利用遗传算法解决 TSP 问题及可视化程序

一、实验目的

- 掌握遗传算法在实际问题中的应用方法
- 理解遗传算法中染色体编码和适应度函数的定义方法
- 熟悉遗传算法中选择、交叉、变异操作在实际问题中的适配问题
- 熟悉遗传算法中的控制参数设置的性能的影响
- 将解决方案可视化并建立和用户交互前端程序

二、实验内容

在本次实验中，我们给定了中国 34 个省会城市的二维坐标，其中部分数据截图如下：

```
重庆,106.54,29.59
拉萨,91.11,29.97
乌鲁木齐,87.68,43.77
银川,106.27,38.47
呼和浩特,111.65,40.82
南宁,108.33,22.84
哈尔滨,126.63,45.75
长春,125.35,43.88
沈阳,123.38,41.8
石家庄,114.48,38.03
太原,112.53,37.87
西宁,101.74,36.56
```

```
济南,117,36.65
郑州,113.6,34.76
南京,118.78,32.04
合肥,117.27,31.86
杭州,120.19,30.26
福州,119.3,26.08
南昌,115.89,28.68
长沙,113,28.21
武汉,114.31,30.52
广州,113.23,23.16
台北,121.5,25.05
```

数据总共由 34 行组成，每一行代表一个城市名字以及对应坐标。两个城市之间的距离可以通过对应坐标求欧氏距离得到。对于给定数据，要求选择始发城市和剩余 33 个城市中的全部城市或部分城市作为需要遍历的城市，通过编写相应的遗传算法代码，求解 TSP 问题中回到始发城市的路径，并且尽可能的使路径总长度最短。

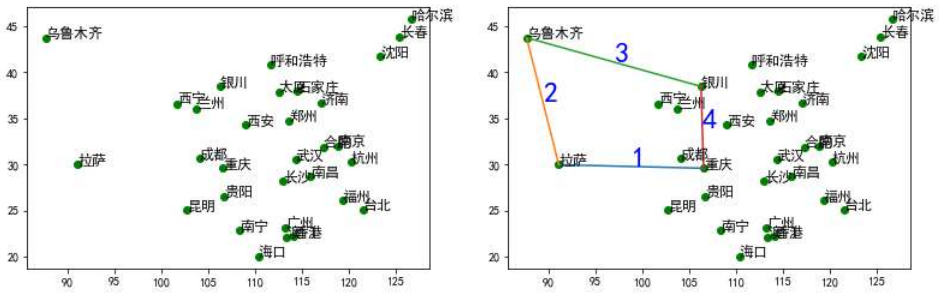


图 1. 地图可视化

设计实现相应的前端可视化程序。例如，34 个城市的可视化结果如图 1 左图所示。两个城市之间的距离可以通过两个城市的坐标来求出，假设在旅行商问题中，旅行商希望经过的城市为乌鲁木齐、银川、拉萨、重庆这四个城市并且起点为重庆，那么一个可行的解的可视化结果如图 1 右图所示。设计的软件程序中包含类似的地图可视化。

三、实验方法设计

介绍实验中程序的总体设计方案、关键步骤的编程方法及思路，主要包括:

1）染色体编码和适应度定义的程序设计（伪代码或源代码截图）及说明解释

染色体编码用数字表示，每个城市对应一个数字，一共有 34 个城市，分别对应数字 0 到 33。以字典的形式存储，代码如下图：

```
datas = {}
with open("../data/data.txt", "r") as f:
    data_list = f.readlines()
for i in range(len(data_list)):
    str = data_list[i].replace("\n", "")
    temp = str.split(',')
    datas[i] = temp
return datas
```

适应度定义为个体的距离，路线的距离越近，适应度越高。

2）种群初始化程序设计（伪代码或源代码截图）及说明解释

首先根据从文件中读取的数据，用 random.choice()函数来随机选择路线，例如一个列表 a = [0, 1, 2, … , 33]，从中随机选择一个城市的编码，添加到一个新的列表 b 中，再移除列表 a 中被选择的城市个体编码。重复上面的步骤，直到列表 a 中的城市编码全部被随机选择，此时就产生了一条城市路线。利用这种方法产生一定数量的初始化路线，我在实验中选择了 300 条初始化路线，在产生的时候做了去重。300 条路线中是没有重复的，然后再对 300 条路线对象化，产生 300 个个体实例。

