模拟退火算法实验————解决TSP问题

> 本报告使用模拟退火算法解决TSP问题

1. 实验背景

旅行商问题，即TSP问题（Traveling Salesman Problem），也叫旅行推销员问题、货郎担问题，是数学领域中著名问题之一。假设有一个旅行商人要拜访n个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

旅行推销员问题是图论中最著名的问题之一，即“已给一个n个点的完全图，每条边都有一个长度，求总长度最短的经过每个顶点正好一次的封闭回路”。Edmonds，Cook和Karp等人发现，这批难题有一个值得注意的性质，对其中一个问题存在有效算法时，每个问题都会有有效算法。

迄今为止，这类问题中没有一个找到有效算法。倾向于接受NP完全问题（NP-Complete或NPC）和NP难题（NP-Hard或NPH）不存在有效算法这一猜想，认为这类问题的大型实例不能用精确算法求解，必须寻求这类问题的有效的近似算法。

2. 模拟退火算法

2.1 算法简介

模拟退火算法（Simulate Anneal，SA）是一种通用概率演算法，用来在一个大的搜寻空间内找寻命题的最优解。

模拟退火的出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。模拟退火算法是一种通用的优化算法，其物理退火过程由加温过程、等温过程、冷却过程这三部分组成。

模拟退火的原理也和金属退火的原理近似：将热力学的理论套用到统计学上，将搜寻空间内每一点想像成空气内的分子；分子的能量，就是它本身的动能；而搜寻空间内的每一点，也像空气分子一样带有“能量”，以表示该点对命题的合适程度。演算法先以搜寻空间内一个任意点作起始：每一步先选择一个“邻居”，然后再计算从现有位置到达“邻居”的概率。

2.2 算法结合TSP

状态空间与状态产生函数

1. 搜索空间也称为状态空间，它由经过编码的可行解的集合组成。
2. 状态产生函数应尽可能保证产生的候选解遍布全部解空间。通常由两部分组成，即产生候选解的方式和候选解产生的概率分布。
3. 候选解一般采用按照某一概率密度函数对解空间进行随机采样来获得。
4. 概率分布可以是均匀分布、正态分布、指数分布等。

状态转移概率

1. 状态转移概率是指从一个状态向另一个状态的转移概率。
2. 通俗的理解是接受一个新解为当前解的概率。
3. 它与当前的温度参数T有关，随温度下降而减小。
4. 一般采用Metropolis准则。

