# 模拟退火实验

## 题目内容：

在TSPLIB中选一个大于100个城市数的TSP问题，然后：

1. 采用多种邻域操作的局部搜索local search策略求解；
2. 在局部搜索策略的基础上，加入模拟退火simulated annealing策略，并比较两者的效果；
3. 要求求得的解不要超过最优值的10％，并能够提供可视化，观察路径的变化和交叉程度。

## 实现环境：

该实验在Windows下使用python3编码实现

## 数据测例：

本次实验使用的测例主要是tsplib上面的ch130问题，是130个城市的旅行商问题，该问题最优tsp解路径长度为6110

另外还使用了ch150和a280进行测试，用于对比得出一些关于调参的结论，这两个问题分别是150个城市和280个城市的旅行商问题

## 算法概述：

**爬山法（局部搜索算法）：**

（1）设置随机产生的初始解

（2）产生临近解，如果临近解更好就接受，并更新当前解，否则丢弃生成的邻居解

产生解的方法（产生邻域的操作）：

（i）开始使用的是交换路径上的两个点，以此产生邻居：

当前解：A->B->C->D->E->F

交换路径上的B、E两个节点

得到邻居解：A->E->C->D->B->F

（ii）后来使用的方法是“逆转两个节点之间的路径”，以此产生邻居：

当前解：A->B->C->D->E->F

逆转子路径B->C->D->E

得到邻居解：A->E->D->C->B->F

（3）一直执行（2）10万次，将最后的解作为结果输出

**模拟退火：**

（1）设置初始温度T=100

（2）温度固定情况下执行：

（i）产生临近解，如果临近解更好就接受，并更新当前解，否则以exp(-ΔE/T) 的概率接受该解

产生解的方法爬山法中的（i）（ii），其中（i）效果很差（有时能限制与最优解的差距到10%以内，大多数时候不行）。

所以最终采用（ii），能够稳定跑到10%之内，调整好参数可以跑到5%之内

（ii）执行（i）1000次，然后跳到（3）

（3）将温度下降：T = T\*0.99，如果温度T小于最低温度T\_min=0.5，结束，输出当前解

## 代码框架简要说明

generate\_random\_list函数产生初始随机解，输入问题规模（有多少个城市），返回一种随机的走法：

def generate\_random\_list(point\_num):

point\_list = [0]\*(point\_num - 1)  
 for i in range(0, point\_num - 1):  
 point\_list[i] = i+1  
 random.shuffle(point\_list)  
 point\_list.insert(0, 0)  
 point\_list.append(0)  
 return point\_list

get\_distance获取两点间的距离，将两个城市的位置作为参数，返回距离，用于计算整个tsp路径长度

def get\_distance(point\_a, point\_b):

temp1 = math.pow(coordinate\_x[point\_a] - coordinate\_x[point\_b], 2)  
 temp2 = math.pow(coordinate\_y[point\_a] - coordinate\_y[point\_b], 2)  
 return math.pow(temp1 + temp2, 0.5)

ExchangeLocalSearch类，其solve函数使用交换两个节点作为邻域操作，使用局部搜索爬山法求解tsp最短路

InverseLocalSearch类，其solve函数使用逆转两个节点之间的路径作为邻域操作（来产生新解），使用局部搜索爬山法求解tsp最短路

SimulateAnneal类，其solve函数使用交换两个节点/逆转两个节点之间的路径作为领域操作，使用模拟退火进行求解

class ExchangeLocalSearch:

class InverseLocalSearch:  
class SimulateAnneal:

show\_figure函数用于将tsp问题可视化，显示出；路径求解的具体过程

store\_result函数用于将求解的结果存入文件中

def show\_figure(self):

def store\_result(self):

swap\_two\_point()，邻域操作函数，交换两个节点来产生新解

swap\_sub\_path()，邻域操作函数，逆转两个节点之间的路径来产生新解

def swap\_two\_point(self):

def swap\_sub\_path(self):

以及三个解题类的solve函数：

ExchangeLocalSearch的solve函数，遍历100\*1000次，不断使用swap\_two\_point邻域操作产生更好的解，显示求解过程，并将最终的解存储：

def solve(self):

# 进行迭代求解  
 for i in range(0, 100\*MARKOV\_LENGTH):  
 if self.swap\_two\_point():  
 print('使用交换点策略的局部搜索 ', ' 第', i, '轮 ', '路径长度更新为：', self.path\_count)  
  
 self.path\_count\_set.append(self.path\_count)  
 if IF\_SHOW\_FIGURE:  
 self.show\_figure()  
 self.store\_result()

