```python
import numpy as np
import random
import math

# Step 1: Define the Problem
class TSP:
    def __init__(self, cities):
        self.cities = cities
        self.num_cities = len(cities)
        self.distances = self.calculate_distances()

    def calculate_distances(self):
        # Calculate the Euclidean distance between all pairs of cities
        distances = np.zeros((self.num_cities, self.num_cities))
        for i in range(self.num_cities):
            for j in range(i + 1, self.num_cities):
                dist = np.linalg.norm(np.array(self.cities[i]) - np.array(self.cities[j]))
                distances[i][j] = distances[j][i] = dist
        return distances

    def get_distance(self, path):
        # Calculate the total distance of a given path (tour)
        distance = 0
        for i in range(len(path) - 1):
            distance += self.distances[path[i]][path[i+1]]
        distance += self.distances[path[-1]][path[0]]  # Return to the starting city
        return distance

# Step 2: Initialize Parameters
class AntColony:
    def __init__(self, problem, num_ants, alpha, beta, rho, iterations):
        self.problem = problem
        self.num_ants = num_ants
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
        self.iterations = iterations
        self.pheromone = np.ones((self.problem.num_cities, self.problem.num_cities))  # Initial pheromone
        self.best_solution = None
        self.best_distance = float('inf')
```

```python
    def choose_next_city(self, current_city, visited):
        # Step 3: Probabilistically choose the next city
        probabilities = []
        for i in range(self.problem.num_cities):
            if i not in visited:
                pheromone = self.pheromone[current_city][i] ** self.alpha
                heuristic = (1.0 / self.problem.distances[current_city][i]) ** self.beta
                probabilities.append(pheromone * heuristic)
            else:
                probabilities.append(0)
        total_prob = sum(probabilities)
        probabilities = [p / total_prob for p in probabilities]
        next_city = random.choices(range(self.problem.num_cities), probabilities)[0]
        return next_city

    def construct_solution(self):
        # Step 3: Construct a solution (path) by all ants
        solution = []
        for _ in range(self.num_ants):
            path = [random.randint(0, self.problem.num_cities - 1)]  # Start from a random city
            visited = set(path)
            while len(path) < self.problem.num_cities:
                current_city = path[-1]
                next_city = self.choose_next_city(current_city, visited)
                path.append(next_city)
                visited.add(next_city)
            solution.append(path)
        return solution

    def update_pheromone(self, solutions):
        # Step 4: Update pheromone trails
        new_pheromone = np.zeros_like(self.pheromone)
        for solution in solutions:
            distance = self.problem.get_distance(solution)
            for i in range(len(solution) - 1):
                current_city = solution[i]
                next_city = solution[i + 1]
                new_pheromone[current_city][next_city] += 1.0 / distance
                new_pheromone[next_city][current_city] += 1.0 / distance
            # Return to the starting city
            current_city = solution[-1]
            next_city = solution[0]
            new_pheromone[current_city][next_city] += 1.0 / distance
            new_pheromone[next_city][current_city] += 1.0 / distance

        # Apply pheromone evaporation
        self.pheromone = (1 - self.rho) * self.pheromone + new_pheromone
```

```
    def run(self):
        # Step 5: Run the ACO algorithm
        for iteration in range(self.iterations):
            solutions = self.construct_solution()
            for solution in solutions:
                distance = self.problem.get_distance(solution)
                if distance < self.best_distance:
                    self.best_solution = solution
                    self.best_distance = distance
            self.update_pheromone(solutions)
            print(f"Iteration {iteration + 1}: Best Distance = {self.best_distance}")
        # Step 6: Output the Best Solution
        return self.best_solution, self.best_distance


# Example usage:
if __name__ == "__main__":
    # Step 1: Define the cities (coordinates)
    cities = [(0, 0), (1, 2), (2, 4), (3, 3), (5, 0), (4, 2)]  # Example cities in 2D space
    tsp = TSP(cities)

    # Step 2: Initialize the Ant Colony with parameters
    num_ants = 20
    alpha = 1.0  # Importance of pheromone
    beta = 2.0   # Importance of heuristic (inverse distance)
    rho = 0.1    # Evaporation rate
    iterations = 10

    colony = AntColony(tsp, num_ants, alpha, beta, rho, iterations)

    # Run the Ant Colony Optimization
    best_solution, best_distance = colony.run()

    # Output the results
    print("\nBest Path: ", best_solution)
    print("Best Distance: ", best_distance)
```

```
Iteration 1: Best Distance = 14.536631057245 56
Iteration 2: Best Distance = 14.536631057245 56
Iteration 3: Best Distance = 14.536631057245 56
Iteration 4: Best Distance = 14.536631057245 56
Iteration 5: Best Distance = 14.536631057245 56
Iteration 6: Best Distance = 14.536631057245 56
Iteration 7: Best Distance = 14.536631057245 56
Iteration 8: Best Distance = 14.536631057245 56
Iteration 9: Best Distance = 14.536631057245 56
Iteration 10: Best Distance = 14.536631057245 56

Best Path:  [0, 1, 2, 3, 5, 4]
Best Distance:  14.536631057245 56
```