MCM思路：

1、巡逻路径的研究：

（1）GA遗传算法对于TSP的应用：

一般流程：编码与种群生成、种群适应度的估计与选择、交叉繁殖、变异；

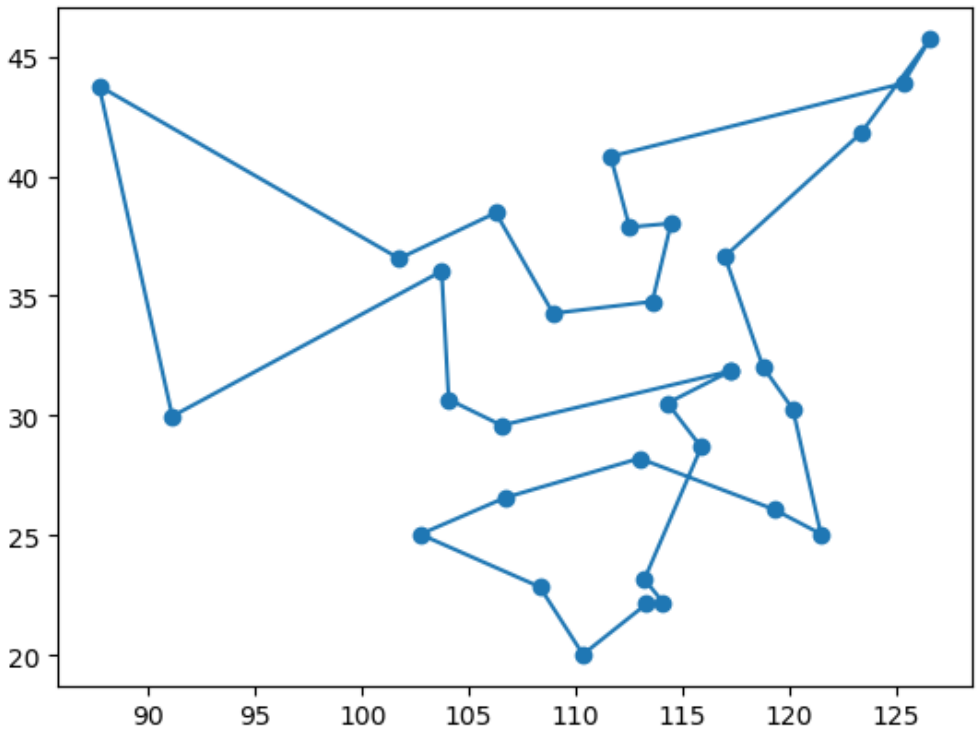
对于城市的编码一般使用十进制直接编码，然后每一组的染色体都是表示一次遍历所有地点的顺序，注意点就是交叉繁殖时应该不能出现重复的元素。例如：

