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

# Define constants for the algorithm
NUM_ANTS = 50
NUM_CITIES = 20  # Now we have 20 cities
ALPHA = 1.0  # Influence of pheromone
BETA = 2.0   # Influence of distance
RHO = 0.1    # Pheromone evaporation rate
Q = 100       # Pheromone deposit constant
MAX_ITER = 100  # Maximum number of iterations

# Predefined 20 cities (coordinates in 2D space)
def generate_cities():
    cities = np.array([
        [5, 10], [11, 5], [14, 9], [12, 15], [8, 13],  # Cities 0-4
        [10, 10], [13, 7], [16, 5], [14, 3], [18, 6],  # Cities 5-9
        [4, 2], [7, 1], [8, 5], [6, 7], [4, 10],  # Cities 10-14
        [15, 18], [12, 17], [3, 18], [17, 12], [19, 8]  # Cities 15-19
    ])
    return cities

# Compute the distance matrix
def compute_distance_matrix(cities):
    num_cities = len(cities)
    distance_matrix = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(i + 1, num_cities):
            dist = np.linalg.norm(cities[i] - cities[j])
            distance_matrix[i, j] = dist
            distance_matrix[j, i] = dist
    return distance_matrix

# Initialize pheromone matrix
def initialize_pheromone_matrix(num_cities):
    pheromone_matrix = np.ones((num_cities, num_cities))  # Pheromone starts as 1 for all edges
    np.fill_diagonal(pheromone_matrix, 0)  # No pheromone on the diagonal (self-loops)
    return pheromone_matrix

# Calculate the total length of a tour
def calculate_tour_length(tour, dist_matrix):
    length = 0
    for i in range(len(tour) - 1):
        length += dist_matrix[tour[i], tour[i + 1]]
    length += dist_matrix[tour[-1], tour[0]]  # Returning to the start
    return length

# Ant solution construction (probabilistic decision on next city)
def construct_solution(num_cities, pheromone_matrix, dist_matrix):
    tour = [random.randint(0, num_cities - 1)]  # Start from a random city
    visited = set(tour)

    while len(tour) < num_cities:
        current_city = tour[-1]
        probabilities = []
        for next_city in range(num_cities):
            if next_city not in visited:
                pheromone = pheromone_matrix[current_city, next_city] ** ALPHA
                distance = (1.0 / dist_matrix[current_city, next_city]) ** BETA
                probabilities.append(pheromone * distance)
            else:
                probabilities.append(0)
```

```
        total_prob = sum(probabilities)
        probabilities = [p / total_prob for p in probabilities]

        # Choose the next city based on the probabilities
        next_city = np.random.choice(range(num_cities), p=probabilities)
        tour.append(next_city)
        visited.add(next_city)

    return tour

# Update the pheromone matrix based on the solutions found by ants
def update_pheromone(pheromone_matrix, all_tours, dist_matrix, best_tour):
    # Evaporate pheromone
    pheromone_matrix *= (1 - RHO)

    # Add pheromone for all ants
    for tour in all_tours:
        tour_length = calculate_tour_length(tour, dist_matrix)
        for i in range(len(tour) - 1):
            pheromone_matrix[tour[i], tour[i + 1]] += Q / tour_length
        pheromone_matrix[tour[-1], tour[0]] += Q / calculate_tour_length(tour, dist_matrix)

    # Add pheromone for the best tour
    best_length = calculate_tour_length(best_tour, dist_matrix)
    for i in range(len(best_tour) - 1):
        pheromone_matrix[best_tour[i], best_tour[i + 1]] += Q / best_length
    pheromone_matrix[best_tour[-1], best_tour[0]] += Q / best_length

# Main ACO algorithm for solving TSP
def ant_colony_optimization(num_cities, dist_matrix, pheromone_matrix, max_iter):
    best_tour = None
    best_tour_length = float('inf')

    # Main loop
    for iteration in range(max_iter):
        all_tours = []

        # Step 1: All ants construct their solutions
        for _ in range(NUM_ANTS):
            tour = construct_solution(num_cities, pheromone_matrix, dist_matrix)
            all_tours.append(tour)
            tour_length = calculate_tour_length(tour, dist_matrix)

            # Step 2: Update the best tour if necessary
            if tour_length < best_tour_length:
                best_tour = tour
                best_tour_length = tour_length

        # Step 3: Update pheromone matrix
        update_pheromone(pheromone_matrix, all_tours, dist_matrix, best_tour)

        # Optional: print progress every 10 iterations
        if iteration % 10 == 0:
            print(f"Iteration {iteration}: Best tour length = {best_tour_length:.2f}")

    return best_tour, best_tour_length


# Main Execution
if __name__ == "__main__":
    # Step 1: Generate predefined cities and distance matrix
    cities = generate_cities()
    dist_matrix = compute_distance_matrix(cities)
```

```
    # Step 2: Initialize pheromone matrix
    pheromone_matrix = initialize_pheromone_matrix(NUM_CITIES)

    # Step 3: Run ACO algorithm
    best_tour, best_tour_length = ant_colony_optimization(NUM_CITIES, dist_matrix, pheromone_matrix, MAX_ITER)

    # Step 4: Output the best tour and visualize it
    print(f"Best tour length: {best_tour_length:.2f}")
```

```
Iteration 0: Best tour length = 107.48
Iteration 10: Best tour length = 81.48
Iteration 20: Best tour length = 80.59
Iteration 30: Best tour length = 80.50
Iteration 40: Best tour length = 79.23
Iteration 50: Best tour length = 79.23
Iteration 60: Best tour length = 77.88
Iteration 70: Best tour length = 77.88
Iteration 80: Best tour length = 77.88
Iteration 90: Best tour length = 77.88
Best tour length: 77.88
```