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	计科 2 班	专业（方向）	计算机科学与技术
学号	20337263	姓名	俞泽斌

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；
  - 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；
  - 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

模拟退火算法：

模拟退火算法首先是将整个问题进行编码，在 tsp 问题下的这串编码就是经过的城市的序列，然后设定一个初始温度值，然后在当前的温度下对这串编码进行变换操作。我用了一个随机概率来确定使用哪一种变换。

这里我使用的变换操作主要有两种，一种是直接取出编码的两个不同的随机

点，然后交换他们，第二种是取出三个编码内的随机点数，通过判断大小的方式对他们的前后顺序进行改变移动，使得顺序为从小到大的，然后通过列表的操作将其中前两个中间的那一串序列放到第三个随机数后面。

变换后的编码在进行综合路径的计算，与变换之前的路线的路径长度进行比较，如果变换之前的路径长度更长，那就说明新的路线是更好的解，就保留下来，更新当前路径和长度。如果变换之前的路径长度更短，为了能得到更多的优化的解的方案，就以 exp（-delta distance/T）的概率来接受他，如果接受的话也得到新的解。这样进行循环 trying_times 的次数，然后按比例降低温度，并输出当前的值

当温度小于预先设定的最小值的时候，终止算法

遗传算法

首先也是产生种群的操作了，最早的是个体的生成，在 tsp 问题中，个体就可以是路径的，也就是经过的城市的序号所产生的列表，然后随机生成种群数量个个体，开始进行循环，进行预定次数的循环

循环内一开始是选择操作，就是选择种群中较好的个体，既要选择好，又要尽量保存个体的多样性，这里我采用的是锦标赛的选择方式，也就是随机选择两个个体，然后比较他们的路径长度，然后选择路径长度更小的那一个放入交配池中

然后就是交配操作，这里我使用了两种不同的交叉操作，一种是最普通的，也就是在 parents 这个列表中随机取出两个个体，然后取一个随机值，将一个个体的随机值之前的所有路径保留，然后将按照第二个个体中的相对顺序填充好后面的路径，第二个子代类似于上面的算法，不过取的是第二个个体的上半部分。

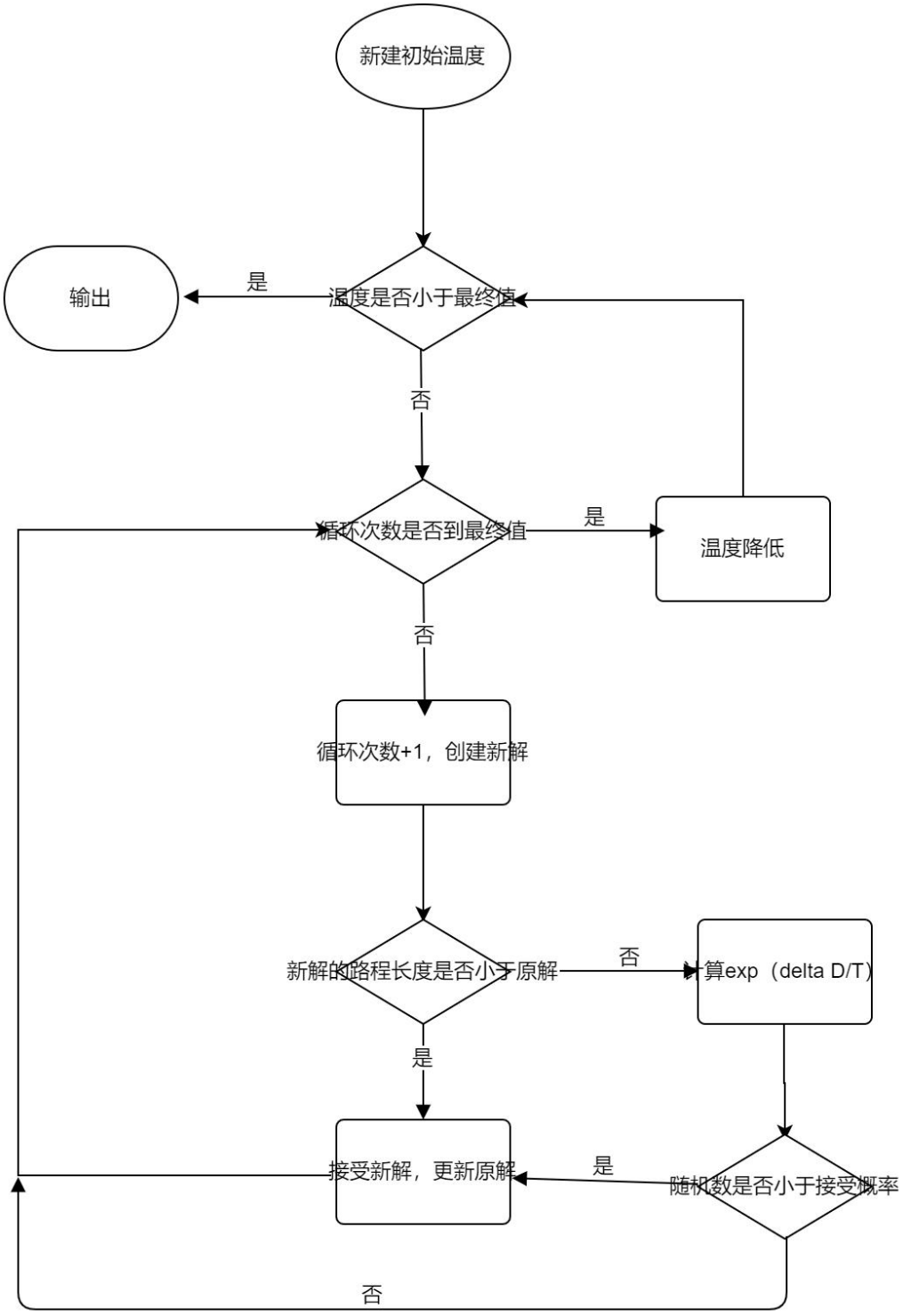
还有一种是网上找来的 cpx 的交叉，也就是 cx 和 px 一起的交叉操作：px：从 parent1 中随机选择 N 个点，在 child1 中的相同位置复制这 N 个点，。将未在 child1 中的点按照在 parent2 中的相对顺序填充到 child1 中的空位中。Child2 类似于上面的算法，不过取的是第二个个体的 N 个点。