InverseLocalSearch的solve函数，遍历100\*1000次，不断使用swap\_sub\_path邻域操作产生更好的解，显示求解过程，并将最终的解存储：

def solve(self):

# 进行迭代求解  
 for i in range(0, 100\*MARKOV\_LENGTH):  
 if self.swap\_sub\_path():  
 print('使用交换路径策略的局部搜索 ', ' 第', i, '轮 ', '路径长度更新为：', self.path\_count)  
  
 self.path\_count\_set.append(self.path\_count)  
 if IF\_SHOW\_FIGURE:  
 self.show\_figure()  
 self.store\_result()

SimulateAnneal的solve函数，外层遍历由温度控制，从初温0.4到末温0.01，降温系数0.99，内层循环固定温度做1000次，内层循环中不断使用swap\_sub\_path邻域操作产生更好的解，并显示求解过程，在温度降到0.01之下，跳出外层循环，将最终的解存储：

def solve(self):

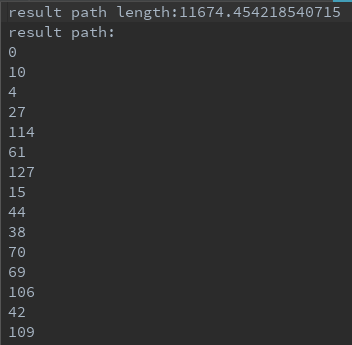
# 进行迭代求解  
 while self.temperature > MIN\_TEMPERATURE:  
 for i in range(0, MARKOV\_LENGTH):  
 if self.swap\_sub\_path():  
 print('使用模拟退火', ' 当前温度：', self.temperature, ' 第', i, '轮 ', '路径长度更新为：', self.path\_count)  
 self.path\_count\_set.append(self.path\_count)  
 if IF\_SHOW\_FIGURE:  
 self.show\_figure()  
 self.temperature \*= ATTENUATION\_QUOTIENT  
 self.store\_result()

