曲阜師範大學

学生实验报告

学院 计算机学院

课程 人工智能实训

姓名 彭清元

学号 20204162

年级 2020 级

专业 计算机科学与技术

曲阜师范大学实验报告

学院 计算机学院 年级、专业 20 级计算机科学与技术二班 学号 2020416240

姓名 彭清元 课程名称 人工智能 上课时间 第 1 节 — 第 11 节

实验日期 2023 年 5 月 27 日 星期 六 教师签名 雷玉霞 成绩

实验三 遗传算法解决 TSP 问题

1. 【实训目的】

(1) 学习遗传算法的基本思想

(2) 使用遗传算法解决实际问题，如 TSP 问题

(3) 将理论知识转成实际代码进行展示，运行

2. 【实验设备】

(1) 计算机一台

(2) Idea 软件

3. 【实验原理】

(1) TSP 问题

TSP 问题（Travelling Salesman Problem）即旅行商问题，又译为旅行推销员问题、货郎担问题，是数学领域中著名问题之一。假设有一个旅行商人要拜访 n 个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

(2) 遗传算法

它主要借用了生物进化中“物竞天择，适者生存”的自然机理，通过选择、遗传和变异等机制，模拟自然进化过程来求解复杂问题的一种搜索算法。以其简单通用、鲁棒性强、适合并行处理及应用范围广，大体流程主要分为：初始化种群、计算适应度函数、选择、交叉、变异然后不断重复直到找到理想的解

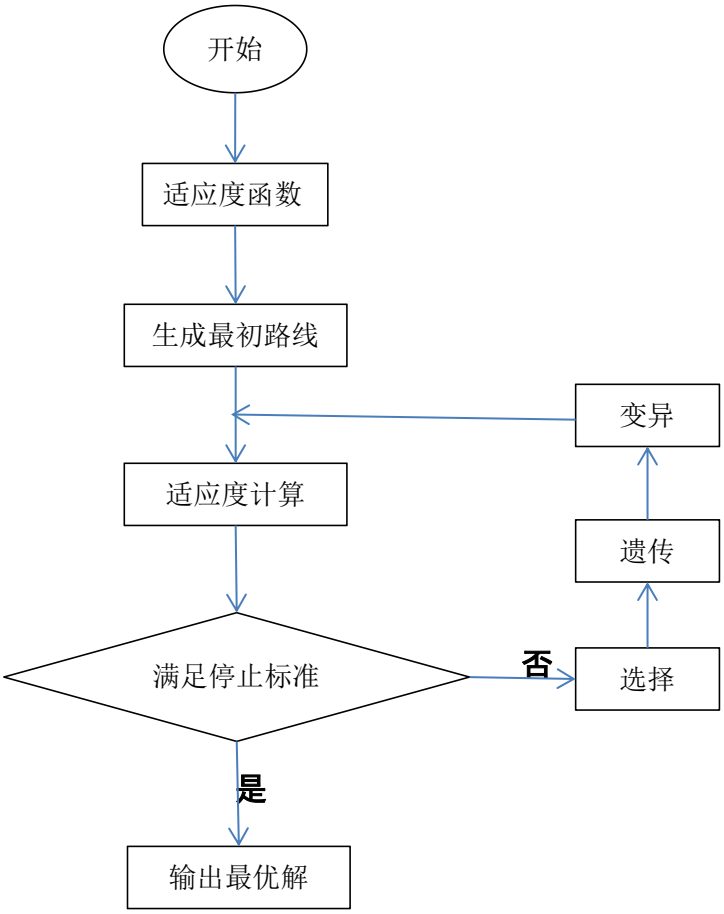
4. 【实验数据】

数据集：cities.txt。

数据集介绍：127 个城市的坐标数据。第一列为城市序号，第二列为城市的 x 坐标，第三列为城市的 y 坐标。

5. 【实验分析】

将我们能够成功访问的所有的城市作为一个种群，将一个路径看成一个个体，在遗传学中的染色体是每一个城市，在这个基础上我们使用遗传算法进行求解，第一步：把问题的解，表示成 “染色体”，在算法中就是以二进制编码的串，给出一群 “染色体”，也就是假设的可行解；第二步：把这些可行解置于问题的 “环境” 中，按适者生存的原则，选取较适应环境的“染色体”进行复制，并通过交叉、变异过程产生更适应环境的新一代 “染色体”群；第三步：经过这样的一代地进化，最后就会收敛到最适应环境的一个 “染色体”上，它就是问题的最优解。将这个应用到旅行商这个具体的问题上，从 cities.txt 获取 127 个城市之间的信息，利用遗传算法，对这些城市之间的信息进行“遗传”“变异”“交叉”，最后得到一个最优解