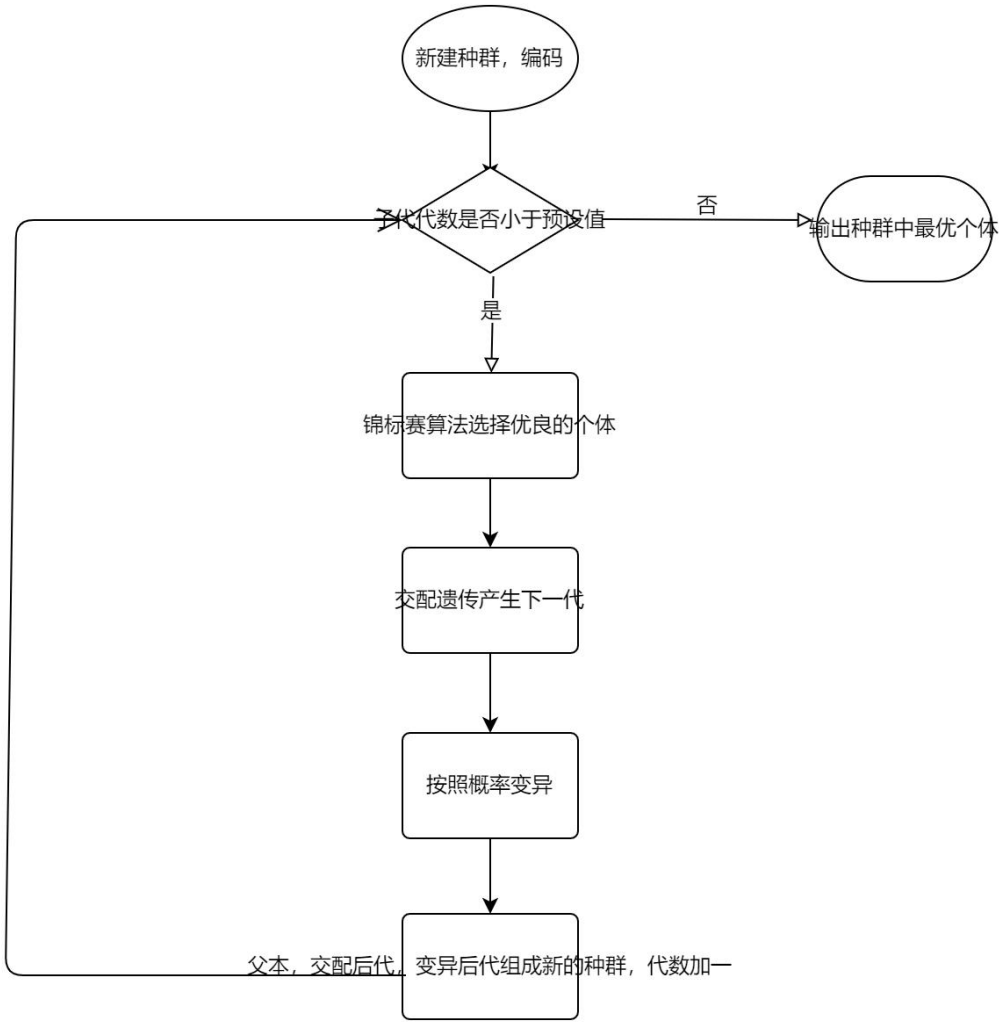
Cx 的交叉操作是：P1 中的第一个点为起点 start，P2 中的第一个点为 end，将 start 添加到子代中。找寻 P1 中 end 的位置 p,令 P2 中 p 对应的点为 end，将 end 添加到子代，重复，直到 end=start

接着是关于变异的操作，变异我这里用了三种变异的操作，然后分别用了 pm1，pm2，pm3 来表示发生对应变异的概率，第一种变异是路径中的两个点交换一下，第二种是路径中的两个随机点中间的路径翻转，第三种是路径中两个随机点中间的路径放到第三随机点之后

总体的操作就是在整个种群中选择亲本，然后让亲本按一定概率选择不同的交叉操作来产生子代，同时按不同的概率来选择不同的变异方式，都加入到子代这个种群中，然后通过锦标赛的选择方式从子代种群中选择相应数量的子代与留下来的亲本一起作为下一个种群进入下一次循环，一直到循环次数结束，输出最终结果。

2. 伪代码


新建初始温度
温度是否小于最终值
是
输出
否
循环次数是否到最终值
是
温度降低
否
循环次数+1，创建新解
新解的路程长度是否小于原解
否
计算exp（delta D/T）
是
接受新解，更新原解
是
随机数是否小于接受概率
否


新建种群，编码
子代代数是否小于预设值
否
输出种群中最优个体
是
锦标赛算法选择优良的个体
交配遗传产生下一代
按照概率变异
父本，交配后代，变异后代组成新的种群，代数加一

3. 关键代码展示（带注释）

```python
# 载入数据
df = pd.read_csv('kroA100.tsp', skiprows=5, delimiter="\t")
# city = np.array(df[0][0:len(df)-2])  # 最后一行为EOF，不读入
# city_name = df.values.tolist()
# city=city_name[0]
# city_name=df.values.split()
# city_name=city_name[1][0]
# print(city_name)
location = []
df = df.values
df = df[0:-1]
for i in range(len(df)):
    tcity = df[i][0].split()
    city_x = int(tcity[1])
    city_y = int(tcity[2])
    city_location = []
    city_location.append(city_x)
    city_location.append(city_y)
    location.append(city_location)
    # print(city_location)
pc=0.5
Tend=1
Tstart=5000
```

首先是数据的载入，我这里使用的是调用了 pandas 的一个库函数来对 tsp 文件进行读取，其实也可以将 tsp 文件的前面几行删去改成 data.txt 进行读取，用 pandas 进行读取数据的时候要注意的是读入的是一个 numpy 文件，也就是..arry[]..之类的数据，然后要通过 values 以及 split 来进行转换，最终将所有的城市坐标录入 location 这个列表中

```python
#两个城市的距离
def dist(a, b):
    distance = ((location[a][0] - location[b][0]) ** 2 + (location[a][1] - location[b][1]) ** 2) ** 0.5
    return distance
#路程总长
def sumval(a):
    sum = 0
    for i in range(len(a)-1):
        sum += dist(a[i], a[i+1])
    sum+=dist(a[len(a)-1],a[0])
    return sum
```

这是比较基础的函数，主要是求两个城市的距离和当前路肩的总长度，上面几个函数都是可以在遗传算法和模拟退火算法中出现的，下面的几个函数就要做出区

分了

模拟退火算法：

```
def creat_new(former):
    newway = former.copy()
    a = random.randint(0,len(location)-1)
    b = random.randint(0,len(location)-1)
    newway[a], newway[b] = newway[b], newway[a]
    return newway
```

新解的产生函数，在老的路线中随机选择两个城市进行交换操作

```
def create_new2(former):
    loc1,loc2,loc3=np.random.randint(0, len(location)-1, 3)
    # 下面的三个判断语句使得loc1<loc2<loc3
    if loc1 > loc2:
        loc1, loc2 = loc2, loc1
    if loc2 > loc3:
        loc2, loc3 = loc3, loc2
    if loc1 > loc2:
        loc1, loc2 = loc2, loc1
    # 将[loc1,loc2)区间的数据插入到loc3之后
    newways=former.copy()
    changepart = newways[loc1:loc2].copy()
    newways[loc1:loc3 - loc2 + 1 + loc1] = newways[loc2:loc3 + 1].copy()
    newways[loc3 - loc2 + 1 + loc1:loc3 + 1] = changepart.copy()
    return newways
```

