



南開大學
Nankai University

人工智能技术实验 实验报告

实验名称：基于蚁群算法的旅行商问题

姓名：张恒硕

学号：2212266

专业：智能科学与技术

人工智能学院

2024 年 12 月

目录

一、 问题简述	3
二、 实验目的	4
三、 实验内容	4
四、 编译环境	4
五、 实验步骤与分析	5
1. 参数与设置	5
2. 路径类	6
3. 城市选择函数（轮盘赌选择）	7
4. 更新信息素矩阵函数	7
5. 展示演化过程	8
6. 绘制路径	9
7. 主函数	10
六、 实验结果	12
1. 旅行商问题调参求解	12
1) 蚂蚁数量	12
2) 迭代次数	13
3) 信息素总量	14
4) 信息素挥发系数	15
5) 因子	16
6) 更新方法	17
2. 可视化展示	18
七、 分析总结	18
1. 成果总结	18
2. 问题分析与改进思路	18

一、问题简述

旅行商问题：设有 N 个互相可直达的城市，给定每对城市之间的距离，某推销商准备从其中的 A 城出发，周游各城市一遍，最后又回到 A 城，要求访问每一座城市并回到起始城市的最短回路。这是非常经典的组合优化问题中的 NP 难问题（多项式复杂程度的非确定性问题），通过常规的暴力枚举，遍历等方法难以计算出最优解，因此，本实验使用蚁群算法进行最优解的计算。

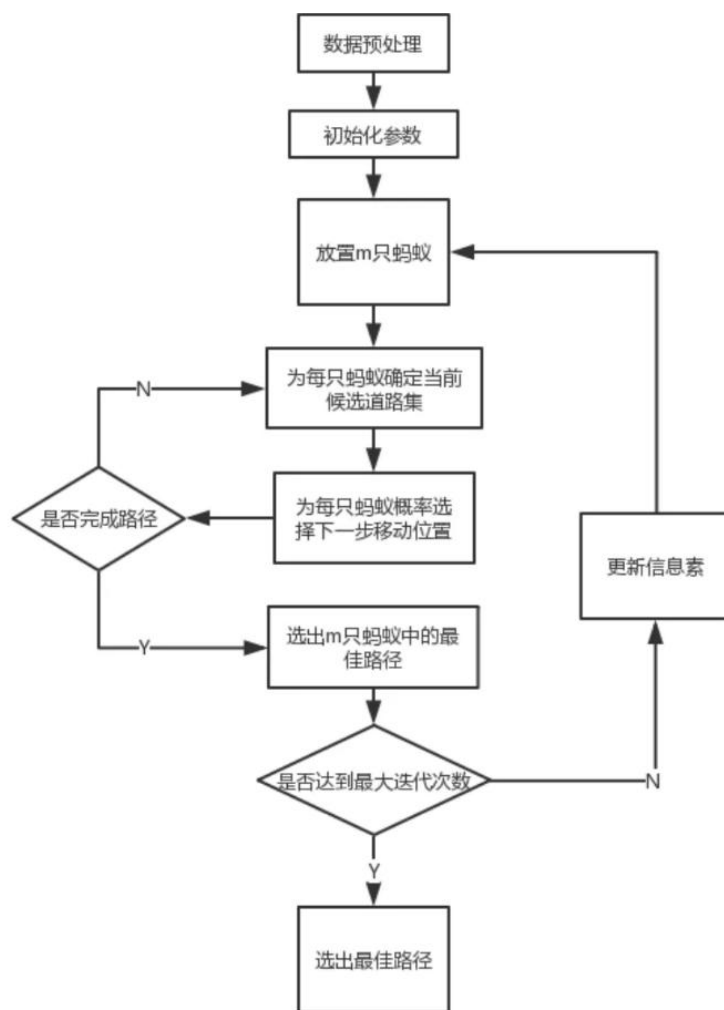
蚁群算法(Ant Colony Optimization, ACO)是一种基于群体智能的优化算法，由 Marco Dorigo 于 1992 年提出。蚁群算法模拟了自然界中蚂蚁群体寻找最短路径的行为，利用蚂蚁在行走过程中留下的信息素来引导后续蚂蚁选择路径。通过不断迭代和信息素更新，算法逐渐找到问题的最优解或近似最优解。

