重 庆 大 学

学 生 实 验 报 告

实验课程名称　　　　　人工智能导论

开课实验室　　　　　DS1502

学　　　　院 软件学院 年级 2021 专业班 软工 X 班

学 生 姓 名 XXX 学　号 2021XXXX

开 课 时 间 2023 至 2024 学年第 1 学期

总 成 绩	
教师签名	

大数据与软件学院制

《人工智能导论》实验报告

开课实验室：DS1502 **2023 年 12 月 2 日**

<table>
<tr><td>学院</td><td>大数据与软件学院</td><td>年级、专业、班</td><td>21 软件工程 X 班</td><td>姓名</td><td>XXX</td><td>成绩</td><td></td></tr>
<tr><td>课程名称</td><td>人工智能导论</td><td>实验项目名　称</td><td>基于粒子群算法解决有能力约束的车辆路径调度（CVRP)</td><td>指导教师</td><td>XX</td><td></td><td></td></tr>
<tr><td>教师评语</td><td colspan="7">教师签名：
2023 年　月　日</td></tr>
</table>

一、 实验目的

本实验要求用粒子群算法找到成本最低的、满足约束条件的一种车辆调度方式。

二、实验内容

（1）场景：

① 单向：纯送货；

② 单配送中心：只有一个配送中心；

③ 单车型：只考虑一种车型；

④ 需求不可拆分：客户需求只能有一辆车满足；

⑤ 车辆封闭：完成配送任务的车辆需回到配送中心；

⑥ 车辆充足：不限制车辆数量；

⑦ 非满载：任意客户点的需求量小于车辆最大载重；

（2）要求：

① 优化目标：最小化车辆启动成本和车辆行驶成本之和；

② 约束条件：车辆行驶距离约束，重量约束；

③ 已知信息：配送中心位置、客户点位置、客户点需求、车辆最大载重、车辆最大行驶距离、车辆启动成本、车辆单位距离行驶成本；

三、使用仪器、材料

1. 操作系统：Windows 11

2. 开发设备：Lenovo Legion R9000P2021H

3. 开发平台：PyCharm 2023.1

四、实验过程原始记录(数据、图表、计算等)：

（一）源代码

```python
import math
import random
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pylab import mpl

# 设置matplotlib字体为中文
mpl.rcParams['font.sans-serif'] = ['SimHei']

1 个用法
def calculate_distance(city_coordinates):
    # 计算城市之间的距离矩阵
    distance_matrix = pd.DataFrame(index=range(len(city_coordinates)), columns=range(len(city_coordinates)))
    for i in range(len(city_coordinates)):
        xi, yi = city_coordinates[i]
        for j in range(len(city_coordinates)):
            xj, yj = city_coordinates[j]
            distance_matrix.loc[i, j] = round(math.sqrt((xi - xj) ** 2 + (yi - yj) ** 2), 2)
    return distance_matrix

1 个用法
def greedy_algorithm(city_coordinates, distance_matrix):
    # 使用贪婪算法生成车辆路径
    distance_matrix = distance_matrix.astype('float64')
    for i in range(len(city_coordinates)):
        distance_matrix.loc[i, i] = float('inf')  # 将对角线标记为无穷，以防止自环
    distance_matrix.loc[:, 0] = float('inf')  # 将第一列标记为无穷，以防止返回起始城市
    route = []
    current_city = random.randint(1, len(city_coordinates) - 1)
    route.append(current_city)
    distance_matrix.loc[:, current_city] = float('inf')
    for i in range(1, len(city_coordinates) - 1):
        next_city = distance_matrix.loc[current_city, :].idxmin()
        route.append(next_city)
        distance_matrix.loc[:, next_city] = float('inf')
        current_city = next_city
    return route