内循环终止准则
也称Metropolis抽样稳定准则，用于决定在各温度下产生候选解的数目。常用的抽样稳定准则包括：

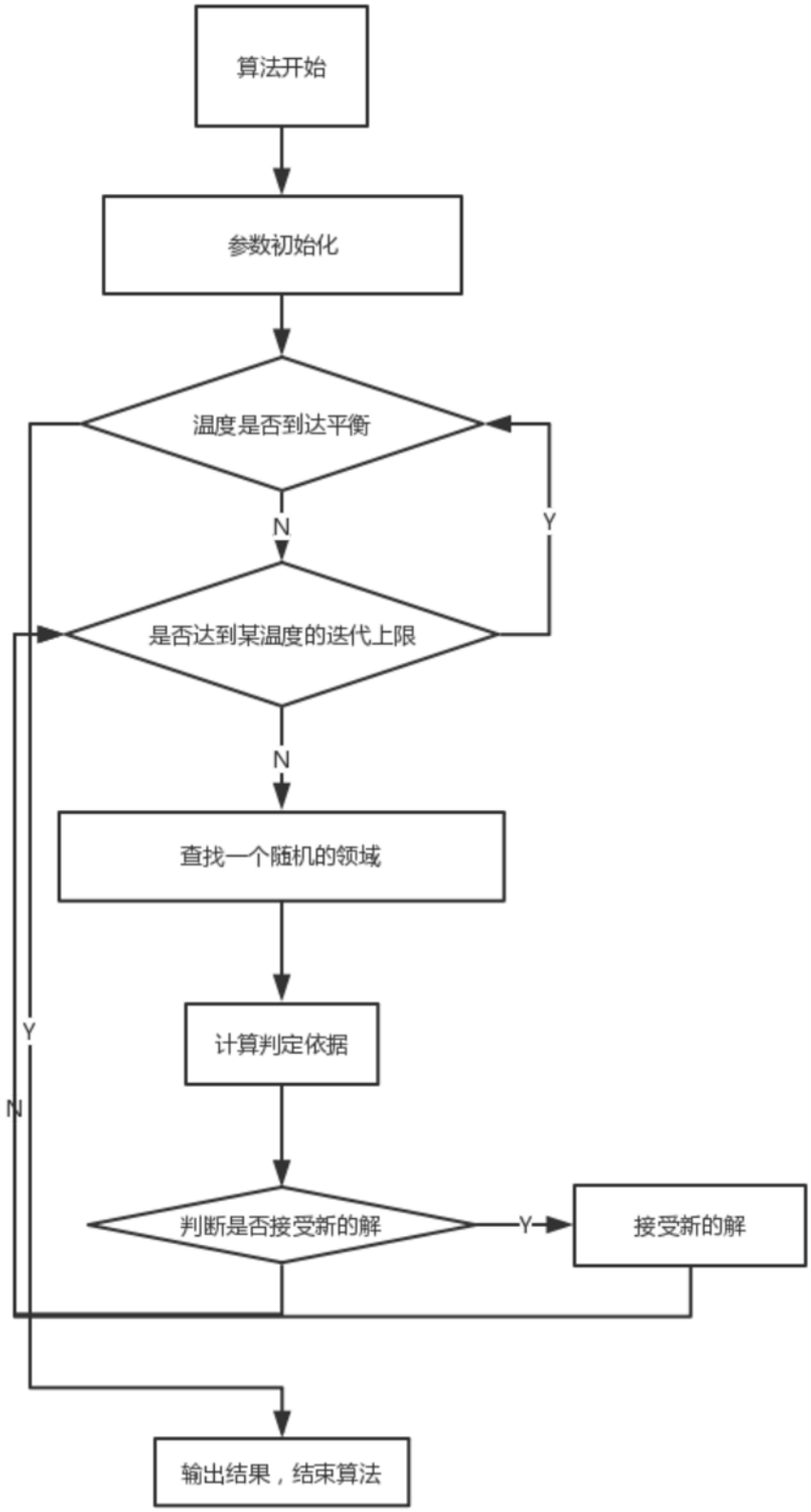
1. 检验目标函数的均值是否稳定。
2. 连续若干步的目标值变化较小。
3. 按一定的步数抽样。

外循环终止准则
即算法终止准则，常用的包括：

1. 设置终止温度的阈值。
2. 设置外循环迭代次数。
3. 算法搜索到的最优值连续若干步保持不变。
4. 检验系统熵是否稳定。

3. 算法设计流程

3.1 算法流程图



3.2 算法实现过程

使用python3作为设计语言，其中本次使用的版本是python3.6，其中使用了常规的的库有：

1. time
2. copy
3. random
4. math

使用的非内置库有：

1. numpy
2. matplotlib

3.2.1 算法公式

> 本算法设计的公式有：
>
> 1. Metropolis准则
> 2. 降温方法

3.2.1.1 Metropolis准则

首先计算两次适应值的差距

$$\Delta f = f_1 - f_2$$

随后计算：

$$e^{-\frac{\Delta f}{T_k}} > \xi$$

其中$\xi \in (0,1)$是一个随机的小数
根据上面这个式子判断是否接受一个新的值

3.2.1.2 降温方法

我这里使用的是用指数更新的方式进行温度更新

$$T = T \times q$$

3.2.2 算法设计

3.2.2.1 TSP的基础类

首先对于TSP问题，首先需要解决对于TSP至少图的问题的解决

1. 点类

```
class Point:
    def __init__(self,x,y):
        self.x = x
        self.y = y
```

2. 图类
   对于TSP最重要的是通过一个行走序列，得到路径长度，所以其中的 `ask_distance_for_plan()` 是非常重要的。

```
class Graph:
    def __init__(self,n):
        self.points = []
        self.point_n = n

    def add_point(self,p):
        self.points.append(p)
        self.point_n += 1

    def ask_distance(self,i,j):
        point_x = self.points[i]
        point_y = self.points[j]
        return math.sqrt(sqr(point_x.x - point_y.x) + sqr(point_x.y - point_y.y))

    def ask_distance_for_plan(self,plan):
        res = 0.0
        for i in range(1,self.point_n):
            res += self.ask_distance(plan[i],plan[i-1])
        res += self.ask_distance(plan[0],plan[self.point_n-1])
        # print(plan,' ',res)
        return res

    def show(self):
        for item in self.points:
            print(item.x," ",item.y)
```