由于逆转子路径的邻域操作效果好很多，所以最终使用逆转子路径作为邻域操作

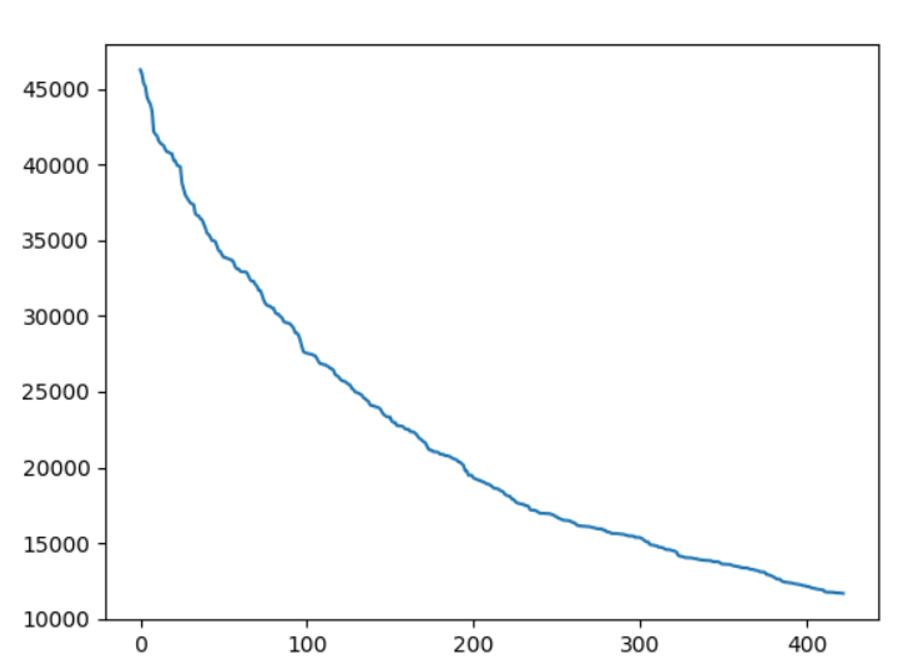
## 实验结果展示：

**使用ExchangeLocalSearch类的solve函数求解tsp所得结果：（局部搜索1）**

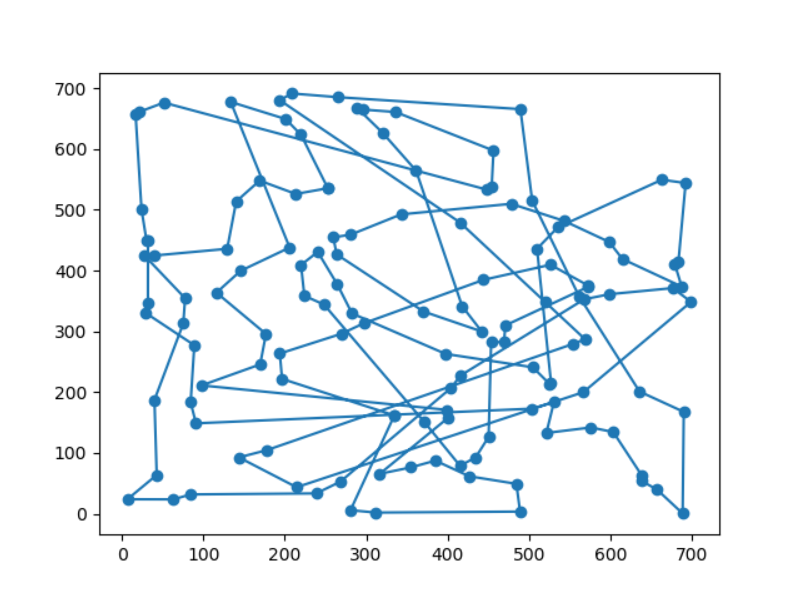
该爬山法最终解决方案：



该方法求解问题时随迭代次数增加，tsp解对应路径长度变化曲线：



最终tsp路径连线图：

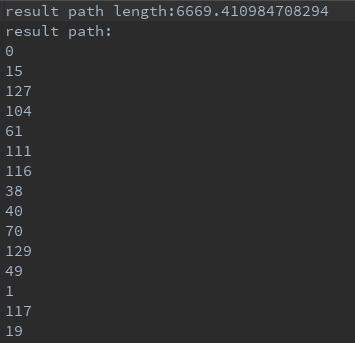


从上面的结果中我们看到：

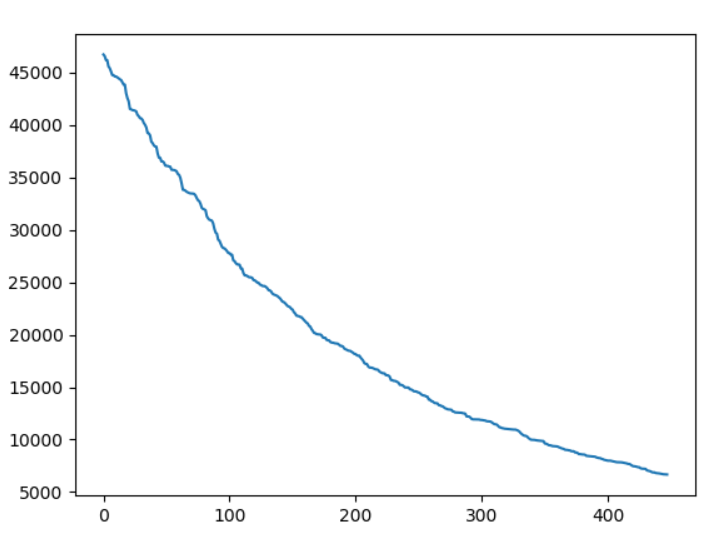
1. 使用爬山法求解过程中随迭代次数增加，tsp解对应路径**长度一直变小**，这是因为爬山法只接受更好的解，所以长度只会不断减小
2. **求解时下降过程非常快这是爬山法的优点**，仅仅用了400多步就收敛，
3. 但是**缺点是最终的解看起来虽然比随机产生路径好，但是还是非常差，路径交错**很多，长度为10000以上，而真正的最优解是6110，超出接近100%，这是这个算法的局限性，由于不能接受差解，所以无法跳过局部最优，达到全局最优（或者其它更优的局部最优）
4. 通过反复测试，每次得到的解有好有坏，这是因为初始解的随机选择，如果初始解选的好，有极小概率达到全局较优解

