实验报告

————遗传算法

姓名：张子玉　　学号：19335276　　日期：2021.10.30

摘要：

用遗传算法求解 TSP 问题（问题规模等和模拟退火求解 TSP 实验同），要求：
1.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）
2.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
3.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。

要求 1 在实验过程中已设计完成，变异操作采用的是和模拟退火的局部搜索一样的操作。
要求 2、3 见 结果分析。

导言

TSP 问题

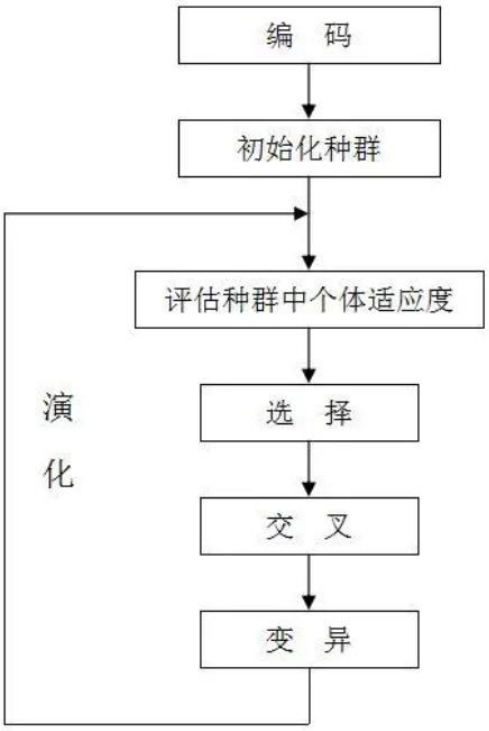
旅行商问题 （最短路径问题）(英语： travelling salesman problem, TSP) 是这样一个问题: 给定一系列城市和每对城市之间的距离, 求解访问每一座城市一次并回到起始城市的最短回路。

遗传算法

遗传算法（Genetic Algorithm）是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群（population）开始的，而一个种群则由经过基因（gene）编码的一定数目的个体(individual)组成。
遗传算法是从代表问题可能潜在的解集的一个种群（population）开始的，而一个种群则由经过基因（gene）编码的一定数目的个体(individual)组成。每个个体实际上是染色体(chromosome)带有特征的实体。
染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，它决定了个体的形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。因此，在一开始需要实现从表现型到基因型的映射即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，如二进制编码。
初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代（generation）演化产生出越来越好的近似解，在每一代，根据问题域中个体的适应度（fitness）大小选择（selection）个体，并借助于自然遗传学的遗传算子（genetic operators）进行组合交叉（crossover）和变异（mutation），产生出代表新的解集的种群。
这个过程将导致种群像自然进化一样的后生代种群比前代更加适应于环境, 末代种群中的最

优个体经过解码（decoding），可以作为问题近似最优解。

遗传算法的过程如下：



1. 首先寻找一种对问题潜在解进行“数字化”编码的方案。（建立表现型和基因型的映射关系）
2. 随机初始化一个种群，种群里面的个体就是这些数字化的编码。
3. 对个体的基因进行解码获得表现型，用适应性函数对个体作适应度评估。
4. 用选择函数按照某种规定择优选择，即淘汰部分表现不好的个体。
5. 选择父母亲进行繁衍，用父母的染色体按照一定的方法进行交叉，生成子代
6. 对子代染色体进行变异，至此，获得一个新的种群
7. 循环 3-6，直至获得最优解

我们可以看到，在遗传算法中，几个重要的影响因素是：

- 初始解的多样性，初始解越多样，找到全局最优的几率越大。
- 种群大小的选择。种群大小越大，越容易跳出局部最优，但是同时也导致运算更慢。同样时间内的迭代代数更少。
- 适应度函数，应该用什么算法来计算解的适应度，影响了解之间的区分度的高低，从而进一步影响了收敛速度。
- 交叉和变异的算法和发生交叉与变异的概率，交叉和变异算法、概率的不同影响了收敛速度。过于保守的算法会导致很难跳出局部最优和收敛慢。但是过于激进的算法也会导致对好解的破坏，导致收敛变慢。
- 自然选择过程的选择，优胜劣汰的方式也决定了种群优化的速度与趋势。

实验过程

本次实验过程中，利用 C++面向对象的特性，将 GA 过程封装成一个类，解封装成类 path。使用 C++进行 GA 的过程，将获得的解存储在外部文件中，再使用 pyhthon，将其可视化。

编码

TSP 问题的解是一个城市的路径，要求城市不能重复出现。所以我使用了城市大小的数组来表示一个路径的解，在遗传算法中相当于一个种群的一个个体。

这里用 C++类 Path 表示一条路径，包含私有变量 len 记录该个体的长度，用于后面的适应值评价函数, path[N]用来记录路径。

初始化种群

在上文中提到，初始解越多样，找到全局最优的几率越大。种群大小越大，越容易跳出局部最优，但是同时也导致运算更慢。同样时间内的迭代代数更少。
所以经过选择，本次实验中，将初始群体的大小，设置为 1000 个个体。

适应性函数

本次实验中，适应性函数是根据个体的长度来设计的，适应值为该个体路径长度的倒数。TSP 问题中解的路径越长，解的适应性越小，所以这里将适应值设置成解的长度的导数，当适应值越大，证明路径长度越小，所以该个体在群体更加优秀，能被选入新的群体并进行遗传操作。
适应性函数如下：

```
1. int GA::fit(Path p) {
2.  return 1 / p.getLength();
3. }
```