3.2.2.2 初始化

对于模拟退火的时候，初始化操作就是规定初始温度，结束温度，每个温度中的迭代次数，温度削减率q

```
def __init__(self,t_max,t_min,L,q):
    self.T_MAX = t_max
    self.T_MIN = t_min
    self.L = L
    self.q = q
```

3.2.2.3 `run()` 最主要的函数

给出核心的代码：

```python
def run(self,n,graph):
    # 随机化种子
    random.seed(time.time())

    # 初始化需要的变量
    init_plan = self.initPlan(n)
    T = self.T_MAX

    res = {
            "fitness": graph.ask_distance_for_plan(init_plan),
            "plan": init_plan
            }

    count = 0
    T_s = []
    best_ans = []

    # 温度处理
    while T > self.T_MIN:

        # 迭代过程
        for i in range(self.L):
            # 寻找新的解
            new_plan = copy.copy(init_plan)
            new_plan = self.get_new_plan(new_plan)
            count += 1

            init_fitness = graph.ask_distance_for_plan(init_plan)
            new_fitness  = graph.ask_distance_for_plan(new_plan)

            if res["fitness"] > init_fitness:
                res["fitness"] = init_fitness
                res["plan"] = copy.copy(init_plan)
            if res["fitness"] > new_fitness:
                res["fitness"] = new_fitness
                res["plan"] = copy.copy(new_plan)

            # print("fitness = ",init_fitness," plan :\n",init_plan)
            # 利用差值，以及之前的结果判断结果是否满足接受
            delta_fitness = new_fitness - init_fitness
            # print("delta = ",delta_fitness)

            # if the fitness get smaller , then it should be good
            # if the fitness get bigger , the it can be accepted by rate
            if delta_fitness >= 0:
                div = random.random()
                metropolis = math.exp(-delta_fitness / T)

                if metropolis <= div:
                    init_plan = copy.copy(new_plan)
            else:
                init_plan = copy.copy(new_plan)

        T_s.append(T)
        best_ans.append(res["fitness"])
        T *= self.q

    print(res)
    return res,T_s,best_ans
```

4. 实验结果

4.1 实验数据

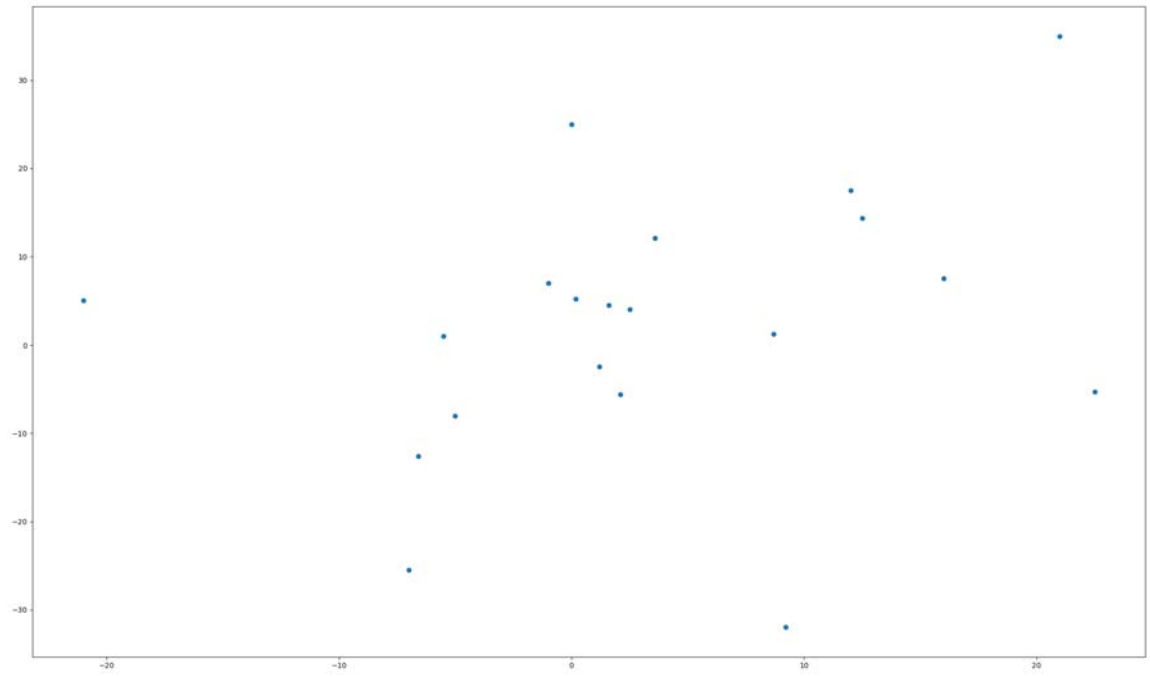
使用的实验数据如下：
给出所有点的坐标

```
A 2.5 4.0
B 1.2 -2.4
C 8.7 1.2
D 3.6 12.1
E -5.5 0.94
F -6.6 -12.6
G 0.18 5.219
H 12.5 14.3609
I 22.5 -5.26
J 1.61 4.5
K 2.1 -5.6
L 0 25
M 9.2 -32
N -1 7
O -5 -8
P 21 35
Q 16 7.5
R -21 5
S -7 -25.5
T 12 17.5
```