随机选择路线：

```
cities_list = list(data)
count = n
routes = []
flag = 0
while count > 0:
    if flag > 300:
        print("please choose reasonable size of according to the length of route.")
        break
    flag += 1
    temp = cities_list.copy()
    route = []
    while temp != []:
        gene = random.choice(temp)
        route.append(gene)
        temp.remove(gene)
    if route not in routes:
        routes.append(copy.deepcopy(route))
        count -= 1
return routes
```

将路线初始化成个体实例：

```
def initIndividual(route_list: list, data: dict)->list:
    """
    according to the route set initial individual set
    :param route_list: route set, different route(list) in in route_list
    :param data: refer to value of getData()
    :return: a list contains instance of class Individual
    """
    result = []
    for i in range(len(route_list)):
        result.append(Individual(route_list[i], data))
    return result
```

个体类定义：

```
class Individual:
    """
    data(dict type): value return getData()
    route(list type): a route travel all the cities
    distance(float): value of fitness function as well as distance of route
    """

    def __init__(self, route, data):
        self.data = data
        self._route = copy.deepcopy(route)
        self.distance = 0
        self._getinstance()

    @property
    def route(self):
        return self._route

    def _getinstance(self):
```

3）选择操作程序设计（伪代码或源代码截图）及说明解释

我采用的是精英保留策略，选择操作选择前 300 个优良的个体，在第一代的时候，选择的个体是就是初始化种群的所有个体(初始种群规模是300)，后面的交叉和变异都会产生新的个体，群体个体数量增加，根据精英保留策略，此时依据个体的适应度（路径的距离）淘汰劣势个体，保留前 300 个优良个体。

```
def select(self):
    """
    I use elitist preservation strategy, maintain the best solution found over time before selection. preserve top N individuals
    """
    if len(self.individuals) == N:
        """
        if self.individuals is initial individuals, return
        """
        return
```

```
for count in range(self.new_born_num):
    distance = self.individuals[0].distance
    flag = 0
    for i in range(len(self.individuals)):
        if distance < self.individuals[i].distance:
            distance = self.individuals[i].distance
            flag = i
    worst = self.individuals[flag]
    self.individuals.remove(worst)
```

4）交叉操作程序设计（伪代码或源代码截图）及说明解释

由选择操作选择出来的优良个体以交叉概率 Pm 进行两两交叉，假如个体 1 可以交叉，则个体 1 和个体 2 交叉，假如个体 1 和 2 的基因如下

个体 1：

1	2	3	4	5	6	7	8	9	0

个体 2：

3	5	1	2	6	9	8	4	0	7

首先随机选择基因片段，假设选中颜色标注的基因片段。

新个体 1 的产生： 遍历个体 2 的基因，如果基因编码不在交叉的基因片段 1 中，就添加到一个序列后面，到索引的长度时直接拼接基因片段 1，接着继续遍历个体 2 的基因，直至遍历完个体 2 的基因。产生新个体的图示如下：

3									
3	1								
3	1	2							
3	1	2	4	5	6	7			
3	1	2	4	5	6	7	9		

3	1	2	4	5	6	7	9	8	
3	1	2	4	5	6	7	9	8	0

同样，另一个新个体的基因型是

1	3	4	2	6	9	8	5	7	0

这样，每一次交叉都会产生两个新的个体。

代码实现如下：

```
def cross(self):
    """
    cross method produces new born individuals and add it to the self.individuals
    self.new_born_num counts new born individuals in total

    """
    self.new_born_num = 0
    for i in range(len(self.individuals)-1):
        rate = random.random()
        if rate < Pc:
            parent1 = copy.deepcopy(self.individuals[i])
        else:
            continue
        parent2 = copy.deepcopy(self.individuals[i+1])
        if parent1.route == parent2.route:
            continue
        gene_length = len(self.individuals[0].route)
        index1 = random.randint(0, gene_length - 1)
        index2 = random.randint(index1, gene_length - 1)
        gene_segment1 = parent2.route[index1:index2]
        route1 = []
```

```
flag = 0
for gene in parent1.route:
    if gene not in gene_segment1:
        if flag == index1:
            route1 += gene_segment1
        route1.append(gene)
        flag += 1
flag = 0
gene_segment2 = parent1.route[index1:index2]
route2 = []
for gene in parent2.route:
    if gene not in gene_segment2:
        if flag == index1:
            route2 += gene_segment2
        route2.append(gene)
        flag += 1
new_individual2 = Individual(route2, self.individuals[0].data)
self.individuals.append(new_individual2)
new_individual1 = Individual(route1, self.individuals[0].data)
self.individuals.append(new_individual1)
self.new_born_num += 2
```