**使用InverseLocalSearch类的solve函数求解tsp所得结果：（局部搜索2）**

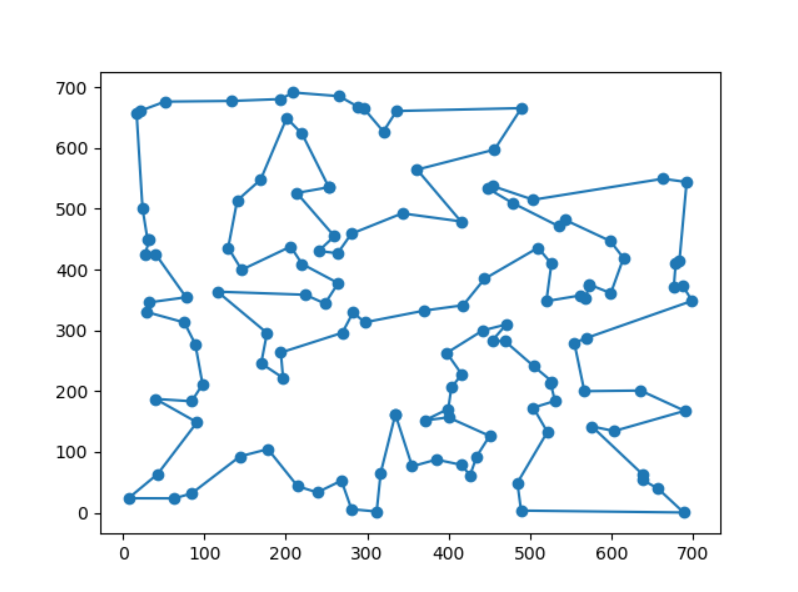
该爬山法最终解决方案：



该方法求解问题时随迭代次数增加，tsp解对应路径长度变化曲线：



最终tsp路径连线图：

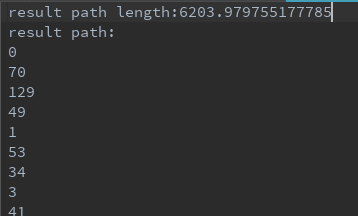


从上面的结果中我们看到：

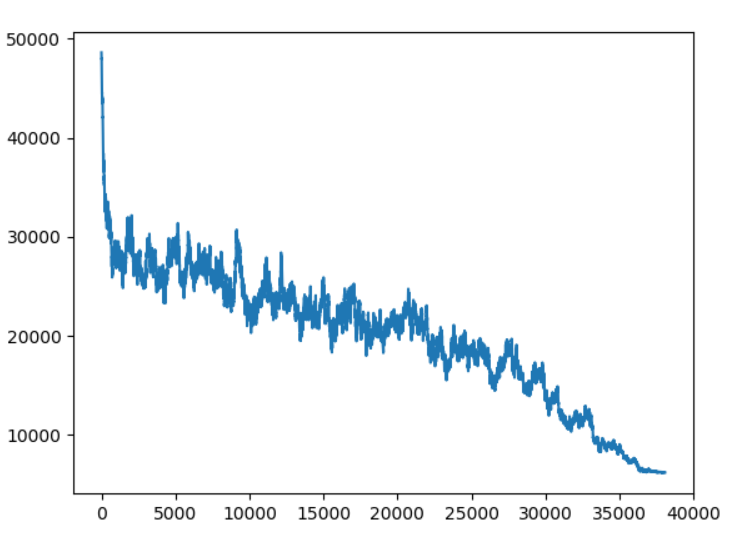
1. 使用爬山法求解过程中随迭代次数增加，tsp解对应路径**长度一直变小**，这是因为爬山法只接受更好的解，所以长度只会不断减小
2. **求解时下降过程非常快这是爬山法的优点**，仅仅用了500多步就收敛，
3. 但是缺点是最终的解看起来虽然比随机产生路径好（看起来没有路径交叉了），但是还是比较差（个别顺序还可以优化），解路径长度6669，而真正的最优解是6110，超出接近10%，**虽然比前面的ExchangeLocalSearch类的解法更好，但是后面马上就能看到，这种方法比模拟退火差不少，而且得到的解时好时坏，这也是缺点**，这是这个算法的局限性，由于不能接受差解，所以无法跳过局部最优，达到全局最优（或者其它更优的局部最优）
4. 通过反复测试，每次得到的解有好有坏，这是因为初始解的随机选择，如果初始解选的好，有极小概率达到全局较优解

