

实验报告

——模拟退火

姓名：张子玉

学号：19335276

日期：2021.9.16

摘要：

在 TSPLIB 中选一个大于 100 个城市数的 TSP 问题，

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%，并能够提供可视化图形界面，观察路径的变化和交叉程度。

导言

TSP 问题

旅行商问题（最短路径问题）（英语：travelling salesman problem, TSP）是这样一个问题：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。

启发式算法

启发式算法（heuristic algorithm）是相对于最优化算法提出的。一个问题的最优算法求得该问题每个实例的最优解。启发式算法可以这样定义：一个基于直观或经验构造的算法，在可接受的花费（指计算时间和空间）下给出待解决组合优化问题每一个实例的一个可行解，该可行解与最优解的偏离程度一般不能被预计。

局部搜索策略

局部搜索是解决最优化问题的一种启发式算法。对于某些计算起来非常复杂的最优化问题，比如各种 NP 完全问题，要找到最优解需要的时间随问题规模呈指数增长，因此诞生了各种启发式算法来退而求其次寻找次优解，是一种近似算法（Approximate algorithms），以时间换精度的思想。局部搜索就是其中的一种方法。

邻域动作是一个函数，通过这个函数，对当前解 s ，产生其相应的邻居解集合。在组合优化问题中（TSP 问题属于组合优化问题），领域一般定义为由给定转化规则对给定的问题域上每结点进行转化所得到的问题域上结点的集合。

局部搜索的一般过程是：

1. 随机选择一个初始的可能解 x , 计算 $P=N(x)$ 为 x 在映射 N 下的领域。
2. 如果满足结束条件则 goto (9), 其中结束条件包括规定的循环次数、 P 为空。
3. Begin
4. 选择 P 的一个子集, y 为此子集的一个最优解。
5. 如果 $f(y) < f(x)$, 则将 y 改为临时的最优解, 并将 $P=N(y)$, 其中 f 为一指标函数。
6. 否则, 从 P 中减去刚才选择的子集。
7. goto (2).
8. End
9. 输出计算结果
10. 结束

所以我们需要明细领域操作 $N(x)$ 和评价解的优秀程度的指标函数 $f(x)$ 。

模拟退火

在分子和原子的世界中, 能量越大, 意味着分子和原子越不稳定, 当能量越低时, 原子越稳定。‘退火’是物理学术语, 指对物体加温在冷却的过程。模拟退火算法来源于晶体冷却的过程, 如果固体不处于最低能量状态, 给固体加热再冷却, 随着温度缓慢下降, 固体中的原子按照一定形状排列, 形成高密度、低能量的有规则晶体, 对应于算法中的全局最优解。而如果温度下降过快, 可能导致原子缺少足够的时间排列成晶体的结构, 结果产生了具有较高能量的非晶体, 这就是局部最优解。因此就可以根据退火的过程, 给其在增加一点能量, 然后在冷却, 如果增加能量, 跳出了局部最优解, 这本次退火就是成功的。

模拟退火的一般过程是:

1. 初始化: 初始温度 T (充分大), 初始解状态 S (是算法迭代的起点), 每个 T 值的迭代次数 L
2. 对 $k=1, \dots, L$, goto (3) - (6) 步:
3. 产生新解 S'
4. 计算增量 $\Delta T = C(S') - C(S)$, 其中 $C(S)$ 为代价函数
5. 若 $\Delta T < 0$ 则接受 S' 作为新的当前解, 否则以概率 $\exp(-\Delta T/T)$ 接受 S' 作为新的当前解。
6. 如果满足终止条件则输出当前解作为最优解, 结束程序。
7. T 逐渐减少, 且 $T \rightarrow 0$, goto (2)

模拟退火算法由两个循环控制:

外循环的终止条件为: 达到终止温度。每次循环更新温度, 新温度 = 当前温度 * 退火系数, 实现降温。

内循环的终止条件为: 达到循环次数 L 。每次循环在同一温度下进行局部搜索。

模拟退火需要确定以下变量:

初始温度
终止温度
退火系数
每个温度下的循环次数

需要注意的几点：

1. 初始的温度 $T(0)$ 应选的足够高，使的所有转移状态都被接受。初始温度越高，获得高质量的解的概率越大，耗费的时间越长。
2. 退火速率：最简单的下降方式是指数式下降： $T(n) = \lambda T(n), n = 1, 2, 3, \dots$ ，本次实验中也采用此方法来进行退火速率的控制。
3. 终止条件：本次实验中，将规定一个终止温度，若温度达到终止温度，则退火完成。

实验过程

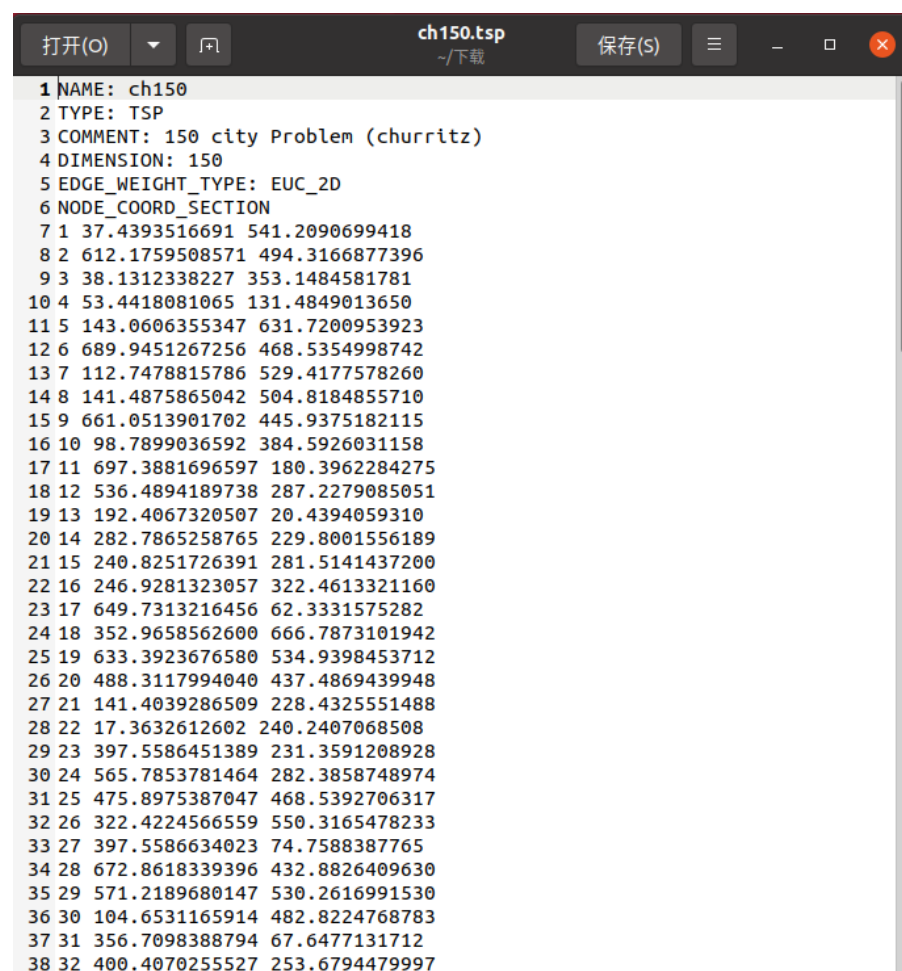
使用 C++ 实现具体算法，将运行得到的不同阶段的解（路径）存在文件“path.txt”中，再使用 python 进行可视化。