第二个新解产生函数，主要是因为如果只采用第一个新解产生函数的话，因为变异的量不够，所以要运行较多的次数才能逐渐收敛到某一个最佳的解，第二个函数主要是将 loc1 和 loc2 中间的数据全部插入到 loc3 后面

```python
def printfig(ans,T):
    #plt.cla()表示清除当前轴
    #plt.clf()表示清除当前数字
    #plt.ion()用于打开交互模式
    #plt.ioff()用于关闭交互模式
    plt.cla()
    plt.clf()
    X = []
    Y = []
    for i in range(len(ans)):
        x = location[ans[i]][0]
        y = location[ans[i]][1]
        X.append(x)
        Y.append(y)
    plt.scatter(x, y)
    plt.plot(X, Y, '-o')
    plt.title("temp answer:")
    #plt.savefig('./img/pic-{}.png'.format(printnum + 1))
    plt.show()
    plt.pause(0.5)
    if T<=Tend:
        plt.ioff()
        plt.show()
```

实现图像化界面的函数，主要是调用了 plt 里面的库函数的操作，一开始为了实现动图的效果我是想先把 plt 中的图片都保存下来，然后通过另一个库来实现 gif 的动图形式，后来发现 plt 提供了打开和关闭交互模式的按钮，打开之后就不需要自己手动关闭才会跳出来下一个图片，然后如果不 pause 一下的话就容易加载不出当前的图片，所以为了让 pause 不特别影响运行速度，我是在每降温 5 次的时候才输出一张路线图，输出路线图的方法也就是将 x，y 导入，然后将每个点中间用线连接

```python
def main():
    tempway=[]
    for i in range(len(location)):
        tempway.append(i)
    T = Tstart
    cnt = 0
    trend = []
    while T > Tend:
        for i in range(1000):
            if np.random.random()<pc:
                newway = creat_new(tempway)
            else:
                newway =create_new2(tempway)
            old_dist = sumval(tempway)
            new_dist = sumval(newway)
            p=0
            if new_dist - old_dist < 0:
                p=1
            else:
                p=exp(-(new_dist - old_dist) / T)
            if np.random.random() < p:
                tempway = newway
        T = T * 0.98
        cnt += 1
        bestlen=sumval(tempway)
        trend.append(bestlen)
        print(cnt,"次降温，温度为：",T," 路程长度为：", bestlen)
        plt.ion()
        if T<=Tend or cnt%5==0:
            printfig(tempway, T)
```

```python
lenfinal=sumval(tempway)
print("最终路程长度为",lenfinal)
print("最终路线为",tempway)
plt.plot(trend)
plt.show()
```

主函数，主要就是模拟退火算法的思想了，设定一个初始温度值，然后在当前的温度下对这串编码进行变换操作。我用了一个随机概率来确定使用哪一种变换，就是 pc，当随机值小于 pc 的时候就采用第一种变换方法，如果不是就采用第二种变换。

变换后再进行综合路径的计算，与变换之前的路线的路径长度进行比较，如

果变换之前的路径长度更长，那就说明新的路线是更好的解，就保留下来，更新当前路径和长度。如果变换之前的路径长度更短，为了能得到更多的优化的解的方案，就以 exp（-delta distance/T）的概率来接受他，如果接受的话也得到新的解。这样进行循环 trying_times 的次数，然后按比例降低温度，每降低 5 次温度来输出一次当前路线图，每次降低温度的时候都在终端输出当前路线长度和温度来监控程序运行 当温度小于预先设定的最小值的时候，终止算法。

接下来是遗传算法：
数据导入，打印图片两个函数都在上面做了具体分析，以下就不做赘述。

```
def init_group(group_size):
    races = []
    route = []
    for i in range(len(location)):
        route.append(i)
    for i in range(group_size):
        newroute = route.copy()
        random.shuffle(newroute)
        races.append(newroute)
    return races
```

生成一个初始种群的操作，对于一个 tsp 问题，他的个体就是一条路径，所以种群就是 group_size 条路径，这里我采用的是 random 里的 shuffle 函数来随机产生路径，要注意的是，如果没有将原路径进行 copy 后再调用 shuffle 函数，最后生成的将会是一样的一个种群，虽然每次生成后 print 出来都不一样，然后最后 print 整个 races 的时候都是一样的，所以每一次的随机变化都要进行拷贝。

```
def sort2(arr1, arr2):
    arr1 = np.array(arr1)
    arr2 = np.array(arr2)
    arrIndex = np.array(arr2).argsort()
    arr1 = arr1[arrIndex]
    arr2 = arr2[arrIndex]
    arr1.tolist()
    arr2.tolist()
```

Sort2 函数主要是用 arr2 里的数字的大小排序是将对应的 arr1 中的次序也进行改变，主要应用就是对每一个个体的路径长度进行排序的时候，将个体也按照相应的顺序进行排序，这里采用了 argsort 函数，来方便运算

```python
def tournament(old_group, size):
    parents = []
    for i in range(size):
        s1, s2 = np.random.randint(0, len(old_group), 2)
        if sumval(old_group[s1]) > sumval(old_group[s2]):
            parents.append(old_group[s2])
        else:
            parents.append(old_group[s1])
    return parents
```

锦标赛选择法来选择优秀个体，就是随机取出两个个体，比较他们的路径长度，将路径长度小的那个放入 parents 列表中，作为交配池

```python
def cross(parent1, parent2):
    a = random.randint(1, len(location) - 1)
    child1, child2 = parent1[:a], parent2[:a]

    for i in range(len(location)):
        if parent2[i] not in child1:
            child1.append(parent2[i])

        if parent1[i] not in child2:
            child2.append(parent1[i])

    return child1, child2
```

交叉操作 1，主要就是很简单原理了，将其中一个个体的前一部分（长度是随机产生的）放到 child2 中，后一部分放到 child1 中，然后将其他的按照另一个亲本的顺序没有重复地放入路径中，但是这个交叉好像交叉部分太多，以致于单纯使用这一个交叉产生好的解需要很久，所以按网上教程引入交叉操作 2

```python
def CPX(parent1, parent2):
    cycle = []  # 交叉点集
    start = parent1[0]
    cycle.append(start)
    end = parent2[0]
    while end != start:
        cycle.append(end)
        end = parent2[parent1.index(end)]
    child = parent1[:]
    cross_points = cycle[:]
    if len(cross_points) < 2:
        cross_points = random.sample(parent1, 2)
    k = 0
    for i in range(len(parent1)):
        if child[i] in cross_points:
            continue
        else:
            for j in range(k, len(parent2)):
                if parent2[j] in cross_points:
                    continue
                else:
                    child[i] = parent2[j]
                    k = j + 1
                    break
    return child
```

交叉操作 2：这个交叉的就比较复杂了，采用的是 cx 和 px 一起的交叉操作：px：从 parent1 中随机选择 N 个点，在 child1 中的相同位置复制这 N 个点，。将未在 child1 中的点按照在 parent2 中的相对顺序填充到 child1 中的空位中。Child2 类似于上面的算法，不过取的是第二个个体的 N 个点。

