实验三：粒子群优化算法解决旅行商问题

一、实验目的

1. 掌握粒子群优化（PSO）算法的基本原理和实现流程。
2. 运用 PSO 算法解决离散优化问题（旅行商问题，TSP）。
3. 对比 PSO 算法与遗传算法的优劣。

二、实验内容

使用离散粒子群优化算法（DPSO）求解 TSP 问题，通过模拟鸟群寻优行为优化路径，找到访问所有城市的最短路径。

三、实验原理

1. **离散 PSO 算法（排序 PSO）**
   - **粒子表示**：每个粒子的位置为城市索引的排列（如 `[2, 0, 1]` 表示访问顺序为城市 2→城市 0→城市 1）。
   - **速度与位置更新**：通过速度向量的排序生成位置，避免直接处理离散排列的复杂性。速度更新公式为：

$$v_i = w \cdot v_i + c_1 \cdot r_1 \cdot (pbest_i - x_i) + c_2 \cdot r_2 \cdot (gbest_i - x_i)$$

   - **适应度函数**：路径总距离，计算公式为：

$$\text{distance} = \sum_{i=0}^{n-1} \text{distance}(city_i, city_{(i+1)\%n})$$

2. **关键步骤**
   - **初始化**：随机生成粒子速度向量，通过排序得到初始位置。
   - **迭代更新**：根据个体最优（`pbest`）和全局最优（`gbest`）更新速度，重新排序生成新位置。
   - **收敛条件**：达到最大迭代次数或适应度不再显著变化。

代码如下：

```
import numpy as np
import random

class City:
    """城市类，存储坐标并计算距离"""
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_to(self, other):
        """计算到另一城市的欧氏距离"""
        return np.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)
```

```
class Particle:
    """粒子类，表示TSP路径"""
    def __init__(self, n_cities):
        self.n = n_cities
        self.velocity = np.random.randn(n_cities)  # 初始速度为随机实数
        self.position = np.argsort(self.velocity).tolist()  # 位置由速度排序生成
        self.pbest_position = self.position.copy()  # 个体最优位置
        self.pbest_distance = float('inf')  # 个体最优距离

    def calculate_distance(self, cities):
        """计算路径总距离"""
        distance = 0
        for i in range(self.n):
            current = self.position[i]
            next_idx = (i + 1) % self.n
            distance +=
cities[current].distance_to(cities[self.position[next_idx]])
        return distance

class PSO:
    """PSO算法类"""
    def __init__(self, cities, num_particles=30, max_iter=1000, w=0.8, c1=2,
c2=2):
        self.cities = cities
        self.n = len(cities)
        self.num_particles = num_particles
        self.max_iter = max_iter
        self.w = w  # 惯性权重
        self.c1 = c1  # 个体学习因子
        self.c2 = c2  # 群体学习因子
        self.particles = [Particle(self.n) for _ in range(num_particles)]  # 初始
化粒子群
        self.gbest_position = None  # 全局最优位置
        self.gbest_distance = float('inf')  # 全局最优距离
        self._initialize_gbest()

    def _initialize_gbest(self):
        """初始化全局最优"""
        for particle in self.particles:
            dist = particle.calculate_distance(self.cities)
            particle.pbest_distance = dist
            if dist < self.gbest_distance:
                self.gbest_distance = dist
                self.gbest_position = particle.position.copy()

    def update_particles(self):
        """更新粒子速度和位置"""
        for particle in self.particles:
            # 将个体最优和全局最优位置转换为速度向量
            pbest_speed = np.array([particle.pbest_position.index(i) for i in
range(self.n)])
            gbest_speed = np.array([self.gbest_position.index(i) for i in
range(self.n)])

            r1 = np.random.rand(self.n)  # 随机数1
```

```
            r2 = np.random.rand(self.n)  # 随机数2

            # 更新速度
            new_velocity = (self.w * particle.velocity +
                            self.c1 * r1 * (pbest_speed - particle.velocity) +
                            self.c2 * r2 * (gbest_speed - particle.velocity))
            particle.velocity = new_velocity

            # 生成新位置（通过速度排序）
            particle.position = np.argsort(particle.velocity).tolist()

            # 计算新适应度并更新最优解
            new_dist = particle.calculate_distance(self.cities)
            if new_dist < particle.pbest_distance:
                particle.pbest_distance = new_dist
                particle.pbest_position = particle.position.copy()
                if new_dist < self.gbest_distance:
                    self.gbest_distance = new_dist
                    self.gbest_position = particle.pbest_position.copy()

    def run(self):
        """运行PSO算法"""
        for iter_num in range(self.max_iter):
            self.update_particles()
            if iter_num % 100 == 0:
                print(f"Iteration {iter_num}, Best Distance:
{self.gbest_distance:.2f}")
        return self.gbest_position, self.gbest_distance

def main():
    """主函数"""
    n_cities = 10  # 城市数量
    random.seed(42)  # 固定随机种子便于复现
    cities = [City(random.uniform(0, 100), random.uniform(0, 100)) for _ in
range(n_cities)]

    # 初始化PSO参数
    pso = PSO(
        cities=cities,
        num_particles=30,
        max_iter=1000,
        w=0.8,
        c1=2,
        c2=2
    )

    # 运行算法
    best_path, best_dist = pso.run()
    print("\nOptimal Path:", best_path)
    print(f"Shortest Distance: {best_dist:.2f}")

if __name__ == "__main__":
    main()
```

与遗传算法对比

对比维度	粒子群算法（PSO）	遗传算法（GA）
原理	基于群体协作与个体经验更新粒子位置	基于自然选择、交叉变异操作优化种群
收敛速度	更快，迭代初期即可快速逼近最优解	较慢，需更多迭代才能收敛
全局搜索能力	易陷入局部最优（缺乏变异机制）	较强（交叉变异保持种群多样性）
参数复杂度	参数较少（惯性权重、学习因子）	参数较多（交叉率、变异率、选择策略）
实现难度	更简单（无需设计交叉变异算子）	较复杂（需设计适合 TSP 的交叉算子，如 OX、PMX）