Ant Colony Optimization for the Traveling Salesman Problem

```
import random
import numpy as np

class AntColonyTSP:
    def __init__(
        self,
        coords,
        n_ants=20,
        n_iterations=200,
        alpha=1.0,
        beta=5.0,
        rho=0.5,
        initial_pheromone=1.0,
        q=100.0,
        seed=None
    ):
        if seed is not None:
            random.seed(seed)
            np.random.seed(seed)

        self.coords = np.asarray(coords)
        self.n_cities = len(self.coords)
        self.dist = self._distance_matrix(self.coords)
        self.heuristic = 1.0 / (self.dist + 1e-12)
        np.fill_diagonal(self.heuristic, 0.0)

        self.n_ants = n_ants
        self.n_iterations = n_iterations
        self.alpha = alpha
        self.beta = beta
        self.rho = rho
        self.q = q

        self.pheromone = np.full((self.n_cities, self.n_cities), initial_pheromone, dtype=float)
        self.best_tour = None
        self.best_length = float("inf")

    @staticmethod
    def _distance_matrix(coords):
        n = len(coords)
        D = np.zeros((n, n))
        for i in range(n):
            for j in range(i + 1, n):
                d = np.linalg.norm(coords[i] - coords[j])
```

```
            D[i, j] = d
            D[j, i] = d
    return D

def _tour_length(self, tour):
    L = 0.0
    for i in range(len(tour) - 1):
        L += self.dist[tour[i], tour[i + 1]]
    L += self.dist[tour[-1], tour[0]]
    return L

def _transition_probabilities(self, current, unvisited):
    pher = self.pheromone[current, unvisited] ** self.alpha
    heur = self.heuristic[current, unvisited] ** self.beta
    num = pher * heur
    s = num.sum()
    if s == 0:
        return np.ones(len(unvisited)) / len(unvisited)
    return num / s

def _construct_solutions(self):
    tours = []
    lengths = []
    for _ in range(self.n_ants):
        start = random.randrange(self.n_cities)
        tour = [start]
        unvisited = list(range(self.n_cities))
        unvisited.remove(start)
        while unvisited:
            current = tour[-1]
            unvisited_arr = np.array(unvisited, dtype=int)
            probs = self._transition_probabilities(current, unvisited_arr)
            chosen_idx = np.random.choice(len(unvisited), p=probs)
            next_city = unvisited.pop(chosen_idx)
            tour.append(next_city)
        L = self._tour_length(tour)
        tours.append(tour)
        lengths.append(L)
        if L < self.best_length:
            self.best_length = L
            self.best_tour = tour.copy()
    return tours, lengths

def _update_pheromones(self, tours, lengths):
    self.pheromone *= (1.0 - self.rho)
    for tour, L in zip(tours, lengths):
```

```
            deposit = self.q / (L + 1e-12)
            for i in range(len(tour)):
                a = tour[i]
                b = tour[(i + 1) % self.n_cities]
                self.pheromone[a, b] += deposit
                self.pheromone[b, a] += deposit

    def run(self, verbose=False):
        for it in range(1, self.n_iterations + 1):
            tours, lengths = self._construct_solutions()
            self._update_pheromones(tours, lengths)
            if verbose and (it % max(1, self.n_iterations // 10) == 0):
                print(f"Iteration {it}/{self.n_iterations} best_length={self.best_length:.4f}")
        return self.best_tour, self.best_length

if __name__ == "__main__":
    import matplotlib.pyplot as plt

    n_cities = 20
    seed = 42
    np.random.seed(seed)
    coords = np.random.rand(n_cities, 2) * 100

    aco = AntColonyTSP(
        coords,
        n_ants=40,
        n_iterations=300,
        alpha=1.0,
        beta=5.0,
        rho=0.4,
        initial_pheromone=1.0,
        q=100.0,
        seed=seed
    )

    best_tour, best_len = aco.run(verbose=True)
    print("Best length:", best_len)
    print("Best tour:", best_tour)

    tour_coords = coords[[*best_tour, best_tour[0]]]
    plt.figure(figsize=(8, 6))
    plt.scatter(coords[:, 0], coords[:, 1])
    for i, (x, y) in enumerate(coords):
        plt.text(x + 0.5, y + 0.5, str(i), fontsize=9)
    plt.plot(tour_coords[:, 0], tour_coords[:, 1], marker='o', linestyle='-')
    plt.title(f"ACO TSP solution length {best_len:.3f}")
```

```
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.show()
```

Output:

```
Iteration 30/300 best_length=388.1978
Iteration 60/300 best_length=388.1978
Iteration 90/300 best_length=388.1978
Iteration 120/300 best_length=388.1978
Iteration 150/300 best_length=388.1978
Iteration 180/300 best_length=388.1978
Iteration 210/300 best_length=388.1978
Iteration 240/300 best_length=388.1978
Iteration 270/300 best_length=388.1978
Iteration 300/300 best_length=388.1978
Best length: 388.19775804129887
Best tour: [4, 12, 0, 5, 16, 3, 13, 8, 11, 7, 2, 18, 9, 15, 10, 14, 6, 19, 1, 17]
```