6. 【实验过程】

(1) 数据准备

从 cities.txt 中获取 127 个城市的相关信息，创建一个 127*127 的矩阵进行存储每两个城市之间的距离，这个距离使用欧式距离进行表示，初始化这个迭代次数以及这个发生变异的概率

```
cities = load_data('cities.txt')  # 导入数据
n_routes = 100  # 路线
epoch = 100000  # 迭代次数
prob = 0.01 # 变异的概率
```

对两个城市之间的距离进行计算，并存入到矩阵当中去，使用 dist_matrix[i,j] 来表示城市 i 和城市 j 之间的距离，

```
def get_cities_distance(cities):
    dist_matrix = np.zeros((127, 127))  # 根据数据的长度来界定---127*127
    n_cities = len(cities)
    for i in range(n_cities - 1):
        for j in range(i + 1, n_cities):
            dist = get_two_cities_dist(cities[i], cities[j])  # 勾股定理求距离
            dist_matrix[i, j] = dist  # 城市 i 到城市 j 的距离
            dist_matrix[j, i] = dist
return dist_matrix
```

(2) 适应性函数

先随机生成 100 路径，使用 np.random.choice 从 n_cities 当中选取 size 个范围是 n_cities 的数据返回的也就是一个一维数组，这个数组作为一个数据样本进行选择处理，每

两个城市之间的距离使用欧式距离计算得到之后，将所有的距离进行相加，将这个倒数作为适应度函数。

```
def init_route(n_route, n_cities):
    routes = np.zeros((n_route, n_cities)).astype(int)
    for i in range(n_route):
        routes[i] = np.random.choice(range(n_cities), size=n_cities, replace=False)
    return routes
def get_route_fitness_value(route, dist_matrix):
    dist_sum = 0
    for i in range(len(route) - 1):
        dist_sum += dist_matrix[route[i], route[i + 1]]  # $$$$$对于一条路线的计算，此时的 route 为 routes[i]
    dist_sum += dist_matrix[route[len(route) - 1], route[0]]  # 由于循环的局限性此处累加单拿出来
    return 1 / dist_sum  # 越短越好，自然选择用倒数
```

(3) 选择操作

选择操作也称为复制（reproduction），是指从当前种群中选出个体以生成交配池（mating pool）的过程。这里使用的是轮盘赌法，轮盘每个区的角度与个体的选择概率成正比，然后产生一个随机数，它落入转盘的哪个区域就选择相应的个体交叉。很显然，选择概率大的个体被选中的可能性较大，获得交叉的机会也就越大。

```
def selection(routes, fitness_values):
    selected_routes = np.zeros(routes.shape).astype(int)
    probability = fitness_values / np.sum(fitness_values)
    n_routes = routes.shape[0]
    for i in range(n_routes):
        choice = np.random.choice(range(n_routes), p=probability)  # 概率越大，取值次数越多
        selected_routes[i] = routes[choice]
    return selected_routes
```

(4) 交叉操作

在这里选择的交叉的方式是使用分割的方法，因此存在许多不同的方案处理，比如存在两条路径 A-B-C-D，C-D-A-B，我们可以将 1，3 号进行交叉，但是不能存在重复的元素，由于这些路径经过不断地选择和交叉之后最终会形成最优的方案。在每次进行分割的时候选择两条路径，将这两条路径进行分割后进行合并操作，具体实现是

```
def crossover(routes, n_cities):
    for i in range(0, len(routes), 2):
        r1_new, r2_new = np.zeros(n_cities), np.zeros(n_cities)

        seg_point = np.random.randint(0, n_cities)
        cross_len = n_cities - seg_point

        r1, r2 = routes[i], routes[i + 1]
        r1_cross, r2_cross = r2[seg_point:], r1[seg_point:]

        r1_non_cross = r1[np.inld(r1, r1_cross) == False]
```

