中山大学计算机学院人工智能本科生实验报告

2022学年春季学期

课程名称：Artificial Intelligence

教学班级	人工智能（陈川）	专业（方向）	计算机科学与技术人工智能与大数据
学号	20337025	姓名	崔璨明

一、实验题目

在TSPLIB（ http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ ，多个地址有备份；其他网站还可以找到有趣的art TSP和national TSP）中选一个大于100个城市数的TSP问题，使用模拟退火和遗传算法求解。

模拟退火

- 采用多种邻域操作的局部搜索local search策略求解（此时可以选取城市数量较少的TSP问题进行求解）
- 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（此时必须选取城市数大于100个的数据集）
- 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。

遗传算法

- 设计较好的交叉操作，并且引入多种局部搜索操作（可替换通常遗传算法的变异操作）
- 和之前的模拟退火算法（采用相同的局部搜索操作）进行比较
- 得出设计高效遗传算法的一些经验，并比较单点搜索和多点搜索的优缺点

二、实验内容

1、算法原理

局部搜索：

局部搜索算法是在一组可行解的基础上，在当前解的领域内进行局部搜索产生新的可行解的过程。其步骤为：

1. 构造初始解s；
2. 定义s的邻域$\delta(s)$；
3. 在邻域$\delta(s)$中搜索新的解$s\prime$；
4. 令$s = s\prime$ 重复上述步骤直到满足停止条件。
   在实际中局部搜索可以作为其它算法的补充, 目的是进一步提升解的质量。

模拟退火算法：

简单来说，模拟退火算法的思想借鉴于固体的退火原理，当固体的温度很高的时候，内能比较大，固体的内部粒子处于快速无序运动，当温度慢慢降低的过程中，固体的内能减小，粒子的慢慢趋于有序，最终，当固体处于常温时，内能达到最小，此时，粒子最为稳定。模拟退火算法便是基于这样的原理设计而成。

模拟退火算法从某一较高的温度出发，这个温度称为初始温度，伴随着温度参数的不断下降，算法中的解趋于稳定，但是，可能这样的稳定解是一个局部最优解，此时，模拟退火算法中会以一定的概率跳出这样的局部最优解，以寻找目标函数的全局最优解。当前的温度越高，跳出局部最优解接受劣解的概率就越大。随着温度的降低，渐渐得出靠近最优的解。

遗传算法：

遗传算法采纳了自然进化模型，如选择、交叉、变异、迁移、局域与邻域等。计算开始时，一定数目N个个体(父个体1、父个体2、父个体3、父个体4.....)即种群随机地初始化,并计算每个个体的适应度函数，第一代即初始代就产生了。如果不满足优化准则，开始产生新一代的计算。为了产生下一代，按照适应度选择个体父代进行基因重组(交叉)而产生子代。所有的子代按一定概率变异。然后子代的适应度又被重新计算，子代被插人到种群中将父代取而代之，构成新的一代(子个体1、子个体 2、子个体 3、子个体 4.....)。这一过程循环执行,直到满足优化准则为止。

2、算法伪代码

局部搜索算法伪代码：

```
input:当前状态curr,,
output:最后的状态curr
for i=1 to MAX:#设置迭代次数
        next=neighbour(curr)
        if evaluate(next)<evaluate(curr):#只接受更优解
                curr=next

return curr
```

```
#----------------------------------
function evaluate()
input:n个城市组成的序列a
output:代价总和

float distance=0
for i in range(1,n):
        distance+=dis(a[n],a[n-1])
distance+=dis(a[0],a[n])
return distance
#----------------------------------
function neighbour()
input:当前状态curr
output:新状态

return 交换两个城市 or 随机将一段逆序 or 随机交换两段
```

