“旅行商”问题实验报告

21215122，何峙，大数据与人工智能

**实验内容**

旅行商问题是指一名商人要到若干城市去推销商品，已知城市个数和各城市间的路程（或旅费），要求找到一条从最开始城市出发，经过所有城市且每个城市只能访问一次，最后回到最开始城市的路线，使总的路程（或旅费）最小。

**实验设计**

本实验使用模拟退火算法求解。模拟退火算法的核心思想为：算法迭代过程中，以一定概率接受比当前解更差的解，然后使用这个更差的解继续搜索。现利用模拟退火算法步骤对旅行商问题建模：

1. 解空间

设经过每个城市形成一个路径为一个解，形成某个排序s\_i = {c1, c2, c3, ....}，其中ci代表每个城市，则解空间为所有这些排列的集合S = {s\_1, s\_2, s\_3, .....}。

1. 初始解

模拟退火算法的最优解与初始解无关，故初始解可为随机生成的一个排列，如s\_0 = {c1, c2, c3, ....}。

1. 目标函数

目标函数即为所有城市的路径总长度（或总旅费）：

cost(c1, c2, ...., cn) =

其中d为计算两个城市之间的距离函数。

1. 新解的产生

本实验采用将两个城市的逆序变换作为新解：任选城市序号j和k，交换j和k的访问顺序，即若交换前解为s\_i = {c1, c2, ..., cj, ...., ck, ...., cn}，则交换后解为s\_i′ = {c1, c2, ..., ck, ...., cj, ...., cn}。

1. 损失函数

目标函数差即为变换前的解和变化后的解的损失函数之差：

Δc = cost(s\_i′) - cost(s\_i)

1. Metropolis接受准则

接受当前解的概率为 p =

其中t用于控制退火速度，其迭代规则为 t := a \* t， 0< a < 1

**实验结果**

本实验使用一组随机创建的坐标数据对的列表作为每个城市的坐标，形如:

coordinates = [[x1, y1], [x2, y2], ...,]。

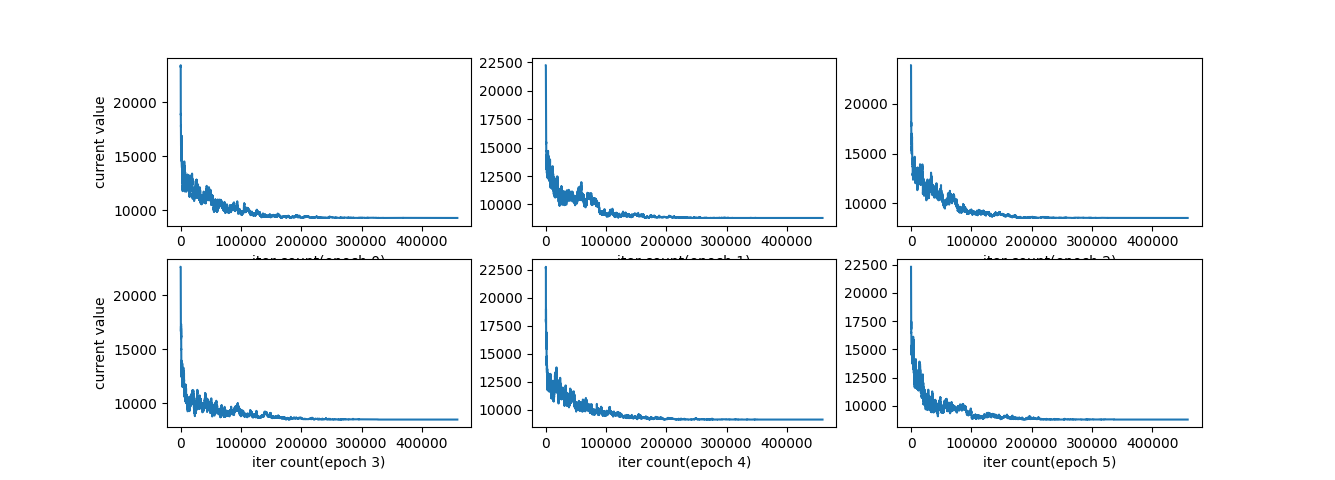


图1 算法各次求最短距离变化趋势

图1显示了跑了6次模拟退火算法的旅行商最短距离随迭代次数变化的趋势。可以看出所求最短距离都是不断减少，符合算法预期，即模拟退火算法是可以求解旅行商问题的。

表1 算法各次求得的最短距离

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch | 1 | 2 | 3 | 4 | 5 | 6 |
| 最短距离 | 9268.03 | 8816.41 | 8521.08 | 8472.09 | 9136.44 | 8785.92 |

表1显示了算法每次所求的最短距离都不一样，说明模拟退火算法并不能求得精确解，只能迫近最优解。

**附代码**

*import numpy as np*

*import os*

*import matplotlib.pyplot as plt*

*# 获取点坐标数据*

*def get\_coordinates(file\_path: str) -> np.ndarray:*

*coordinates = np.array([[565.0,575.0],[25.0,185.0],[345.0,750.0],[945.0,685.0],[845.0,655.0],*