5）变异操作程序设计（伪代码或源代码截图）及说明解释

交叉之后群体中的每个个体以变异概率 Pm 变异，假设个体的基因型如下：

1	2	3	4	5	6	7	8	9	0

变异时随机选择两个编码，然后进行交换，这样就完成了变异操作。假如选中的交互的基因编码位置是 0(对应的基因编码是 1)和 8(对应的基因编

码是 9)，则变异之后产生的新个体基因型如下：

9	2	3	4	5	6	7	8	1	0

实现的代码如下：

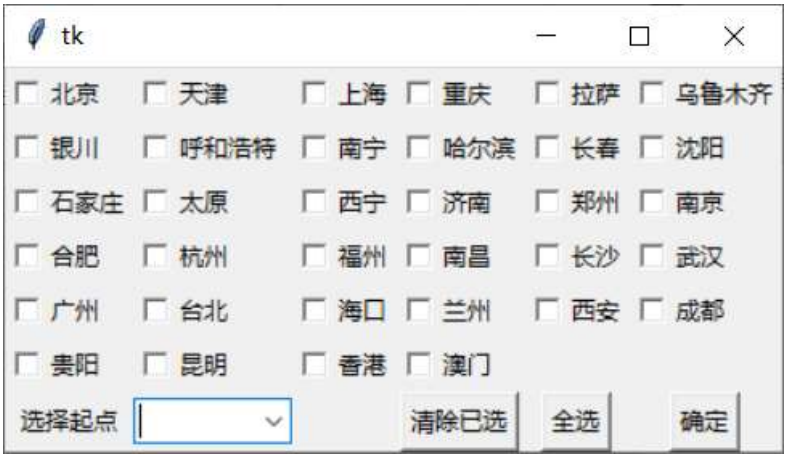
```
def mutate(self):
    gene_length = len(self.individuals[0].route)
    for i in range(len(self.individuals)):
        while True:
            index1 = random.randint(0, gene_length - 1)
            index2 = random.randint(0, gene_length - 1)
            if index1 != index2:
                self.individuals[i].route[index1],
self.individuals[i].route[index2] = self.individuals[i].route[index1],
self.individuals[i].route[index2]
                break
```