蚁群算法的核心思想是利用信息素(Pheromone)的正反馈机制来逐步优化解。蚂蚁在行走过程中会释放信息素，信息素的浓度与路径的质量（通常是路径的长度或花费的时间）有关。路径越短，信息素越浓，后续蚂蚁选择该路径的概率就越高。这种机制可以通过多次迭代不断强化优质路径，逐渐逼近最优解。

对于本实验，求解我国 34 个城市的旅行商问题，给定 34 个城市的名称和坐标数据，求遍历一次所有城市回到起点期间，所经过路程的较小值。

基本思想：

1. 根据具体问题设置多只蚂蚁，分头并行搜索。
2. 每只蚂蚁完成一次周游后，在行进的路上释放信息素，信息素量与解的质量成正比。
3. 蚂蚁路径的选择根据信息素强度大小（初始信息素量设为相等），同时考虑两点之间的距离，采用随机的局部搜索策略。这使得距离较短的边，其上的信息素量较大，后来的蚂蚁选择该边的概率也较大。
4. 每只蚂蚁只能走合法路线（经过每个城市 1 次且仅 1 次）。
5. 所有蚂蚁都搜索完一次就是迭代一次，每迭代一次就对所有的边做一次信息素更新，然后新的蚂蚁进行新一轮搜索。
6. 更新信息素包括原有信息素的蒸发和经过的路径上信息素的增加。
7. 达到预定的迭代步数，或出现停滞现象（所有蚂蚁都选择同样的路径，解不再变化），则算法结束，以当前最优解作为问题的最优解。



二、 实验目的

1. 了解旅行商问题的基本概念，理解旅行商问题的求解难度；
2. 了解蚁群算法的基本原理及实现流程；
3. 能够利用蚁群算法解决旅行商问题；
4. 进一步理解可视化设计。

三、 实验内容

1. 实现基于蚁群算法的 34 个城市的旅行商问题，求解访问每一座城市一次并最终回到起始城市的最短回路；
2. 对比蚁群算法下不同蚂蚁数量 m 、信息素挥发系数 ρ 、信息素总量 Q 、信息素因子 α 、启发函数因子 β 等参数下，蚁群算法对问题求解的效果影响并分析原因。
3. 进行蚁群算法可视化，展示搜索过程中每次迭代过程中种群最优路径走势的变化和迭代最优结果。

四、 编译环境

Python3.11+numpy+pandas+matplotlib+cartopy

cartopy 用于获取真实投影地图来绘制路径，不使用则 plot_path 函数无法直接使用，对其他部分没有影响。

五、 实验步骤与分析

1. 参数与设置

● 内容与原理

设置绘图使用的字体，设置蚁群算法参数，读取城市经纬距离矩阵和城市经纬度两个文件的数据。

● 代码

```
# 设置中文字体

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode_minus'] = False


# 参数

length = 34      # 城市数量

num_ants = 30    # 蚂蚁数量

generations = 100 # 迭代次数

Q = 10           # 信息素释放总量

rho = 0.5        # 信息素挥发因子

alpha = 1        # 信息素重要性因子

beta = 5         # 启发式信息重要性因子


# 读取数据

file_path_distances = '城市经纬距离矩阵.xlsx'
```

```

df_distances = pd.read_excel(file_path_distances, index_col=0)

distance_matrix = df_distances.iloc[:length, :length].values

city_names = df_distances.index[:length]

file_path_cities = '城市经纬度.xlsx'

df_cities = pd.read_excel(file_path_cities, index_col=0)

```

- 两个文件存储的数据分别用作蚁群算法的距离矩阵和绘制地图的位置标定。

2. 路径类

● 内容与原理

定义路径类，其包含一个蚂蚁选择的城市序列和对应的总距离。

● 代码

```

# 路径类

class Path:

    def __init__(self, path):

        self.path = path

        self.len = None

    def length(self):

        if self.len is None:

            self.len = sum([distance_matrix[self.path[i]][self.path[(i + 1) % length]] for i
in range(length)])

        return self.len