*[880.0,660.0],[25.0,230.0],[525.0,1000.0],[580.0,1175.0],[650.0,1130.0],*

*[1605.0,620.0],[1220.0,580.0],[1465.0,200.0],[1530.0, 5.0],[845.0,680.0],*

*[725.0,370.0],[145.0,665.0],[415.0,635.0],[510.0,875.0],[560.0,365.0],*

*[300.0,465.0],[520.0,585.0],[480.0,415.0],[835.0,625.0],[975.0,580.0],*

*[1215.0,245.0],[1320.0,315.0],[1250.0,400.0],[660.0,180.0],[410.0,250.0],*

*[420.0,555.0],[575.0,665.0],[1150.0,1160.0],[700.0,580.0],[685.0,595.0],*

*[685.0,610.0],[770.0,610.0],[795.0,645.0],[720.0,635.0],[760.0,650.0],*

*[475.0,960.0],[95.0,260.0],[875.0,920.0],[700.0,500.0],[555.0,815.0],*

*[830.0,485.0],[1170.0, 65.0],[830.0,610.0],[605.0,625.0],[595.0,360.0],*

*[1340.0,725.0],[1740.0,245.0]])*

*return coordinates*

*# 计算点与点之间（城市之间）距离矩阵*

*def getdismat(coordinates: np.ndarray) -> np.ndarray:*

*num = coordinates.shape[0]*

*distmat = np.zeros((num, num))*

*for i in range(num):*

*place\_i = coordinates[i]*

*for j in range(num):*

*place\_j = coordinates[j]*

*distmat[i][j] = np.sqrt(np.power(place\_i[0] - place\_j[0], 2) + np.power(place\_i[1] - place\_j[1], 2))*

*return distmat*

*# 进行模拟退火过程*

*def sa\_run():*

*coordinates = get\_coordinates('./a280.tsp')*

*num = coordinates.shape[0]*

*dist\_mat = getdismat(coordinates)*

*solution\_new = np.arange(num)*

*solution\_current = solution\_new.copy()*

*solution\_best = solution\_new.copy()*

*value\_current = 9999999*

*value\_best = 9999999*

*alpha = 0.99*

*t\_range = (1, 100)*

*markovlen = 1000*

*t = t\_range[1]*

*epochcount = 6*

*epoch\_best = []*

*epoch\_current = []*

*for epoch in range(epochcount):*

*result\_best = [] # 记录迭代过程中的最优解*

*result\_current = []*

*while t > t\_range[0]:*

*for i in range(markovlen):*

*# 使用将两个坐标逆序的方式产生新解*

*while True:*

*loc1 = int(np.ceil(np.random.rand() \* (num - 1)))*

*loc2 = int(np.ceil(np.random.rand() \* (num - 1)))*

*if loc1 != loc2:*

*break*

*solution\_new[loc1], solution\_new[loc2] = solution\_new[loc2], solution\_new[loc1]*

*value\_new = 0*

*for j in range(num - 1):*

*value\_new += dist\_mat[solution\_new[j]][solution\_new[j + 1]]*

*value\_new += dist\_mat[solution\_new[0]][solution\_new[num - 1]]*

*if value\_new < value\_current:*

*# 接受该解*

*# print('accept1')*

*value\_current = value\_new*

*solution\_current = solution\_new.copy()*

*if value\_new < value\_best:*

*value\_best = value\_new*

*solution\_best = solution\_new.copy()*

*else:*

*# 以一定概率接受该解*

*if np.random.rand() < np.exp(-(value\_new - value\_current) / t):*

*# print('accept2')*

*value\_current = value\_new*

*solution\_current = solution\_new.copy()*

*else:*

*# print('not accept')*

*solution\_new = solution\_current.copy()*

*result\_current.append(value\_current)*

*t = alpha \* t*

*result\_best.append(value\_best)*

*print(f't: {t}, value: {value\_current}')*

*epoch\_current.append(result\_current)*

*epoch\_best.append(result\_best)*

*print(f'epoch {epoch} finish!!!!!!!!')*

*solution\_new = np.arange(num)*

*solution\_current = solution\_new.copy()*

*solution\_best = solution\_new.copy()*

*value\_current = 9999999*

*value\_best = 9999999*

*t = t\_range[1]*

*# print(f'best values: {value\_best}')*

*# print(f'best solution: {solution\_best}')*

*fig\_col = 3 # 每行多少个子图*

*fig\_row = epochcount // fig\_col*

*fig = plt.figure()*

*ax = fig.subplots(fig\_row, fig\_col)*

*for r in range(fig\_row):*

*for c in range(fig\_col):*

*index = r \* fig\_col + c*

*if c == 0:*

*ax[r, c].set\_ylabel('current value')*

*ax[r, c].set\_xlabel(f'iter count(epoch {index})')*

*ax[r, c].plot(np.array(epoch\_current[index]))*

*plt.show()*

*for i in range(len(epoch\_best)):*

*print(f'best value {i}: {epoch\_best[i][-1]}')*

*if \_\_name\_\_ == '\_\_main\_\_':*

*sa\_run()*