

**软件学院学生实验报告**

**实验课程名称：** 智能算法设计 **教师： 肖如良**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **背包问题的分布估计算法** | | | **实验成绩** |  |
| **学生姓名** | **高强** | **学 号** | **123032014010** | **年级专业班级** | **软三** |
| **小组成员** |  | | | **实验日期** | **2016.11.22** |

1. **实验目的**

旅行商问题的解决的模拟退火算法：

TSP 问题描述为：设有n个城市的集合city={C1, C2，…，Cn}，对于城市Ci，Cj∈city，从 Ci 到 Cj 的距离记为 dij∈R+，这里假设 dij=dji，即考虑对称 TSP 问题。 TSP 问 题的解， 就是在集合 city 中找到一个不重复的全排列 Ck1，Ck2，…，Ckn，使其之间距离最短,也就是要确定一条经过每个城市的当且仅当一次并返回出发点的最短路线。

1. **实验软硬件环境**

Windows,PyCharm,python

1. **实验内容及要求**

选择一种熟悉的语言实现解决关于旅行商问题算法。实验过程要记录其最终结果，并将结果可视化。实验有代码部分。

1. **实验记录**

**1.模拟退火算法（SA）实现**

1. 给定初始温度 T0，及初始点向量，计算该点的函数值f（x)；

初始化温度为200，利用多次试验之后总结得到相应结果。f（x）为按照相应书序访问各个城市之后的总路径长度之和。

（2）随机产生扰动 Δx，得到新点 x′=x+Δx，计算新 点函数值 f（x′），及函数值差 Δ f=f（x′）-f（x）；

（3）若Δf≤0，则接受新点，作为下一次模拟的初始点;

（4）若 Δf>0，则计算新点接受概率:p（Δf）=exp（-Δf K·T）,产生[0,1] 区间上均匀分布的伪随机数 r, r∈[0,1], 如果 p（Δf）≥r,则接受新点作为下一次模拟的初始点; 否则放弃新点, 仍取原来的点作为下一次模拟的初始点。