四、实验结果展示

展示程序界面设计、运行结果及相关分析等，主要包括：

1）可视化程序界面展示及各功能组件介绍

界面入下图：

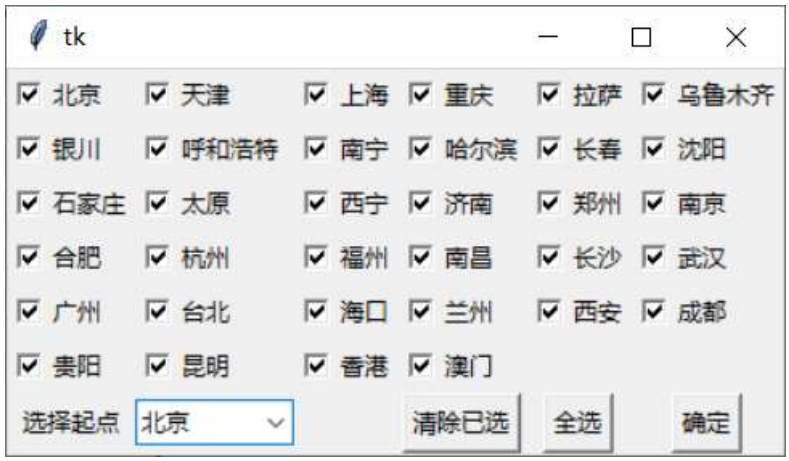


通过复选框勾选选择要经过的城市，通过下拉菜单选择起点城市，清除已

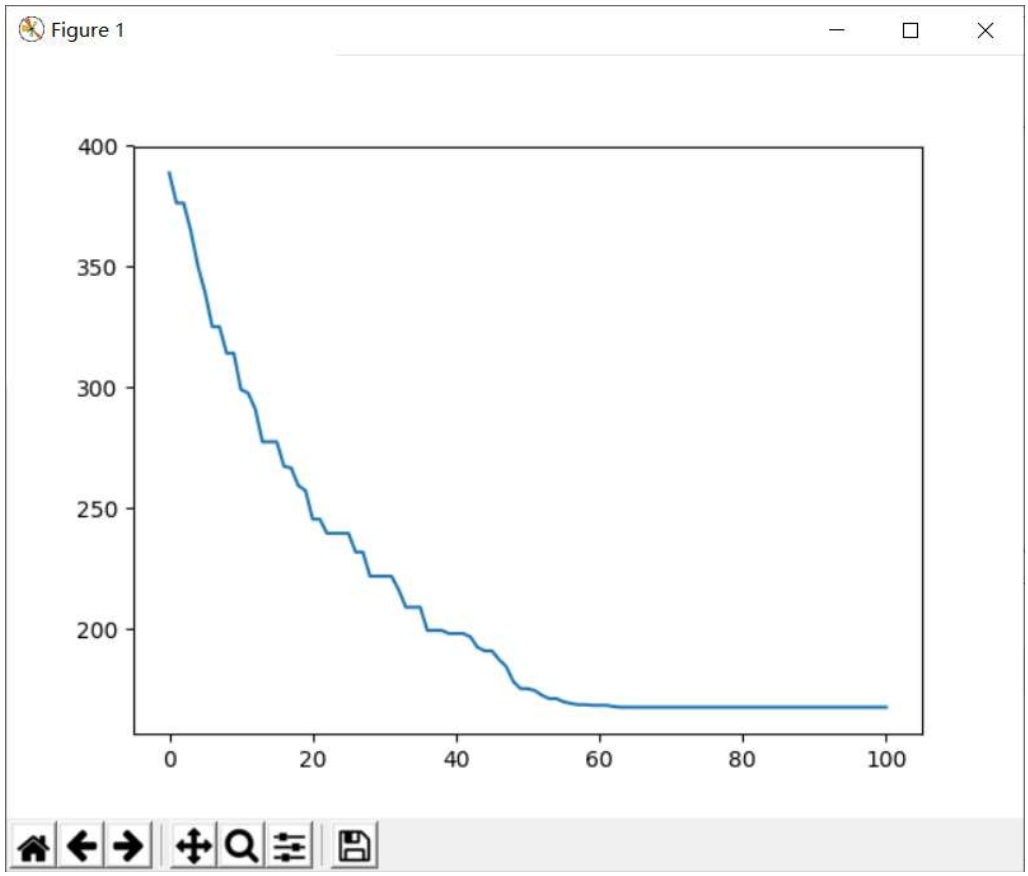
选按钮可以全部清除勾选的城市，全选按钮可以选择所有城市，在选择好起点和要经过的城市之后，点击确定按钮就可以开始。结果可以参照 2）。