```

2 用法

```python
def calculate_fitness(bird_pop, demand, distance_matrix, capacity, distance_limit, c0, c1):
    # 计算种群的适应度值
    bird_pop_routes, fitness_values = [], []
    for j in range(len(bird_pop)):
        bird = bird_pop[j]
        routes = []
        route = [0]
        total_distance = 0
        current_distance, load = 0, 0
        i = 0
        while i < len(bird):
            if route == [0]:
                current_distance += distance_matrix.loc[0, bird[i]]
                route.append(bird[i])
                load += demand[bird[i]]
                i += 1
            else:
                if (distance_matrix.loc[route[-1], bird[i]] + distance_matrix.loc[bird[i], 0] + current_distance <= distance_limit) & (
                        load + demand[bird[i]] <= capacity):
                    current_distance += distance_matrix.loc[route[-1], bird[i]]
                    route.append(bird[i])
                    load += demand[bird[i]]
                    i += 1
                else:
                    current_distance += distance_matrix.loc[route[-1], 0]
                    route.append(0)
                    total_distance += current_distance
                    routes.append(route)

                    current_distance, load = 0, 0
                    route = [0]

        current_distance += distance_matrix.loc[route[-1], 0]
        route.append(0)
        total_distance += current_distance
        routes.append(route)

        bird_pop_routes.append(routes)
        fitness_values.append(round(c1 * total_distance + c0 * len(routes), 1))

    return bird_pop_routes, fitness_values

```

1 个用法

```python
def crossover_operator(bird, personal_line, global_line, w, c1, c2):
    # 交叉操作生成新的个体
    crossed_bird = [None] * len(bird)
    parent1 = bird

    rand_num = random.uniform(0, sum([w, c1, c2]))
    if rand_num <= w:
        parent2 = [bird[i] for i in range(len(bird) - 1, -1, -1)]
    elif rand_num <= w + c1:
        parent2 = personal_line
    else:
        parent2 = global_line

    start_pos = random.randint(0, len(parent1) - 1)
    end_pos = random.randint(0, len(parent1) - 1)
    if start_pos > end_pos:
        start_pos, end_pos = end_pos, start_pos
    crossed_bird[start_pos:end_pos + 1] = parent1[start_pos:end_pos + 1].copy()

    list2 = list(range(0, start_pos))
    list1 = list(range(end_pos + 1, len(parent2)))
    list_index = list1 + list2
    j = -1
    for i in list_index:
        for j in range(j + 1, len(parent2) + 1):
            if parent2[j] not in crossed_bird:
                crossed_bird[i] = parent2[j]
                break

    return crossed_bird
```

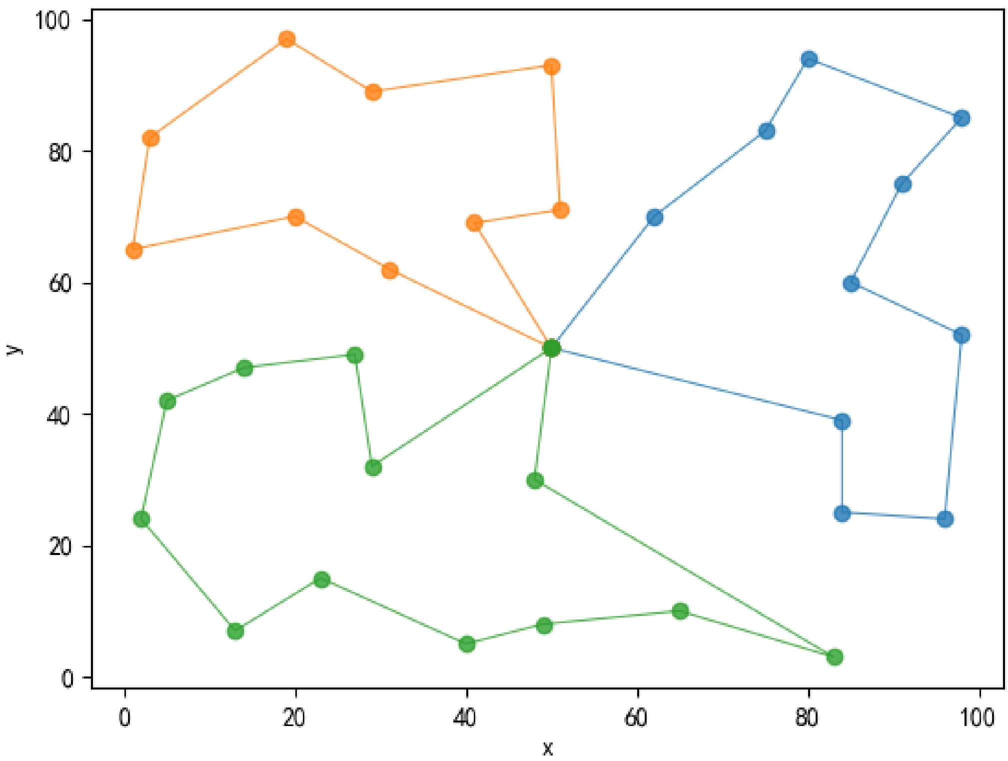
```python

1 个用法
def draw_routes(car_routes, city_coordinates):
    # 绘制车辆路径
    for route in car_routes:
        x, y = zip(*[city_coordinates[i] for i in route])
        plt.plot(x, y, 'o-', alpha=0.8, linewidth=0.8)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