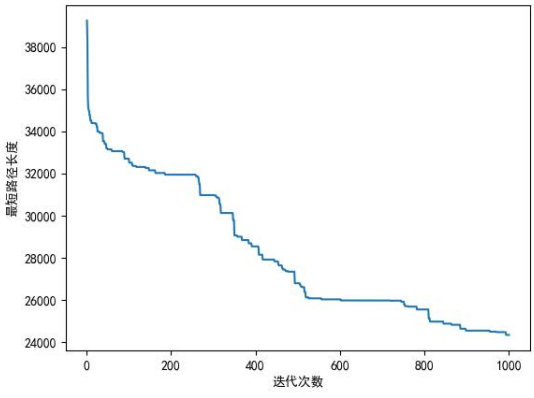
```
        r2_non_cross = r2[np.inld(r2, r2_cross) == False]

        r1_new[:cross_len], r2_new[:cross_len] = r1_cross, r2_cross
        r1_new[cross_len:], r2_new[cross_len:] = r1_non_cross, r2_non_cross

        routes[i], routes[i + 1] = r1_new, r2_new
    return routes
```

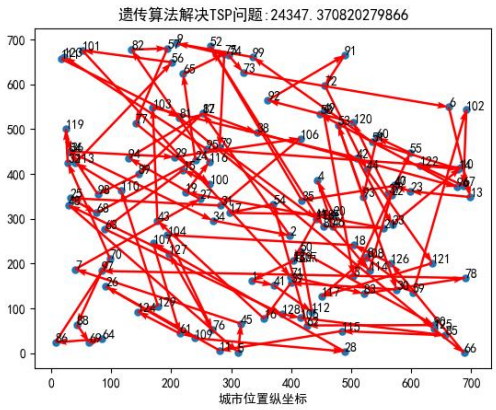
(5) 变异操作

在样本中选取的变异的概率是 0.01，我们只需要在交叉之后，再随机选择几个位置进行改变值就可以了。当然变异的概率是很小的，并且是随机的，这一点要注意。并且由于变异是随机的，所以不排除生成比原来还更加糟糕的个体。具体的变异实现是

```
def mutation(routes, n_cities):
    prob = 0.01
    p_rand = np.random.rand(len(routes))  # 此处的长度的计算返回列表的行数
    # 此处的随机数是生成的 01 之间的小数，长度为 1000
    for i in range(len(routes)):
        if p_rand[i] < prob:
            mut_position = np.random.choice(range(n_cities), size=2, replace=False)
            # 随机抽取不重复的两个数交换
            l, r = mut_position[0], mut_position[1]
            routes[i, l], routes[i, r] = routes[i, r], routes[i, l]
    return routes
```