**使用SimulateAnneal类的solve函数求解tsp所得结果：**

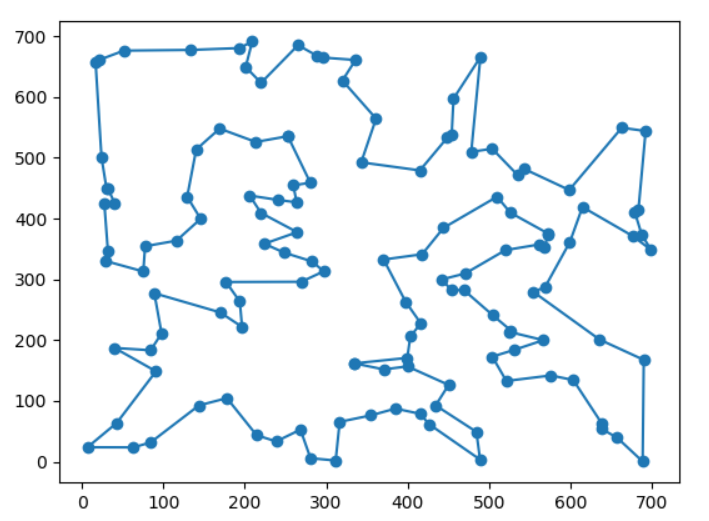
使用模拟退火求解的最终解决方案：



该方法求解问题时随迭代次数增加，tsp解对应路径长度变化曲线：



最终tsp路径连线图：



从上面的结果中我们看到：

1. 使用模拟退火求解过程中随迭代次数增加，tsp解对应路径**长度不断波动**，并且在求解的前期波动剧烈，求解后期波动平缓甚至没有波动。这是因为模拟退火算法能够以一定的概率接受差解，所以随着迭代次数增加，tsp路径长度能够变大或者变小，温度高时（前期）接受差解概率大，所以波动大，温度低时（后期）接受差解的概率小所以波动小，趋于平缓；
2. 求解时解的值（路径长度）**下降非常慢（比前面的局部搜索慢几倍），这是模拟退火算法的缺点**，用了40000多步才完成整个求解过程，前期波动下降，后期稳定下降；
3. **解路径长度6203，而真正的最优解是6110，偏差最优解仅仅1%~2%，完全符合老师要求**，这已经非常接近最优解了，比其前面两种局部搜索优秀太多，我们看到最终tsp路径连线图中，相比之前的InverseLocalSearch局部搜索方法求出的解，在一些路径的选择上更好，这是**因为模拟退火为了防止陷入局部最优，会以一定的概率接受差解，从而跳出局部最优，接受差解的概率会随着温度下降、差解劣质程度上升而降低，因此在遇到比较好的解时则跳出概率较小（遇到最优解或者较优解时不会放弃该解），从而跳过一些较差的局部最优，最终搜索到较优解，这是模拟退火的优点** ；
4. 使用**模拟退火能够稳定得到比较好的解**，而不像局部搜索，时好时坏（差的时候非常差），而模拟退火即使差的时候也能距离最优解5%之内
5. 因为**模拟退火有一定接受差解的概率，所以有可能求得的解不是整个过程中遇到的最优解，这也是缺点** ；

**关于路径变化过程的更多展示截图可以到我的github相关项目上看：**

https://github.com/ousuixin/TSP-/tree/master

## 总结

1. 使用局部搜索（爬山法）求解tsp问题用时很短，收敛快速，模拟退火用时较长波动到趋于稳定；
2. 虽然局部搜索（爬山法）求解分配方案用时短，但是求得的解效果很差，在上述的两种爬山法中，第一种爬山法得出的方案距离最优解超出100%之多，完全无法接受，第二种爬山法比第一种好了不少，但是仍然距离最优解在10%左右，无法稳定达到10%之内，达不到老师的要求，足以看出**爬山法的特点**：搜索过程中快速接近最优解，但是极易陷入局部极小点而停止更新解，最终解的好坏与初始解的选择有很大关系（因此只有很小概率得到比较好的解）
3. 而模拟退火算法则在该问题中有比较好的表现，虽然用时较长，但是能够稳定得出距离最优解5%之内的解，完全符合老师的要求，甚至有时候能够到达2%之内（比如上面的截图给出的解法） 。**模拟退火算法的特点概括为**：（1）具有一定的跳过局部最优解的能力，能够以较快的速度得到问题的近似最优解（较优解），如果参数（初温、末温、降温系数）设置的很好，模拟退火能够表现出相当好的效果（2）但是正因为它会有概率接受差解，所以最终的解有可能比搜索过程中的某些解更差（3）对参数非常敏感，如果降温系数、初温、末温稍微设置不当，就会表现出较差的效果（初温要较高，否则一开始就很快收敛，末温要设置较低，否则无法充分冷却收敛，降温系数越接近1越好，能够充分退火，并且内层循环迭代次数至少要1000次才能有比较好的效果）