## **广州大学学生实验报告**

**开课学院及实验室：**计算机科学与工程实验室 **2021年11月4日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | **计算机科学与网络工程学院** | **年级/专业/班** | **网络193** | **姓名** | 吴伟俊 | **学号** | 1906200107 |
| **实验课程名称** | **人工智能导论实验** | | | | | **成绩** |  |
| **实验项目名称** | **TSP问题的遗传算法实现** | | | | | **指导老师** | 张少宏 |

(\*\*\*报告只能为文字和图片,老师评语将添加到此处,学生请勿作答\*\*\*)

**一、实验内容**

**问题描述**：旅行商问题，即TSP问题（Traveling Salesman Problem）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。

**内容提要**：以N个节点的TSP（旅行 商问题）问题为例，应用遗传算法并用选定的编程语言，设计简单的遗传优化系统对问题进行求解，求出问题的最优解，通过实验培养学生利用遗传算法进行问题求解的基本技能。

**二、实验设备**

1. 实验设备：计算机；

2. 平台：Windows操作系统，vscode

**三、实验步骤**

* 随机生成N个二维坐标节点。

先进行初始化，随机生成矩阵mat，设置10个城市，500个群体数目

文本

描述已自动生成

* 应用遗传算法并用选定的编程语言，设计简单的遗传优化系统对问题进行求解，求出问题的最优解。

图表, 箱线图

描述已自动生成

* 分析适应度函数对启发式搜索算法的影响。

目标函数g = \sum\sqrt((x\_u-x\_v)^2+(y\_u-y\_v)^2)

我们要求g\_min,因此我们的适应度函数f应和g呈负相关,则：f\*g=1

f=1/g

为了减少进度误差，我们将根号去掉，因此f=1/(\sum(x\_u-x\_v)^2+(y\_u-y\_v)^2)

去掉根号之后的收敛速度更快

文本

描述已自动生成

* \*扩展选做题：考虑不同数值N对最终结果和求解性能的影响。对于比较大的N，是否设计更快速的近似方法代替原有算法。

**四、分析说明**

* 实验结果：

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

最优解以及相应的遗传代数的关系：

图表, 折线图

描述已自动生成

图片包含 文本

描述已自动生成

通过分析图不难知道，每一代的解处于抖动状态，优秀的基因（码）被传下去，因此总体一直再找到更优解

**五、实验总结：**

本次实验，我通过把TSP问题转换成适合遗传算法的数学模型，将遗传算法运用到求解TSP问题的较优解的过程中.一方面使我更加了解并熟悉遗传算法的原理和运用，另一方面体会到合适的数学模型的重要性，比如根据目标函数构建合理的适应度函数，以及不同适应度函数的收敛速度不一样。

**函数源代码如下：**

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

def dis(x, y, p1, p2):

    p1 = int(p1)

    p2 = int(p2)

    return (x[p1] - x[p2]) \*\* 2 + (y[p1] - y[p2]) \*\* 2

def f(F, N, n, x, y):

    # 适应度函数

    for x1 in range(N):

        square\_sum = 0

        for x2 in range(n):

            square\_sum += dis(x, y, F[x1][x2], F[x1][x2 + 1])

        F[x1][-1] = round(1 / np.sqrt(square\_sum), 7)

def matching(mat, pc, n, pm, num):

    match\_matrix = np.array(1 - (np.random.rand(num) > pc), dtype = np.bool\_)

    a = list(mat[match\_matrix][:, :-1])  # 进行交配的个体

    b = list(mat[np.array(1 - match\_matrix, dtype=bool)][:, :-1])  # 未进行交配的个体,直接放到下一代

    b = [list(i) for i in b]

    if len(a) % 2 != 0:

        b.append(a.pop())

    for i in range(int(len(a) / 2)):

        p1 = np.random.randint(1, int(n / 2) + 1)

        p2 = np.random.randint(int(n / 2) + 1, n)

        x1 = list(a.pop())

        x2 = list(a.pop())

        x1[:p1], x2[:p1] = x2[:p1], x1[:p1]

        x1[:p2], x2[:p2] = x2[:p2], x1[:p2]

        c1 = x1[p1:p2]

        c2 = x2[p1:p2]

        while True:

            for i in x1[:p1]:

                if i in c1:

                    x1[x1[:p1].index(i)] = c2[c1.index(i)]

                    break

            if np.intersect1d(x1[:p1], c1).size == 0:

                break

        while True:

            for i in x1[p2:]:

                if i in c1:

                    x1[x1[p2:].index(i) + p2] = c2[c1.index(i)]

                    break

            if np.intersect1d(x1[p2:], c1).size == 0:

                break

        while True:

            for i in x2[:p1]:

                if i in c2:

                    x2[x2[:p1].index(i)] = c1[c2.index(i)]

                    break

            if np.intersect1d(x2[:p1], c2).size == 0:

                break

        while True:

            for i in x2[p2:]:

                if i in c2:

                    x2[x2[p2:].index(i) + p2] = c1[c2.index(i)]

                    break

            if np.intersect1d(x2[p2:], c2).size == 0:  # 如果不存在交集，则循环结束

                break

        # 交配之后产生的个体进行一定概率上的变异

        variation(x1, pm, n)

        variation(x2, pm, n)

        b.append(x1)

        b.append(x2)

    return np.column\_stack((b, np.zeros((num, 1))))

def variation(list\_a, pm, n):

    '''变异函数'''

    if np.random.rand() < pm:

        p1 = np.random.randint(1, int(n / 2) + 1)

        p2 = np.random.randint(int(n / 2) + 1, n)

        #         print(p1,p2)

        temp = list\_a[p1:p2]

        temp.reverse()

        list\_a[p1:p2] = temp

if \_\_name\_\_ == '\_\_main\_\_':

    gen = []  # 代数

    dist = []  # 每一代的最优距离

    # 初始化

    F = []  #存放访问顺序和每个个体适应度

    N = 500  #初始化群体的数目

    n = 10  #城市数

    pc = 0.9  #交配概率

    pm = 0.2  #变异概率

    pos\_x = np.random.randint(0, 100, size = n)

    pos\_y = np.random.randint(0, 100, size = n)

    x = np.append(pos\_x, pos\_x[0])

    y = np.append(pos\_y, pos\_y[0])

    for i in range(N):

        F.append(np.random.permutation(np.arange(0, n)))

    zero = np.zeros((N, 1))

    F = np.column\_stack((F, zero))  # 矩阵的拼接

    f(F, N, n, x, y)

    for i in range(200):

        a, b = np.argmin(F[:, -1]), F[np.argmax(F[:, -1])]

        if (i + 1) % 10 == 0:

            cur\_dis = 0

            for v in F:

                cur\_dis += 1 / v[-1]

            print("当前实验最优解：", cur\_dis)

            gen.append(i+1)

            dist.append(cur\_dis)

            index = np.array(b[:-1], dtype=np.int32)

            x = np.append(pos\_x[index],pos\_x[[index[0]]])

            y = np.append(pos\_y[index],pos\_y[[index[0]]])

            fig = plt.figure()

            plt.plot(x,y,'-o')

            plt.xlabel('x',fontsize = 16)

            plt.ylabel('y',fontsize = 16)

            plt.title('{iter}'.format(iter=i+1) + 'th gen')

            plt.show()

        mat\_temp = matching(F, pc, n, pm, N)  # 交配变异

        f(F, N, n, x, y)

        c, d = np.argmin(F[:, -1]), F[np.argmax(F[:, -1])]

        F[c] = b

    # 绘制进化曲线

    plt.plot(gen, dist, '-r')

    plt.show()