Cx 的交叉操作是：P1 中的第一个点为起点 start，P2 中的第一个点为 end，将 start 添加到子代中。找寻 P1 中 end 的位置 p,令 P2 中 p 对应的点为 end，将 end 添加到子代，重复，直到 end=start

```python
# 变异
def variation(path):
    i1, i2, = random.sample(range(0, len(location) - 1), 2)
    path[i1], path[i2] = path[i2], path[i1]
    return path
```

为了增加多样性，我的变异操作用了三种不同的变异

变异操作 1：首先是最简单的从路径中随机取出两个城市来交换他们的位置

```python
def inversion_mutation(path):
    a, b = random.sample(range(0, len(location) - 1), 2)
    for i in range(a, (a + b) // 2 + 1):
        path[i], path[b + a - i] = path[b + a - i], path[i]

    return path
```

变异操作 2：取出路径中的一段随机路径，然后将它翻转

```python
def slide_mutation(path):
    a, b = random.sample(range(0, len(location) - 1), 2)
    if a > b:
        a, b = b, a
    t = path[a]
    for i in range(a + 1, b + 1):
        path[i - 1] = path[i]
    path[b] = t
    return path
```

变异操作 3，取出路径中的一段随机路径，然后将它向前滑动一个单位，将前面挤出来的放到最后面。

```
def nextgene(group, parent):
    new_races = []
    while len(new_races) < group_size:
        i1, i2, i3 = np.random.randint(0, len(parent), 3)
        p1 = parent[i1].copy()
        p2 = parent[i2].copy()
        p3 = parent[i3].copy()
        # 进行交叉
        if np.random.random() < pc:
            p4 = CPX(p1, p2)
            new_races.append(p4)
        else:
            p5, p6 = cross(p1, p2)
            new_races.append(p5)
            new_races.append(p6)
        # 概率小于pm就进行变异
        if np.random.random() < pm1:
            # 变异
            p1 = variation(p1)
        if np.random.random() < pm2:
            p2 = inversion_mutation(p2)
        if np.random.random() < pm3:
            p3 = slide_mutation(p3)
        new_races.extend([p1, p2, p3])
    # sort2(new_races,groupfunc(new_races))
```

```
#final_race = []
#for i in range(len(group) - parent_size):
    #final_race.append(new_races[i])
final_race = tournament(new_races, group_size - parent_size)
for i in range(parent_size):
    final_race.append(parent[i])
return final_race
```

接下来是最核心的一部分内容了，为了让子代更好一点，我将 new_races 的大小设定为 groupsize 而不是 group_size-parents_size，让里面能够进行排序来选择更好的,让 parents 的亲本在 pc 的概率下进行第一种交叉，否则进行第二种交叉，然后将所得到的全部放入 new_races 中，然后将随机的亲本进行不同中变异，这里我设定了三种变异的概率，一定会发生一种变异，在后期也尽可能保留多样性，产生的子代也放入 new_races 当中，最后对 new_races 按照锦标赛算法得出新一代一部分，然后将亲本保留前一部分，因为锦标赛算法的随机性，这一部分也是随机的。

```python
def main():
    route = init_group(group_size)
    tempdis = []
    printnum=0
    for i in range(trying_times):
        parent = tournament(route, parent_size+100)
        route = nextgene(route, parent)
        # result = min([sumval(j) for j in route])
        result = 9999999
        tpath = []
        for j in range(len(route)):
            if sumval(route[j]) < result:
                result = sumval(route[j])
                tpath = route[j]
        tempdis.append(result)
        if (i+1) % 10 == 0:
            print('第', i+1, '次遗传，现在的距离为', result)
            print('现在的路线为', tpath)
            plt.ion()
            printfig(tpath, i+1)
    print('最终路线为：',route[-1])
    print('最终距离为',tempdis[-1])
```

```python
# 绘制进化次数-距离图
plt.plot(tempdis)
plt.title('Distance change with evolution times')
plt.xlabel('evolution times')
plt.ylabel('distance')

plt.show()
```

主函数了，其实就是将上面所有的子函数全都串联起来，先是产生种群，然后不断地选择 parents，交配变异产生后代，然后输出种群中最好的那一个的长度和路径，然后每 10 次遗传输出一次路径距离和路线图，最后输出一个整个路径长度随遗传次数变化而变化所产生的曲线。

4. 创新点&优化（如果有）

```python
# 载入数据
df = pd.read_csv('kroA100.tsp', skiprows=5, delimiter="\t")
# city = np.array(df[0][0:len(df)-2])  # 最后一行为EOF，不读入
# city_name = df.values.tolist()
# city=city_name[0]
# city_name=df.values.split()
# city_name=city_name[1][0]
# print(city_name)
location = []
df = df.values
df = df[0:-1]
for i in range(len(df)):
    tcity = df[i][0].split()
    city_x = int(tcity[1])
    city_y = int(tcity[2])
    city_location = []
    city_location.append(city_x)
    city_location.append(city_y)
    location.append(city_location)
    # print(city_location)
pc=0.5
Tend=1
Tstart=5000
```

调用了 pandas 的一个库函数来对 tsp 文件进行读取，其实也可以将 tsp 文件的前面几行删去改成 data.txt 进行读取，用 pandas 进行读取数据的时候要注意的是读入的是一个 numpy 文件，也就是..arry[]..之类的数据，然后要通过 values 以及 split 来进行转换，最终将所有的城市坐标录入 location 这个列表中

```python
def printfig(ans, T):
    #plt.cla()表示清除当前轴
    #plt.clf()表示清除当前数字
    #plt.ion()用于打开交互模式
    #plt.ioff()用于关闭交互模式
    plt.cla()
    plt.clf()
    X = []
    Y = []
    for i in range(len(ans)):
        x = location[ans[i]][0]
        y = location[ans[i]][1]
        X.append(x)
        Y.append(y)
    plt.scatter(x, y)
    plt.plot(X, Y, '-o')
    plt.title("temp answer:")
    #plt.savefig('./img/pic-{}.png'.format(printnum + 1))
    plt.show()
    plt.pause(0.5)
    if T<=Tend:
        plt.ioff()
        plt.show()
```

调用了 plt 里面的库函数的操作，使用了 plt 提供的打开和关闭交互模式的按钮，打开之后就不需要自己手动关闭才会跳出来下一个图片，然后如果不 pause 一下的话就容易加载不出当前的图片，所以为了让 pause 不特别影响运行速度，我是在每降温 5 次的时候才输出一张路线图。

```python
def create_new2(former):
    loc1,loc2,loc3=np.random.randint(0, len(location)-1, 3)
    # 下面的三个判断语句使得loc1<loc2<loc3
    if loc1 > loc2:
        loc1, loc2 = loc2, loc1
    if loc2 > loc3:
        loc2, loc3 = loc3, loc2
    if loc1 > loc2:
        loc1, loc2 = loc2, loc1
    # 将[loc1,loc2)区间的数据插入到loc3之后
    newways=former.copy()
    changepart = newways[loc1:loc2].copy()
    newways[loc1:loc3 - loc2 + 1 + loc1] = newways[loc2:loc3 + 1].copy()
    newways[loc3 - loc2 + 1 + loc1:loc3 + 1] = changepart.copy()
    return newways
```