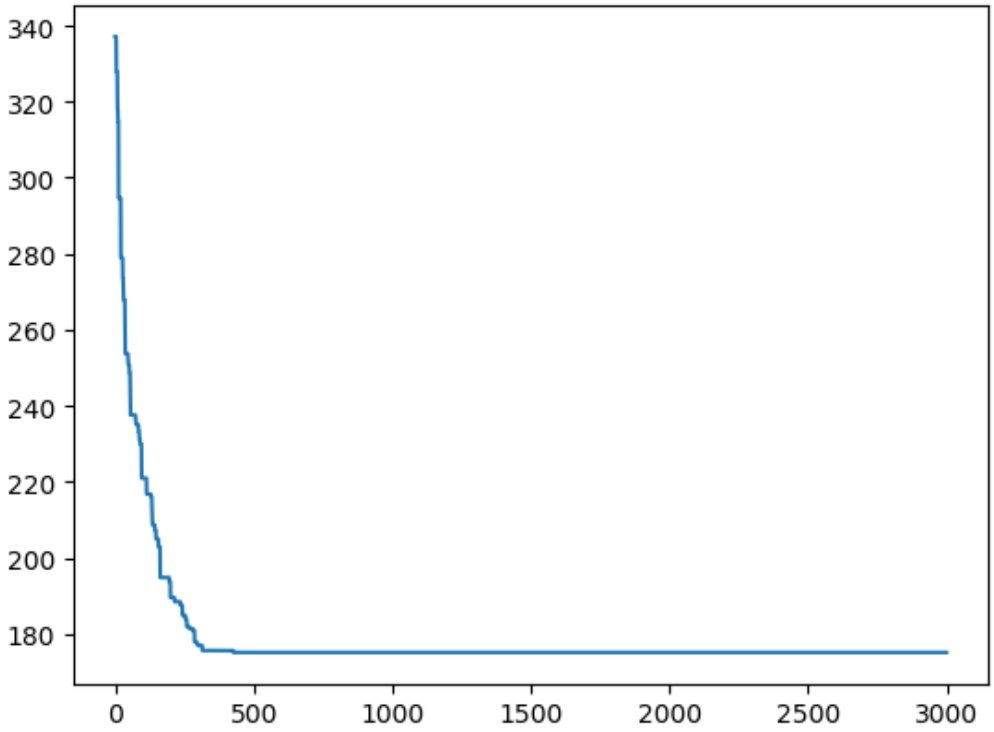
```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
import pandas

# load the data of the city
city_name=[]
city_condition=[]
with open('tsp.txt','r') as f:
    lines = f.readlines()
    for line in lines:
        line=line.split('\n')[0]
        line=line.split(',')
        city_name.append(line[0])
        city_condition.append([float(line[1]),float(line[2])])
city_condition=np.array(city_condition)

#show the map
#plt.scatter(city_condition[:,0],city_condition[:,1])
#plt.show()

# the distance matrix
city_count=len(city_name)
Distance=np.zeros((city_count,city_count))
for i in range(city_count):
    for j in range(city_count):
        Distance[i][j]=math.sqrt((city_condition[i][0]-city_condition[j][0])**2+
(city_condition[i][1]-city_condition[j][1])**2)

count=300 #population amount
improve_count=10000 #improve amount
itter_time=3000 #iteration amount
retain_rate=0.3#the strong rate
random_select_rate=0.5#the weak select rate
mutation_rate=0.1#varitation probability

origin=15#set the start pos
index=[i for i in range(city_count)]
index.remove(15)

#get the distance of each array
def get_total_distance(x):
    distance=0
```

```
    #get the init distance between the origin pos and the first pos
    distance+=Distance[origin][x[0]]
    for i in range(len(x)):
        if i==len(x)-1:
            #accumulate the later pos
            distance+=Distance[origin][x[i]]
        else:
            distance+=Distance[x[i]][x[i+1]]
    return distance

#to random swap the order of the pos in the array to check
#if the distance if shorter than before so that we can choose the better one
def improve(x):
    i=0
    distance=get_total_distance(x)
    while i<improve_count:
        u=random.randint(0,len(x)-1)
        v=random.randint(0,len(x)-1)
        if u!=v:
            #swap function
            new_x=x.copy()
            t=new_x[u]
            new_x[u]=new_x[v]
            new_x[v]=t
            new_distance=get_total_distance(new_x)
            #check the distance
            if new_distance<distance:
                distance=new_distance
                x=new_x.copy()
        else:
            continue
        i+=1

#to sort the fitness, and select the live ones
#and remain some luck one whose fitness is small
def selection(population):
    graded = [[get_total_distance(x),x] for x in population]
    #get the x in order
    graded=[x[1] for x in sorted(graded)]
    #get the length of the strong
    retain_len=int(retain_rate*len(graded))
    #select
    parents= graded[:retain_len]
    #but remain some luck
    for chromo in graded[retain_len:]:
        if random.random()<random_select_rate:
            parents.append(chromo)
    return parents

#cross
def crossover(parents):
    #guarantee the number of the child
    target_count=count-len(parents)
    children=[]
    while len(children)<target_count:
        #to choose the parents in the group
        male_index=random.randint(0,len(parents)-1)
        female_index=random.randint(0,len(parents)-1)
```

```
        if male_index!=female_index:
            male=parents[male_index]
            female=parents[female_index]

            #the cross segment
            left=random.randint(0,len(male)-2)
            right=random.randint(left+1,len(male)-1)
            gene1=male[left:right]
            gene2=female[left:right]

            child1_c=male[right:]+male[:right]
            child2_c=female[right:]+female[:right]
            child1=child1_c.copy()
            child2=child2_c.copy()

            for o in gene2:
                child1_c.remove(o)

            for o in gene1:
                child2_c.remove(o)

            child1[left:right]=gene2
            child2[left:right]=gene1

            child1[right:]=child1_c[0:len(child1)-right]
            child1[:left]=child1_c[len(child1)-right:]

            child2[right:]=child2_c[0:len(child2)-right]
            child2[:left]=child2_c[len(child2)-right:]

            children.append(child1)
            children.append(child2)
    return children

#change the order of the child
def mutation(children):
    for i in range(len(children)):
        if random.random()<mutation_rate:
            child=children[i]
            u = random.randint(1,len(child)-4)
            v = random.randint(u+1,len(child)-3)
            w=random.randint(v+1,len(child)-2)
            child=child[0:u]+child[v:w]+child[u:v]+child[w:]

def get_result(population):
    graded=[[get_total_distance(x),x] for x in population]
    graded=sorted(graded)
    return graded[0][0],graded[0][1]

#init the random group
population=[]
for i in range(count):
    x=index.copy()
    random.shuffle(x)
    improve(x)
    population.append(x)

register=[]
```

```
i=0
while i<itter_time:
    parents=selection(population)

    children=crossover(parents)

    mutation(children)

    population=parents+children

    distance,result_path=get_result(population)

    register.append(distance)

    i=i+1

print(distance)
print(result_path)

result_path=[origin]+result_path+[origin]
X=[]
Y=[]
for index in result_path:
    X.append(city_condition[index,0])
    Y.append(city_condition[index,1])
plt.figure()
plt.plot(X,Y,'-o')
plt.show()
plt.figure()
plt.plot(list(range(len(register))),register)
plt.show()
```