模拟退火算法伪代码：

```
input:当前状态curr,,
output:最后的状态curr
for t=1 to infinity:#时间更新
        T=updtae(t)#温度更新
        if T==0 then: return curr#温度到达阈值，返回

        for i=1 to 100000:#当前温度下的稳定状态
                next=neighbour(curr)
                if evaluate(next)<evaluate(curr):
                        curr=next
                else:
                        k=evaluate(next)-evaluate(curr)
                        以概率pow(e,-k/T)执行:
                                curr=next

#----------------------------------
function evaluate()
input:n个城市组成的序列a
output:代价总和

float distance=0
for i in range(1,n):
        distance+=dis(a[n],a[n-1])
distance+=dis(a[0],a[n])
return distance
#----------------------------------
function neighbour()
```

```
input:当前状态curr
output:新状态

return 交换两个城市 or 随机将一段逆序 or 随机交换两段
```

遗传算法伪代码：

```
#设待解决的TSP问题有k个城市
population=[]
#生成含有10个不相同个体的初始种群
i=0
do
        随机生成一个长度为k的字符串,字符取自{0，…，k-1}且不重复,记此字符串为
individual
        if population不含有individual then
                population=population U {individual}
        i←i+1
while(i>=9)
num iteration=0;

repeat
        new_population=[]
        for i=1 to len(population):
                parentl=random_select(population, evaluate())
                parent2-random_select(population,evaluate())
                child=reproduce(parentl,parent2)
                在[0，1]内生成一个随机数r
                if r<=β then:
                        mutate(child) #变异
                new_population.append(child)
        population=new population
num iteration++
until num_iteration >= 100000

#-----------------------------------
function random_select()
Input:
        population，种群集
        evaluate,评估个体的适应度函数
Output:
        被选择的个体 sum←0

for i =1 to len(population) do
        evaluate(population[i])
        sum=sum+population[i].value
for i=1 to len(population) do
```

```
        生成一个[0,1]内的随机数r
        if (r<(population[i].value /sum)) then
                individual=population[i]
                return individual

#-----------------------------------
function reproduce(x,y)
Input: x,y两个个体
Output: x和y交叉得到的下一代个体z
n=len(x)
c=random number from 1 to n
return APPEND(sub_str(x,1,C)+sub_str(y,c+1,n))
```

3、关键代码展示

在TSP问题中，产生邻域的方法有许多种，在本次实验中，我经过多次实验和对比，最终采取了三种产生下一状态的方法（随机交换两个城市、随机选取一段城市序列并将其逆序、将序列随机分成四段并交换其中两段），并每次随机选取一种来产生新的状态，增加随机性以使算法达到更好的性能，产生下一状态的函数如下：

```python
#生成下一个状态
def neighbour3(path):
    leng=len(path)
    charge=random.random()
    if charge<0.334:#分成四段，交换中间两段
        a1,a2,a3=random.sample(range(1, leng-1), 3)
        if a1>a2:
            a1,a2=a2,a1
        if a2>a3:
            a2,a3=a3,a2
        if a1>a2:
            a1,a2=a2,a1
        tmp=path[0:a1]+path[a2:a3]+path[a1:a2]+path[a3:leng]

    elif charge<0.666:#交换两个城市
        i=random.randint(0,leng-2)
        j=random.randint(1,leng-1)
        #print(i,"<->",j)
        #print(path)
        if i!=j:
            path[i],path[j]=path[j],path[i]
            tmp=path[:]
            path[i],path[j]=path[j],path[i]
        else:
            tmp=path[:]

    else:#分成三段，将中间逆序
        k1,k2=random.sample(range(1, leng-1), 2)
        if k1>k2:
            k1,k2=k2,k1
        tmp=path[0:k1]+path[k1:k2][::-1]+path[k2:leng]
    return tmp
```

采用模拟退火算法来求解的过程中，我设置初始温度为1000000，采用指数降温，每次温度乘以系数0.98，每个温度下的温度状态则为进行1000次领域操作，算法的关键代码

如下：

```
tem=1000000#设置初始温度
while tem>0.0001:
    tem=0.98*tem#降温操作，每次乘0.98
    if tem==0:
        break

    for i in range(1000):   #稳定状态定为每个温度进行1000次
        next_node=neighbour3(curr)

        e_new=evaluate(next_node)
        e_now=evaluate(curr)
        dis_change.append(e_now)
        #接受更好状态
        if e_new<e_now :
            curr=next_node

        #以一定概率接受
        else :
            k=min(1,math.exp((e_now-e_new)/tem))
            if k>random.random():
                print(math.exp((e_now-e_new)/tem),e_now-e_new,tem)
                curr=next_node

    #每隔一段时间进行采样
    if ctt%100==0:…
    ctt+=1
print(evaluate(curr))
```

