**使用仿生类的智能搜素算法实现旅行商问题**

**实验报告**

|  |  |
| --- | --- |
| **班级** | **9191062302** |
| **学号** |  |
| **姓名** |  |

目录

[1 实验目的 3](#_Toc125931863)

[2 实验内容 3](#_Toc125931864)

[3 实验代码 3](#_Toc125931865)

[4 实验结果 6](#_Toc125931866)

# 实验目的

通过实验让学生掌握以遗传进化为代表的仿生类随机优化搜索算法，并根据实际问题灵活应用这些算法。

# 实验内容

随机生成100个结点（或更多结点）的图，在任意两结点之间赋予一条边，然后在这些边上赋予随机代价。最后从遗传算法、蚁群算法、粒子群算法、鱼群算法（上课介绍过），以及蛙跳算法、萤火虫算法，蝙蝠算法（课上没有介绍过）等仿生类算法中选择一个算法在这个图上求解从一个结点开始，经过且只经过每个结点一次，最后返回起点的最短路径的问题。

# 实验代码

*import* random

*import* numpy *as* np  
*def* get\_matrix(num):  
 dist\_matrix = np.zeros((num, num))  
 *for* i *in* range(num - 1):  
 *for* j *in* range(i + 1, num):  
 dist = random.randint(100, 500)  
 dist\_matrix[i, j] = dist  
 dist\_matrix[j, i] = dist  
 *return* dist\_matrix  
  
  
*# 计算某一路线的适应度  
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]]  
 dist\_sum += dist\_matrix[route[len(route) - 1], route[0]]  
 *return* 1 / dist\_sum  
  
  
*# 计算所有路线的适应度  
def* get\_all\_routes\_fitness\_value(routes, dist\_matrix):  
 fitness\_values = np.zeros(len(routes))  
 *for* i *in* range(len(routes)):  
 f\_value = get\_route\_fitness\_value(routes[i], dist\_matrix)  
 fitness\_values[i] = f\_value  
 *return* fitness\_values  
  
  
*# 随机初始化路线  
def* init\_route(n\_route, num):  
 routes = np.zeros((n\_route, num)).astype(int)  
 *for* i *in* range(n\_route):  
 routes[i] = np.random.choice(range(num), size=num, replace=*False*)  
 *return* routes  
  
  
*# 选择  
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)  
 selected\_routes[i] = routes[choice]  
 *return* selected\_routes  
  
  
*# 交叉  
def* crossover(routes, num):  
 *for* i *in* range(0, len(routes), 2):  
 r1\_new, r2\_new = np.zeros(num), np.zeros(num)  
 seg\_point = np.random.randint(0, num)  
 cross\_len = num - seg\_point  
 r1, r2 = routes[i], routes[i + 1]  
 r1\_cross, r2\_cross = r2[seg\_point:], r1[seg\_point:]  
 r1\_non\_cross = r1[np.in1d(r1, r1\_cross, invert=*True*)]  
 r2\_non\_cross = r2[np.in1d(r2, r2\_cross, invert=*True*)]  
 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  
  
  
*# 变异  
def* mutation(routes, num):  
 prob = 0.01 *# 变异概率为 0.01* p\_rand = np.random.rand(len(routes))  
 *for* i *in* range(len(routes)):  
 *if* p\_rand[i] < prob:  
 mut\_position = np.random.choice(range(num), size=2, replace=*False*)  
 l, r = mut\_position[0], mut\_position[1]  
 routes[i, l], routes[i, r] = routes[i, r], routes[i, l]  
 *return* routes  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 num = 100 *# 节点个数* n = 106 *# 种群大小* epoch = 5000 *# 迭代次数* dist\_matrix\_ = get\_matrix(num) *# 随机生成节点距离矩阵* routes\_ = init\_route(n, dist\_matrix\_.shape[0]) *# 初始化所有路线* fitness\_values\_ = get\_all\_routes\_fitness\_value(routes\_, dist\_matrix\_) *# 计算所有初始路线的适应度* best\_index = fitness\_values\_.argmax()  
 best\_route, best\_fitness = routes\_[best\_index], fitness\_values\_[best\_index] *# 保存最短路线及其适应度  
  
 # 开始迭代* not\_improve\_time = 0  
 *for* i\_ *in* range(epoch):  
 routes\_ = selection(routes\_, fitness\_values\_) *# 选择* routes\_ = crossover(routes\_, num) *# 交叉* routes\_ = mutation(routes\_, num) *# 变异* fitness\_values\_ = get\_all\_routes\_fitness\_value(routes\_, dist\_matrix\_)  
 best\_route\_index = fitness\_values\_.argmax()  
 best\_route, best\_fitness = routes\_[best\_route\_index], fitness\_values\_[best\_route\_index] *# 保存最短路线及其适应度  
 if* (i\_ + 1) % 500 == 0:  
 print('迭代次数: {}, 当前最短路线距离： {}'.format(i\_ + 1, 1 / get\_route\_fitness\_value(best\_route, dist\_matrix\_)))  
  
 print('最短路线为：')  
 print(best\_route)  
 print('总距离为： {}'.format(1 / get\_route\_fitness\_value(best\_route, dist\_matrix\_)))

# 实验结果

选择遗传算法进行求解

