**《最优化技术》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级计算机科学与技术（卓越）02班** | | | **姓名** | **李燕琴** |
| **实验题目** | 遗传算法的应用 | | | | | |
| **实验时间** | **2021年5月28日** | | **实验地点** | **DS3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 ■设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  理解遗传算法的基本思想，并应用于求解实际问题。 | | | | | | |
| 二、实验项目内容  请利用遗传算法解决以下的旅行商问题：  一个旅行者需要到国内的10个城市旅行，各城市的坐标见cities.csv文档。请设计一个合理的线路使旅行者所行的路程之和最小。注意：每个城市只能访问一次，且最后要回到原来出来的城市。  要求：输出适应度函数的进化曲线和最终选择的线路图。  注意：所有程序请用python语言实现。只提交本电子文档，注意本文件末尾的文件命名要求；源程序一节请用代码备注的方式说明你的算法和思路；实验结果一节需要提供测试结果截图并给出结果分析。 | | | | | | |
| 三、实验过程或算法（源程序）  import os  from math import sqrt  from numpy.lib.function\_base import select  import pandas as pd  import matplotlib.pyplot as plt  import numpy as np  import copy  import itertools  # 前往工作路径  os.chdir(r'D:\2021study\1.课程学习\03-最优化技术\Hw\_Ex/')  df = pd.read\_csv('cities.csv',encoding='utf-8')  col = df.columns  city\_x = df[col[1]]  city\_y = df[col[2]]  city\_n = len(city\_x)  # plt.plot(city\_x,city\_y,"\*")  # 参数 PPT中的A3-B2-C1-D3的参数设置  N = 500 # 初始化种群数目  iter\_n = 100 # 迭代次数  select\_operator = 0.2  # 选择算子  cross\_rate = 0.4  # 交叉算子  mutation\_rate = 0.1 # 变异算子  all\_cnt = []  # 迭代次数  all\_dis = []  # 每次迭代的最优适应度  # ==================== 适应度函数 =====================  def distance(citylist,n):      ''' 当前路径下的总距离 '''      total\_dis = 0      for i in range(n):          curr\_city = int(citylist[i])          next\_city = int(citylist[(i+1)%n])          total\_dis += dis[curr\_city][next\_city]      return total\_dis  def fitness(citylist,n):      ''' 适应度计算 '''      return 1/distance(citylist,n)  # ================== 选择操作 ========================  # 将群体中适应度较大的C个个体直接替换适应度较小的C个个体。  def select(city\_mat,mat\_n):      ''' 选择函数，将群体中适应度较大的C个个体直接替换适应度较小的C个个体。 '''      c = int(select\_operator\*mat\_n)      new\_city\_mat = sorted(city\_mat,key=lambda x:x[-1])      for i in range(c):          new\_city\_mat[i] = copy.copy(new\_city\_mat[mat\_n-i-1])      return new\_city\_mat  # ================== 交叉 ========================  def cross(f\_father,f\_mother):      ''' 交叉 '''      # 输入的可能是一个float类型的list，故需要转换为Int的list      father = np.array(f\_father,dtype=np.int).tolist()      mother = np.array(f\_mother,dtype=np.int).tolist()      # print(type(father))        # 交叉点的选择      city\_n = len(father)      point1 = np.random.choice(range(city\_n))      point2 = np.random.choice(range(city\_n))      if point1>point2:          point1,point2 = point2,point1        # 子序列重建      son1 = copy.copy(father[0:point1])      son1.extend(mother[point1:point2])      son1.extend(copy.copy(father[point2:city\_n]))      son2 = copy.copy(mother[0:point1])      son2.extend(father[point1:point2])      son2.extend(copy.copy(mother[point2:city\_n]))      # 交叉结果（子序列）去重      num\_idx1 = np.zeros(city\_n)  # 记录子序列idx个数      num\_idx2 = np.zeros(city\_n)      for i in range(city\_n):          num\_idx1[son1[i]] += 1          num\_idx2[son2[i]] += 1      for i in range(city\_n):          if(num\_idx1[i]==0):  # 子序列元素替换              for j in range(city\_n):                  if(num\_idx1[son1[j]]>1):                      num\_idx1[son1[j]] -= 1                      son1[j] = i                      num\_idx1[i] += 1                      break          if(num\_idx2[i]==0):              for j in range(city\_n):                  if(num\_idx2[son2[j]]>1):                      num\_idx2[son2[j]] -= 1                      son2[j] = i                      num\_idx2[i] += 1                      break      # DEBUG      # print(f"son1 = {son1}\nson2 = {son2}")      return son1,son2  # ================== 变异 ========================  def mutation(citylist,n):      ''' 变异 '''      # 变异点选择      point1 = np.random.choice(range(city\_n))      point2 = np.random.choice(range(city\_n))      if point1>point2:          point1,point2 = point2,point1        # 变异序列重建      res = copy.copy(citylist[0:point1])      res.extend(list(reversed(citylist[point1:point2])))      res.extend(citylist[point2:n])      return res  if \_\_name\_\_=='\_\_main\_\_':      # ================== 计算距离矩阵 ========================      dis = np.zeros((city\_n,city\_n))      for i in range(0,city\_n):          for j in range(i+1,city\_n):              dis[i][j] = sqrt(pow(city\_x[i]-city\_x[j],2)+pow(city\_y[i]-city\_y[j],2))              dis[j][i] = dis[i][j]        # ================== 初始化种群 ========================      cnt = 0      init\_group = []      nums = range(city\_n)      # done\_XXX 这样是顺序产生的，不晓得影不影响，可以用random打乱      # 已验证不影响      for num in itertools.permutations(nums):          init\_group.append(num)          cnt+=1          if cnt>=N:              break      cnt = 0  # 迭代次数记录      city\_mat = np.zeros((N,city\_n+1))        # 动态图构建      # plt.ion()      # ion\_x = [0,120]      # ion\_y = [0,120]      while(cnt<=iter\_n):          # 计算适应度得分，并加载city\_mat后面          for i in range(N):              city\_mat[i][0:city\_n] = copy.copy(init\_group[i][0:city\_n])              city\_mat[i][city\_n] = fitness(init\_group[i],city\_n)          fit = np.array(city\_mat)[:,city\_n]          # 最优结果保存          ion\_dis = 1/max(fit)          all\_dis.append(ion\_dis)          all\_cnt.append(cnt)          # 绘制动态图          # ion\_x[0] = ion\_x[1]          # ion\_y[0] = ion\_y[1]          # ion\_x[1] = cnt          # plt.plot(ion\_x, ion\_y)          # plt.pause(0.1)          # if cnt==1:          #     ion\_y[0] = ion\_dis          # ion\_y[1] = ion\_dis            # DEBUG          print("迭代%4d次，最优结果为%.3f"%(cnt,ion\_dis))          # =============== 选择 =================          new\_city\_mat = select(city\_mat,N);  # list          # DEBUG          # print(f"选择:{new\_city\_mat[0][0:city\_n]}")          # =============== 交叉 =================          half\_N = int(N/2)          for i in range(half\_N):              if np.random.random()<cross\_rate:                  father = new\_city\_mat[i][0:city\_n].tolist()  # 单独取出来是ndarray，需要tolist()                  mother = new\_city\_mat[i+half\_N-1][0:city\_n].tolist()                  son1,son2 = cross(father,mother)                  new\_city\_mat[i][0:city\_n] = son1                  new\_city\_mat[i+half\_N-1][0:city\_n] = son2          # DEBUG          # print(f"交叉:{new\_city\_mat[0][0:city\_n]}")          # =============== 变异 =================          for i in range(N):              if np.random.random()<mutation\_rate:                  target = new\_city\_mat[i][0:city\_n].tolist()                  new\_city\_mat[i][0:city\_n] = mutation(target,city\_n)          # DEBUG          # print(f"变异:{new\_city\_mat[0][0:city\_n]}")          temp = np.array(new\_city\_mat).astype(np.uint8)          init\_group = temp[:,0:city\_n]          cnt += 1        # =============== 绘图 =================      plt.figure(1)      path\_x = []      path\_y = []      for i in range(len(city\_mat[0])-1):          path\_x.append(city\_x[city\_mat[0][i]])          path\_y.append(city\_y[city\_mat[0][i]])      # 回到起点      path\_x.append(city\_x[city\_mat[0][0]])      path\_y.append(city\_y[city\_mat[0][0]])      plot = plt.plot(path\_x, path\_y,'r\*-')      plt.xlabel('X axis')      plt.ylabel('Y axis')      plt.title("Route")      plt.show()        plt.figure(2)      plot = plt.plot(all\_cnt,all\_dis)      plt.xlabel('Generation')      plt.ylabel('Shortest Length')      plt.title("Fitness Evolution")      plt.show() | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  输出适应度函数的进化曲线和最终选择的线路图。  根据遗传算法原理，最初始可行结果种群进行选择、变异、交叉等操作，其中参数设置如图 1，得到最终适应度函数的进化曲线，如图 2，其中可以发现，遗传算法收敛速度很快，当有10个城市的时候，迭代二十多次即可收敛到最优解。其中最优路线图如图 3，即：  最优总距离：109.988  最优线路图: 1-->0-->7-->6-->5-->4-->3-->8-->2-->9  城市路线图: 天津-->北京-->呼和浩特-->银川-->乌鲁木齐-->拉萨-->重庆-->南宁-->上海-->哈尔滨    图 1参数设置    图 2 适应度函数的进化曲线    图 3 最优线路图 | | | | | | |

注：电子文档命名要求：学号+姓名+实验序号