采用遗传算法求解的过程中，关键部分是种群的更新迭代，在我编写的程序中，我采用轮盘赌的方式选取亲本，适应度选取为距离合的倒数乘上系数1000000，适应度越大则越容易被选中；变异的概率我设置为0.3，且为了防止陷入种群中所有个体一致的情况，当产生的两个子代相同时，两者进行变异操作。算法关键代码如下（population为事先生成的初始种群）：

```
while iteration<10000:
    new_population=[]#新种群
    for count in range(0,10):
        plen=len(population)
        the_weight=[]#代表权重的数组
        for cnt in range(0,len(population)):
            the_weight.append(1000000/evaluate(population[cnt]))#距离总和越小权重越大

        #轮盘赌选择亲代
        ch1=random.choices(population,the_weight,k=1)
        ch2=random.choices(population,the_weight,k=1)
        #杂交出两个子代
        child1=reproduce_ox(ch1[0],ch2[0])
        child2=reproduce_ox(ch2[0],ch1[0])
        #有一定概率变异
        if random.random()<mutate :
            child1==neighbour3(child1)
            child2==neighbour3(child2)
```

```
        #并入新种群
        new_population.append(child1)
        new_population.append(child2)
    flag=-1
    max=9999999
    #选取最优良的个体保存到下一代
    for i2 in range(0,len(population)):
        if evaluate(population[i2])<max:
            max=evaluate(population[i2])
            flag=i2
    temp_one=population[flag][:]
    #显示当前迭代次数和最优值
    print(iteration, evaluate(temp_one))
    dis_change.append(max)
    population.clear()
    #更新
    population=new_population[0:19]
    new_population.clear()
    population.append(temp_one)
    iteration+=1
    #每隔一段时间进行采样
    if iteration%10==0:…
```

对于可视化部分，为了能更加清楚的展示结果，采用了python中的matplotlib.pyplot库和matplotlib.animation库，进行路径变化动态图的绘制、最终路径的显示和距离的下降过程展示等等，关键代码便是其画图部分：

```
#每隔一段时间进行采样
if ctt%100==0:
    x1=[]
    y1=[]
    for var in curr:
        x1.append(map_[var-1][1])
        y1.append(map_[var-1][2])
    x1.append(map_[0][1])
    y1.append(map_[0][2])
    im=plt.plot(x1, y1, marker = '.', color = 'red',linewidth=1)
    ims.append(im)
ctt+=1
```

```
#保存动态图
ani = animation.ArtistAnimation(fig1, ims, interval=200, repeat_delay=1000)
ani.save("SA.gif",writer='pillow')

fig3 = plt.figure(3)#用于显示总共费用的降低过程
plt.title('the evolution of the cost')
x_=[i for i in range(len(dis_change))]
plt.plot(x_,dis_change)
plt.show()
```