```

```python
if __name__ == '__main__':
    CAPACITY = 120
    DISTANCE_LIMIT = 250
    C0 = 30
    C1 = 1
    bird_num = 50
    w = 0.2
    c1_value = 0.4
    c2_value = 0.4
    personal_best, personal_line = float('inf'), []
    global_best, global_line = float('inf'), []
    max_iterations = 1000
    iteration_count = 1
    best_fitness_values = []

    customer_coordinates = [(50, 50), (96, 24), (40, 5), (49, 8), (13, 7), (29, 89), (48, 30), (84, 39), (14, 47),
                            (2, 24), (3, 82), (65, 10), (98, 52), (84, 25), (41, 69), (1, 65), (51, 71), (75, 83),
                            (29, 32), (83, 3), (50, 93), (80, 94), (5, 42), (62, 70), (31, 62), (19, 97), (91, 75),
                            (27, 49), (23, 15), (20, 70), (85, 60), (98, 85)]
    demand_values = [0, 16, 11, 6, 10, 7, 12, 16, 6, 16, 8, 14, 7, 16, 3, 22, 18, 19, 1, 14, 8, 12, 4, 8, 24, 24, 2, 10, 15, 2, 14, 9]
    #demand_values = [0, 24, 9, 3, 27, 15, 12, 30, 6, 4, 12, 20, 10, 9, 18, 11, 8, 24, 23, 6, 10, 25, 3, 2, 30, 19, 13, 2, 11, 22, 18, 6]
    distance_matrix = calculate_distance(customer_coordinates)

    bird_pop = [greedy_algorithm(customer_coordinates, distance_matrix) for _ in range(bird_num)]

    bird_pop_routes, fitness_values = calculate_fitness(bird_pop, demand_values, distance_matrix, CAPACITY, DISTANCE_LIMIT, C0, C1)

    global_best = personal_best = min(fitness_values)
    global_line = personal_line = bird_pop[fitness_values.index(min(fitness_values))]
    global_line_routes = personal_line_routes = bird_pop_routes[fitness_values.index(min(fitness_values))]
    best_fitness_values.append(global_best)

    while iteration_count <= max_iterations:
        for i in range(bird_num):
            bird_pop[i] = crossover_operator(bird_pop[i], personal_line, global_line, w, c1_value, c2_value)

        bird_pop_routes, fitness_values = calculate_fitness(bird_pop, demand_values, distance_matrix, CAPACITY, DISTANCE_LIMIT, C0, C1)
        personal_best, personal_line, personal_line_routes = min(fitness_values), bird_pop[
            fitness_values.index(min(fitness_values))], bird_pop_routes[
                                                                 fitness_values.index(min(fitness_values))]
        if personal_best <= global_best:
            global_best, global_line, global_line_routes = personal_best, bird_pop[
                fitness_values.index(min(fitness_values))], bird_pop_routes[
                                                                 fitness_values.index(min(fitness_values))]

        best_fitness_values.append(global_best)
        print(iteration_count, global_best)
        iteration_count += 1

    print(global_line_routes)
    draw_routes(global_line_routes, customer_coordinates)
```