采用了两个新解产生函数，加快新解产生的多样性和到最终状态的变化，将 loc1 和 loc2 中间的数据全部插入到 loc3 后面

```python
def sort2(arr1, arr2):
    arr1 = np.array(arr1)
    arr2 = np.array(arr2)
    arrIndex = np.array(arr2).argsort()
    arr1 = arr1[arrIndex]
    arr2 = arr2[arrIndex]
    arr1.tolist()
    arr2.tolist()
```

在对后面一个需要排序并改动前一个的相对顺序的函数中采用了使用矩阵的方式来排序两个列表的操作，运用了 numpy 库的 argsort 函数

```python
# 变异
def variation(path):
    i1, i2, = random.sample(range(0, len(location) - 1), 2)
    path[i1], path[i2] = path[i2], path[i1]
    return path
```

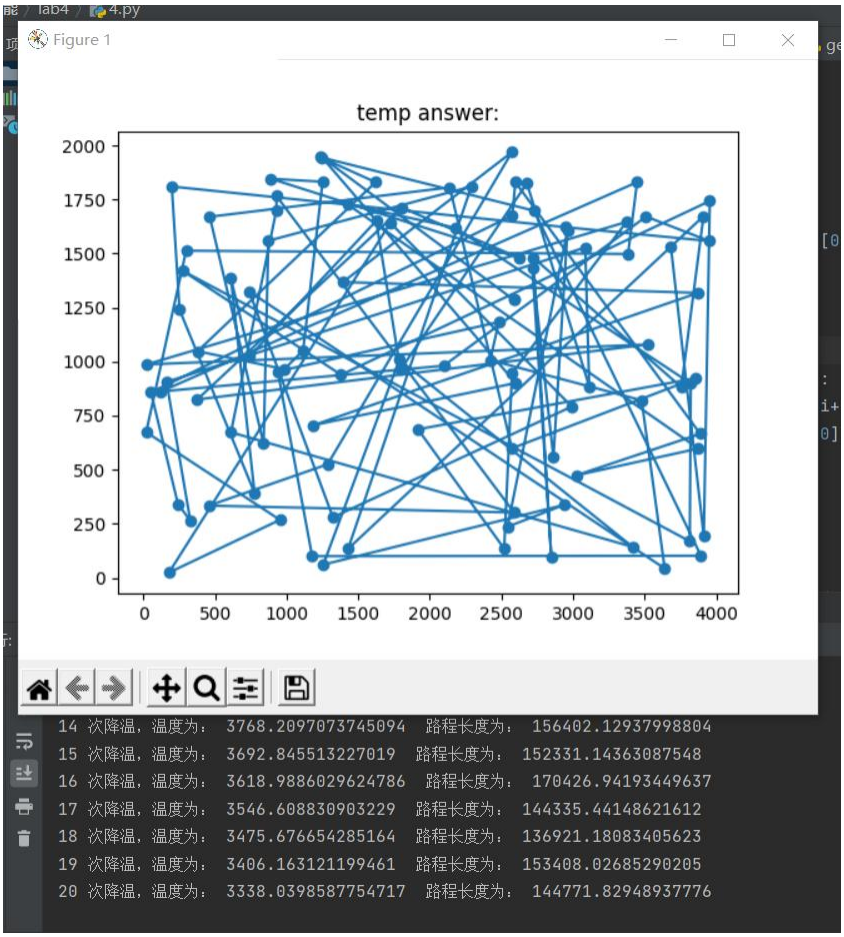
```python
def inversion_mutation(path):
    a, b = random.sample(range(0, len(location) - 1), 2)
    for i in range(a, (a + b) // 2 + 1):
        path[i], path[b + a - i] = path[b + a - i], path[i]

    return path
```

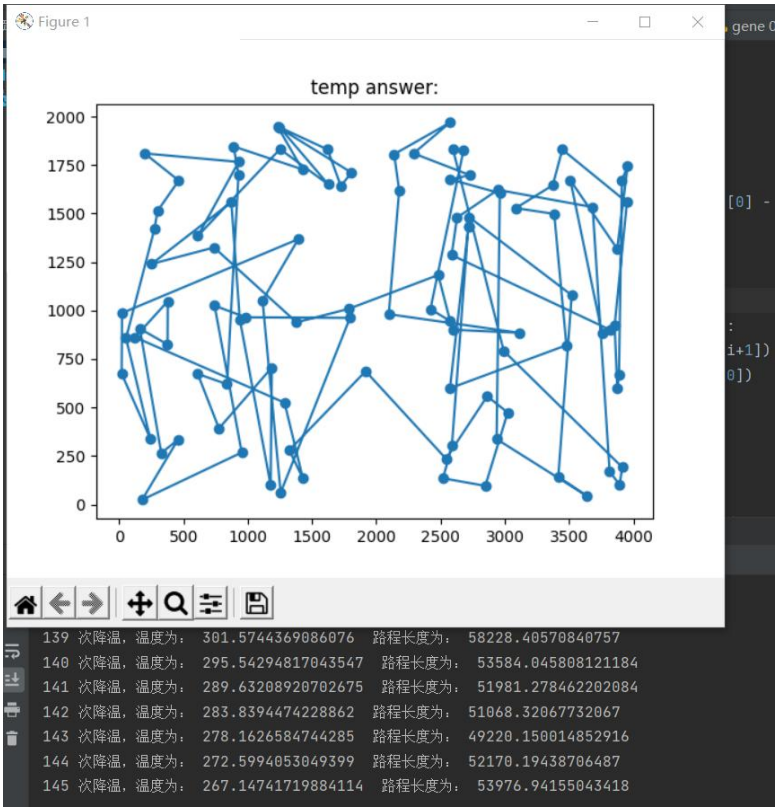
```python
def slide_mutation(path):
    a, b = random.sample(range(0, len(location) - 1), 2)
    if a > b:
        a, b = b, a
    t = path[a]
    for i in range(a + 1, b + 1):
        path[i - 1] = path[i]
    path[b] = t
    return path
```