```

- 生成路径时只需给出相应的城市序列。
- 路径生成后，在第一次被访问时会计算总距离值，之后储存起来，减少

评估耗时。总距离值通过计算整个回路的路径长度获得。

3. 城市选择函数（轮盘赌选择）

● 内容与原理

蚂蚁在当前城市时，会通过信息素浓度信息和启发式信息综合选择下一个去往的城市，在概率选择时使用轮盘赌选择方法。

● 代码

```
# 选择下一个城市：轮盘赌选择

def select_next_city(current_city, unvisited_cities):

    pheromones = pheromone_matrix[current_city][unvisited_cities]    # 信息素部分

    heuristic = 1 / distance_matrix[current_city][unvisited_cities]  # 启发式信息部分（距离倒数）

    probabilities = (pheromones ** alpha) * (heuristic ** beta)        # 用因子指数化

    probabilities /= probabilities.sum()    # 归一化

    next_city_index = np.random.choice(len(unvisited_cities), p=probabilities) # 轮盘赌选择

    return unvisited_cities[next_city_index]
```

- 在当前城市节点 i 时，去往城市节点 j 的选择因数如下：

$$m_{ij} = (\tau_{ij})^{\alpha} (\eta_{ij})^{\beta}$$

其中， τ_{ij} 是两个节点间的信息素浓度， $\eta_{ij} = \frac{1}{d_{ij}}$ 是两个节点间的距离倒数，表示启发信息。 α 、 β 是调节二者权重的指数因子。

- 依据去往各可行城市的选择因数进行轮盘赌选择：

$$P_{ij} = \frac{m_{ij}}{\sum_{k \in allowed} m_{ik}}$$

4. 更新信息素矩阵函数

● 内容与原理

在更新信息素浓度时，首先按一定比例挥发，其次计算各蚂蚁在各边释放的信息素。

● 代码

更新信息素矩阵

```
def update_pheromones(paths):
```

```
    global pheromone_matrix      # 全局变量保证更改
```

```
    pheromone_matrix *= 1 - rho   # 挥发
```

```
    # 增加
```

```
    for path in paths:           # 所有蚂蚁
```

```
        for j in range(length): # 所有边
```

```
            start, end = path.path[j], path.path[(j + 1) % length]
```

```
            pheromone_matrix[start][end] += Q / path.length() # 释放总量除以距离
```

```
            pheromone_matrix[end][start] = pheromone_matrix[start][end] # 对称
```

矩阵

- 更新信息素矩阵时，最重要的是保证信息素矩阵是全局变量，更改可以作用到函数外。
- 由于信息素矩阵是对称的，在更新一个城市去往另一个城市的信息素时，也要同步更新返程的。
- 挥发时，整个矩阵按同一比例进行挥发。
- 增加时，需要遍历所有蚂蚁的路径，在其经过的边上，增加单位长度的释放总量：

$$\Delta\tau_{ij} = \sum_{k=1}^m \frac{Q}{L^k}$$

其中， L^k 是该路径的总长度， Q 是蚂蚁一次搜索释放的信息素总量。

5. 展示演化过程

● 内容与原理

使用世代最优距离数据绘制迭代演化过程。

● 代码


```
# 演化过程
```

```
def evolution(best_lengths):
```

```
    plt.figure(figsize=(10, 6))
```

```
    plt.plot(range(1, len(best_lengths) + 1), best_lengths)
```

```
    plt.xlabel('迭代次数')
```

```
    plt.ylabel('最短距离')
```

```
    plt.title('演化过程')
```

```
    plt.show()
```