在 TSPLIB 中选取“ch150”数据进行求解。

该城市的 TSP 问题的最优解：• **ch150 : 6528**

文件的读取和存放

从 TSPLIB 中下载文件，将文件解压后得到“ch150.tsp”：



```
ch150.tsp
~/下载
保存(S)

1 NAME: ch150
2 TYPE: TSP
3 COMMENT: 150 city Problem (churritz)
4 DIMENSION: 150
5 EDGE_WEIGHT_TYPE: EUC_2D
6 NODE_COORD_SECTION
7 1 37.4393516691 541.2090699418
8 2 612.1759508571 494.3166877396
9 3 38.1312338227 353.1484581781
10 4 53.4418081065 131.4849013650
11 5 143.0606355347 631.7200953923
12 6 689.9451267256 468.5354998742
13 7 112.7478815786 529.4177578260
14 8 141.4875865042 504.8184855710
15 9 661.0513901702 445.9375182115
16 10 98.7899036592 384.5926031158
17 11 697.3881696597 180.3962284275
18 12 536.4894189738 287.2279085051
19 13 192.4067320507 20.4394059310
20 14 282.7865258765 229.8001556189
21 15 240.8251726391 281.5141437200
22 16 246.9281323057 322.4613321160
23 17 649.7313216456 62.3331575282
24 18 352.9658562600 666.7873101942
25 19 633.3923676580 534.9398453712
26 20 488.3117994040 437.4869439948
27 21 141.4039286509 228.4325551488
28 22 17.3632612602 240.2407068508
29 23 397.5586451389 231.3591208928
30 24 565.7853781464 282.3858748974
31 25 475.8975387047 468.5392706317
32 26 322.4224566559 550.3165478233
33 27 397.5586634023 74.7588387765
34 28 672.8618339396 432.8826409630
35 29 571.2189680147 530.2616991530
36 30 104.6531165914 482.8224768783
37 31 356.7098388794 67.6477131712
38 32 400.4070255527 253.6794479997
```

从该文件中选取我们所需要的城市编号、x 坐标、y 坐标数据，复制到“tspdata.txt”中，后续我们通过读取该文件来导入城市坐标。

文件的读取：

（详见下面的“初始化操作”）

文件的存放：

局部搜索：每 10000 次循环获得新解时，将当前的解（路径）存入“path.txt”

模拟退火：温度每更新 100 次，将当前的解（路径）存入“path.txt”

使用局部搜索策略求解

实现框架：

```
1. 随机获得一个初始解 x，令最优解=x
2. while 当前循环次数≤规定的最大循环次数
3.     依据领域操作，产生新解 y
4.     if f(y)<f(x)           //新解更优秀
5.         最优解=y
```

初始化操作：

在初始化中，需要将城市的坐标从外部文件“tspdata.txt”中读入并存储，并生成一个初始解。

在全局范围中，设置：

```
int city_list[N]; // 存放一个解
double city_pos[N][2]; // 存放城市的坐标
```

city_list[]：存放当前的解，这里 city_list 最后没有存放回到起点的城市坐标，因为该坐标就为 city_list[0]，所以，在路径中总共要走过 N+1 个城市，而 city_list 大小设置为 N。

city_pos[][]：存放每个城市的坐标。city_pos[i]表示是第 i 个城市的坐标，所以此二维数组的第一维度大小为 N；city_pos[i][0]表示是第 i 个城市的 x 坐标，city_pos[i][1]表示是第 i 个城市的 y 坐标。

编写函数 **void init()**来完成初始化：将城市的坐标从外部文件“tspdata.txt”中读入并存储，并生成一个初始解。

评价解的优秀程度的指标函数 **f (x)**：

利用解的路径长度作为评价该解的优秀程度。

计算解的路径长度的方法：从头到尾遍历解数组，邻近的两个城市根据坐标算直线距离，最后累加起来即可。

计算两个城市之间的直线距离函数：

```
double distance(double* city1, double* city2)
```

计算解的路径长度函数：

```
double path_len(int* list)
```

设置四种局部搜索的策略来进行找邻域中的解：（详见 `void mutation()`）

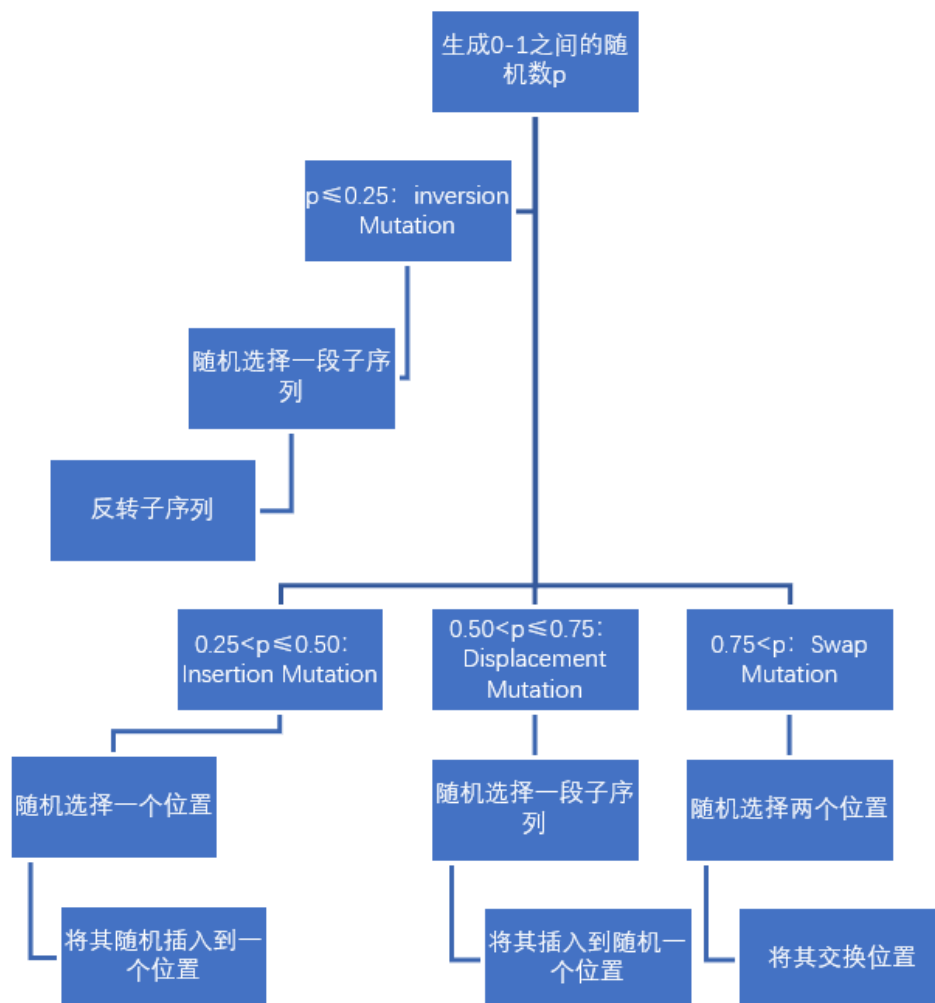
生成 0-1 之间的一个随机数 p ，若：

$p \leq 0.25$: Inversion Mutation

$0.25 < p \leq 0.50$: Insertion Mutation

$0.50 < p \leq 0.75$: Displacement Mutation

$0.75 < p$: Swap Mutation



参考老师给出的伪代码：

6.3 Mutation Operators

1. Inversion Mutation

```

procedure : Inversion Mutation
input : chromosome  $v_1, v_2$ ,
        length of chromosome  $l$ 
output : offspring  $v'$ 
begin
    // step 1: select subtour at random
     $s \leftarrow \text{random}[1:l-1]$ ;
     $t \leftarrow \text{random}[s+1:l]$ ;
    // step 2: produce offspring by
        copying inverse string of
        substring
     $S \leftarrow \text{invert}(v[s:t])$ ;
     $v' \leftarrow v[1:s+1] \parallel S \parallel v[t+1:l]$ ;
    output offspring  $v'$ ;
end
    
```

step 1: select subtour at random



step 2: produce offspring by copying inverse string of substring



v : parent chromosome
 v' : offspring chromosome
 t : end position of substring
 S : inverse string of substring
 $\text{invert}(string)$: inversely changing order of $string$