使用了三种变异操作以及 cpx 交叉操作来增加物种的多样性，产生更多更可能好的解

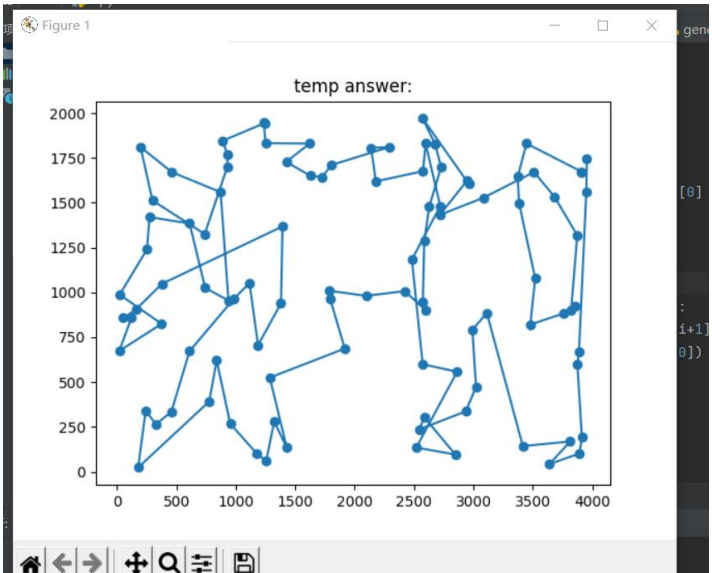
三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

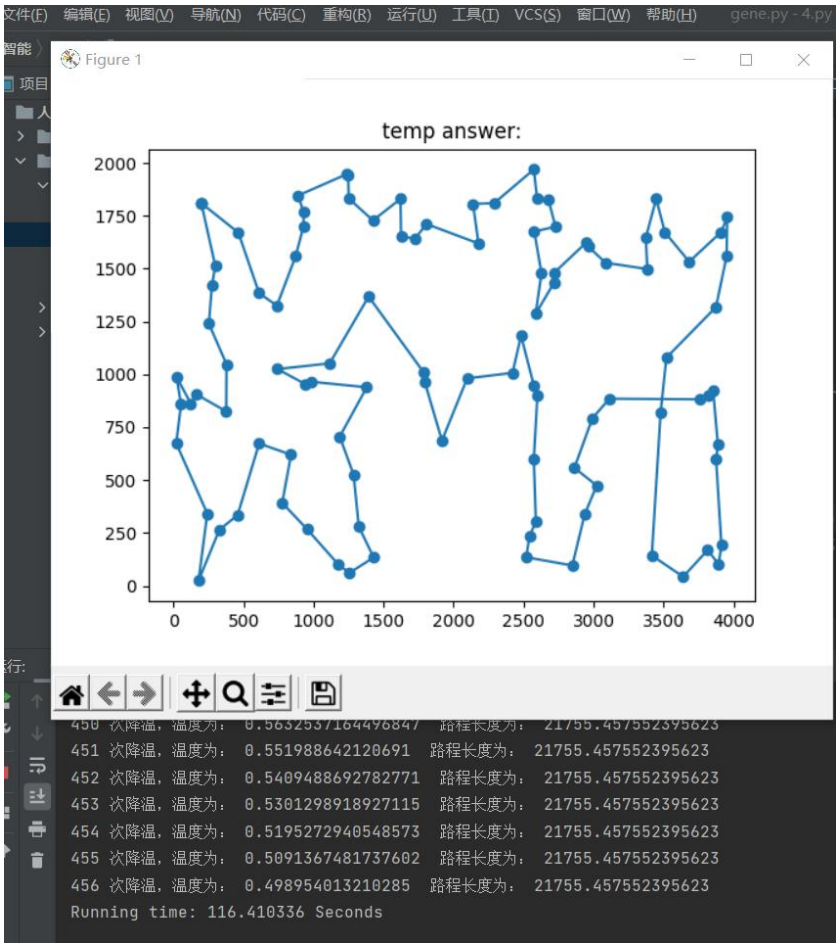


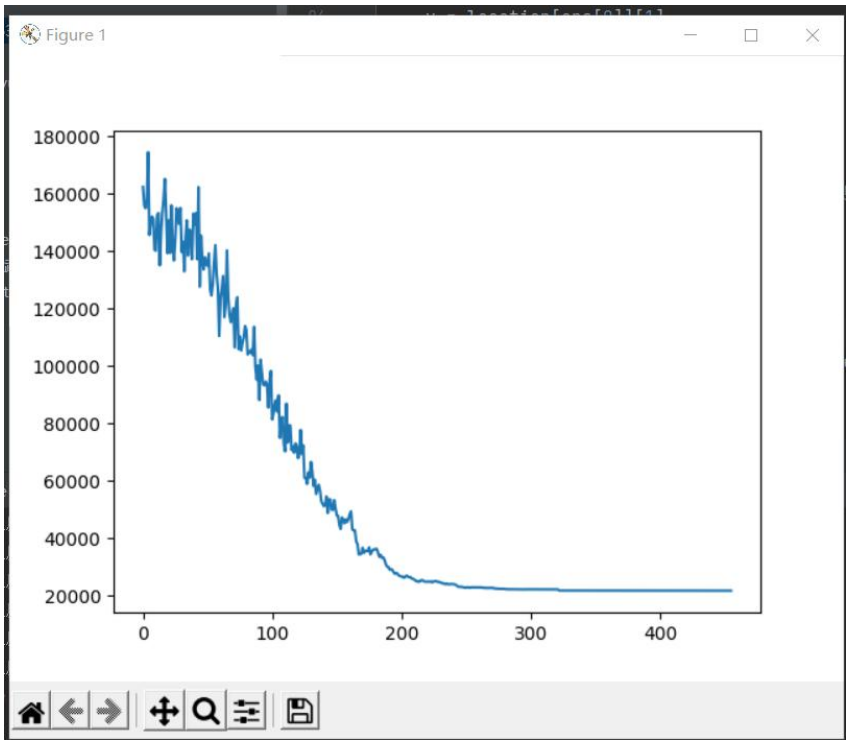



```
139 次降温，温度为： 301.5744369086076  路程长度为： 58228.40570840757
140 次降温，温度为： 295.54294817043547  路程长度为： 53584.045808121184
141 次降温，温度为： 289.63208920702675  路程长度为： 51981.278462202084
142 次降温，温度为： 283.8394474228862  路程长度为： 51068.32067732067
143 次降温，温度为： 278.1626584744285  路程长度为： 49220.150014852916
144 次降温，温度为： 272.5994053049399  路程长度为： 52170.19438706487
145 次降温，温度为： 267.14741719884114  路程长度为： 53976.94155043418
```