给出读入数据的python代码：

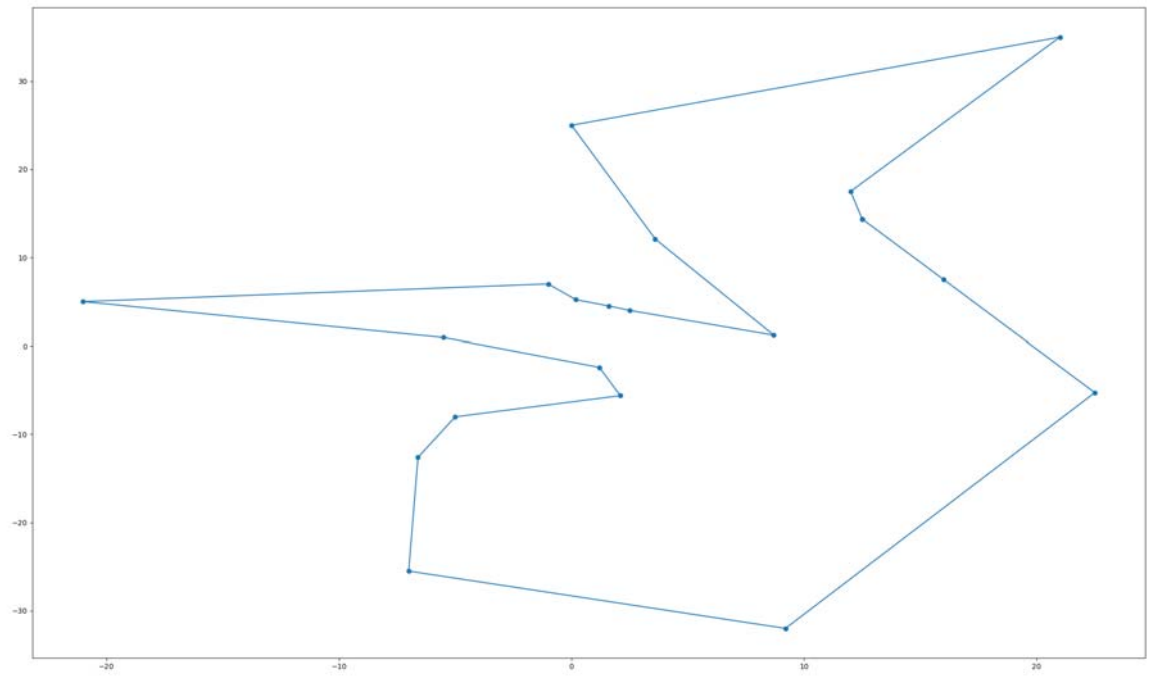
```
graph = Graph(0)
with open("in.txt","r") as f:
    lines = f.readlines()
    for line in lines:
        line = str(line)
        # print(line)
        items = line.split(' ')
        x = float(items[1])
        y = float(items[2])
        # print('x =',x,' y = ',y)
        graph.add_point(Point(x,y))
```