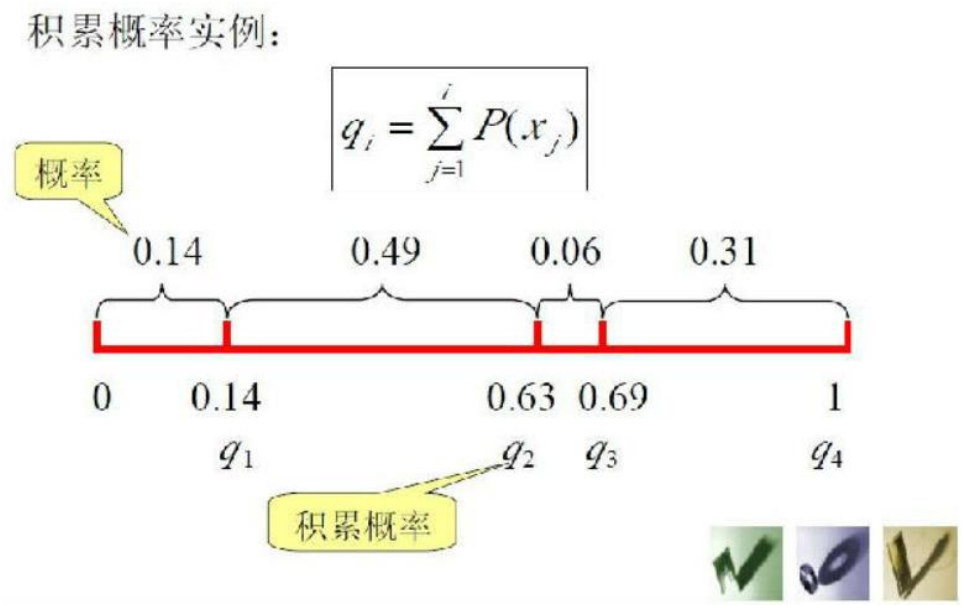
选择

这里的选择是指在经过了交叉、变异后的种群中选择一些优秀的个体。如何判断其优秀程度就是通过适应性函数来进行评价。
使用轮盘选择法来进行选择，这样能让低适应值的个体也有机会去被选择，只是概率比较低，更符合自然界实际情况。

轮盘选择法的具体操作如下：

1. 计算出群体中每个个体的适应度 f(i=1，2，…，M)，M 为群体大小；
2. 计算出每个个体被遗传到下一代群体中的概率；
3. 计算出每个个体的累积概率；
4. 在[0，1]区间内产生一个均匀分布的伪随机数 r；

5. 若 r<q[1]，则选择个体 1，否则，选择个体 k，使得：q[k-1]<r≤q[k] 成立；
6. 重复(4)、(5)共 M 次 如图：



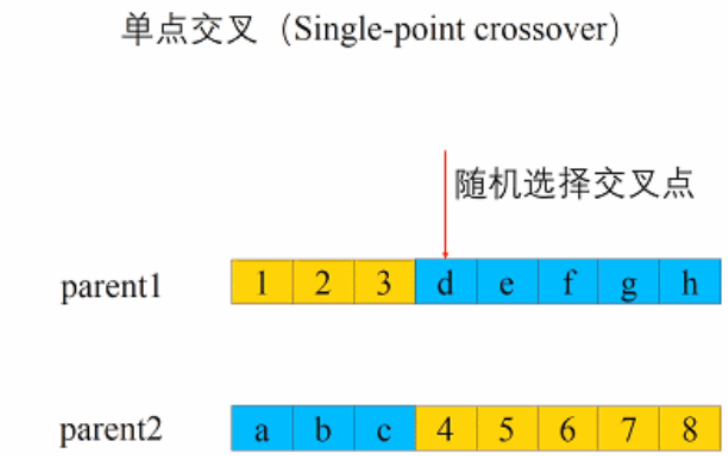
伪代码如下：

```
1. void GA::choose(vector<Path>& group) {
2.  sum <- 0
3.  for i <- 0 to group_size
4.   fitness[i] <- fit(group[i])//fitness 存储每个个体的适应值，
   fit 为适应函数
5.   sum <- sum + fitness[i]  //计算所有适应值之和
6.
7.  for i <- 0 to group_size
8.   chance[i] <- fitness[i] / sum//计算每个个体被选择的概率
9.
10.for i <- 1 to group_size
11.  chance[i] <- chance[i] + chance[i-1]//计算累计概率
12.
13. for i <- 0 to group_size
14.  pick <- 0 到 1 的随机数
15.  for j <- 0 to group_size
16.   //当 pick<=chance 时选择该个体，chance 越大越有机会
17.   if pick <= chance[i]
18.    选择保留该个体，退出循环
19.   //若到最后一个还未选出
20.   if j = group_size - 1
21.    选择保留该个体
22.}
```

交叉

交叉指的是选择种群中的两个个体来进行部分基因的交换，可以随机两个个体、随机个体的某一段基因。
本次实验中，生成一个随机数，当该随机数小于交叉概率设定时，就将随机的临近的两个个体来进行交叉。

这里采用的是单点交叉。
单点交叉通过选取两条染色体，在随机选择的位置点上进行分割并交换右侧的部分，从而得到两个不同的子染色体。
单点交叉是经典的交叉形式，与多点交叉或均匀交叉相比，它交叉混合的速度较慢（因为将染色体分成两段进行交叉，这种方式交叉粒度较大），然而对于选取交叉点位置具有一定内在含义的问题而言，单点交叉可以造成更小的破坏。
本次在多种交叉算子中选择单点交叉，主要是考虑到本次问题属于交叉点位置具有一定内在含义的问题，并且单点交叉实现简单。
图解如下：