57

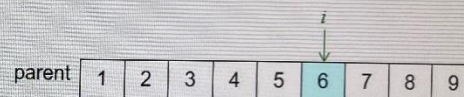
6.3 Mutation Operators

2. Insertion Mutation

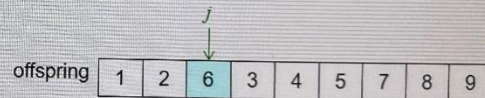
```

procedure : Insertion Mutation
input : chromosome  $v_1, v_2$ ,
        length of chromosome  $l$ 
output : offspring  $v'$ 
begin
    // step 1: select a position in
        parent 1 at random
     $i \leftarrow \text{random}[1:l]$ ;
    // step 2: insert selected value in
        randomly selected
        position parent 2
     $j \leftarrow \text{random}[1:l-1]$ ;
     $W \leftarrow v[1:i-1] \parallel v[i+1:l]$ ;
     $v' \leftarrow W[1:j-1] \parallel v[i] \parallel W[j:l-1]$ ;
    output offspring  $v'$ ;
end
    
```

step 1: select a position in parent 1 at random



step 2: insert selected value in randomly selected position of parent 2



v : parent chromosome
 v' : offspring chromosome
 j : selected position in parent 2
 W : working data set

58

6.3 Mutation Operators

3. Displacement Mutation

procedure : Displacement Mutation

input : chromosome v_1, v_2 ,
length of chromosome l

output : offspring v'

begin

// step 1: select subtour

$s \leftarrow \text{random}[1:l-1]$;

$t \leftarrow \text{random}[s+1:l]$;

// step 2: insert subtour in a
random position

$n \leftarrow t-(s-1)$;

$i \leftarrow \text{random}[1:l-n]$;

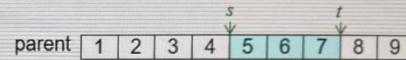
$W \leftarrow v[1:s-1] \parallel v[t+1:l]$;

$v' \leftarrow W[1:i-1] \parallel v[s:t] \parallel W[i:l-n]$;

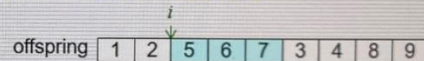
output offspring v' ;

end

step 1: select subtour



step 2: insert subtour in a random position



v : parent chromosome l : length of chromosome
 v' : offspring chromosome s : start position of substring
 t : end position of substring n : length of subtour
 i : insert position
 W : working data set

59

6.3 Mutation Operators

4. Swap Mutation

procedure : Swap Mutation

input : chromosome v_1, v_2 ,
length of chromosome l

output : offspring v'

begin

// step 1: select two position at random

$i \leftarrow \text{random}[1:l-1]$;

$j \leftarrow \text{random}[i+1:l]$;

// step 2: produce offspring by swapping

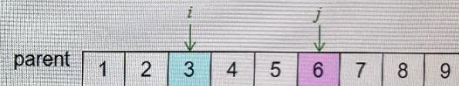
selected positions

$v' \leftarrow v[1:i-1] \parallel v[j] \parallel v[i+1:j-1] \parallel v[i] \parallel v[j+1:l]$;

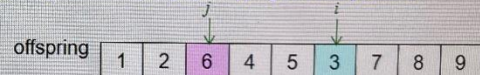
output offspring v' ;

end

step 1: select two position at random



step 2: produce offspring by swapping selected positions



v : parent chromosome l : length of chromosome
 v' : offspring chromosome i : selected position
 j : selected position

60

当没达到设定的最大循环次数时，循环局部搜索：

伪代码如下：

1. **while** 当前循环次数 \leq 规定的最大循环次数

```

2. 依据领域操作，产生新解 y
3. if f(y)<f(x) //若新解更优秀，则接受
4. 最优解=y
5. else //否则，保持旧解
6. 最优解=x

```

使用模拟退火策略求解

在局部搜索的基础上，不改变局部搜索的策略，仅仅添加上温度的控制，以此来让算法在一定的概率下接受差解。

模拟退火的温度控制机制由以下属性控制：

初始温度：T0

终止温度：T_end

退火系数：q

模拟退火算法由两个循环控制：

外循环的终止条件为：达到终止温度。每次循环更新温度，新温度 = 当前温度 * 退火系数，实现降温。

内循环的终止条件为：达到循环次数 L。每次循环在同一温度下进行局部搜索。

```

#define T0 50000.0 // 初始温度
#define T_end (1e-8) //终止温度
#define q 0.99 // 退火系数
#define L 10000 // 每个温度时的迭代次数

```

核心框架：

内循环中，找到一个新解时，判断其与旧解的差值 df。若 df<0（表示是更优解），则无条件接收新解；否则，以概率 $\exp(-df/T)$ 接收差解。

执行完一次温度的 L 次内循环，进行降温操作。

伪代码如下：

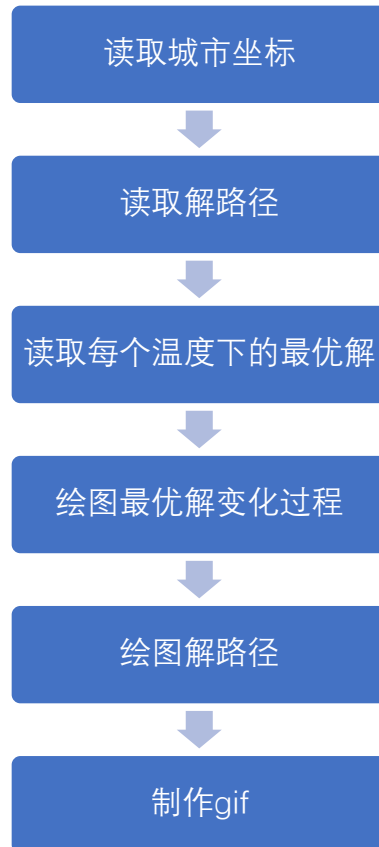
```

1. while 温度 T>终止温度 T_end
2.   for i <-0 to 每个温度下循环次数 L
3.     产生新解
4.     df<-f1-f2 //df 为新解和旧解之差
5.     if df >=0 //新解更差
6.       按概率  $\exp(-df/T)$  采用新解
7.     else
8.       保留旧解
9.   降温

```


可视化

使用 python 编写。
流程图如下：



读取“tspdata.txt”，获得城市坐标：

使用 pandas 的 `read_table`，返回一个 DataFrame（二维）。
详见函数：

```
read_tsp_data(filePath)
```

将最终获得的二维表打印，结果如下：

```

          x          y
0    37.439352  541.20907
1   612.175951  494.316688
2    38.131234  353.148458
3    53.441808  131.484901
4   143.060636  631.720095
..      ...      ...
145  299.588137  530.588962
146  334.274876  152.149457
147  690.965859  134.579331
148   48.079812  270.968067
149   91.646765  166.354116
|
[150 rows x 2 columns]

```

行对应的是城市的编号-1，x 列对应城市的 x 坐标，y 列对应城市的 y 坐标

读取“path.txt”，获得多个解的路径：

同读取坐标时一样，使用 pandas 的 read_table。返回 paths 数组，其中每个元素为每个解对应的路径。

详见函数：

```
read_tsp_path(filePath)
```

读取“len.txt”，获得每个温度下的最优解：

同读取坐标时一样，使用 pandas 的 read_table。返回 pathData 数组，其中每个元素为每个温度下的最优解。

详见函数：

```
read_tsp_optimum(filePath)
```

根据每个温度下的最优解，绘图：

令标题为“TSP simulated annealing”，x 轴为“退火次数”，y 轴为“当前全局最优解”