4、创新点&优化

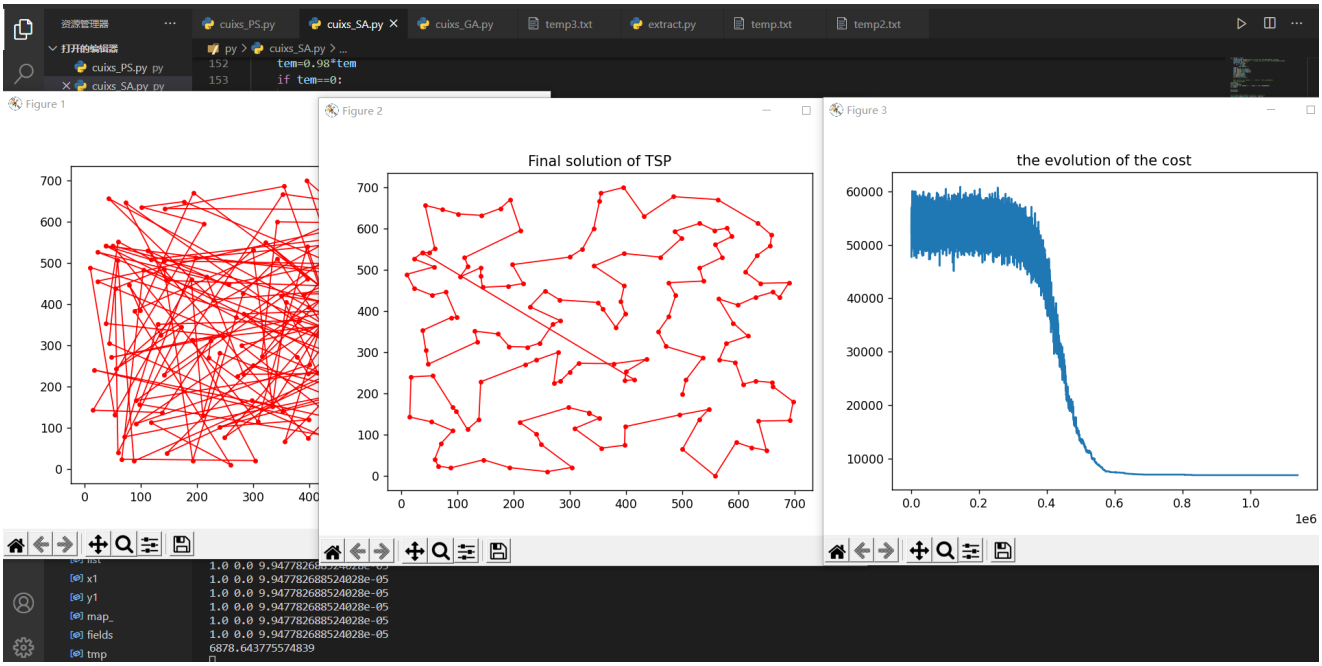
1. 在生成新状态的函数中，我对多种方法进行了尝试，在同一个局部搜索程序中进行实验，然后将结果进行对比（比较的结果在第三部分中进行展示），最终选取了三种方法：随机交换两个城市、随机选取一段城市序列并将其逆序、将序列随机分成四段并交换其中两段。每次随机选择，取得了良好的效果。
2. 在遗传算法的设计中，我在根据一开始设计的伪代码进行编写程序进行测试时，发现效果并不理想，即使增加繁衍的代数、增加每代种群含有的个体数对算法最终得到的结果影响也十分有限，且会陷入最终种群全部一样，难以再提升的困境中。为解决此问题，我在查阅资料后针对原有伪代码做出了两点优化：1、产生初始种群时，采用随机生成的方法产生一半的个体，然后采用贪心算法，随机选取起点生成另外一半个体，这样得到的初代种群的效果明显比单纯随机产生初代种群和全用贪心产生初代种群要好。2、在杂交得到子代个体的过程中，我适当增大了变异的概率，并且将变异的操作进行了修改，原有的变异只是单纯地交换两个城市，优化后的变异函数产生的新状态将比原来更加多样化。
3. 在遗传算法中的亲代进行交叉的操作中，常用的有PMX、OX、PBX、OBX、CX、SEX，在本次实验中我采取的是顺序交叉（OX），即第一步随机选择一对染色体（父代）中几个基因的起止位置（两染色体被选位置相同），第二步，生成一个子代，并保证子代中被选中的基因的位置与父代相同，第三步先找出第一步选中的基因在另一个父代中的位置，再将其余基因按顺序放入上一步生成的子代中。这样交叉的好处是不用进行查重且速度较快，能够满足交叉要求的同时提升算法的效率。