6. 绘制路径

- 内容与原理

使用 `cfeature` 获取地图投影，标记城市坐标和城市名，绘制路径。

- 代码

```
# 绘制路径
```

```
def plot_path(best_path, city_names, distance_matrix):
```

```
    # 模拟地图
```

```
    fig = plt.figure(figsize=(10, 8))
```

```
    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
```

```
    ax.add_feature(cfeature.COASTLINE)
```

```
    ax.add_feature(cfeature.LAND, edgecolor='black')
```

```
    ax.add_feature(cfeature.OCEAN)
```

```
    # 绘制城市点并标注城市名
```

```
    path = [city_names[i] for i in best_path.path]
```

```
    lats = distance_matrix.loc[path, 'Latitude'].values
```

```

lons = distance_matrix.loc[path, 'Longitude'].values

ax.scatter(lons, lats, 60, marker='o', color='k', zorder=5,
transform=ccrs.PlateCarree(), label='城市')

for i, name in enumerate(path):

    ax.text(lons[i], lats[i], name, fontsize=10, ha='right', va='bottom',
transform=ccrs.PlateCarree())

# 绘制路径（回到起点）

ax.plot(lons, lats, 'r-', linewidth=2, label='路径', transform=ccrs.PlateCarree())

ax.plot([lons[-1], lons[0]], [lats[-1], lats[0]], 'r-', linewidth=2,
transform=ccrs.PlateCarree())

plt.show()

```

- 模拟地图的部分并不必须，可以去掉。
- 在绘制路径的时候，需要保证回到起点。

7. 主函数

● 内容与原理

调用前面的函数，进行蚂蚁搜索、信息素浓度更新的迭代过程，记录世代最优，并可视化展示。

● 代码

```

# 主程序

start_time = time.time()

pheromone_matrix = np.ones((length, length)) # 信息素矩阵

best_length_history = [] # 世代最优

best_path = Path(list(range(34))) # 保留迭代过程中的最优个体

```

```
for generation in range(generations):

    paths = []

    for ant in range(num_ants):

        # 每个蚂蚁逐步构建自己的路径

        current_city = np.random.randint(0, length)

        unvisited_cities = set(range(length)) - {current_city}

        path = [current_city]

        while unvisited_cities:

            next_city = select_next_city(current_city, list(unvisited_cities))

            path.append(next_city)

            unvisited_cities.remove(next_city)

            current_city = next_city

        full_path = Path(path)

        paths.append(full_path)

        # 比较最优

        if full_path.length() < best_path.length():

            best_path = full_path

    # 更新信息素矩阵（循环结束后全局更新）

    update_pheromones(paths)

    # 最佳路径

    best_length_history.append(best_path.length())

    if (generation + 1) % 10 == 0:
```

```

        print(f'第{generation + 1}轮: 当前最优方案{best_path.path}的花费为
{best_path.length():.5f}')

# 输出最终结果

print(f'最短路径:{best_path.path}')

print(f'最短距离:{best_path.length():.5f}经纬距离")

end_time = time.time()

print(f'运行时间:{end_time - start_time:.5f}s")

evolution(best_length_history)

plot_path(best_path, city_names, df_cities)

```

- 初始化信息素矩阵为 1 阵。
- 一个轮次的所有蚂蚁搜索完路径后，一起更新信息素。这里还有其他更新逻辑，在以下有进行对比。

六、实验结果

1. 旅行商问题调参求解

基础参数（调节某个参数时，会保证其他参数如下）

蚂蚁数量 num_ants=30

迭代次数 generations=10

信息素释放总量 Q=10

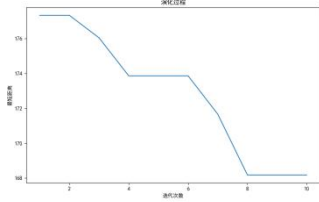

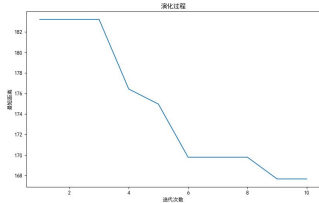
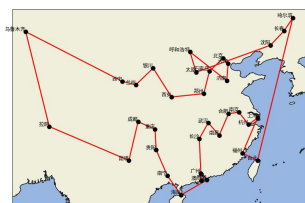


