```python
import numpy as np
import random

# Euclidean distance function
def euclidean_distance(city1, city2):
    return np.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)

# Ant Colony Optimization for TSP
def ant_colony_optimization(cities, num_ants=30, num_iterations=100, alpha=1.0, beta=2.0, rho=0.1, Q=100, initial_pheromone=1.0):
    # Number of cities
    num_cities = len(cities)

    # Distance matrix (distance between each pair of cities)
    dist_matrix = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(num_cities):
            dist_matrix[i][j] = euclidean_distance(cities[i], cities[j])

    # Initialize pheromone matrix
    pheromone_matrix = np.ones((num_cities, num_cities)) * initial_pheromone

    # Best solution found
    best_solution = None
    best_solution_length = float('inf')

    # ACO Main loop
    for iteration in range(num_iterations):
        all_paths = []
        all_path_lengths = []

        # Construct solutions (paths) for all ants
        for ant in range(num_ants):
            path = [random.randint(0, num_cities - 1)]  # Ant starts at a random city
            visited = [False] * num_cities
            visited[path[0]] = True

            # Construct path for the ant
            while len(path) < num_cities:
                current_city = path[-1]
                probabilities = []

                for next_city in range(num_cities):
                    if not visited[next_city]:
                        pheromone = pheromone_matrix[current_city][next_city] ** alpha
                        distance = dist_matrix[current_city][next_city] ** (-beta)
                        probability = pheromone * distance
                        probabilities.append(probability)
                    else:
                        probabilities.append(0)

                # Normalize probabilities
                total_probability = sum(probabilities)
                probabilities = [p / total_probability for p in probabilities]

                # Select the next city based on probabilities
                next_city = np.random.choice(range(num_cities), p=probabilities)
                path.append(next_city)
                visited[next_city] = True

            # Complete the cycle by returning to the starting city
            path_length = sum(dist_matrix[path[i], path[i + 1]] for i in range(num_cities - 1))
            path_length += dist_matrix[path[-1], path[0]]

            all_paths.append(path)
            all_path_lengths.append(path_length)

            # Update the best solution
            if path_length < best_solution_length:
                best_solution = path
                best_solution_length = path_length

        # Pheromone update
        pheromone_matrix *= (1 - rho)  # Apply evaporation

        # Add new pheromone
        for i in range(num_ants):
            path = all_paths[i]
            path_length = all_path_lengths[i]
            pheromone_contribution = Q / path_length

            for i in range(num_cities - 1):
```

```python
                pheromone_matrix[path[i], path[i + 1]] += pheromone_contribution
            pheromone_matrix[path[-1], path[0]] += pheromone_contribution

        # Print progress for each iteration
        print(f"Iteration {iteration + 1}, Best path length: {best_solution_length}")

    return best_solution, best_solution_length

# Generate random cities (for example, 10 cities)
num_cities = 10
cities = np.random.rand(num_cities, 2)  # Random positions of cities in a 2D plane

# Run Ant Colony Optimization for TSP
best_path, best_length = ant_colony_optimization(cities, num_ants=30, num_iterations=100, alpha=1.0, beta=2.0, rho=0.1, Q=100)

# Print the best path and its length
print("\nBest path found:", best_path)
print("Best path length:", best_length)
```

```
Iteration 46, Best path length: 2.633770590867323
Iteration 47, Best path length: 2.633770590867323
Iteration 48, Best path length: 2.633770590867323
Iteration 49, Best path length: 2.633770590867323
Iteration 50, Best path length: 2.633770590867323
Iteration 51, Best path length: 2.633770590867323
Iteration 52, Best path length: 2.633770590867323
Iteration 53, Best path length: 2.633770590867323
Iteration 54, Best path length: 2.633770590867323
Iteration 55, Best path length: 2.633770590867323
Iteration 56, Best path length: 2.633770590867323
Iteration 57, Best path length: 2.633770590867323
Iteration 58, Best path length: 2.633770590867323
Iteration 59, Best path length: 2.633770590867323
Iteration 60, Best path length: 2.633770590867323
Iteration 61, Best path length: 2.633770590867323
Iteration 62, Best path length: 2.633770590867323
Iteration 63, Best path length: 2.633770590867323
Iteration 64, Best path length: 2.633770590867323
Iteration 65, Best path length: 2.633770590867323
Iteration 66, Best path length: 2.633770590867323
Iteration 67, Best path length: 2.633770590867323
Iteration 68, Best path length: 2.633770590867323
Iteration 69, Best path length: 2.633770590867323
Iteration 70, Best path length: 2.633770590867323
Iteration 71, Best path length: 2.633770590867323
Iteration 72, Best path length: 2.633770590867323
Iteration 73, Best path length: 2.633770590867323
Iteration 74, Best path length: 2.633770590867323
Iteration 75, Best path length: 2.633770590867323
Iteration 76, Best path length: 2.633770590867323
Iteration 77, Best path length: 2.633770590867323
Iteration 78, Best path length: 2.633770590867323
Iteration 79, Best path length: 2.633770590867323
Iteration 80, Best path length: 2.633770590867323
Iteration 81, Best path length: 2.633770590867323
Iteration 82, Best path length: 2.633770590867323
Iteration 83, Best path length: 2.633770590867323
Iteration 84, Best path length: 2.633770590867323
Iteration 85, Best path length: 2.633770590867323
Iteration 86, Best path length: 2.633770590867323
Iteration 87, Best path length: 2.633770590867323
Iteration 88, Best path length: 2.633770590867323
Iteration 89, Best path length: 2.633770590867323
Iteration 90, Best path length: 2.633770590867323
Iteration 91, Best path length: 2.633770590867323
Iteration 92, Best path length: 2.633770590867323
Iteration 93, Best path length: 2.633770590867323
Iteration 94, Best path length: 2.633770590867323
Iteration 95, Best path length: 2.633770590867323
Iteration 96, Best path length: 2.633770590867323
Iteration 97, Best path length: 2.633770590867323
Iteration 98, Best path length: 2.633770590867323
Iteration 99, Best path length: 2.633770590867323
Iteration 100, Best path length: 2.633770590867323

Best path found: [4, 5, 1, 8, 7, 3, 0, 2, 6, 9]
Best path length: 2.633770590867323
```