4. 采用路径变化GIF图、最终路径图、随迭代次数增大的距离下降图三种可视化的方法来展示搜索结果。

三、实验结果分析&&对比

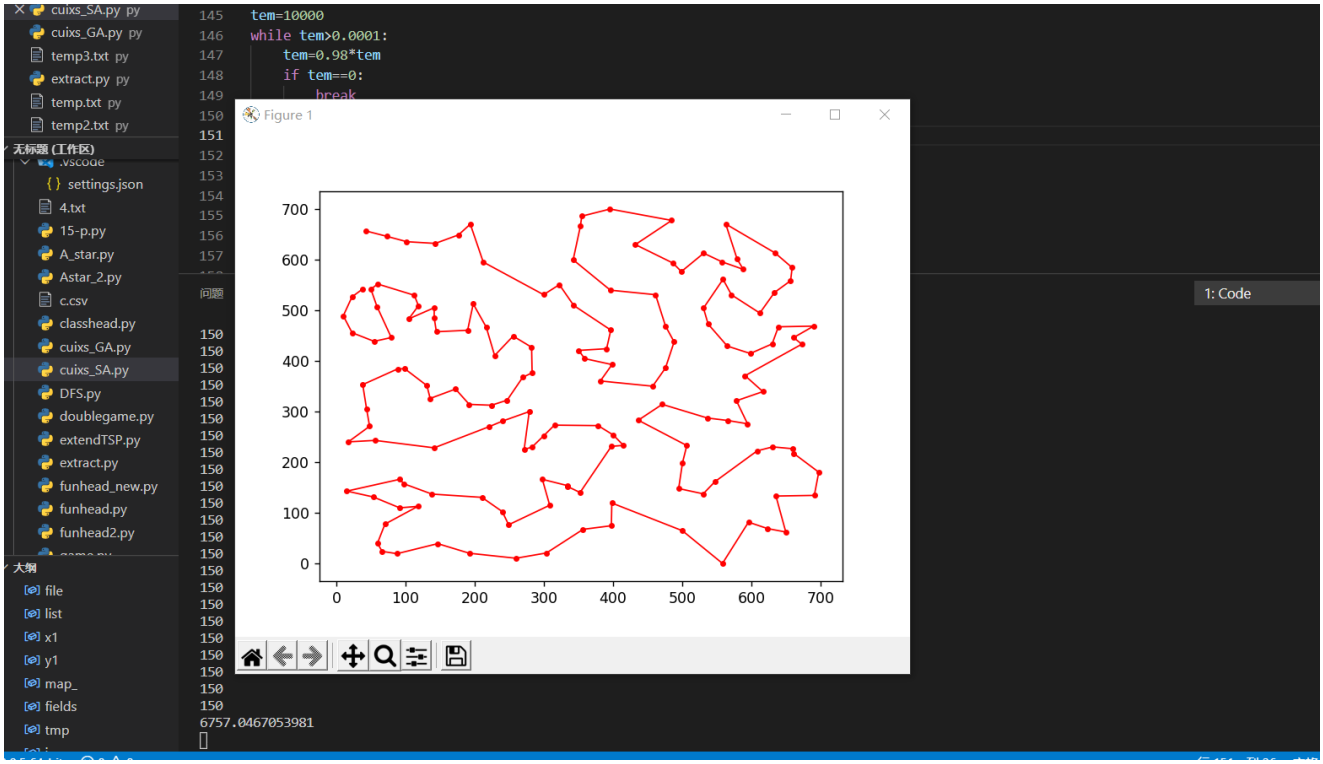
实验结果展示：

运行程序，将要查询的城市保存在同一目录下的temp.txt文本文件中，运行程序即可生成解答，程序运行最后可以得到最终的解、搜索过程中的路径变化GIF图、随迭代次数增大的距离下降图，效果如下：