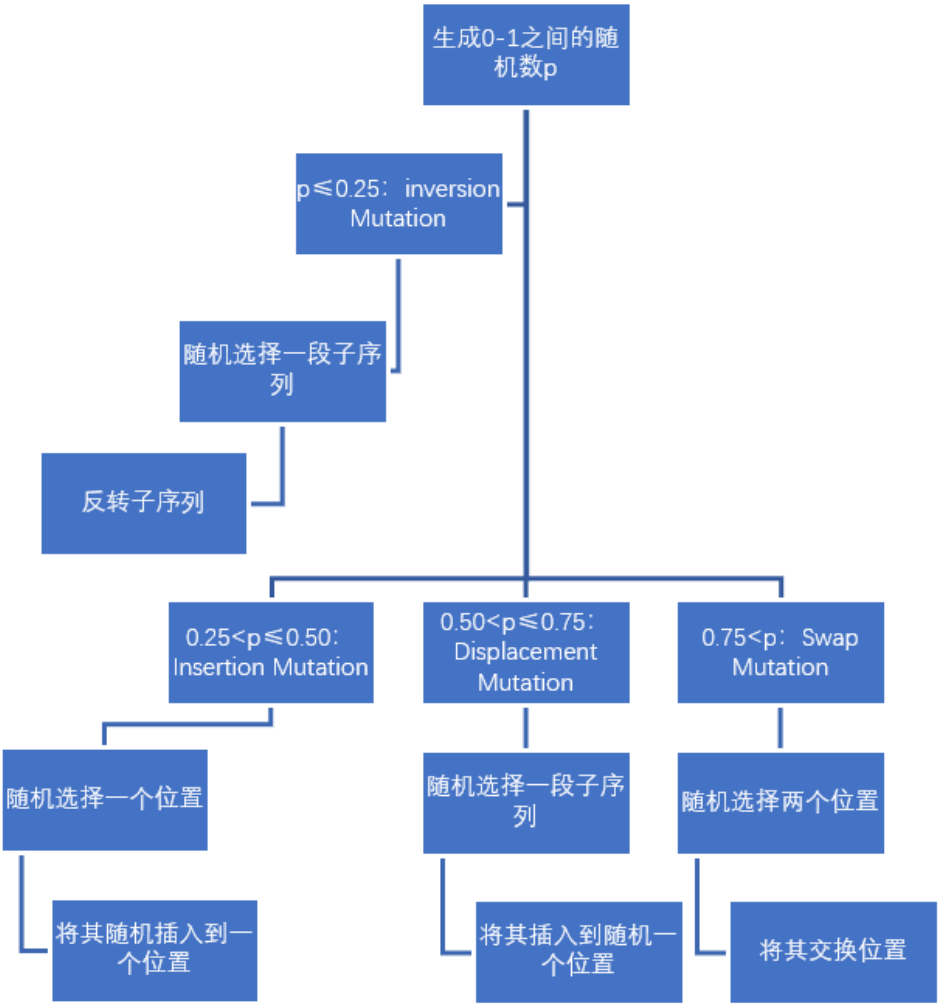
特别需要注意的是，要判断交叉后的结果是否合法，因为交叉时两个基因片段交换，可能会使得一个基因里面出现重复的编号。
判断解的合法性，需要对这个路径进行遍历，查找全部城市是否都存在有且只有一次。
将不合法的解变成合法解，逻辑是将上面的判断结果进行处理，如果第二次访问到这个城市，那么就等待第一个解与第二个解的重复城市，将它们进行交换。这样处理到解的最后一个城市，就能解决不合法解的情况。

变异

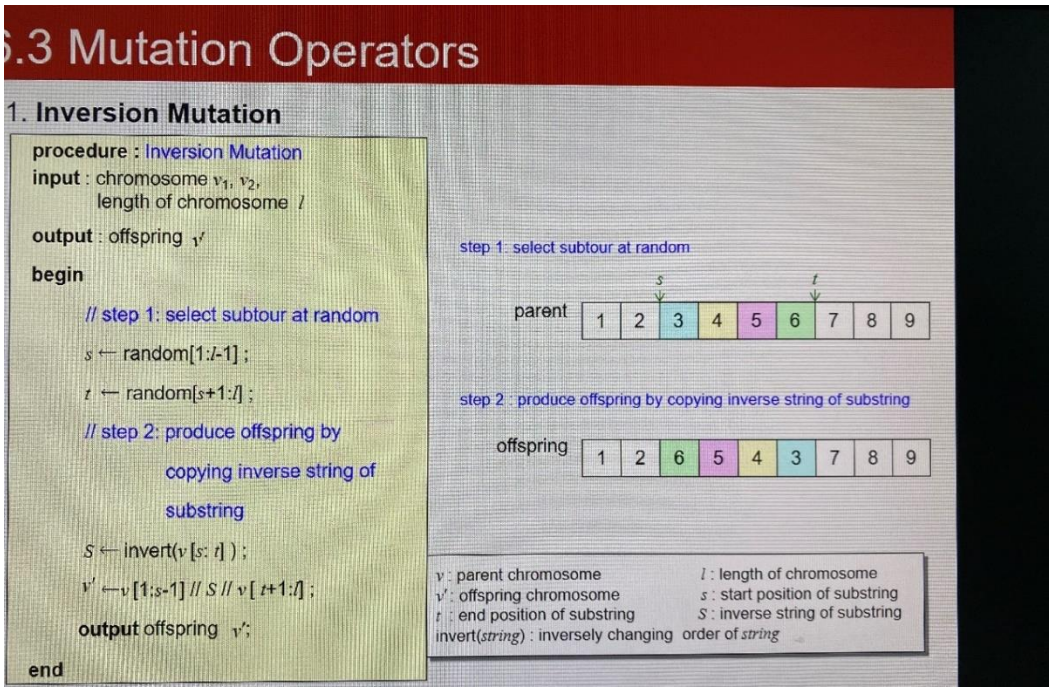
变异，又可以认为是邻域操作。由于实验要求中提到，要和之前的模拟退火算法（采用相同的局部搜索操作）进行比较，所以这里使用了之前模拟退火的操作。

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：

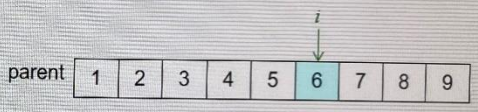


6.3 Mutation Operators

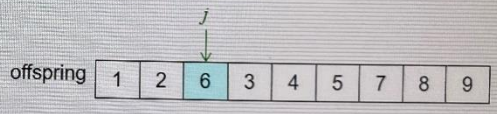
2. Insertion Mutation

```
procedure : Insertion Mutation
input : chromosome v1, v2,
        length of chromosome l
output : offspring v'
begin
    // step 1 : select a position in
                 parent 1 at random
    i ← random[1:l] ;
    // step 2: insert selected value in
                 randomly selected
                 position parent 2
    j ← random[1:l-1] ;
    W ← v [1:i-1] // v[i+1:l] ;
    v'← W[1:j-1] // v[i] // 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



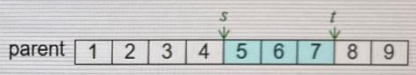
v : parent chromosome	l : length of chromosome
v' : offspring chromosome	i : selected position in parent 1
j : selected position in parent 2	W : working data set

6.3 Mutation Operators

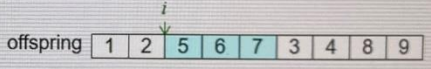
3. Displacement Mutation