给出一个简单的图：



4.2 实验结果展示

使用已有代码进行运行，运行结果如下：

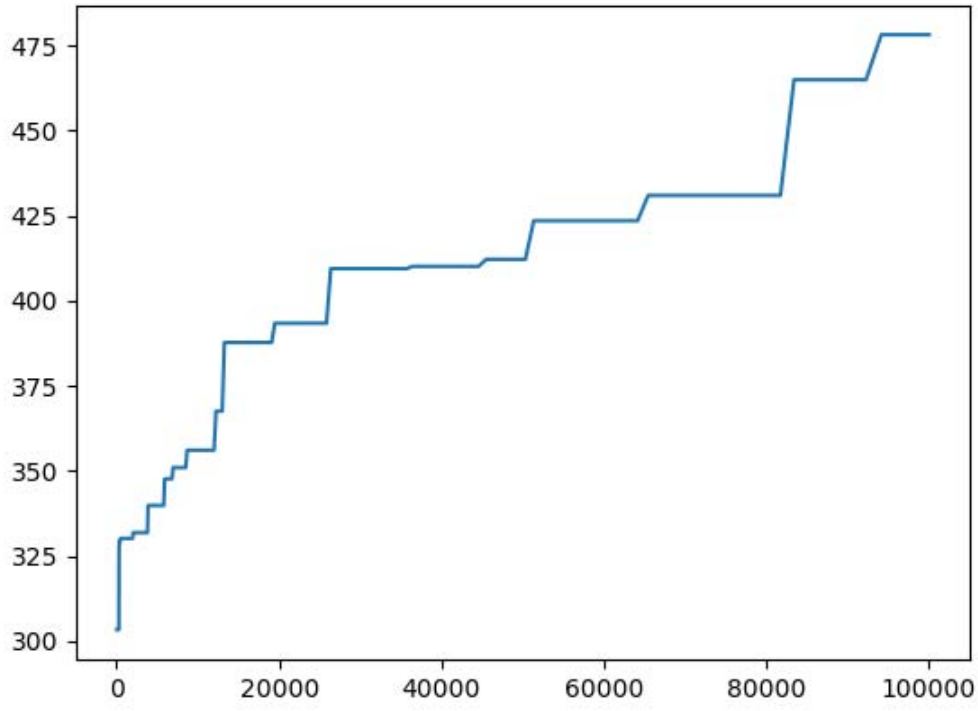


最终的结果：

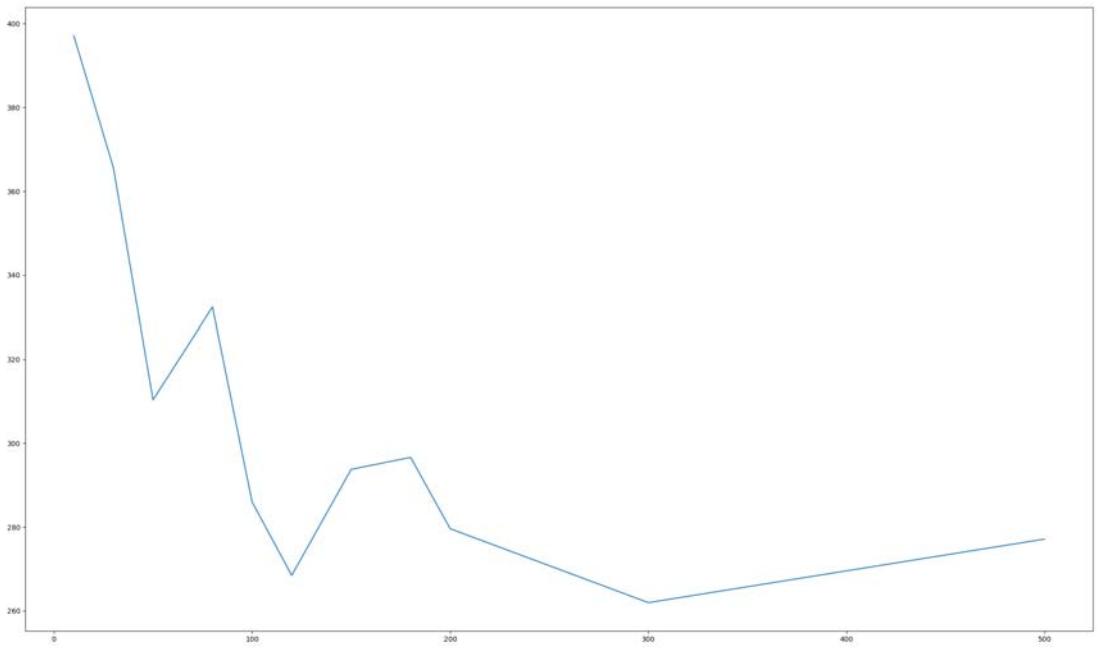
```
{'fitness': 224.65153949772267,
'plan': [7, 16, 8, 12, 18, 5, 14, 10, 1, 4, 17, 13, 6, 9, 0, 2, 3, 11, 15, 19]}
```