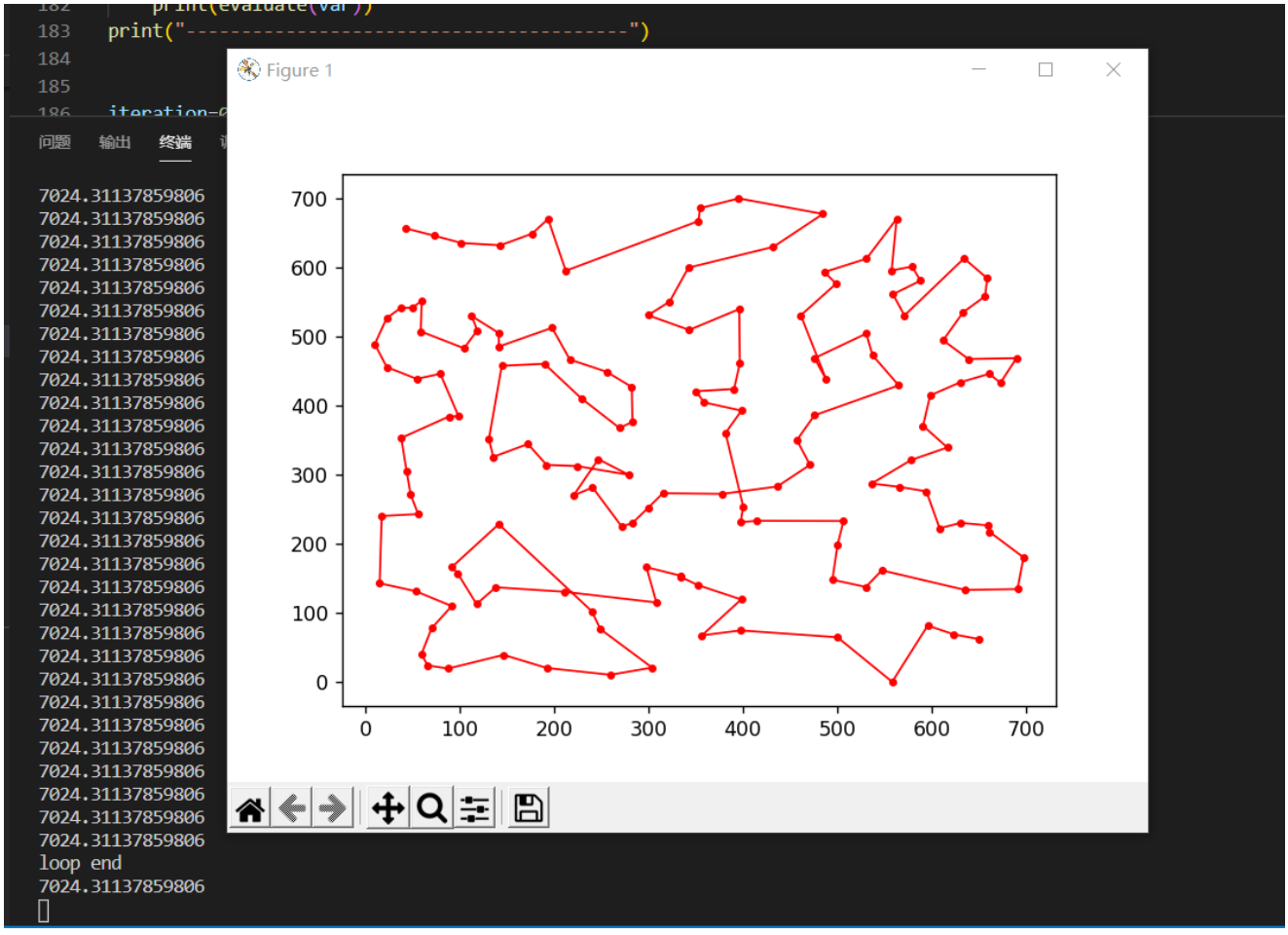


在本次实验中，我从Teaching (uni-heidelberg.de)sh上选取了 **150** 个城市的样例ch150进行解决， 该样例的最优路径查询为 **6528** ，经过实验，模拟退火程序和遗传算法都可以在可观的时间内给出误差范围内的解答，实验结果如下：

