```
import numpy as np
import random

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]]
    return length


class Particle:
    def __init__(self, num_cities, dist_matrix):

        self.position = np.random.permutation(num_cities)
        self.velocity = np.zeros(num_cities)
        self.best_position = self.position.copy()
        self.best_value = calculate_tour_length(self.position, dist_matrix)


def pso_tsp(cities, dist_matrix, num_particles=50, max_iterations=1000):
    num_cities = len(cities)
    w = 0.5
    c1 = 1.5
    c2 = 1.5

    particles = [Particle(num_cities, dist_matrix) for _ in range(num_particles)]
    global_best_position = particles[0].best_position.copy()
    global_best_value = particles[0].best_value


    for t in range(max_iterations):
        for particle in particles:

            r1, r2 = np.random.rand(num_cities), np.random.rand(num_cities)
            particle.velocity = (w * particle.velocity +
                                 c1 * r1 * (particle.best_position - particle.position) +
                                 c2 * r2 * (global_best_position - particle.position))


            particle.position = update_position(particle.position, particle.velocity)


            tour_length = calculate_tour_length(particle.position, dist_matrix)


            if tour_length < particle.best_value:
                particle.best_value = tour_length
                particle.best_position = particle.position.copy()


            if tour_length < global_best_value:
                global_best_value = tour_length
                global_best_position = particle.position.copy()

    return global_best_position, global_best_value


def update_position(position, velocity):

    i, j = np.random.randint(0, len(position), 2)
    position[i], position[j] = position[j], position[i]
    return position


def define_cities():
    return np.array([
        [10, 10], [20, 20], [50, 30], [40, 40], [80, 60],
        [15, 15], [125, 90], [100, 150], [200, 200], [180, 250],
        [250, 30], [300, 100], [220, 150], [60, 250], [120, 190]
    ])


def compute_distance_matrix(cities):
    num_cities = len(cities)
    dist_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])
            dist_matrix[i, j] = dist
            dist_matrix[j, i] = dist
    return dist_matrix

if __name__ == "__main__":
    cities = define_cities()
    dist_matrix = compute_distance_matrix(cities)

    best_tour, best_tour_length = pso_tsp(cities, dist_matrix, num_particles=50, max_iterations=1000)

    print(f"Best Tour: {best_tour}")
    print(f"Best Tour Length: {best_tour_length}")

print("Vatsal - 1BM22CS323")
```

```
Best Tour: [ 3  2 10 11 12  9  8 14  6  4 13  7  5  0  1]
Best Tour Length: 1298.6719763228218
Vatsal - 1BM22CS323
```

Start coding or generate with AI.