```
procedure : Displacement Mutation
input : chromosome v1, v2,
        length of chromosome l
output : offspring v'
begin
    // step 1: select subtour
    s ← random[1:l-1] ;
    t ← random[s+1:l] ;
    // step 2: insert subtour in a
                 random position
    n ← t-(s-1);
    i ← random[1:l-n ] ;
    W ←v [1:s-1] // v [t+1:l];
    v'← W[1:i-1] // v [s : t ] // W[ i:l-n] ;
    output offspring v';
end
```

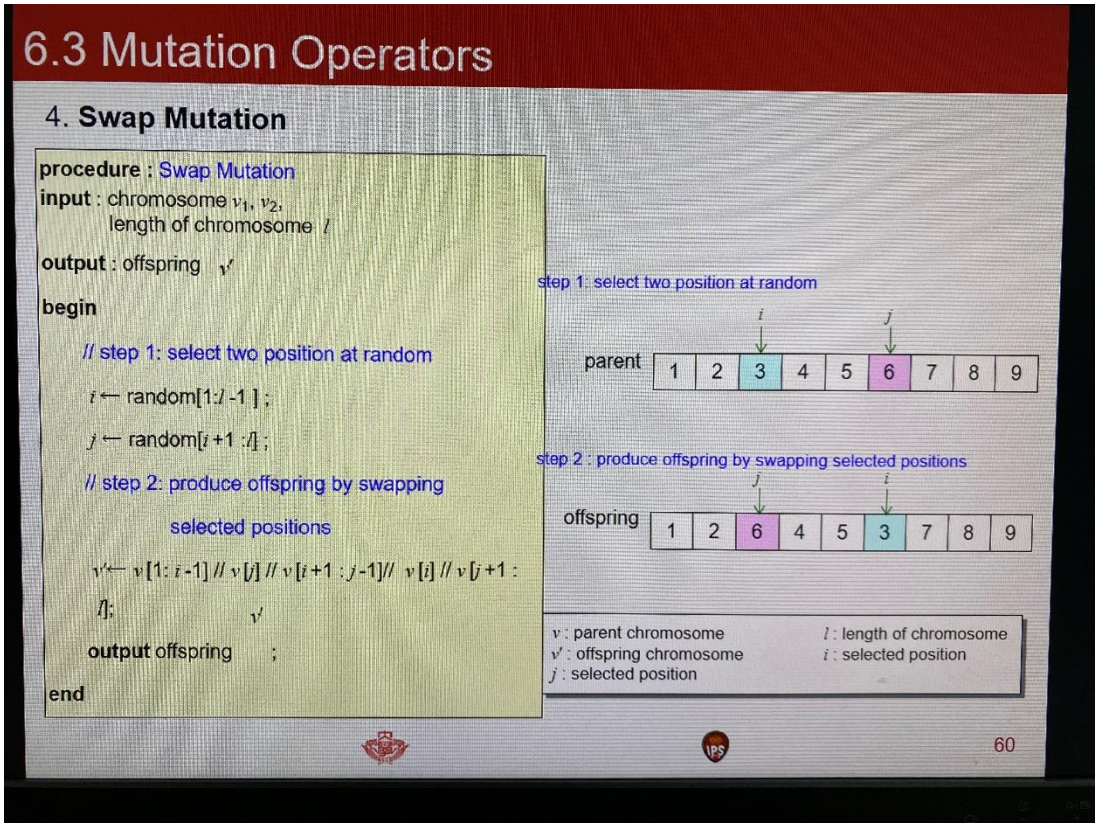
step 1 : select subtour



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	



遗传算法总过程

```
1. read_file()  //读取数据
2. 初始化种群
3.
4. while 次数<循环次数上限
5.     old_group <- group
6.     choose(group)  //选择
7.     cross(group)  //交叉
8.     for i<-0 to 5   //变异5次
9.         variation(group)
10.
11.获得种群中最优解 best
12.
```

代码框架

Path 类定义如下：

```
1. class Path
2. {
3. public:
```

```
    // 计算路径的长度
    void calculate_len();

    // 随机生成一个解
    Path() ;

// 用于随机生成解
void generate_random();

// 和别的path 交叉
void get_cross(Path& t)

// 变异
void get_variation();

// 获得 len
double get_len();

// 获得path
int* get_path();

private:
 double len;//长度
 int path[N];//路径
};
```

Path 类表示一个个体，为本次问题中的解（即路径和路径长度）。
成员变量自然为 len 和 path[N]，记录路径长度和路径。
成员函数包括了对个体进行操作的函数：交叉、变异。除此之外，还有包括基本的函数：随机生成个体、计算路径长度、获得路径、获得路径长度。
特别要注意的是，在每次更新了 path 后（如交叉变异后），要使用 calculate_len 函数更新 len 的值。

GA 类定义如下：

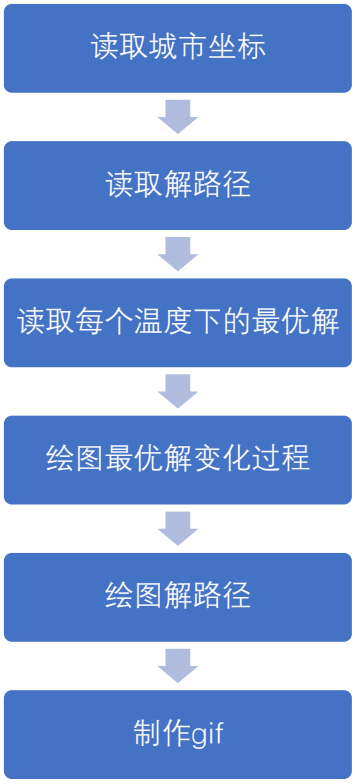
```
class GA {
public:
 // 记录城市坐标
 struct node {
  int num;
  double x;
  double y;
 }nodes[N];

