prob = (pheromone[current_city][city] ** alpha) * ((1 / distance_matrix[current_city][city]) ** beta)
            probabilities.append(prob)
        else:
            probabilities.append(0)
    probabilities = np.array(probabilities)
    return probabilities / probabilities.sum()
# Main ACO loop
best_path = None
best_distance = float('inf')
for iteration in range(num_iterations):
    all_paths = []
    all_distances = []
    for ant in range(num_ants):
        visited = []
        current_city = np.random.randint(0, num_cities)
        visited.append(current_city)
        while len(visited) < num_cities:
            probabilities = calculate_probabilities(current_city, visited, pheromone, distance_matrix)
```

```
            next_city = np.random.choice(range(num_cities),
p=probabilities)
            visited.append(next_city)
            current_city = next_city
        # Complete the tour by returning to the starting city
        visited.append(visited[0])
        all_paths.append(visited)
        # Calculate the total distance of the path
        distance = sum(distance_matrix[visited[i]][visited[i + 1]]
for i in range(len(visited) - 1))
        all_distances.append(distance)
        # Update best path
        if distance < best_distance:
            best_distance = distance
            best_path = visited
    # Update pheromone levels
    pheromone *= (1 - evaporation_rate)  # Evaporation
    for path, distance in zip(all_paths, all_distances):
        for i in range(len(path) - 1):
            pheromone[path[i]][path[i + 1]] +=
pheromone_constant / distance
# Output the best path and distance
```

```
print("Best Path:", best_path)
print("Best Distance:", best_distance)
```