模拟退火算法可以得到的路径长度为6757：



遗传算法可以得到的路径长度为7024：



求得的解不超过最优值的10%，完成了实验要求。

多种邻域操作的局部搜索策略求解：

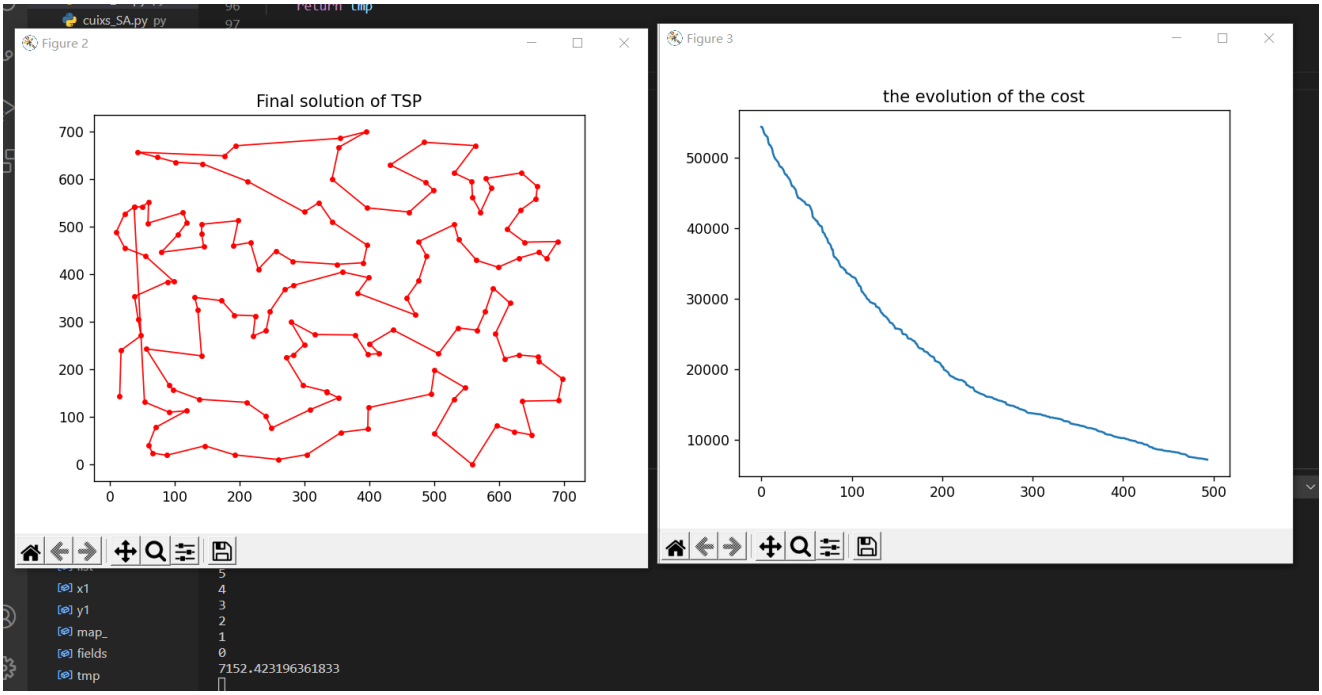
邻域操作	10次搜索的平均结果
op1(交换两个城市)	12720.916435306479
op2(随机选取一段逆序)	8467.348217607696
op3(随机交换两段)	9450.512527916393
op4(每次随机执行其中一种)	7186.486823186904