2）遗传算法收敛图（适应度值随迭代增加的变化趋势）

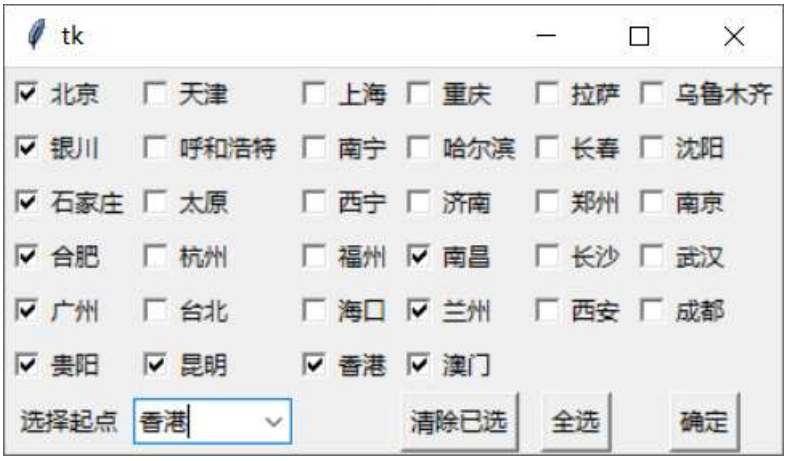
选择全部城市的结果：



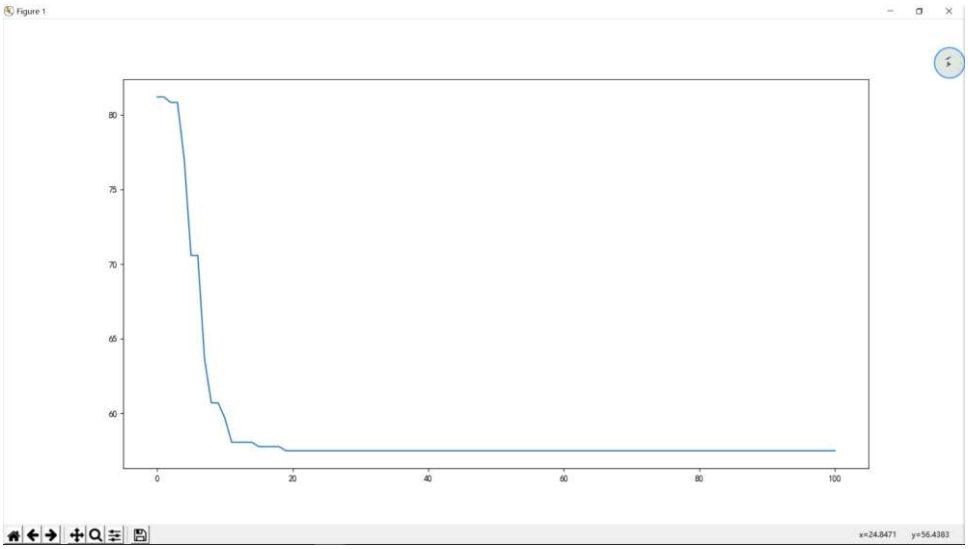
收敛曲线如下（纵坐标是最有个体的路线距离，横坐标是进化的代数）：



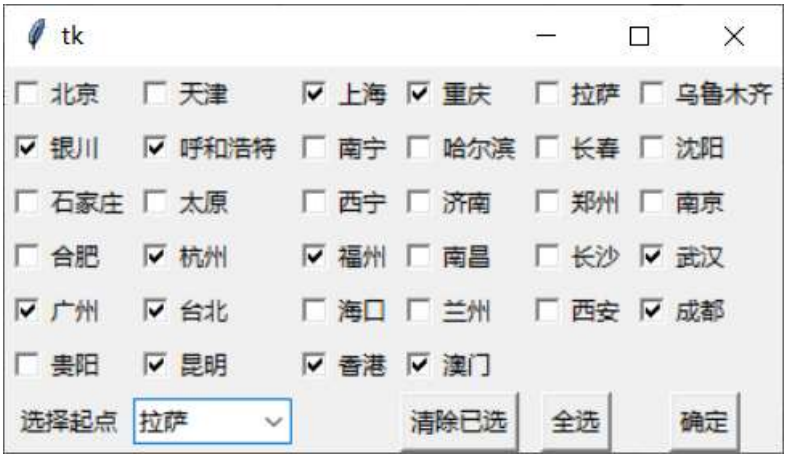
选择部分城市：



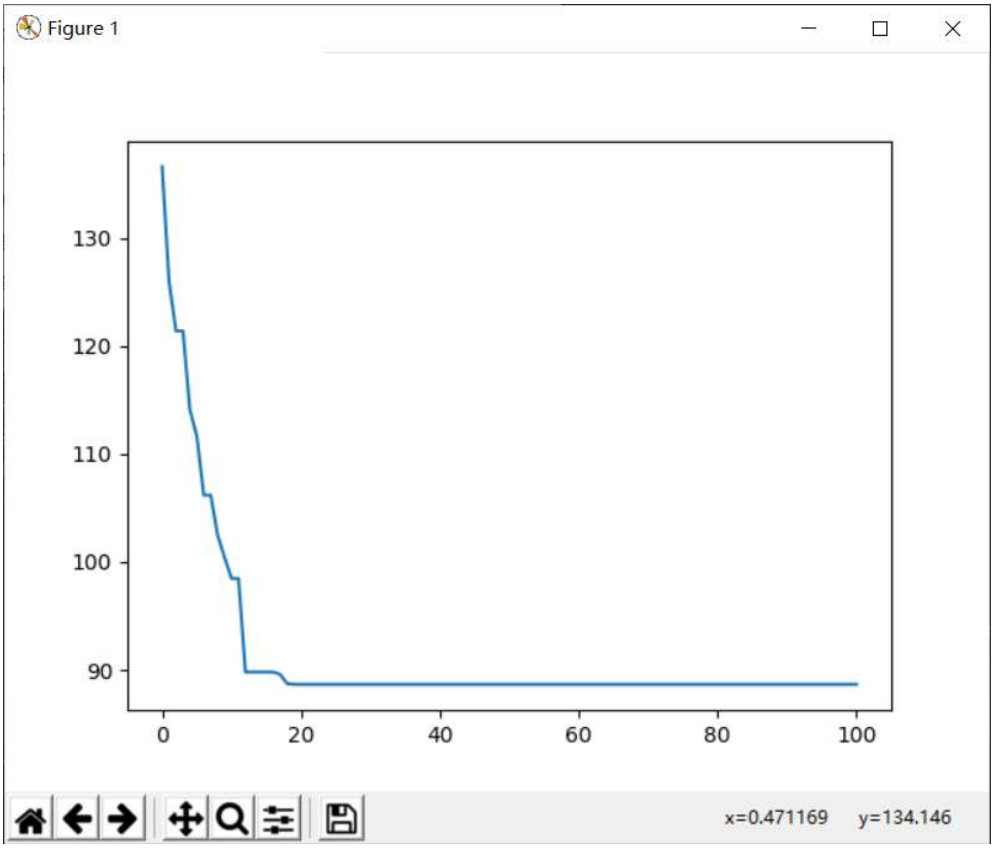
收敛曲线：