（二）实现效果

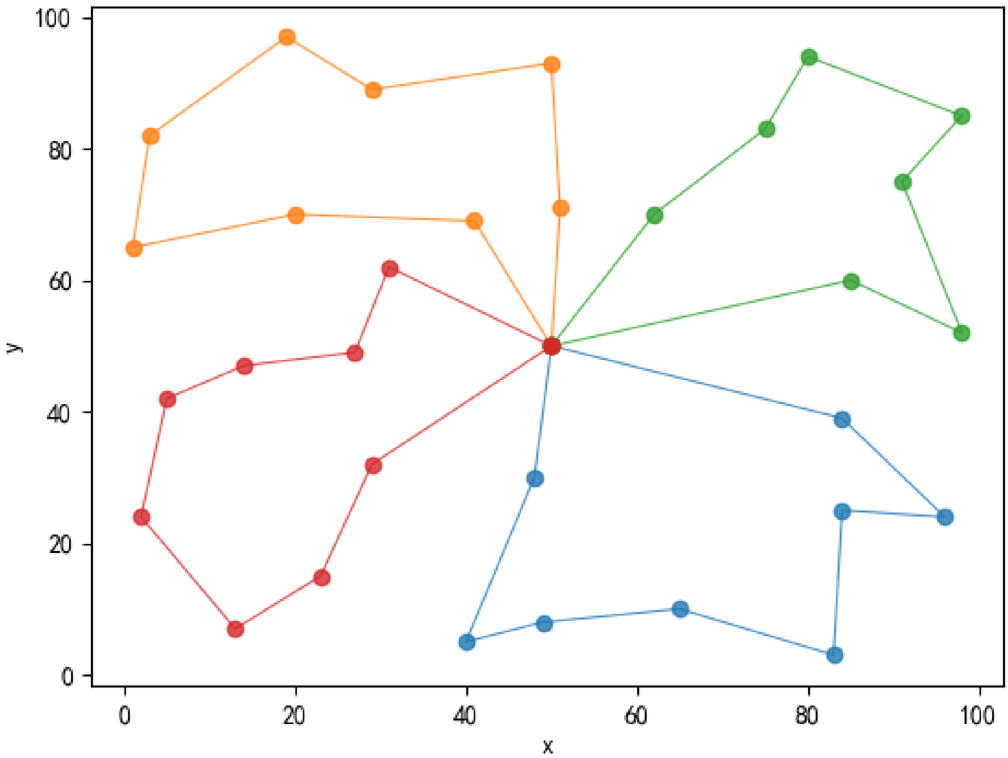
1. 客户需求=[16, 11, 6, 10, 7, 12, 16, 6, 16, 8, 14, 7, 16, 3, 22, 18, 19, 1, 14, 8, 12, 4, 8, 24, 24, 2, 10, 15, 2, 14, 9]时；

```
1000 728.1
[[0, 23, 17, 21, 31, 26, 30, 12, 1, 13, 7, 0], [0, 24, 29, 15, 10, 25, 5, 20, 16, 14, 0], [0, 6, 19, 11, 3, 2, 28, 4, 9, 22, 8, 27, 18, 0]]
```



2. 客户需求=[24, 9, 3, 27, 15, 12, 30, 6, 4, 12, 20, 10, 9, 18, 11, 8, 24, 23, 6, 10, 25, 3, 2, 30, 19, 13, 2, 11, 22, 18, 6]时；

```
1000 796.6
[[0, 6, 2, 3, 11, 19, 13, 1, 7, 0], [0, 14, 29, 15, 10, 25, 5, 20, 16, 0], [0, 30, 12, 26, 31, 21, 17, 23, 0], [0, 24, 27, 8, 22, 9, 4, 28, 18, 0]]
```



3. 总结

本实验旨在用粒子群算法找到成本最低的、满足约束条件的一种车辆调度方式。以下是主要实现的功能：

① 城市距离计算： calculate_distance 函数生成城市之间的距离矩阵，通过欧氏距离计算。

② 贪婪算法生成初始解： greedy_algorithm 函数使用贪婪算法生成初始车辆路径。采用随机选择起始城市，然后选择距离最短的下一个城市，直至所有城市被访问一次。

③ 适应度计算： calculate_fitness 函数计算种群的适应度值。对每个个体（车辆路径），考虑行驶距离和载重情况，计算总行驶距离和路径数量的线性组合作为适应度值。

④ 交叉操作生成新个体： crossover_operator 函数实现了交叉操作，生成新的个体。根据权重随机选择两个个体进行交叉，得到新的路径方案。

⑤ 车辆路径可视化： draw_routes 函数用于绘制车辆路径，以直观展示最优路径。

⑥ 主程序： 定义了问题的具体参数，包括车辆容量、行驶距离限制、适应度函数的权重等。通过迭代粒子群算法，不断更新路径方案，最终输出最优路径并可视化。

⑦ 问题场景与要求： 代码基于一系列约束条件，包括单向纯送货、单配送中心、单车型、需求不可拆分、车辆封闭、车辆充足、非满载等。优化目标是最小化车辆启动成本和车辆行驶成本之和，同时满足距离和载重的约束条件。

总而言之，通过粒子群算法，代码在给定场景下找到了最优的车辆路径，以满足约束条件并达到最小化成本的目标。算法在实际物流调度问题中具有潜在的应用前景。