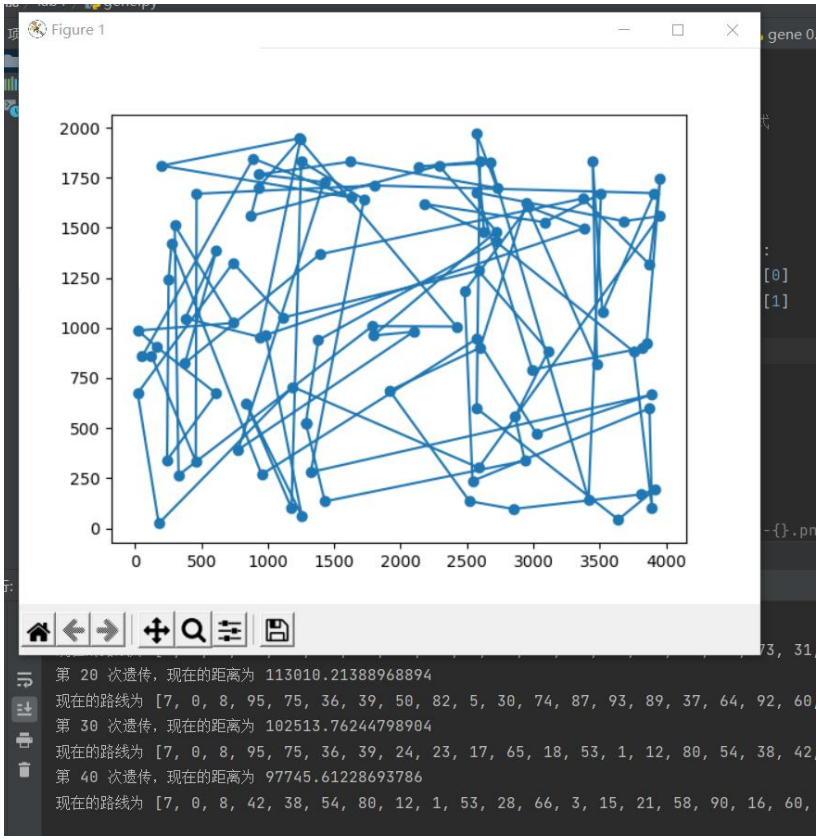

```
209 次降温，温度为： 73.31955561492431  路程长度为： 30696.35041429924
210 次降温，温度为： 71.85316450262583  路程长度为： 30259.083242857334
211 次降温，温度为： 70.41610121257331  路程长度为： 29920.53280014928
212 次降温，温度为： 69.00777918832185  路程长度为： 29622.053941195532
213 次降温，温度为： 67.62762360455541  路程长度为： 29969.833331932467
214 次降温，温度为： 66.2750711324643  路程长度为： 29962.246897496
215 次降温，温度为： 64.94956970981502  路程长度为： 29680.975882477625
```