7. 【实验结果及分析】



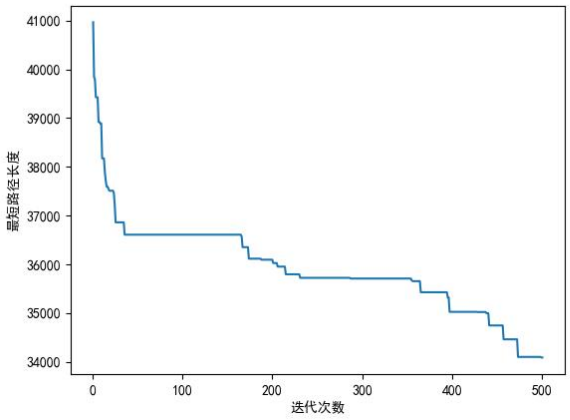


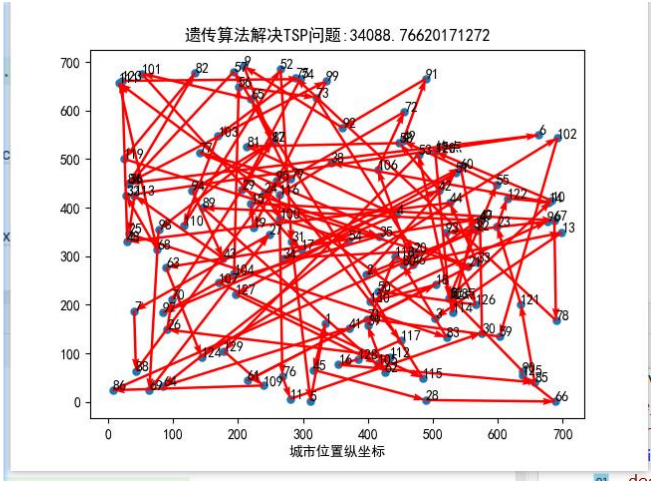
最佳路径为：

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

当前最佳距离为：24347.370820279866

在控制变量的情况下，将交叉概率由原来的 0.95 变成 0.9，变异概率由 0.5 变成 0.1，迭代次数由原来的 1000 次变成 500 次，进行计算，得出最小路径为



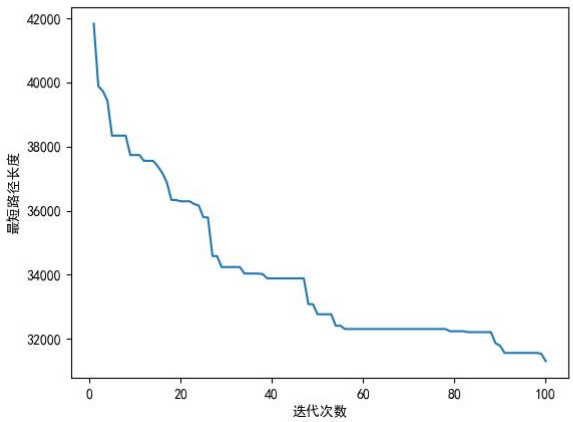


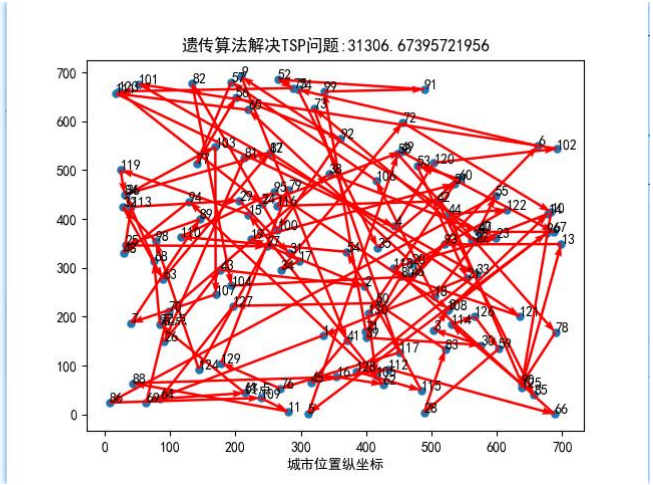
最佳路径为：

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

当前最佳距离为：34088.76620171272

在控制变量的情况下，将交叉概率保持原来的 0.95，变异概率保持原来的 0.5，迭代次数由原来的 1000 次变成 100 次，进行计算，得出最小路径为





最佳路径为：

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

```
当前最佳距离为: 31306.67395721956
```

通过多组数据表示，交叉概率和变异概率变化对最终结果影响较大

8. 【实验心得】

本次实验学习了这遗传算法的基本原理，并且将这个遗传算法结合了 TSP 问题，使用代码将理论和实践进行结合，解决了关于 TSP 最短距离问题。

在使用遗传算法解决这个 TSP 问题，最重要的也就是遗传算法的选择、交叉、变异等操作。在选择过程中，计算出每一条路线与适应性相关的概率；在进行交叉的时候，顺序选出两条路线，数据切分执行后再合并操作；在变异过程中，使用随机函数，随机生成小数与设定突变概率进行比较，将交换位置的数值进行交换

通过这次实训进一步掌握了关于使用遗传算法来解决实际问题，将理论和实践进一步结合，提高了编码能力，加深了对遗传算法的理解。

9. 【附录】

代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ["SimHei"]

# 载入数据
def load_data():
    df = pd.read_csv('cities.txt', sep=" ", skiprows=6, header=None)
    city = np.array(df[0][0:len(df) - 1])  # 最后一行为 EOF，不读入
    city_name = city.tolist()
    city_x = np.array(df[1][0:len(df) - 1])
    city_y = np.array(df[2][0:len(df) - 1])
    city_location = list(zip(city_x, city_y))
    return city_name, city_location