// 种群，大小为group_size
vector<Path> group;
```

```
12. // 读取数据
13. void read_file();
14.
15. GA();
16.
17. ~GA();
18. // GA 总过程
19. Path get_answer();
20.
21.private:
22. // 选择
23. void choose(vector<Path>& group);
24. // 交叉
25. void cross(vector<Path>& group);
26. // 变异
27. void variation(vector<Path>& group);
28. // 决定子代是否能取代亲本，获取的优秀种群
29. void judge(vector<Path>& old_group, vector<Path>& group);
30. // 初始化种群
31. void init();
32.};
```

GA 类表示遗传算法的过程。
成员变量为 nodes[N]，记录城市坐标，和 group，记录种群（中的个体们）。
成员函数实现遗传算法中的每个步骤：初始化种群、选择、交叉、变异，除此之外，还包括了读取文件获得城市坐标数据以及计算城市间距离的函数，决定子代亲代谁更优的函数。所有的这些函数都服务于遗传算法的过程，在 get_answer 函数中调用。

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 100 次迭代的最优解，绘图：

令标题为“TSP genetic algorithm”，x 轴为“迭代次数”，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：

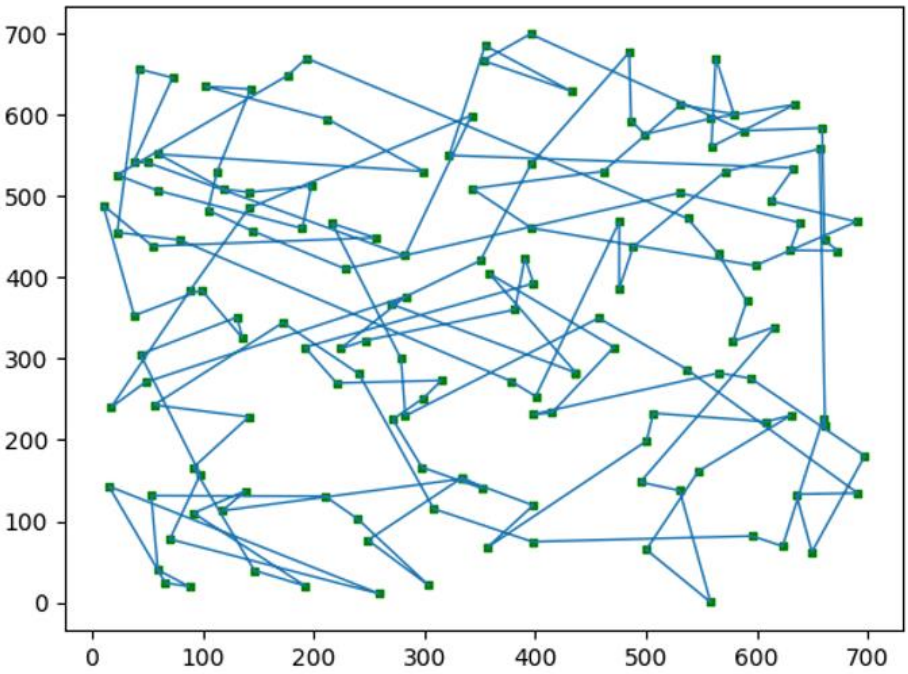
利用 matplotlib 绘图。

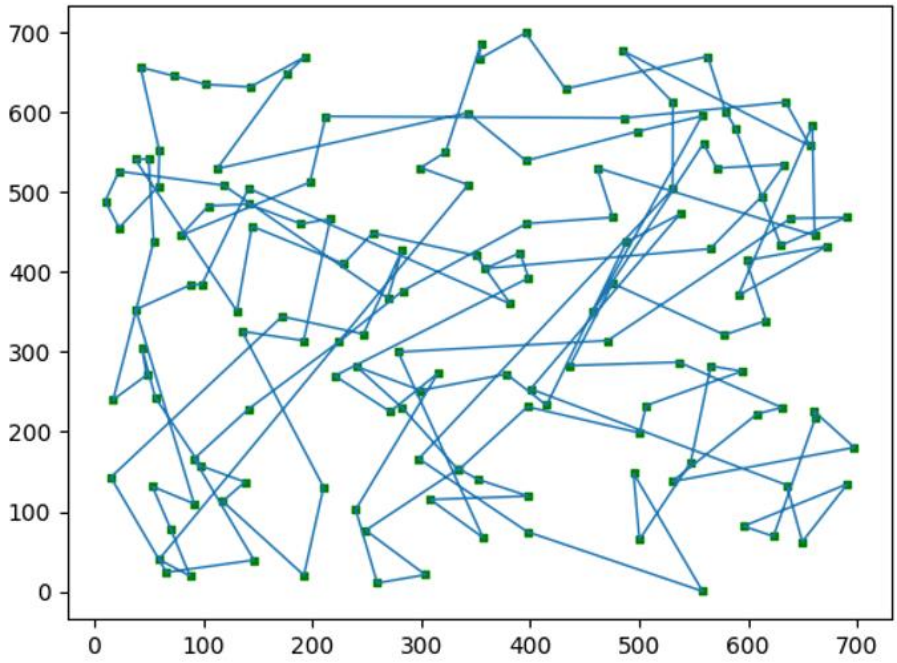
详见函数：

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

打印图片展示：

下为抽选的打印的图片中的两个，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