详见函数：

```
create_figure_optimum(optimum)
```

根据城市坐标以及路径，绘图：

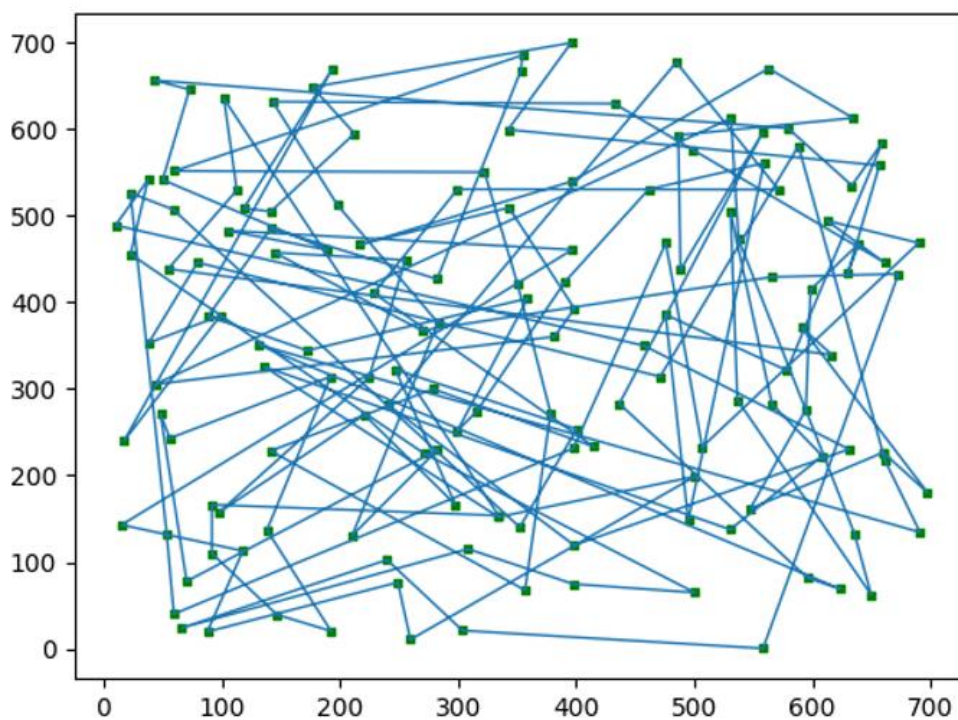
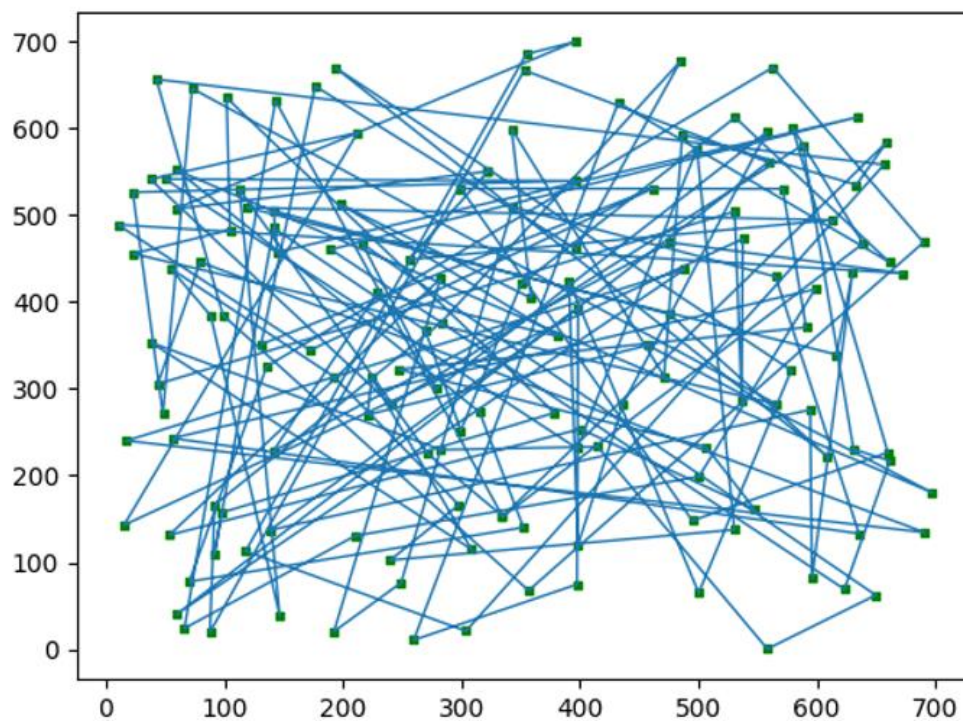
利用 matplotlib 绘图。

详见函数：

```
create_figure(citys, paths)
```

打印图片展示：

下为抽选的图片中的两个，更详细的请见“结果分析”。



由多幅图片，创建 gif:

详见函数：

```
create_gif(gif_name, paths, duration = 0.3)
```

结果分析

局部搜索

输出内容：

- 每产生一个新解，输出当前最优解（路径长度）
- 运算结束后，输出最佳路径（用城市编号来指明路径）
- 得到的最优解和实际最优解之间的误差
- 运算耗时

程序执行结果示例：

运行过程中，会不断输出当前最优解。

```
当前最优解: 7619.89
当前最优解: 7613.88
当前最优解: 7610.89
当前最优解: 7577.13
当前最优解: 7546.54
当前最优解: 7541.49
当前最优解: 7527.82
当前最优解: 7512.17
当前最优解: 7486.74
当前最优解: 7485.86
当前最优解: 7428.1
当前最优解: 7411.99
当前最优解: 7387.75
当前最优解: 7361.94
当前最优解: 7350.26
当前最优解: 7341.36
当前最优解: 7294.71
当前最优解: 7282.28
当前最优解: 7274.53
当前最优解: 7227.26
当前最优解: 7176.68
当前最优解: 7171.39
当前最优解: 7165.14
当前最优解: 7162.86
当前最优解: 7160.73
当前最优解: 7138.22
当前最优解: 7138.22
```

运行结束后，会输出最佳路径、路径长度、误差、耗时。

```
最佳路径:
56-->33-->126-->52-->79-->59-->16-->111-->105-->54-->92-->90-->46-->138-->134-->131-->32-->23-->38-->67-->43-->109-->51-->20-->25-->81-->110-->47-->2-->37-->6-->28-->9-->42-->120-->139-->40-->53-->12-->116-->101-->24-->118-->127-->69-->36-->61-->11-->148-->130-->17-->66-->60-->140-->117-->57-->39-->41-->129-->27-->31-->123-->74-->13-->136-->112-->64-->145-->144-->147-->49-->72-->80-->14-->122-->77-->133-->15-->78-->21-->44-->71-->45-->106-->91-->119-->68-->128-->104-->4-->115-->150-->125-->22-->149-->62-->3-->48-->63-->30-->89-->96-->113-->10-->94-->88-->121-->35-->93-->124-->8-->84-->7-->34-->73-->76-->87-->1-->98-->103-->82-->95-->107-->5-->100-->143-->97-->146-->26-->75-->18-->142-->85-->65-->132-->137-->102-->114-->99-->19-->29-->86-->108-->70-->135-->50-->55-->58-->141-->83
最佳路径长度: 7109.05
误差: 0.0890086
耗时: 38.983sec
```