信息素挥发因子 rho=0.5

信息素重要性因子 alpha=1

启发式信息重要性因子 beta=5

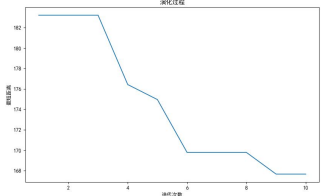

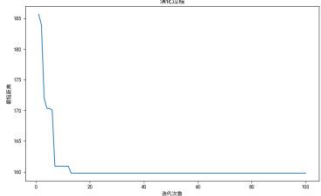

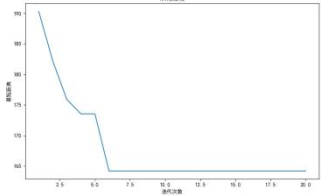
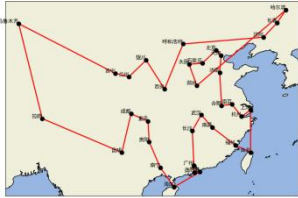
1) 蚂蚁数量

nu	最 短 距	演化过程	最优路径	运 行 时
----	-------	------	------	-------

m_ants	离			间 (s)
20	168.18298			0.11601
30	167.68280			0.17838
40	162.57633			0.24967
<p>分析：蚂蚁数量与构建路径次数成正比，更多的蚂蚁会带来更长的运行时间，但同时也会使长短边的信息素浓度差异累积得更多，带来更优质的解。</p>				

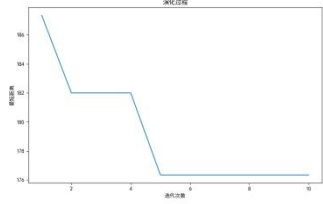

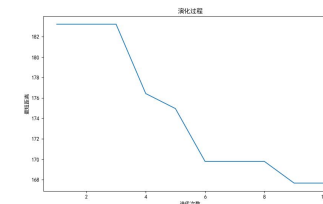

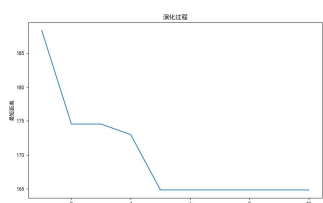
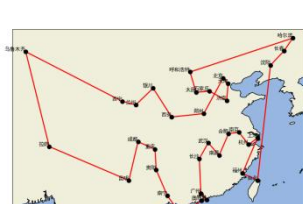
2) 迭代次数

generation	最短距离	演化过程	最优路径	运行时间 (s)

10	167.682 80			0.17838
10 0	159.787 13			1.93177
迭代 到 收 敛	164.187 31（20 轮，连续 十代结 果改变 小于1）			0.38214
<p>分析：实验推荐的100轮迭代次数是不必要的，第一轮的结果已经达到180左右，而在20轮左右便收敛到165以下。迭代次数与运行时间成正比，更多的迭代次数在一定程度上可以带来更好的结果。</p>				

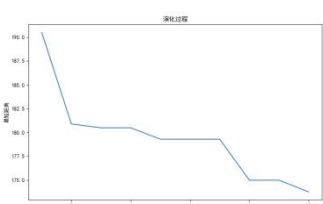
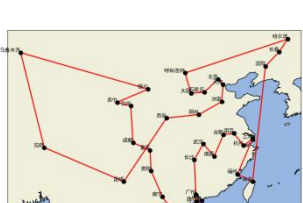
3) 信息素总量

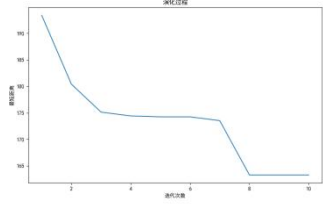

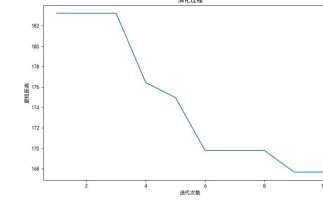

Q	最 短 距 离	演化过程	最优路径	运 行 时 间 (s)
---	------------	------	------	----------------

1	176.339 08			0.19270
10	167.682 80			0.17838
10 0	164.827 91			0.19956

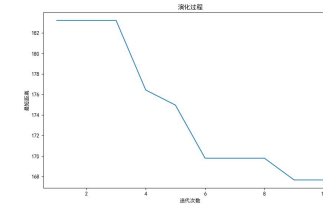

分析：信息素总量与每次更新后长短边的信息素浓度差有关。其较大时，差异累积快，收敛迅速；其较小时，迭代慢，但多样性好，可能收敛到更优质的解。实验对比中，由于迭代次数并未达到收敛次数，信息素总量越大越好。