部分路线：

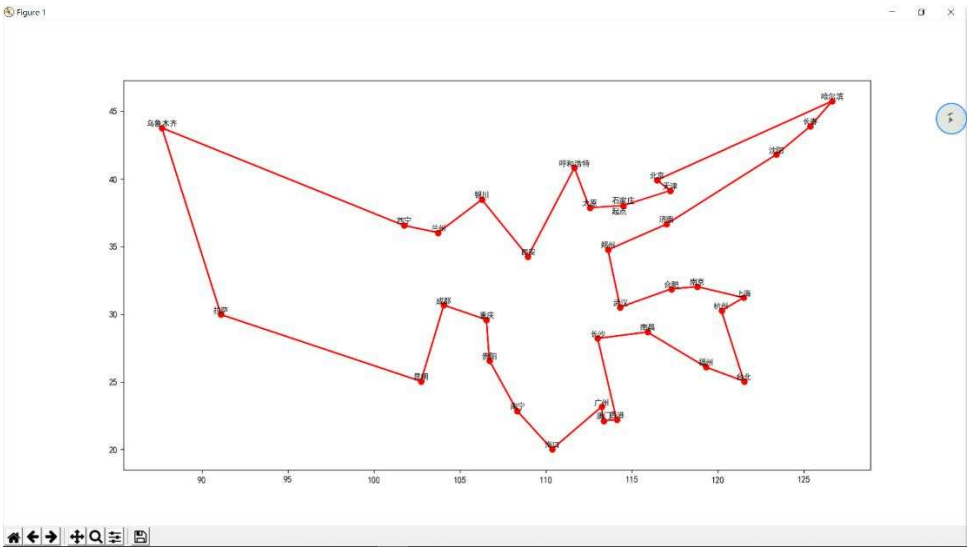


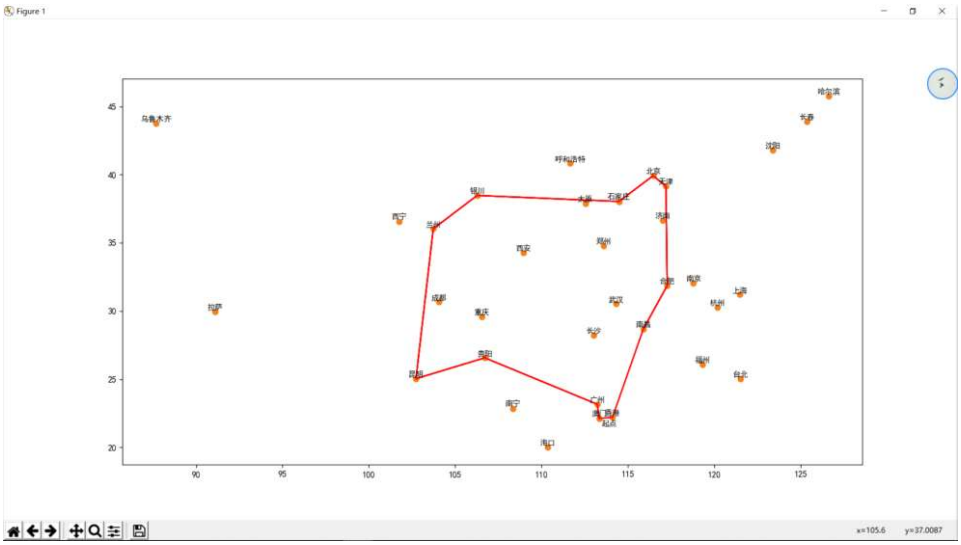
收敛曲线：

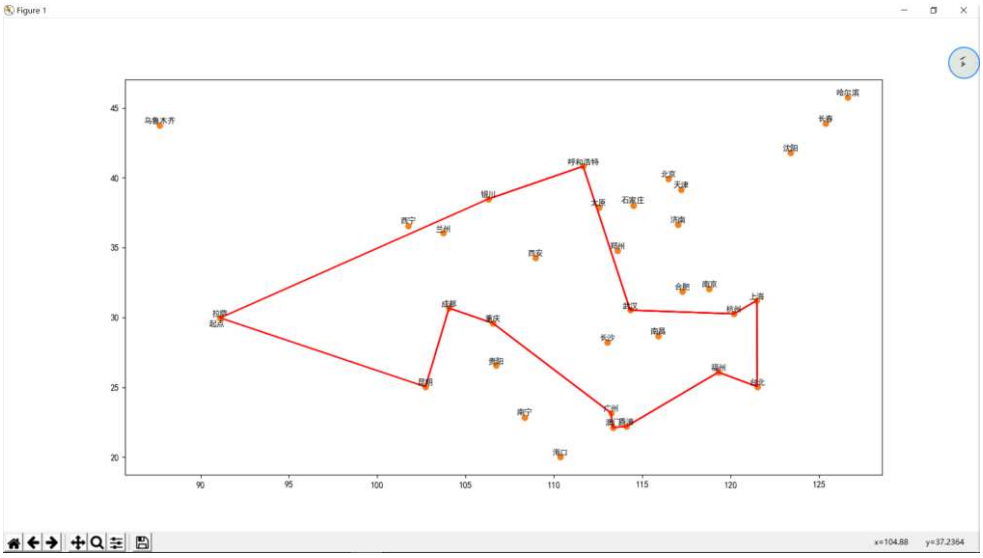


3）不同城市组合下的最优结果城市路径展示及分析

下列最优路径的图像与 2）中的收敛图一一对应。由下列不同的图像可以看出，最优解都呈现出无交叉的闭合回路，第一个图是全部城市一个比较好的最优，但并不是真正的最优解，第二和第三个图可以看是得到的是真正的最优解。







4）不同初始种群个数对结果的影响分析

初始化种群数量的大小会影响收敛速度和结果，初始种群过小，收敛速度快，但是全局搜索能力弱，得出了的最优解与真正的最优解可能相差比较大；反之，如果初始化种群数量过大，收敛速度就变慢了，但是全局搜索能力强，得出的最优解往往比较好。所以应该选择合适的初始化种群数量。34 个城市，初始化种群数量在 200 到 300 合适。

5）不同交叉概率和变异概率对结果的影响分析

对于交大的交叉概率和变异概率，梯度下降比较快。如果交叉概率和变异概率都很大，遗传算法就类似穷举搜索了。