上面的图片显示了最短距离的规划以及距离的收敛方式，能够得到急速的收敛。

（2）改进遗传算法：

第一个是：首先对于初始群体的优化：先对他们进行一个自我改良交换，经过多次迭代后找到当前可能的最短路径的状态（见代码improve部分）

第二个是：Kmeans算法对于初始种群的优化，使用扇形区域的方法对于地点进行分类，然后先把每一个区域的点随机遍历后再进入另一个区域：

第三个：同样是对于扇形区域的不同理解，因为需要多台飞机进行搜索，因此需要划分区域使用不同飞机分别只用搜索固定区域，相当于给定了遍历的初始种群区域，更快收敛。

第四个：用2-opt邻域搜索方法：如下图：

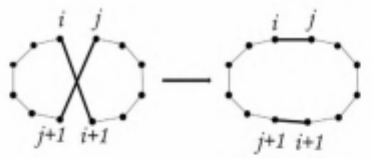


Figure 10: Illustration of 2-opt Optimizer

m of 2-opt optimizer are shown as follows.

Require: $T_1, T_2, ..., T_n$ and Tour
Evaluate the distance matrix
for i = 1 : 1 : n-2
  for j = i+2 : 1 : n
    evaluate $d_1 = E_{i,i+1} + E_{j,j+1}$, $d_2 = E_{i,j} + E_{i+1,j+1}$
    if ($d_1 > d_2$)
      swap indices of the targes in Tour
    end
  end
end

通过这样的遍历查询可以有效加快收敛速率，但是可能会很费时，所以可以规定对于同一个染色体只能进行一次邻域搜索，可以在初始化进行，也可以在交叉中进行但是要使用记录矩阵记录，一旦执行过就不再执行，加快速度。

2、对于港口位置的选取：

使用到的是01规划问题：对于一个事务是否选择它进行公式表示，是表示为1，不表示为0，然后列出损失函数，对损失函数进行优化最小化。

3、整体思路：

01规划处理港口分配问题，使用飞机消耗模型进行分配医药、使用遗传算法对地点巡逻进行路线规划。