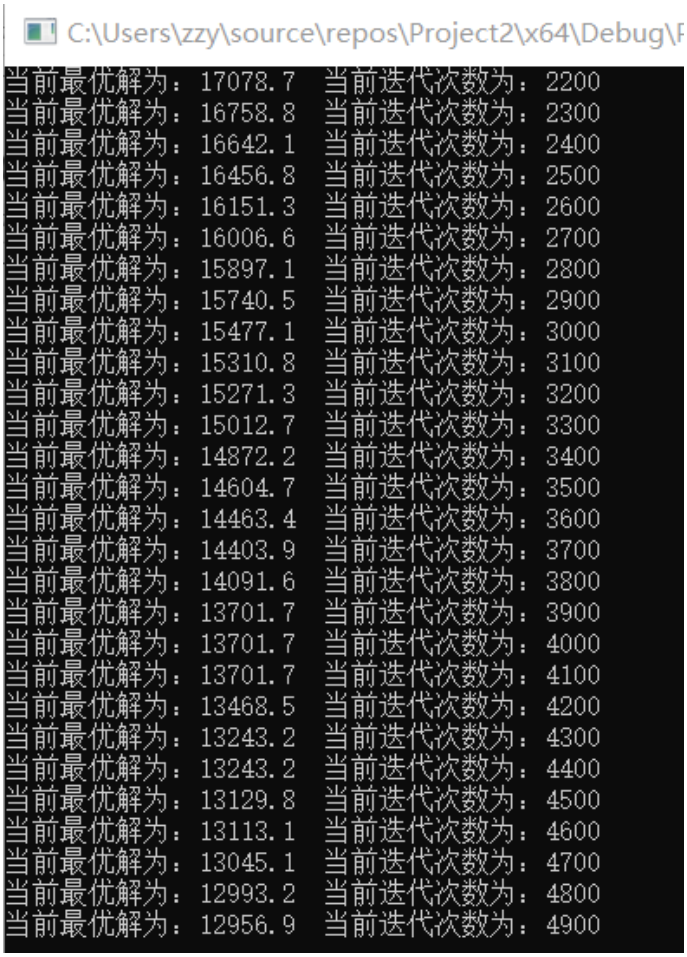
单独分析遗传算法

输出内容：

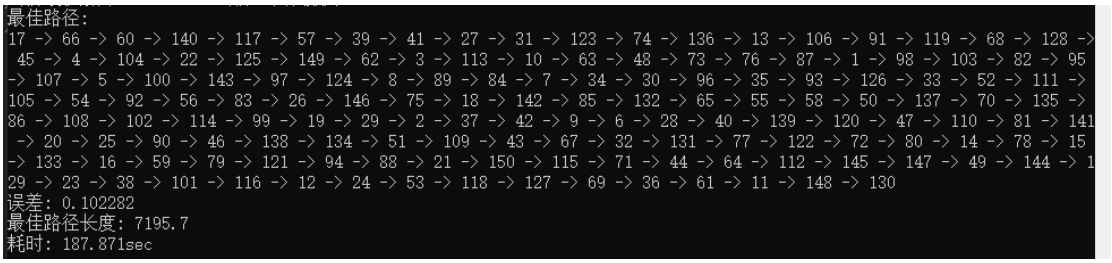
每 100 次迭代，输出当前最优解（路径长度）
运算结束后，输出最佳路径（用城市编号来指明路径）及长度
得到的最优解和实际最优解之间的误差
运算耗时

程序执行结果示例：

运行过程中，会不断输出当前最优解和当前迭代次数（每 100 次迭代输出一次）：



运行结束后，会输出最佳路径、最佳路径长度、误差、耗时：



多次运行结果：

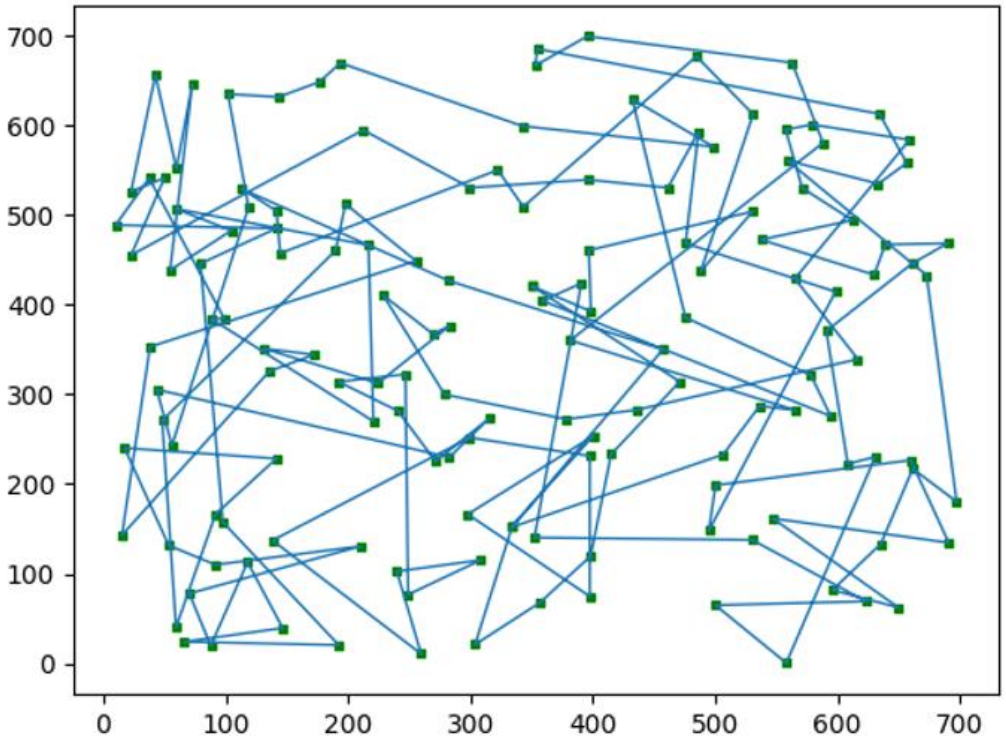
次数	最优解	误差	耗时
1	7195.7	10.23%	187.871
2	7213.14	10.49%	205.494
3	7142.32	9.41%	184.285
4	7151.55	9.55%	183.928
5	7168.02	9.95%	190.34
6	7213.45	10.64%	188.37
7	7029.98	7.69%	149.733
8	7288.03	11.64%	193.546
9	7072.97	8.34%	195.421
10	7225.92	10.69%	148.934

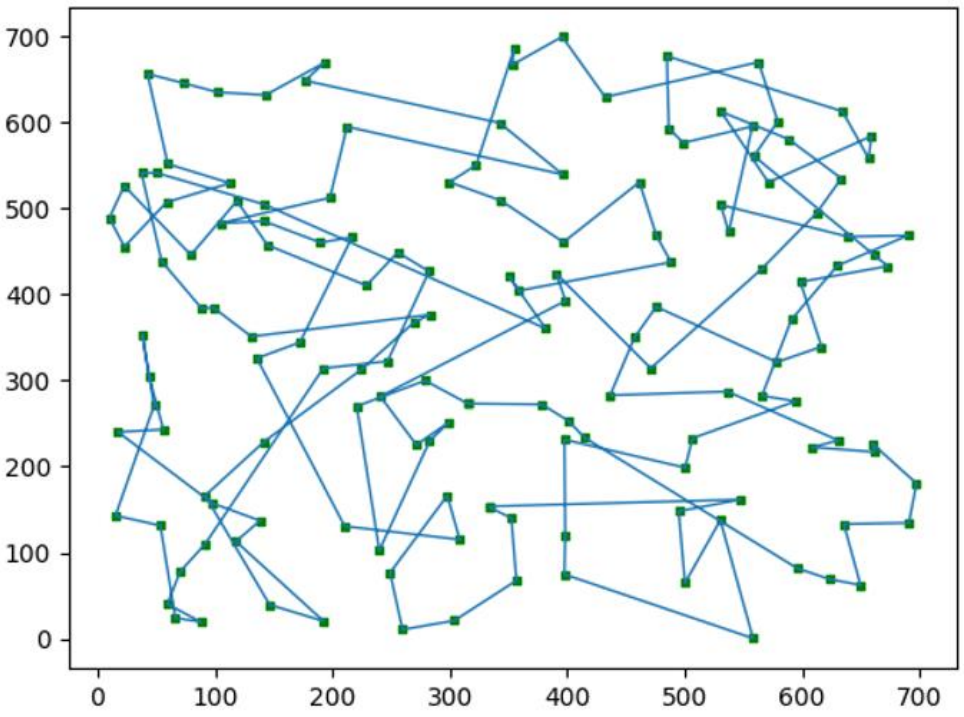
根据上述十次测试结果可以得出:

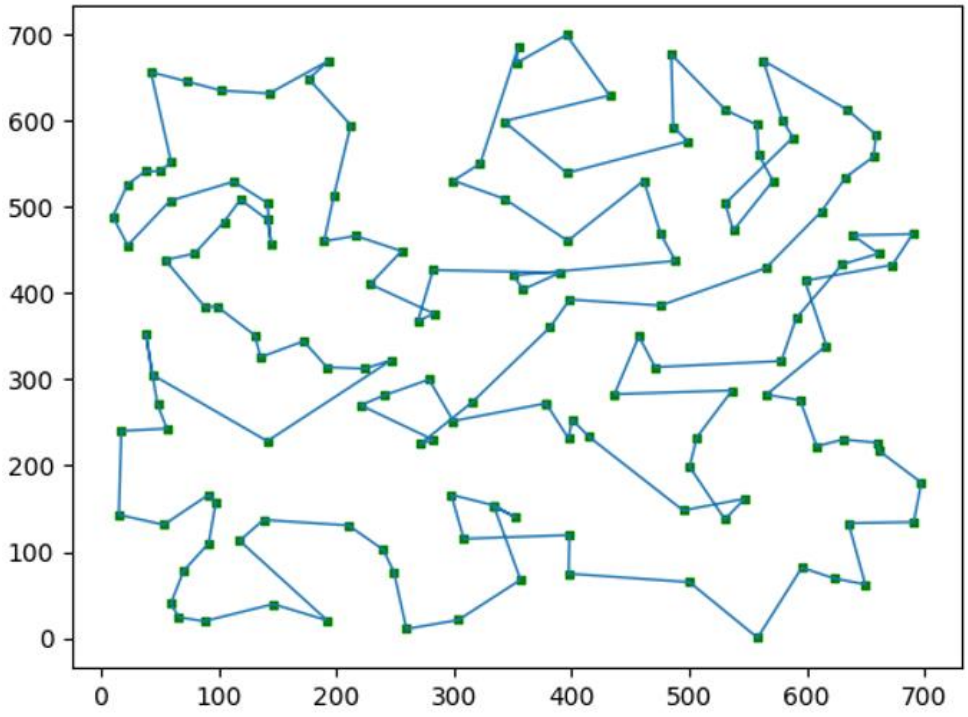
最好解	7029.98
最差解	7288.03
平均解	7171.66
平均误差	9.86%
平均耗时	182.81

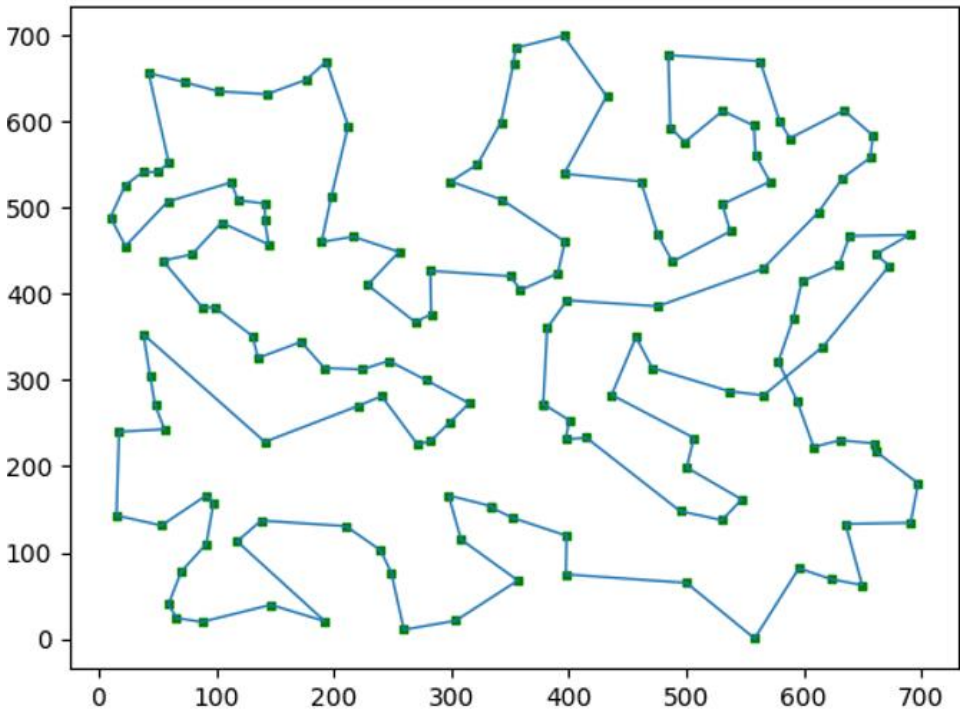
显示路径变化：

（以某次运行得到的结果为例）


700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700


700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700

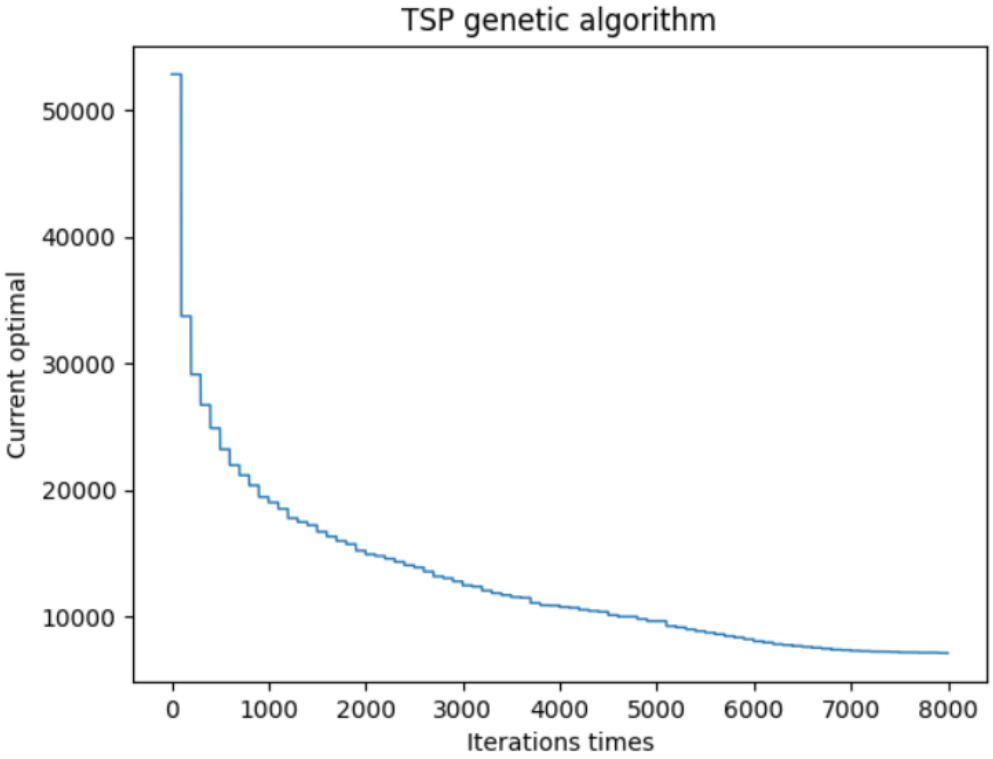




可以看到随着循环次数的增加，解的路径的交叉程度明显变小，最终获得的路径中已无交叉，找到了较好的路径解。