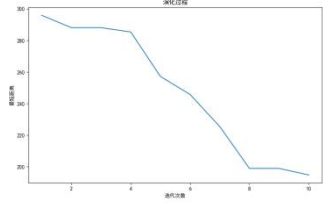
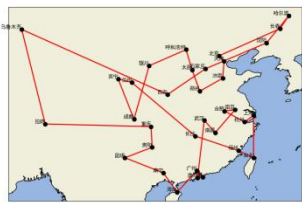
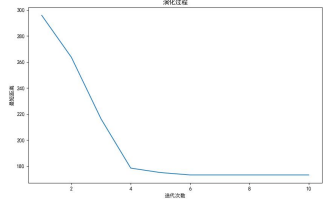
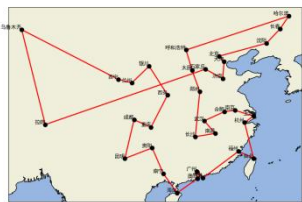
4) 信息素挥发系数

rho	最短距离	演化过程	最优路径	运行时间 (s)
0.2	173.745 56			0.19657

0.35	163.306 26			0.18513
0.5	167.682 80			0.17838
分析：信息素挥发系数会减弱长短边信息素浓度差异的累积速度，减缓收敛，并带来多样性。				

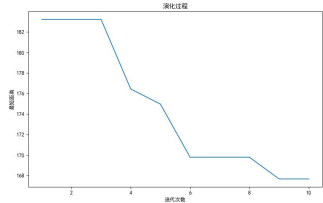
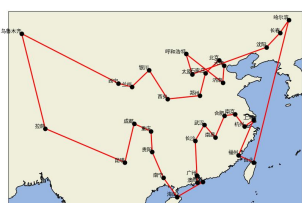
5) 因子

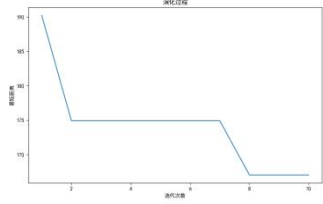
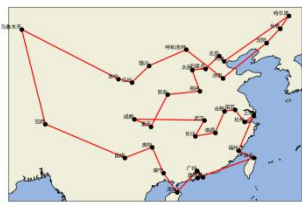
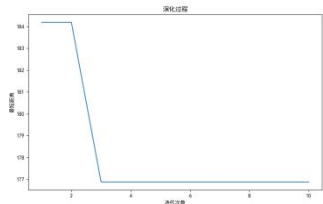

alpha: beta	最短距离	演化过程	最优路径	运行时间 (s)
1: 5	167.682 80			0.17838

1: 1	194.828 04			0.18601
5: 1	173.164 69			0.19866

分析：启发式信息和信息素浓度信息都会给蚂蚁的路径选择带来指引。在本问题中，放大任何一个的占比，都会提高选取短边的可能性，加快收敛。在其他的问题中，增加启发式信息的比重，会提高短边的选择概率；而增加信息素浓度信息，会增加重复选择相同边的概率。这两种增加都会加快收敛。

6) 更新方法

更新方法	最短距离	演化过程	最优路径	运行时间 (s)
每轮集体更新	167.682 80			0.17838

每轮最优更新	167.02502			0.17797
每个立刻更新	176.87651			0.17919
分析：不同的更新方式会对信息素的应用逻辑产生影响。以上采用的都是每轮集体更新。每轮最优更新会带来更强的定向性，能加快一个方向的收敛。每个立刻更新相当于单个蚂蚁迭代更多次数，信息素矩阵更加频繁地更新。				

2. 可视化展示

以上展示了多组可视化结果，这里不作额外展示。

七、 分析总结

1. 成果总结

本次实验成功实现了 TSP 问题的蚁群算法求解和相应的可视化，并且对参数进行了调试。

2. 问题分析与改进思路

- 第一轮即取得良好结果：蚁群算法的种群初始化并不是随机初始化，在第一轮便使用启发式信息构建路径，因此获得优质初始种群。
- 与遗传算法比较：在本问题中，蚁群算法比遗传算法更高效，因为它更丰富地利用了启发式信息，而不是依赖随机性和交叉变异带来的可能优质区间。