# 计算两个城市的欧式距离
def dist_cal(x, y):
    return ((x[0] - y[0]) ** 2 + (x[1] - y[1]) ** 2) ** 0.5

# 求距离矩阵
def matrix_dis(city_name, city_location):
    city_num = len(city_name)
    res = np.zeros((city_num, city_num))
    for i in range(city_num):
        for j in range(i + 1, city_num):
            res[i, j] = dist_cal(city_location[i], city_location[j])  # 求两点欧式距离
            res[j, i] = res[i, j]  # 距离矩阵：对角线为 0，对称
    return res
```

```
# 初始化种群
def rand_pop(city_num, pop_num, pop, distance, matrix_distance):
    rand_ch = np.array(range(city_num))
    for i in range(pop_num):
        np.random.shuffle(rand_ch)
        pop[i, :] = rand_ch
        distance[i] = comp_dis(city_num, matrix_distance, rand_ch)  # 这里的适应度其实是距离

# 计算每个个体的总距离
def comp_dis(city_num, matrix_distance, one_path):
    res = 0
    for i in range(city_num - 1):
        res += matrix_distance[one_path[i], one_path[i + 1]]
    res += matrix_distance[one_path[-1], one_path[0]]  # 最后一个城市和第一个城市的距离，需单独处理
    return res

# 打印出当前路径
def print_path(city_num, one_path):
    res = str(one_path[0] + 1) + '-->'
    for i in range(1, city_num):
        res += str(one_path[i] + 1) + '-->'
    res += str(one_path[0] + 1)
    print("最佳路径为：")
    print(res)

# 轮盘赌的方式选择子代
def select_sub(pop_num, pop, distance):
    fit = 1. / distance  # 适应度函数
    p = fit / sum(fit)
    q = p.cumsum()  # 累积概率
    select_id = []
    for i in range(pop_num):
        r = np.random.rand()  # 产生一个[0,1)的随机数
        for j in range(pop_num):
            if r < q[0]:
                select_id.append(0)
                break
            elif q[j] < r <= q[j + 1]:
                select_id.append(j + 1)
                break
    next_gen = pop[select_id, :]
    return next_gen

# 交叉操作-每个个体对的某一位置进行交叉
def cross_sub(city_num, pop_num, next_gen, cross_prob, evbest_path):
    for i in range(0, pop_num):
```

```
        best_gen = evbest_path.copy()
        if cross_prob >= np.random.rand():
            next_gen[i, :], best_gen = intercross(city_num, next_gen[i, :], best_gen)

# 具体的交叉方式：部分映射交叉
def intercross(city_num, ind_a, ind_b):
    r1 = np.random.randint(city_num)
    r2 = np.random.randint(city_num)
    while r2 == r1:
        r2 = np.random.randint(city_num)
    left, right = min(r1, r2), max(r1, r2)
    ind_a1 = ind_a.copy()
    ind_b1 = ind_b.copy()
    for i in range(left, right + 1):
        ind_a2 = ind_a.copy()
        ind_b2 = ind_b.copy()
        ind_a[i] = ind_b1[i]
        ind_b[i] = ind_a1[i]
        # 每个个体包含的城市序号是唯一的，因此交叉时若两个不相同，就会产生冲突
        x = np.argwhere(ind_a == ind_a[i])
        y = np.argwhere(ind_b == ind_b[i])
        # 产生冲突，将不是交叉区间的数据换成换出去的原数值，保证城市序号唯一
        if len(x) == 2:
            ind_a[x[x != i]] = ind_a2[i]
        if len(y) == 2:
            ind_b[y[y != i]] = ind_b2[i]
    return ind_a, ind_b

# 变异方式：翻转变异
def mutation_sub(city_num, pop_num, next_gen, mut_prob):
    for i in range(pop_num):
        if mut_prob >= np.random.rand():
            r1 = np.random.randint(city_num)
            r2 = np.random.randint(city_num)
            while r2 == r1:
                r2 = np.random.randint(city_num)
            if r1 > r2:
                temp = r1
                r1 = r2
                r2 = temp
            next_gen[i, r1:r2] = next_gen[i, r1:r2][::-1]

