**计算智能 作业3**

2021年10月24日

**要求：**

1. 在文档中说明解题思路、方法实现、求解结果等，必要时需要对结果进行分析和讨论。
2. 将源代码作为附录粘贴于作业文档末尾，并同时作为提交的附件与作业文档一起放在压缩包内备查。
3. 独立完成，严禁抄袭。

**内容：**

1. 编写程序，用遗传算法（GA）求解下列问题：

旅行商问题（Travelling Salesman Problem, 简记TSP，亦称货郎担问题)：设有n个城市和距离矩阵D=[dij]，其中dij表示城市i到城市j的距离，i，j=1，2 … n，则问题是要找出遍访每个城市恰好一次的一条回路并使其路径长度为最短。说明：

1) 回路：从某个城市出发，最后回到这个城市。

2) 编程语言不限（需提交源代码和运行结果截图）。

3) 作业相关数据说明（15个城市，最短路径距离291）：

a) p01.tsp, the TSP specification of the data.

b) p01\_d.txt, the intercity distance table.

c) p01\_s.txt, an itinerary that minimizes the total distance.

d) p01\_sxy.txt, the XY coordinates of the minimal itinerary.

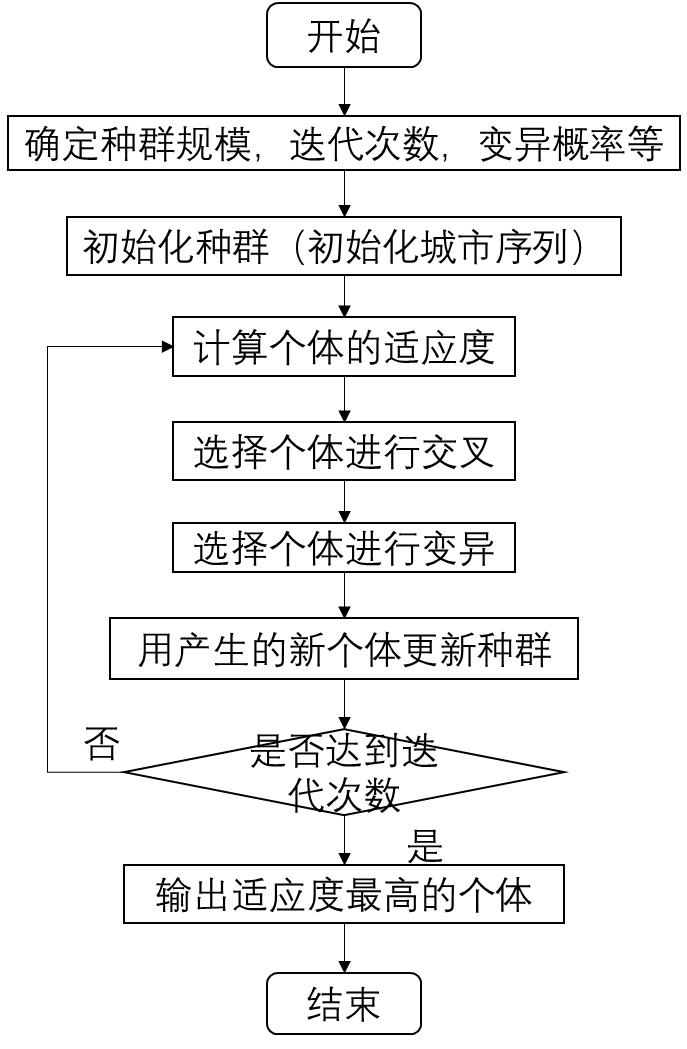
e) p01\_sxy.png, an image of the minimal itinerary.

f) p01\_xy.txt, a set of XY coordinates for the cities, inferred from the distances.

g) p01\_xy.png, an image of the XY coordinates

3) 截止时间：**11月08日 0:00**。

流程图：



解决方法描述：

（1）输入为15个城市坐标，种群规模设置为100，变异概率设置为0.01，迭代次数初步设置为5000。

（2）个体适应度代表的是15个城市连成路线的欧式距离。

（3）选择个体进行交叉操作的时候采用轮盘赌策略。

qa 表示个体a的累积概率，如上图所示个体1、2、3、4的累积概率分别为0.14、0.53、0.6，1。随机生成一个0到1的浮点数f，若 qa < f <= qb，则个体b被选中。当采用轮盘赌策略选择交叉父体之后，采用顺序交叉法进行交叉操作：