由实验可知，采取第四种领域操作比较理想。

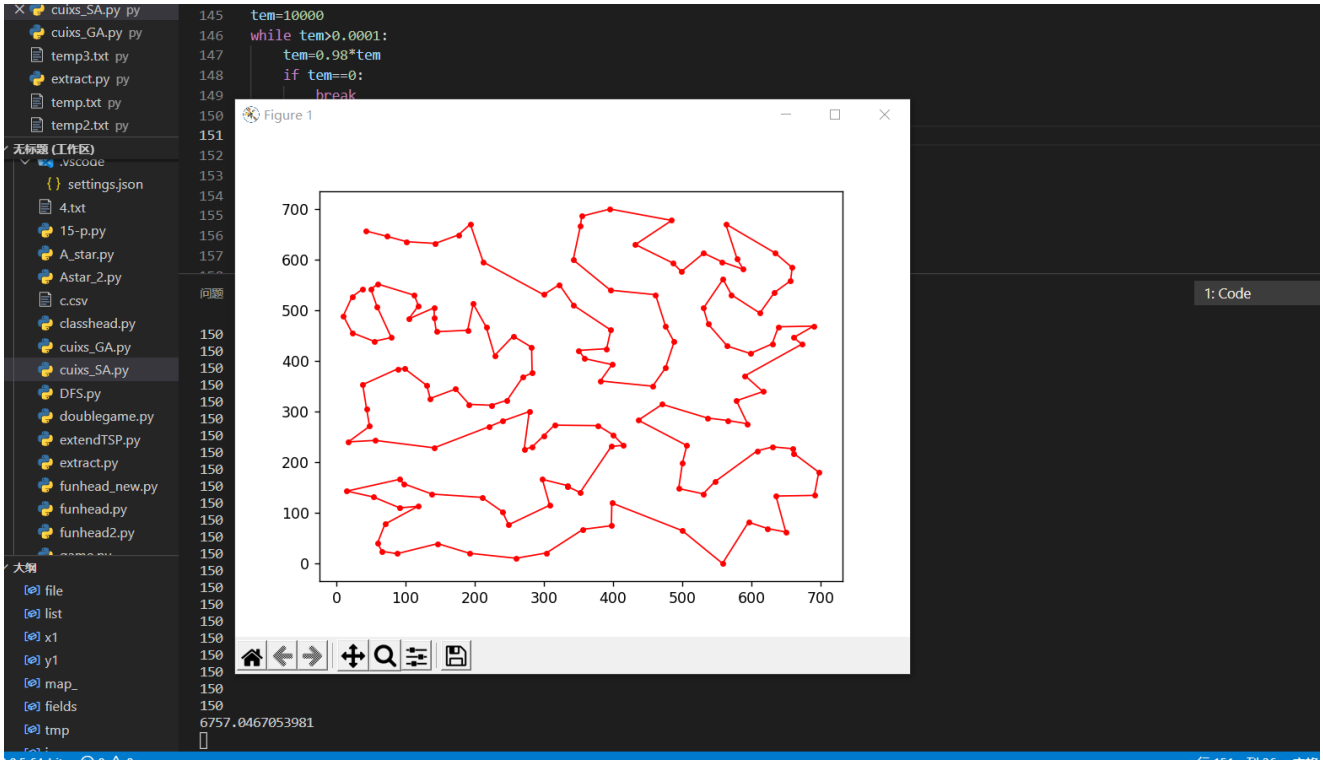
局部搜索策略和模拟退火的比较:

在原有的局部搜索算法上采用模拟退火策略后，算法的性能大大提升，原本的局部搜索策略经常卡在7000到8000之间，且解的有劣性很大一部分取决于初始状态，加入模拟退火策略后可以达到6700甚至更小，更加逼近最优解：

局部搜索：



模拟退火策略：



遗传算法和模拟退火策略的比较:

模拟退火策略和遗传算法都可以得到较为逼近最优解的结果，但遗传算法更依赖于时间，即可能为了得到规定范围内的答案，遗传算法所花费的时间要多，但这也和模拟退火策略的降温系数和达到稳定状态的时间有关。

从生成解的过程GIF来看，模拟退火策略的路径变化较快，遗传算法的路径变化较慢且每次变化差别并不是很明显。

设计高效遗传算法的经验:

1. 确定符合问题的遗传编码，如TSP问题中采取城市序列；01背包问题中可以采取二进制编码。
2. 选取合适的交叉操作，如上文提到的PMX、OX、PBX、OBX、CX、SEX等等。
3. 选取合适的变异操作，变异可以不仅仅是交换两个城市，也可以采用局部搜索中产生邻域的方法。
4. 调参，多次实验选取合适的参数，如变异概率、选取亲本进行交叉的概率等等。
5. 设置初始种群，可以通过改进产生初始种群的方法来减少盲目的操作，提高效率。

单点搜索和多点搜索的优缺点:

名称	优点	缺点
单点搜索	收敛速度较快	容易陷入局部最优解。
多点搜索	有利于跳出局部最优，全局择优	收敛速度较慢。

四、参考资料

- 遗传算法中几种交叉算子小结