# 绘制路径图
def draw_path(city_num, city_location, pop, distance):
    fig, ax = plt.subplots()
    x, y = zip(*city_location)
```

```
    ax.scatter(x, y, linewidths=0.1)
    for i, txt in enumerate(range(1, len(city_location) + 1)):
        ax.annotate(txt, (x[i], y[i]))
    res0 = pop
    x0 = [x[i] for i in res0]
    y0 = [y[i] for i in res0]
    ax.annotate("起点", (x0[0], y0[0]))
    ax.annotate("终点", (x0[-1], y0[-1]))
    # 绘制箭图
    for i in range(city_num - 1):
        plt.quiver(x0[i], y0[i], x0[i + 1] - x0[i], y0[i + 1] - y0[i], color='r', width=0.005, angles='xy',
scale=1,
                   scale_units='xy')
    plt.quiver(x0[-1], y0[-1], x0[0] - x0[-1], y0[0] - y0[-1], color='r', width=0.005, angles='xy', scale=1,
               scale_units='xy')
    plt.title("遗传算法解决 TSP 问题:" + str(distance))
    plt.xlabel("城市位置横坐标")
    plt.xlabel("城市位置纵坐标")
    plt.savefig("map.png")
    plt.show()

# 绘制最优解随迭代次数的关系
def draw_iter(iteration, best_distance_list):
    iteration = np.linspace(1, iteration, iteration)
    plt.plot(iteration, best_distance_list)
    plt.xlabel("迭代次数")
    plt.ylabel("最短路径长度")
    plt.savefig("figure.png")
    plt.show()

def main():
    city_name, city_location = load_data()
    matrix_distance = matrix_dis(city_name, city_location)
    city_num = len(city_name)  # 城市数量
    pop_num = 300  # 群体个数
    cross_prob = 0.95# 交叉概率
    mut_prob = 0.5  # 变异概率
    iteration = 1000  # 迭代次数

    # 初始化初代种群和距离，个体为整数，距离为浮点数
    pop = np.array([0] * pop_num * city_num).reshape(pop_num, city_num)
    #print(pop)
    distance = np.zeros(pop_num)
    #print(distance)
    # 初始化种群
    rand_pop(city_num, pop_num, pop, distance, matrix_distance)
```

```
    #draw_path(city_num, city_location, pop[0], distance)  # 绘制初代图像

    evbest_path = pop[0]
    evbest_distance = float("inf")
    best_path_list = []
    best_distance_list = []
    # 循环迭代遗传过程
    for i in range(iteration):
        # 选择
        next_gen = select_sub(pop_num, pop, distance)
        # 交叉
        cross_sub(city_num, pop_num, next_gen, cross_prob, evbest_path)
        # 变异
        mutation_sub(city_num, pop_num, next_gen, mut_prob)

        # 更新每个个体距离值(1/适应度)
        for j in range(pop_num):
            distance[j] = comp_dis(city_num, matrix_distance, next_gen[j, :])
        index = distance.argmin()  # 记录最小总路程

        # 为了防止曲线波动，每次记录最优值，如迭代后出现退化，则将当前最好的个体回退,取历史最佳结果
        if distance[index] <= evbest_distance:
            evbest_distance = distance[index]
            evbest_path = next_gen[index, :]
        else:
            distance[index] = evbest_distance
            next_gen[index, :] = evbest_path
        # 存储每一步的最优路径(个体)及距离
        best_path_list.append(evbest_path)
        best_distance_list.append(evbest_distance)

    # 绘制迭代次数和最优解的关系曲线
    draw_iter(iteration, best_distance_list)

    best_path = evbest_path
    best_distance = evbest_distance
    # 迭代完成，打印出最佳路径
    print_path(city_num, best_path)
    print("当前最佳距离为:", best_distance)
    # 绘制路径图
    draw_path(city_num, city_location, best_path, best_distance)

if __name__ == '__main__':
    main()
```

实验的过程不是消极的观察，而是积极的、有计划的探测，一个成功的实验需要的是眼光、勇气和毅力。

——曲阜师范大学名誉校长

诺贝尔奖获得者丁肇中