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。

显示过程中最优解（路径长度）的变化：



可以看到遗传算法的过程是在中间某一时刻，解发生剧烈变化，最佳路径的长度急剧下降，之后较为缓慢的下降。

和模拟退火算法（采用相同的局部搜索操作）进行比较

精确度比较：

分别进行十次实验，模拟退火的平均误差为 4%左右，而遗传算法的平均误差在 9%左右，这说明模拟退火比遗传算法能找到更加精确的解。

稳定性比较：

遗传算法的极差（最好解-最差解）为 200 多，模拟退火的极差也为 200 多，这说明模拟退火和遗传算法在稳定性上相当，都比较稳定。

用时比较：

模拟退火的平均耗时为 367.4，遗传算法的平均耗时为 182.81。这里由于两次算法设计的迭代次数不同，所以不太好直接进行比较。

模拟退火过程中，在设定的起始温度和终止温度下，大约要进行 1200 次的温度更新，每个温度下设定循环 10000 次。而遗传算法设定总迭代次数为 8000 次，可以看出来，遗传算法的迭代次数是远远小于模拟退火的，但是遗传算法的运行时间并没有小很多倍，仅仅是小了一半不到。所以其实，单次的遗传算法过程是比单次的模拟退火过程要更慢的，这里考虑主要有以下几个原因：

1、遗传算法的"变异"过程就等于模拟退火寻找邻解的过程，但是遗传算法比模拟退火

多出一个交叉的过程，交叉过程中的映射部分需要计算较长时间。

2、遗传算法保存的是一个“种群”，即它是一个多点搜素算法。而模拟退火是单点搜索算法。遗传算法的解集更加复杂且庞大，所以花费时间更多。

过程比较：

模拟退火的过程中，最优解的值是上下波动的；遗传算法的过程中，最优解要不保持不变，要不下降，所以呈现出来的图像为锯齿状。

设计高效遗传算法的经验，单点搜索和多点搜索的优缺点

高效遗传算法的经验：

1. 一个好的编码方式，有利于遗传操作，在交叉，选择，变异下更加简单
2. 增加初始解的多样性，初始解越多样，找到全局最优的几率越大。
3. 种群大小的选择要合理。种群大小越大，越容易跳出局部最优，但是同时也导致运算更慢。同样时间内的迭代代数更少。
4. 选择合适的适应度函数，应该用什么算法来计算解的适应度，影响了解之间的区分度的高低，从而进一步影响了收敛速度。