多次运行结果：

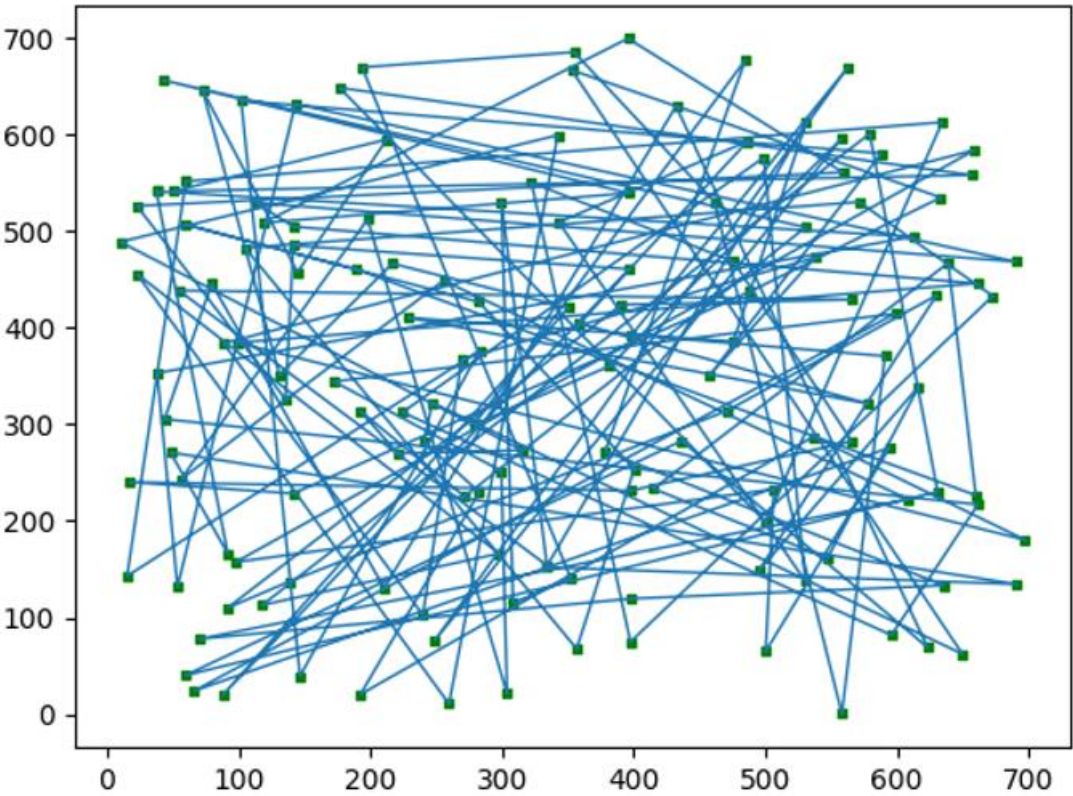
次数	最优解	误差	耗时
1	7044.41	7.91%	17.89
2	7109.05	8.90%	16.37
3	7179.43	9.97%	16.1
4	7418.44	13.64%	15.58
5	7178.02	9.95%	15.5
6	7211.45	10.64%	16.545
7	6973.02	6.82%	15.647
8	7317.81	12.09%	14.69
9	6984.08	7.01%	15.8
10	7116.18	9.12%	13.946

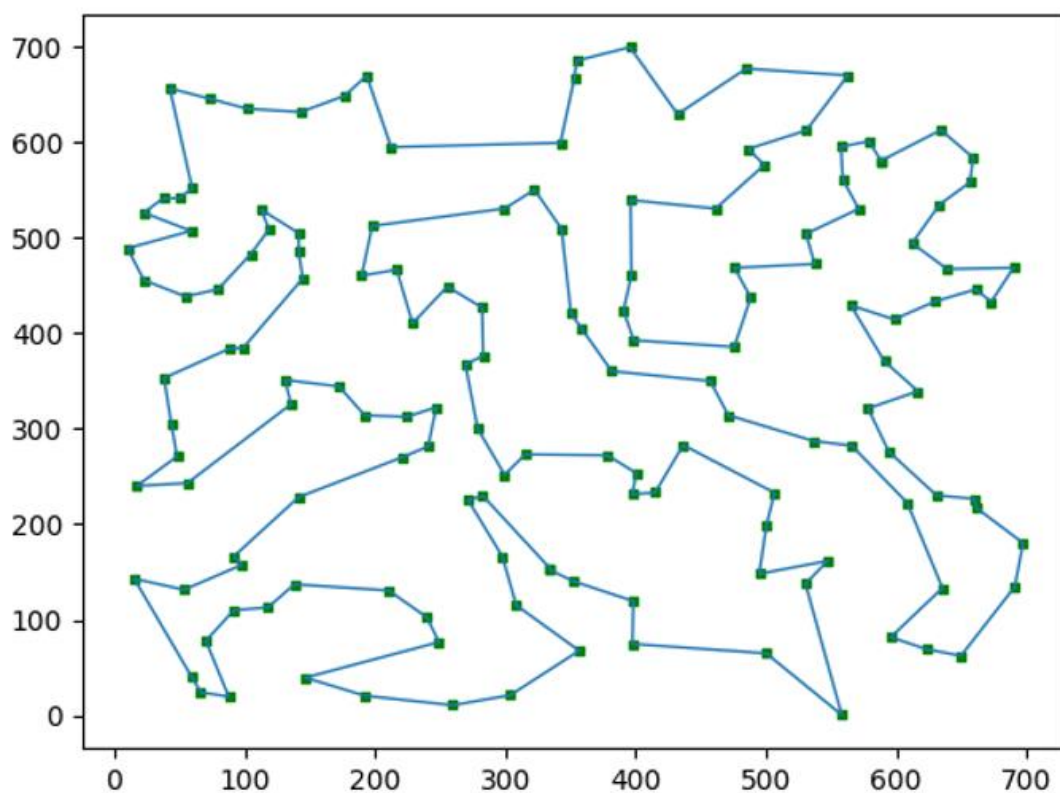
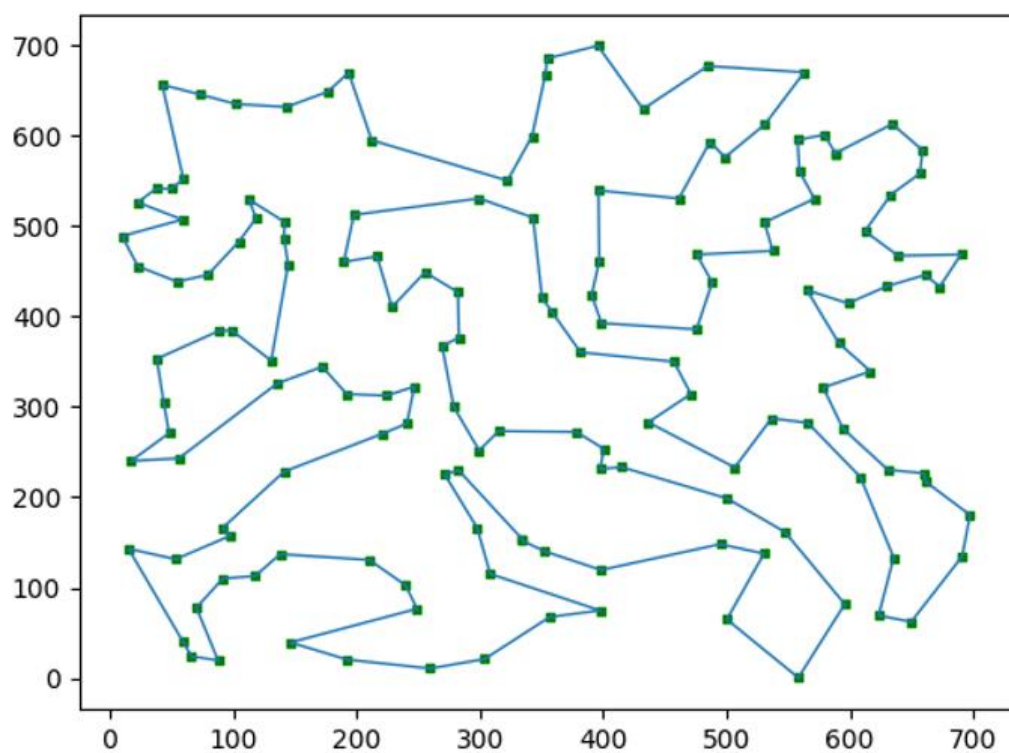
根据上述十次测试结果可以得出：