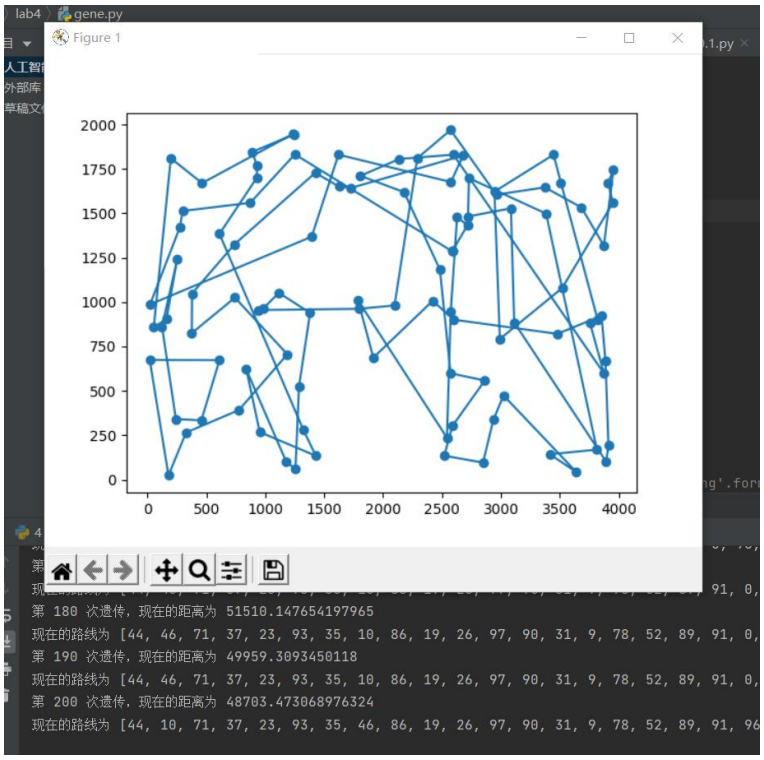


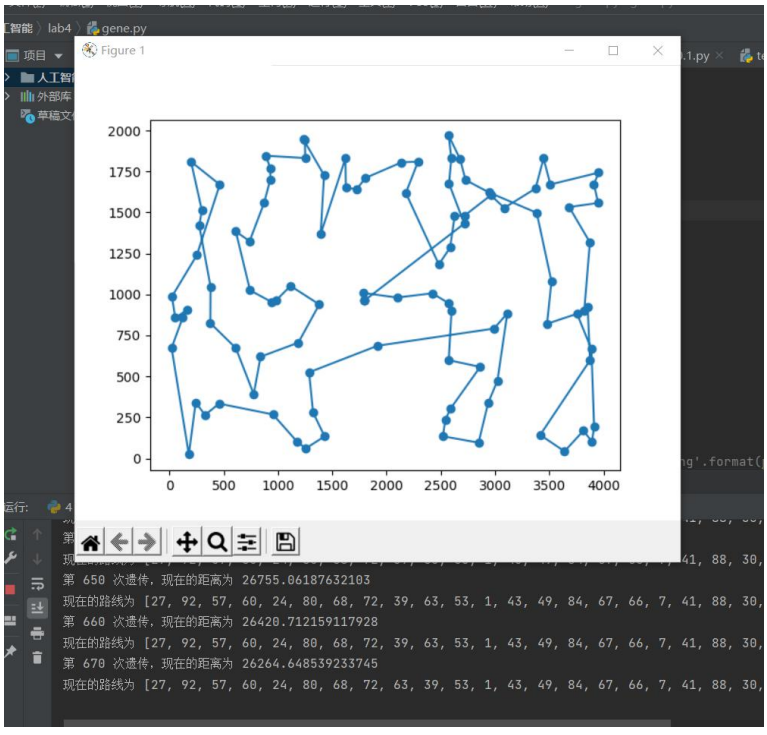


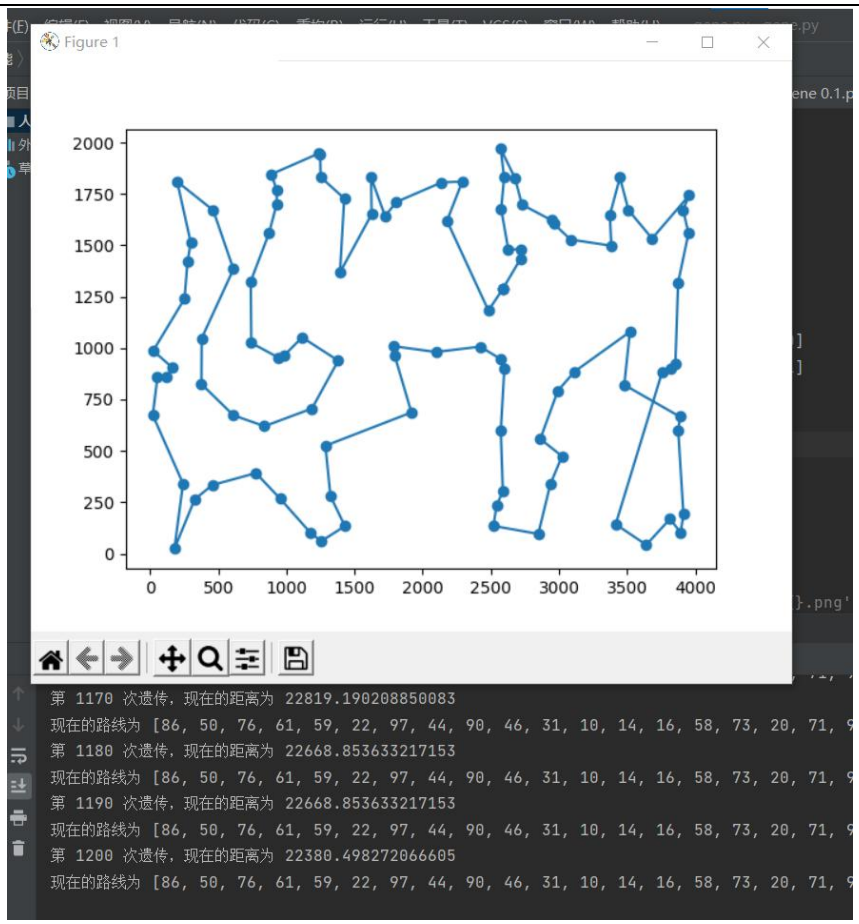
首先是模拟退火算法因为是动图的原因，我这里可能只是截取其中的几个部分图片，然后具体的所有运行操作我用了屏幕录制的方式，并将所得到的视频文

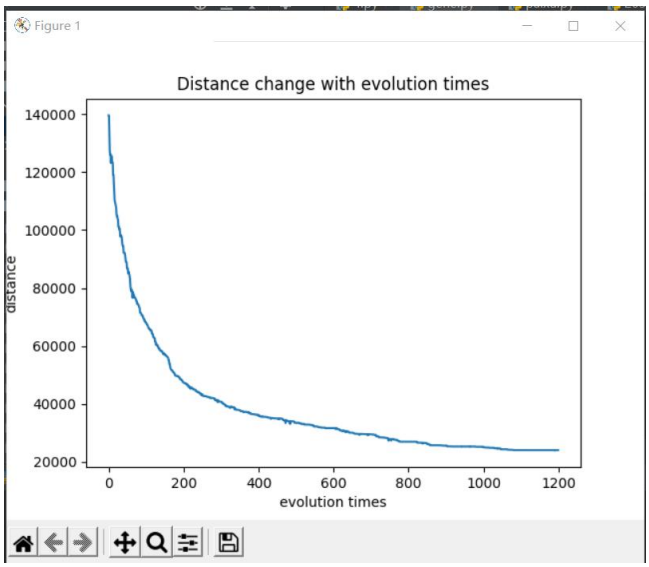
件放在了 result 文件夹中，可以具体地去看，（后来发现大于 50M 就删掉了）但也可以看到，模拟退火算法在逐渐改变里面的路线，然后一开始会更多地接受不好的解，后面接受不好的解的概率减小，然后 逐渐趋向于收敛，也可以看做一个理线的过程，慢慢就理清楚了，然后最后得到的结果是 21755 ， 与最好的在网站中的结果 21282 相比，差距在 2.2%，可以接受




lab4 gene.py
Figure 1
2000
1750
1500
1250
1000
750
500
250
0
0 500 1000 1500 2000 2500 3000 3500 4000
第 180 次遗传，现在的距离为 51510.147654197965
现在的路线为 [44, 46, 71, 37, 23, 93, 35, 10, 86, 19, 26, 97, 90, 31, 9, 78, 52, 89, 91, 0,
第 190 次遗传，现在的距离为 49959.3093450118
现在的路线为 [44, 46, 71, 37, 23, 93, 35, 10, 86, 19, 26, 97, 90, 31, 9, 78, 52, 89, 91, 0,
第 200 次遗传，现在的距离为 48703.473068976324
现在的路线为 [44, 10, 71, 37, 23, 93, 35, 46, 86, 19, 26, 97, 90, 31, 9, 78, 52, 89, 91, 96


lab4 gene.py
Figure 1
2000
1750
1500
1250
1000
750
500
250
0
0 500 1000 1500 2000 2500 3000 3500 4000
第 650 次遗传，现在的距离为 26755.06187632103
现在的路线为 [27, 92, 57, 60, 24, 80, 68, 72, 39, 63, 53, 1, 43, 49, 84, 67, 66, 7, 41, 88, 30,
第 660 次遗传，现在的距离为 26420.712159117928
现在的路线为 [27, 92, 57, 60, 24, 80, 68, 72, 39, 63, 53, 1, 43, 49, 84, 67, 66, 7, 41, 88, 30,
第 670 次遗传，现在的距离为 26264.648539233745
现在的路线为 [27, 92, 57, 60, 24, 80, 68, 72, 63, 39, 53, 1, 43, 49, 84, 67, 66, 7, 41, 88, 30,





遗传算法的具体例子，这里可能只是截取其中的几个部分图片，然后具体的所有运行操作我用了屏幕录制的方式，并将所得到的视频文件放在了 result 文件夹中，（大于 50M，如果需要也有）但可以发现的是，遗传算法的运行速度要低于退火算法，但是可能是因为有多种的交叉和变异的方式，最后虽然很缓慢地变化，

但还是 在不断地优化的，最后得到的结果时 22380，比较网页上最好结果 21282，差了 5.1%，可以接受。

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

从运行时间的角度来看，因为遗传算法中每一次运行都会在种群中进行多次锦标赛的选择，而且还有在种群中选择一个最好的路线进行输出的操作，所以时间复杂度较高，而模拟退火算法只对单个样本进行变换，所以操作的时间需求比较小，但其实更多的时间操作还是看具体设定的运行次数的，比如遗传算法的传递几代，模拟退火的开始温度和结束温度等，具体的运行时间如下图

模拟退火：

遗传：

从最终得到的答案更优的角度来看，因为遗传算法中是对于整个种群进行操作，所以理论上保留的多样性更多，加上交叉变异等变换操作，得到的新解的多样性更明显，也就应该更容易找到最优的解，而模拟退火算法主要通过对单个样本的变化，多样性靠的是概率性地接受差解，但也有一大部分原因还是看具体设定的运行次数的，比如遗传算法的传递几代，模拟退火的开始温度和结束温度等所以这两种方法都有好的方面，可以选择使用

四、 参考资料

https://blog.csdn.net/ha_ha_ha233/article/details/91364937?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216511281111678239534973 2%2522%252C%2522scm%2522%253A%252220140713.

130102334..%2522%257D&request_id=165112811116782395349732&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-91364937.142^v9^control,157^v4^control&utm_term=%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95python&spm=1018.2226.3001.4187

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想