（3）变异操作对每一个个体以变异概率确定是否变异，如果变异的话，随机在个体中选择两个城市，然后交换这两个城市的位置得到变异的效果。

（4）产生新的个体之后，采用精英保留策略，即适应度最好的20个体会被保留下来，其他个体按照适应度进行保留。

3.程序流程

1.初始化城市序列的坐标

2.用欧式距离计算城市序列中每个个体的适应度

3.根据适应度来选择个体作为交叉操作的父体，选择完之后用顺序交叉来进行交叉操作

4.以一定的变异个体才确定是否对个体进行变异，如果需要进行变异，侧随机选择个体的两个城市进行交换。

5.选择适应度最好的20个个体直接保留到下一代，下一代的其他个体按照个体的适应度进行选择。

6.判断是否达到迭代次数，如果没有转到第2步，达到的话转到第7步。

7.输出适应度最好的个体。

代码：

import numpy as np

import random

import copy

import matplotlib.pyplot as plt

class City:

    def \_\_init\_\_(self, idx, x, y):

        self.idx = idx

        self.x = x

        self.y = y

    def \_\_repr\_\_(self):

        return str(self.idx)

def distance(ca, cb):

    dx = abs(ca.x - cb.x)

    dy = abs(ca.y - cb.y)

    distance = int(np.sqrt((dx \*\* 2) + (dy \*\* 2)))

    return distance

def init\_pop(city\_list, popSize):

    pop = []

    for i in range(popSize):

        new\_city\_list = random.sample(city\_list, len(city\_list))

        pop.append(new\_city\_list)

    return pop

def fitness(pop):

    dis\_citys = distance\_citys(pop)

    return 1.0 / dis\_citys

def distance\_citys(pop):

    temp\_dis = 0

    for i in range(len(pop) - 1):

        temp\_dis += distance(pop[i], pop[i + 1])

    temp\_dis += distance(pop[len(pop) - 1], pop[0])

    return temp\_dis

def rank(poplulation):

    rankPop\_dic = {}

    for i in range(len(poplulation)):

        fit = fitness(poplulation[i])

        rankPop\_dic[i] = fit

    return sorted(rankPop\_dic.items(), key=lambda x: x[1], reverse=True)

def select(pop, pop\_rank, eliteSize):

    select\_pop = []

    for i in range(eliteSize):

        select\_pop.append(pop[pop\_rank[i][0]])

    cumsum = 0

    cumsum\_list = []

    temp\_pop = copy.deepcopy(pop\_rank)

    for i in range(len(temp\_pop)):

        cumsum += temp\_pop[i][1]

        cumsum\_list.append(cumsum)

    for i in range(len(temp\_pop)):

        cumsum\_list[i] /= cumsum

    for i in range(len(temp\_pop) - eliteSize):

        rate = random.random()

        for j in range(len(temp\_pop)):

            if cumsum\_list[j] > rate:

                select\_pop.append(pop[pop\_rank[i][0]])

                break

    return select\_pop

def breed(pop, eliteSize):

    breed\_pop = []

    for i in range(eliteSize):

        breed\_pop.append(pop[i])

    i = 0

    while i < (len(pop) - eliteSize):

        a = random.randint(0, len(pop) - 1)

        b = random.randint(0, len(pop) - 1)

        if a != b:

            fa, fb = pop[a], pop[b]

            genea, geneb = random.randint(0, len(pop[a]) - 1), random.randint(0, len(pop[b]) - 1)

            startgene = min(genea, geneb)

            endgene = max(genea, geneb)

            child1 = []

            for j in range(startgene, endgene):

                child1.append(fa[j])

            child2 = []

            for j in fb:

                if j not in child1:

                    child2.append(j)

            breed\_pop.append(child1 + child2)

            i = i + 1

    return breed\_pop

def mutate(pop, mutationRate):

    mutation\_pop = []

    for i in range(len(pop)):

        for j in range(len(pop[i])):

            rate = random.random()

            if rate < mutationRate:

                a = random.randint(0, len(pop[i]) - 1)

                pop[i][a], pop[i][j] = pop[i][j], pop[i][a]

        mutation\_pop.append(pop[i])

    return mutation\_pop

def next\_pop(population, eliteSize, mutationRate):