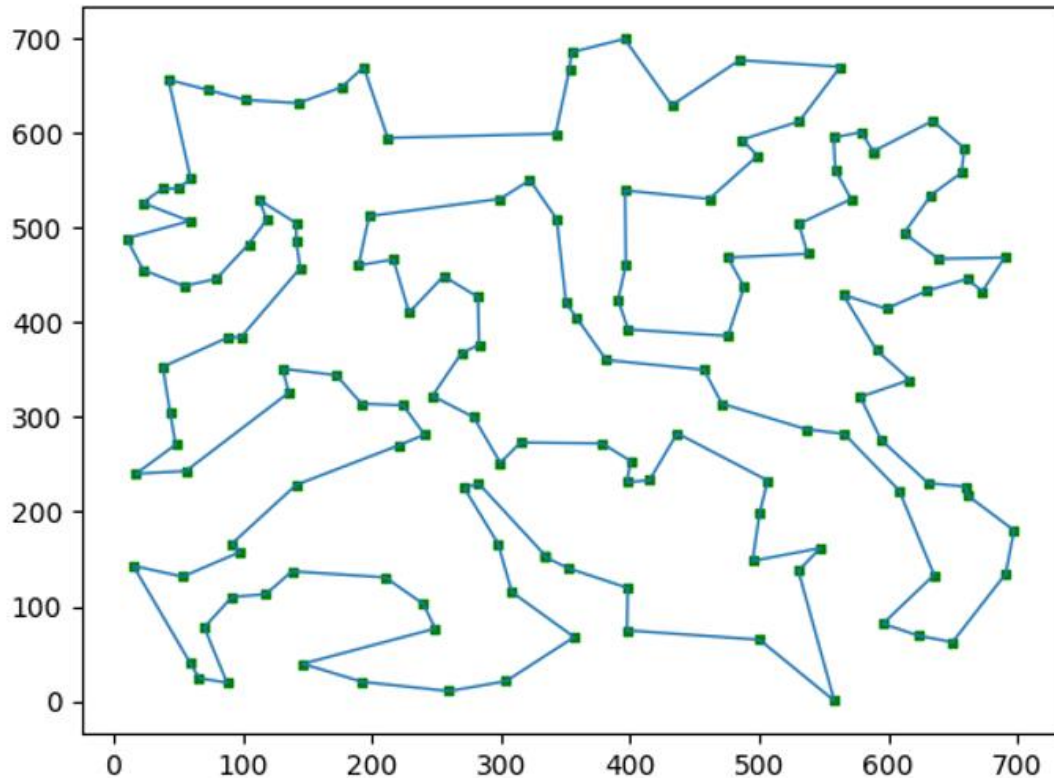
最好解	6984.08
最差解	7418.44
平均解	7152.96
平均误差	9.57%
平均耗时	15.7

显示路径变化：

(以某次运行得到的结果为例)







可以看到随着循环次数的增加，解的路径的交叉程度明显变小，最终获得的路径中已无交叉，找到了较好的路径解。

路径的变化一开始非常的迅速，迅速的由交叉程度非常大的解变化为交叉程度较小的解，但是后面解的变化非常微小。

模拟退火

输出内容：

每产生一个新解，输出当前温度和当前最优解（路径长度）
运算结束后，输出最佳路径（用城市编号来指明路径）
得到的最优解和实际最优解之间的误差
运算耗时

程序执行结果示例：

运行过程中，会不断输出当前温度和当前最优解。

```
当前温度: 36985 当前最优解: 54995.3
当前温度: 36615.2 当前最优解: 51634.7
当前温度: 36249 当前最优解: 54449.3
当前温度: 35886.5 当前最优解: 50082.7
当前温度: 35527.7 当前最优解: 51498.4
当前温度: 35172.4 当前最优解: 54508
当前温度: 34820.7 当前最优解: 54423.3
当前温度: 34472.5 当前最优解: 50041.1
当前温度: 34127.7 当前最优解: 54585
当前温度: 33786.5 当前最优解: 49628.2
当前温度: 33448.6 当前最优解: 55494.1
当前温度: 33114.1 当前最优解: 54203.3
当前温度: 32783 当前最优解: 56538.7
当前温度: 32455.1 当前最优解: 54878.1
当前温度: 32130.6 当前最优解: 55419.2
当前温度: 31809.3 当前最优解: 51312.7
当前温度: 31491.2 当前最优解: 53331.1
当前温度: 31176.3 当前最优解: 53536.8
当前温度: 30864.5 当前最优解: 53611.1
当前温度: 30555.9 当前最优解: 52325.1
当前温度: 30250.3 当前最优解: 53244.8
当前温度: 29947.8 当前最优解: 50910.6
当前温度: 29648.3 当前最优解: 57163.2
当前温度: 29351.8 当前最优解: 52782.8
当前温度: 29058.3 当前最优解: 57545.6
当前温度: 28767.7 当前最优解: 56457.5
当前温度: 28480.1 当前最优解: 49804.3
当前温度: 28195.3 当前最优解: 54068.1
当前温度: 27912.2 当前最优解: 54085.2
```

运行结束后，会输出最佳路径、路径长度、误差、耗时。

```
最佳路径:
21-->150-->104-->4-->115-->44-->71-->45-->128-->68-->119-->91-->106-->13-->74-->123-->136-->112-->64-->80-->14-->72-->49-->147-->1
44-->145-->31-->27-->129-->41-->101-->116-->39-->57-->117-->140-->60-->66-->17-->130-->148-->11-->61-->36-->69-->127-->118-->40-->
139-->53-->24-->12-->43-->67-->38-->23-->32-->131-->77-->122-->133-->16-->59-->15-->78-->79-->121-->88-->94-->10-->113-->63-->30-->
34-->98-->103-->7-->84-->8-->89-->96-->35-->124-->93-->126-->52-->111-->105-->33-->92-->54-->90-->46-->138-->134-->109-->51-->20-->
47-->120-->42-->9-->28-->6-->37-->2-->19-->99-->114-->102-->108-->70-->135-->86-->29-->81-->110-->25-->141-->58-->55-->50-->137-->
132-->65-->85-->142-->18-->75-->83-->56-->26-->146-->97-->143-->100-->5-->107-->95-->82-->1-->87-->76-->73-->48-->3-->62-->149-->
22-->125
最佳路径长度: 6908.89
误差: 0.0583472
耗时: 385.955sec
```

多次运行结果：

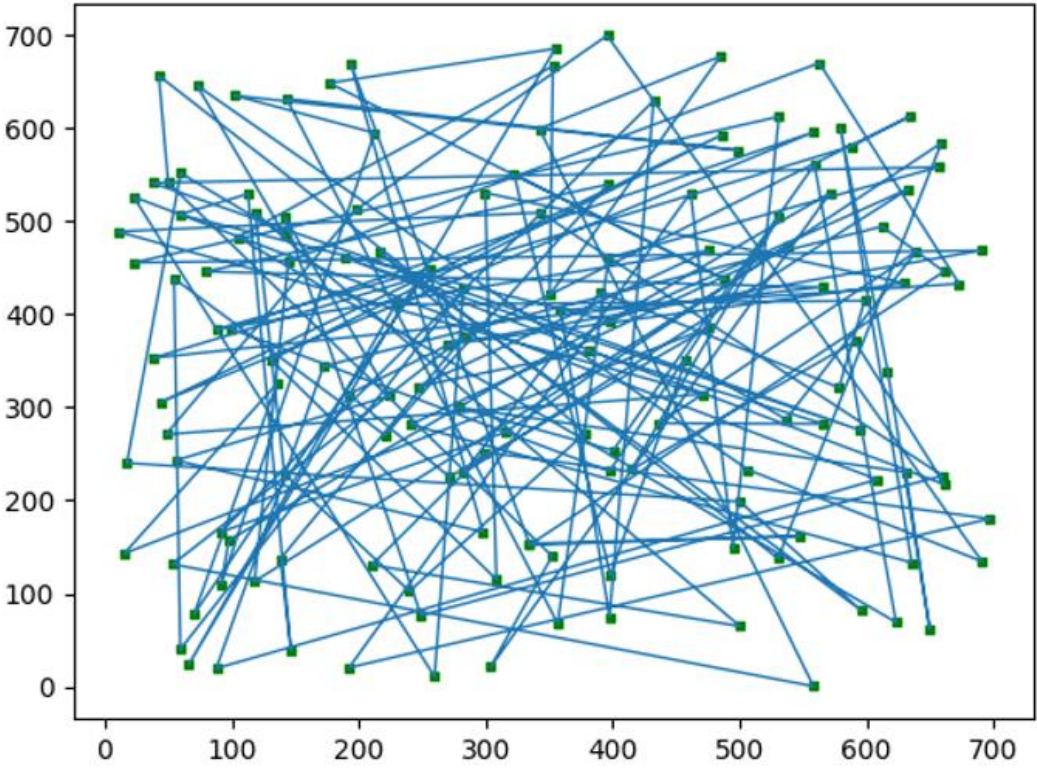
次数	最优解	误差	耗时(秒)
1	6908.89	5.83%	385.955
2	6744.64	3.32%	366.68
3	6871.37	5.26%	409.577
4	6773.32	3.76%	349.483
5	6843.31	4.83%	337.971
6	6701.37	2.66%	408.386
7	6804.02	4.22%	346.389
8	6709.65	2.78%	358.395
9	6696.69	2.58%	357.483
10	6953.53	6.52%	353.751