**2.实现代码**

**import numpy as np**

**import matplotlib.pyplot as plot**

**import math**

**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]])**

**def getdistmat(coordinates):**

**num = coordinates.shape[0]**

**distmat = np.zeros((52, 52))**

**for i in range(num):**

**for j in range(i, num):**

**distmat[i][j] = distmat[j][i] = np.linalg.norm(coordinates[i] - coordinates[j])#求矩阵的范数例如：**

**#（（3,4）的范数为5）**

**return distmat**

**def initpara()：**

**t = (1, 200) #初始化温度状态，开始为200**

**markovlen = 10000**

**return alpha, t, markovlen**

**num = coordinates.shape[0] #求矩阵的第一纬空间的长度**

**distmat = getdistmat(coordinates) #求各点之间的距离**

**solutionnew = np.arange(num) #记录了各个城市的访问顺序**

**valuenew = np.max(100000000) #开始取值为一个足够大的数值**

**solutioncurrent = solutionnew.copy()**

**valuecurrent = np.max(100000000)**

**solutionbest = solutionnew.copy()**

**valuebest = np.max(100000000)**

**alpha, t2, markovlen = initpara()**

**t = t2[1]**

**result = [] # 记录迭代过程中的最优解**

**index = 0**

**while t > t2[0]:**

**for i in np.arange(markovlen):**

**# 下面的两交换和三角换是两种扰动方式，用于产生新解**

**if np.random.rand() > 0.5: # 两交换**

**# np.random.rand()产生[0, 1)区间的均匀随机数**

**while True: # 产生两个不同的随机数**

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

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

**if loc1 != loc2:**

**break**

**solutionnew[loc1], solutionnew[loc2] = solutionnew[loc2], solutionnew[loc1]**

**else: # 三交换**

**while True:**

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

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

**loc3 = np.int(np.ceil(np.random.rand() \* (num - 1)))**

**if ((loc1 != loc2) & (loc2 != loc3) & (loc1 != loc3)):**

**break**

**# 下面的三个判断语句使得loc1<loc2<loc3**

**if loc1 > loc2:**

**loc1, loc2 = loc2, loc1**

**if loc2 > loc3:**

**loc2, loc3 = loc3, loc2**

**if loc1 > loc2:**

**loc1, loc2 = loc2, loc1**

**# 下面的三行代码将[loc1,loc2)区间的数据插入到loc3之后**

**tmplist = solutionnew[loc1:loc2].copy()**

**solutionnew[loc1:loc3 - loc2 + 1 + loc1] = solutionnew[loc2:loc3 + 1].copy()**

**solutionnew[loc3 - loc2 + 1 + loc1:loc3 + 1] = tmplist.copy()**

**valuenew = 0**

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

**valuenew += distmat[solutionnew[j]][solutionnew[j + 1]]**

**valuenew += distmat[solutionnew[0]][solutionnew[51]]**

**if valuenew < valuecurrent: # 接受该解**

**# 更新solutioncurrent 和solutionbest**

**valuecurrent = valuenew**

**solutioncurrent = solutionnew.copy()**

**if valuenew < valuebest:**

**valuebest = valuenew**

**solutionbest = solutionnew.copy()**

**else: # 按一定的概率接受该解**

**if np.random.rand() < np.exp(-(valuenew - valuecurrent) / t):**

**valuecurrent = valuenew**

**solutioncurrent = solutionnew.copy()**

**else:**

**solutionnew = solutioncurrent.copy()**

**index += 1**

**t /= math.log(1 + index)**

**result.append(valuebest)**

**#print(t) # 程序运行时间较长，打印t来监视程序进展速度**

**print(result)**

**#可视化部分的代码**

**print(valuebest)**

**x = np.linspace(1, len(result), len(result))**

**plot.plot(x, result)**

**plot.ylabel("bestvalue")**

**plot.xlabel("times")**

**plot.show()**

**3.过程总结**

冷却控制:指状态由高温向低温状态靠近，以达到最优解又称降温。降温手段可以有：

1. 经典模拟退火算法的降温方式t = t / lg(1+k)
2. 快速降温方式为 t = t / 1 + k

内层平衡：又称Metropolis抽样稳定准则，用于决定在个温度下产生候选街的数目

在该试验中采取内层的平衡条件为循环1000次，每次循环交换当前策略（访问城市的向量）的两个或三个位置并计算其适应函数的值，更新记录局部或全局最优接。

1. 实验结果的展现

用python自带的可视化模块画出实验的结果，代码如上面可视化代码的部分。

二维坐标图以退火的次数为x轴，以实验结果值为y轴。

**4.可视化部分**

退火次数对应的值

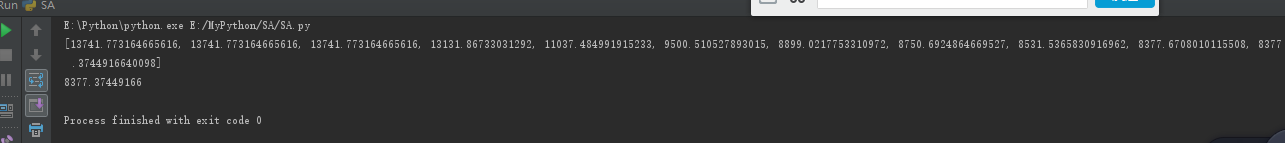


图1

迭代退火求得最优解过程

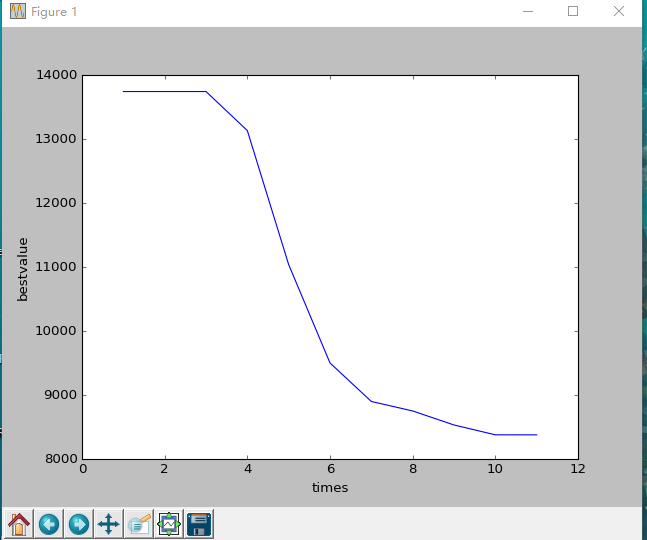
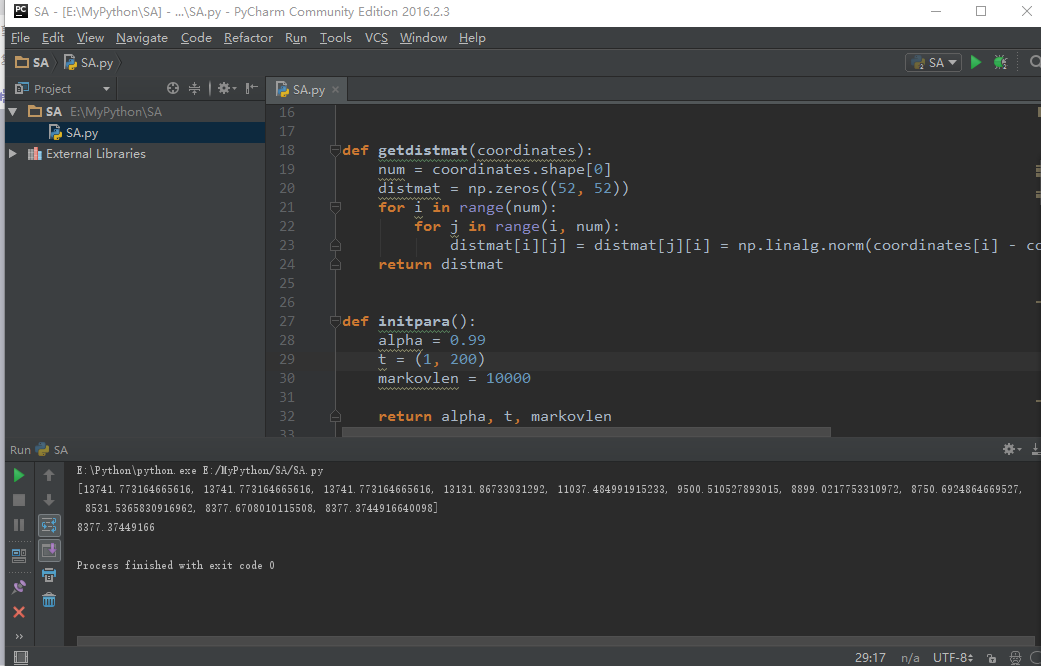


图2



1. **实验小结（心得体会、遇到的问题及解决方式、未解决/需进一步研讨的问题或建议新实验方法等）**

1.初始化温度的确定可以通过多次实验获取。

2.模拟退火算法通过接受低能状态向高能状态的转移跳出局部解求得最优解。

3.退火过程要设定一个较合适的初始温度以防止收敛次数不够，从而得不到最优解。