    pop\_rank = rank(population)  # 按照适应度排序

    select\_pop = select(population, pop\_rank, eliteSize)  # 精英选择策略，加上轮盘赌选择

    breed\_pop = breed(select\_pop, eliteSize)  # 繁殖

    next\_generation = mutate(breed\_pop, mutationRate)  # 变异

    return next\_generation

# 画出路线图的动态变化

def GA\_plot\_dynamic(city\_list, popSize, eliteSize, mutationRate, generations):

    plt.figure('Map')

    plt.ion()

    population = init\_pop(city\_list, popSize)

    print("initial distance:{}".format(1.0 / (rank(population)[0][1])))

    for i in range(generations):

        plt.cla()

        population = next\_pop(population, eliteSize, mutationRate)

        idx\_rank\_pop = rank(population)[0][0]

        best\_route = population[idx\_rank\_pop]

        city\_x = []

        city\_y = []

        for j in range(len(best\_route)):

            city = best\_route[j]

            city\_x.append(city.x)

            city\_y.append(city.y)

        city\_x.append(best\_route[0].x)

        city\_y.append(best\_route[0].y)

        plt.scatter(city\_x, city\_y, c='r', marker='\*', s=200, alpha=0.5)

        plt.plot(city\_x, city\_y, "b", ms=20)

        plt.pause(0.1)

    plt.ioff()

    plt.show()

    print("final distance:{}".format(1.0 / (rank(population)[0][1])))

    bestRouteIndex = rank(population)[0][0]

    bestRoute = population[bestRouteIndex]

    return bestRoute

def GA(city\_list, popSize, eliteSize, mutationRate, generations):

    population = init\_pop(city\_list, popSize)  # 初始化种群

    process = []

    print("initial distance:{}".format(1.0 / (rank(population)[0][1])))

    for i in range(generations):

        population = next\_pop(population, eliteSize, mutationRate)  # 产生下一代种群

        process.append(1.0 / (rank(population)[0][1]))

    plt.figure(1)

    print("final distance:{}".format(1.0 / (rank(population)[0][1])))

    plt.plot(process)

    plt.ylabel('Distance')

    plt.xlabel('Generation')

    plt.savefig(str(generations) + '\_' + str(1.0 / (rank(population)[0][1])) + '\_' + str(mutationRate) + '\_process.jpg')

    plt.figure(2)

    idx\_rank\_pop = rank(population)[0][0]

    best\_route = population[idx\_rank\_pop]

    print(best\_route)

    city\_x = []

    city\_y = []

    for j in range(len(best\_route)):

        city = best\_route[j]

        city\_x.append(city.x)

        city\_y.append(city.y)

    city\_x.append(best\_route[0].x)

    city\_y.append(best\_route[0].y)

    plt.scatter(city\_x, city\_y, c='r', marker='\*', s=200, alpha=0.5)

    plt.plot(city\_x, city\_y, "b", ms=20)

    for i in range(len(coordinates)):

        plt.text(coordinates[i][0], coordinates[i][1], r'  ' + str(i + 1))

    plt.savefig(str(generations) + '\_' + str(mutationRate) + '\_route.jpg')

    plt.show()

num\_city = 15

city\_list = []

coordinates = np.array([[-0.0000000400893815, 0.0000000358808126],

                        [-28.8732862244731230, -0.0000008724121069],

                        [-79.2915791686897506, 21.4033307581457670],

                        [-14.6577381710829471, 43.3895496964974043],

                        [-64.7472605264735108, -21.8981713360336698],

                        [-29.0584693142401171, 43.2167287683090606],

                        [-72.0785319657452987, -0.1815834632498404],

                        [-36.0366489745023770, 21.6135482886620949],

                        [-50.4808382862985496, -7.3744722432402208],

                        [-50.5859026832315024, 21.5881966132975371],

                        [-0.1358203773809326, 28.7292896751977480],

                        [-65.0865638413727368, 36.0624693073746769],

                        [-21.4983260706612533, -7.3194159498090388],

                        [-57.5687244704708050, 43.2505562436354225],

                        [-43.0700258454450875, -14.5548396888330487]])

for i in range(0, num\_city):

    city\_list.append(City(i + 1, coordinates[i][0], coordinates[i][1]))

GA(city\_list, 100, 20, 0.01, 500)