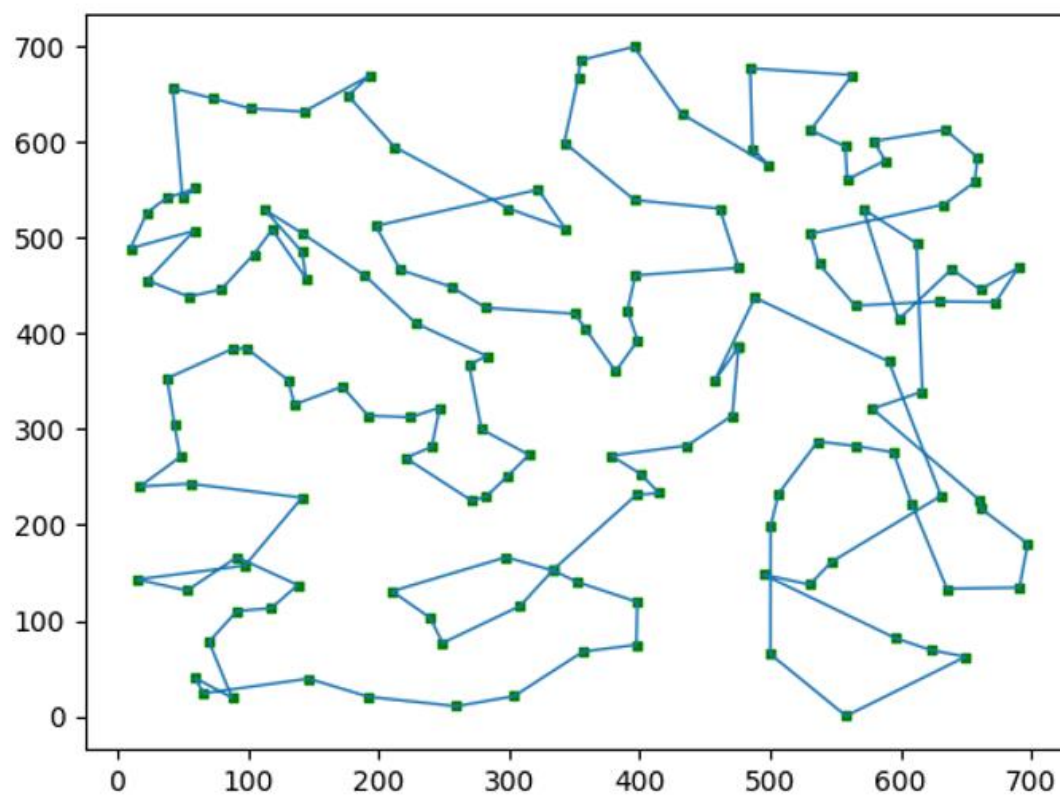
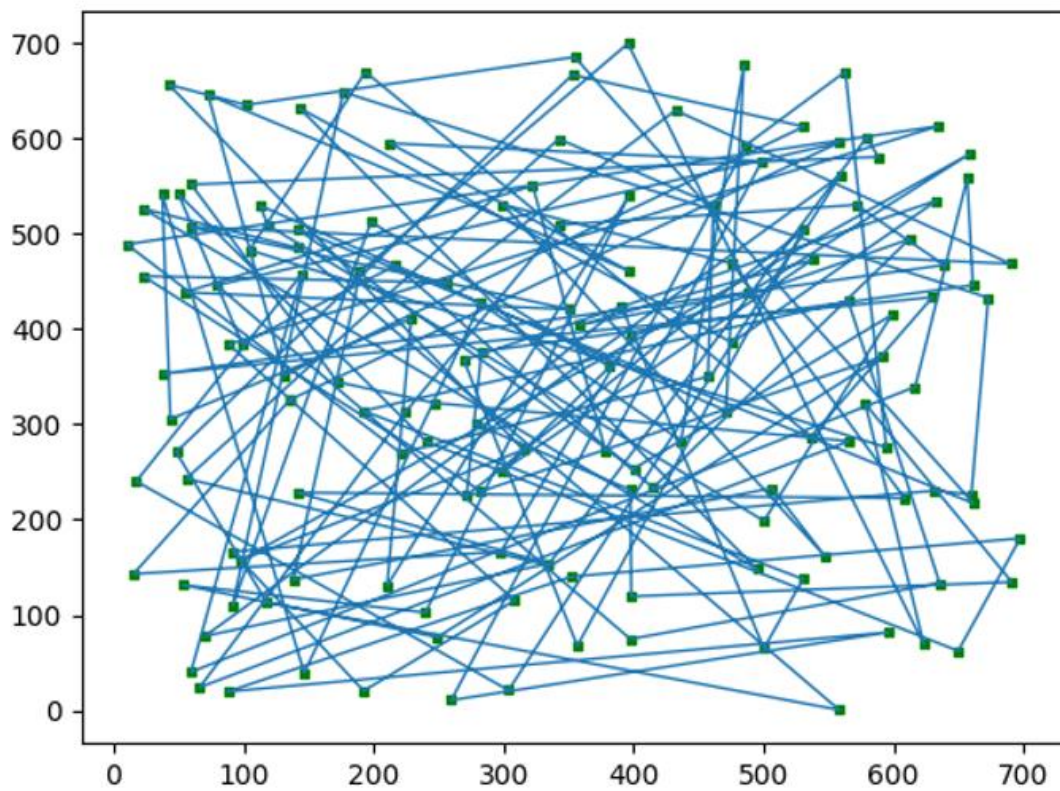
根据上述十次测试结果可以得出：