5. 选择合适的交叉和变异的算法，一般来说，交叉和变异算法的种类越多样，收敛速度会越快。但是需要注意，过于单调保守的算法会导致很难跳出局部最优和收敛慢。但是过于激进的算法也会导致对好解的破坏，导致收敛变慢。
6. 选择合适的发生交叉与变异的概率。概率和算法的影响接近，概率太低导致收敛慢和陷入局部最优。但是概率太低破坏好解也容易收敛慢
7. 自然选择过程的选择，优胜劣汰的方式也决定了种群优化的速度与趋势。淘汰适应率低的个体，不仅优化了种群组成，还控制了种群数量，保证计算速度。

单点搜索和多点搜索的对比：

单点搜索的优点是局部搜索能力强，效率更高，缺点是全局搜索能力差，容易受参数影响。以模拟退火为例子。模拟退火算法虽然以随机搜索技术从概率的意义上找出目标函数的全局最小点，具有一定的摆脱局部最优解的能力，但是由于模拟退火算法对整个搜索空间的状况了解不多，不便于使搜索过程进入最有希望的搜索区域，全局搜索能力仍不如多点搜索。

多点搜索(如遗传算法)的优点是更容易找到全局最优解，缺点是收敛速度慢，且需要保存一个多点解集，开销更大。以遗传算法为例子。遗传算法具有良好的全局搜索能力，可以快速地将解空间中的全体解搜索出，而不会陷入局部最优解的快速下降陷阱；并且利用它的内在并行性，可以方便地进行分布式计算，加快求解速度。但是遗传算法的局部搜索能力较差，导致单纯的遗传算法比较费时，在进化后期搜索效率较低。

结论

通过本次实验，对于 TSP 问题我有了更深入的了解，并且直至现在，我能够使用 3 种方式去求解它，对于每一种方式都有了较为清晰的认知。并且本次实验对比了遗传算法和模拟退火的区别，让我更深刻的体会到了单点搜索和多点搜索各自的优点缺点，在实际应用中应当更具自己的需求和设备的性能来选择合适的方法。同时，在不断更换适应值函数、选择更

优的交叉算法、增加变异算法、调整算法的交叉率、变异率，调整种群大小的过程中逐渐了解算法优化的过程和优化的关键。并且通过这些调整，使得解在一步步接近最优解，在这样的优化的过程中我得到了成就感。

主要参考文献

TSPLIB (uni-heidelberg.de)
遗传算法 - 中文维基百科【维基百科中文版网站】 (gaogevip.com)
谢胜利，唐敏，董金祥. 求解 TSP 问题的一种改进的遗传算法[J]. 计算机工程与应用，2002，38(008):58-60.