4.2 实验结果分析

1. 退火过程和最优解
   在中间算法过程中，记录最优结果，查看最优结果和退火中温度的变化
   结果如图所示：



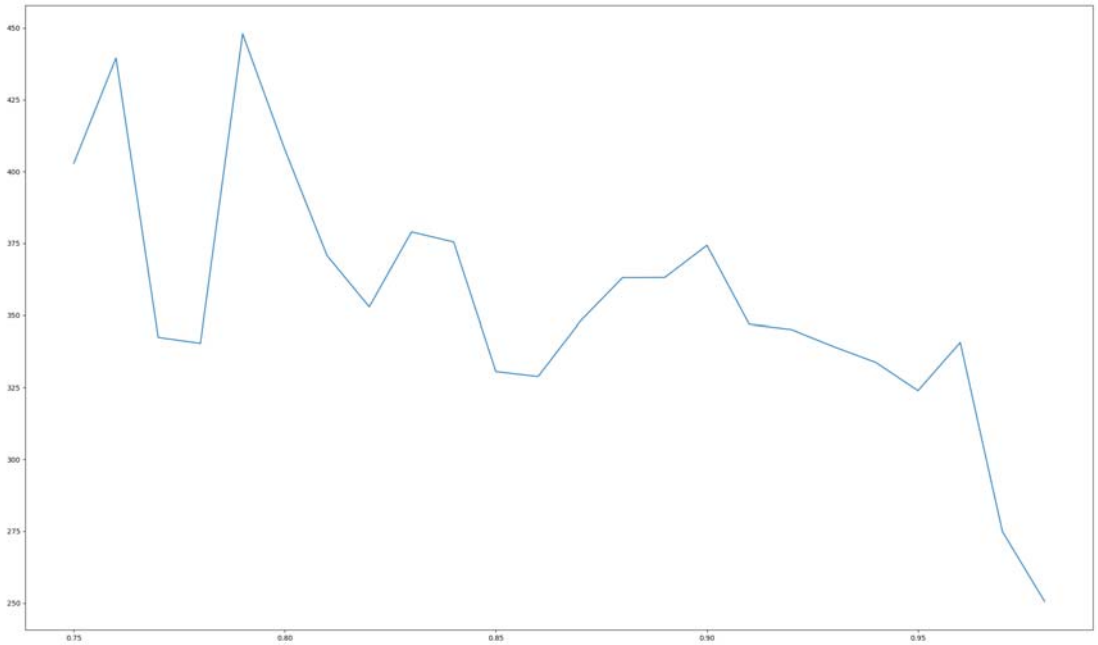
2. 改变迭代次数L和最终结果
   改变每个温度的迭代次数，查看最终的结果。



中间折线的结果，是取得这些结果：

```
L = [10,30,50,80,100,120,150,180,200,300,500]
```

3. 改变温度改变率q
   改变温度的修改，修改q的值



中间折现的结果，是取得如下的改变率：

```
q_s = [0.75,0.76,0.77,0.78,0.79,0.8,0.81,0.82,0.83,0.84,
0.85,0.86,0.87,0.88,0.89,0.9,0.91,0.92,0.93,0.94,0.95,0.96,0.97,0.98]
```

4. 总结
   通过上述两个数据结果，我们可以明显看到，模拟退火在一定程度上可以得到非常好的解，但是在实际的运用中，却不是非常稳定的可以得到比较好的结果。随着迭代次数求解的过程，并不一定结果稳定的变好。