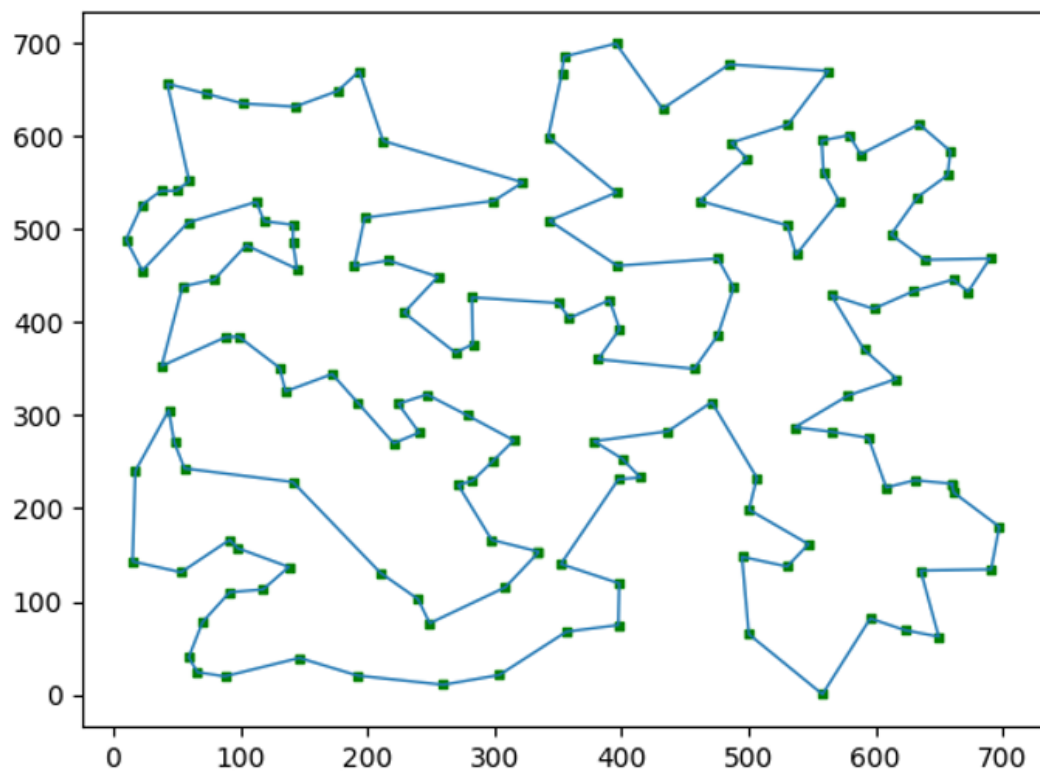
最好解	6696.69
最差解	6953.53
平均解	6800.76

平均误差	4.18%
平均耗时	367.4

显示路径变化：
(以某次运行得到的结果为例)

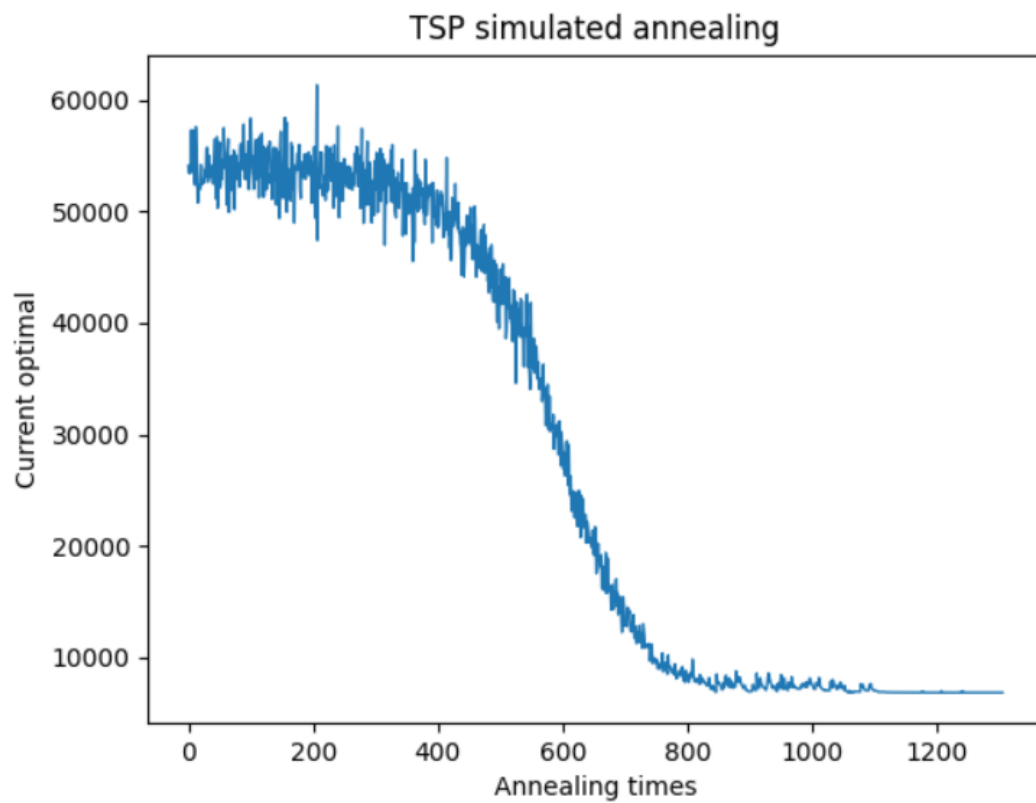






可以看到随着退火的过程，解的路径的交叉程度明显变小，最终获得的路径中已无交叉，找到了较好的路径解。

显示退火过程中最优解（路径长度）的变化：



可以看到模拟退火的过程是在中间某一时刻，解发生剧烈变化，快速接近最优解，且时时刻刻解都是在上下波动的。

局部搜索和模拟退火比较

精确度比较：

通过上述十次实验可以看出，模拟退火的平均误差为 4% 左右，而局部搜索的平均误差在 9% 左右，这说明模拟退火比局部搜索能找到更加精确的解。

稳定性比较：

局部搜索的极差（最好解-最差解）为 400 多，而模拟退火的极差为 200 多，这说明模拟退火运算获得的结果更加稳定。

用时比较：

但是模拟退火使用的时间更长，模拟退火的平均耗时是局部搜索的 20 倍多，这是因为模拟退火在每一个温度都进行了相当多次的局部搜索，通过概率接受差解来跳出局部最优，达到更高的精度。

结论

一开始的实验中，为了简单起见，我只选用了老师给出的局部搜索策略中的 2 个，不管怎么调整循环次数，误差范围均大于 10%。后来，考虑到搜索过程中，循环到后面，都不再增加新解，考虑到应该是无法再演变出新解，即变化方式不够多，所以尝试添加多种搜索方法，之后误差能很容易下降到 10% 以内。

局部搜索的循环次数我定为了 500000。这个数字是经过多次实验得出来的，若循环次数过小，那么误差将会很大；若循环次数过大，程序运行到后期，将不再产生新解，而徒增循环次数，只会增加运行时间，这是没有意义的行为。

经过对局部搜索策略和循环次数的调整，单单靠局部搜索得到的解的平均误差能控制在 9% 左右，但也只是仅仅小于 10% 一点点。而新加入了模拟退火的机制后，误差减小的非常明显，误差下降到了 4% 左右。

模拟退火算法是基于随机搜索的，即在解的空间中展开随机搜索的。当问题的空间很大，而可行解比较多，并且对解的精度要求不高时，随机搜索是很有效的解决办法，因为其他的做法在这个时候时空效率不能让人满意。而借助演化思想和群集智能思想改进过的随机算法更是对解的分布有规律的复杂问题有良好的效果。

利用模拟退火方法能较好的解决 TSP 问题，调整退火速度，初温，接受温度等参数可以控制到解的精度达到自己的满意范围内。

经过多次实验，决定采用初始温度 = 50000；退火速度使用简单的指数减小，系数为 0.99；在终止温度上的调整较大，一开始参考一些博客，将终止温度设置的非常小（例如 $1e-8$ ），但是这样设置的话，运行的时候，后面阶段解的变化很小而且运行速度较慢，所以我增大了终止温度至 0.1，这时候程序运行速度大大提升，而且误差也没有增加很多。

总而言之，经过本次实验我对局部搜索和模拟退火有了更深的理解，并通过编写算法运行测试体会到了两种算法在结果上的区别和操作中的联系，收获非常大。

主要参考文献

https://en.wikipedia.org/wiki/Simulated_annealing

[TSPLIB \(uni-heidelberg.de\)](http://www.tsplib.uni-heidelberg.de)

范展, 梁国龙, 林旺生,等. 求解 TSP 问题的自适应邻域搜索法及其扩展[J]. 计算机工程与应用, 2008(12):75-78.