中山大学计算机学院

人工智能

本科生实验报告

(2022学年春季学期)

课程名称：Artificial Intelligence

教学班级	20 级计科一班	专业（方向）	计算机科学与技术
学号	20337056	姓名	李昊伟

一 . 实验题目

在TSPLIB中选一个大于100个城市数的TSP问题，使用模拟退火和遗传算法求解。

二 . 实验内容

1. 算法原理

模拟退火算法：

模拟退火算法是一种加强版的爬山算法，旨在通过按照一定的概率接受差解来摆脱山肩和山谷。模拟物理上退火的过程时，温度按照指数的速度下降(快速退火)，而接受差解的概率，由温度和差解与当前最优的距离决定，温度越高、距离越小，该差解越容易被接受。

在算法进行的初期，接受差解的概率大，对于局部搜索来讲，有效避免陷入局部极小。在算法进行的后期，温度降低，接受差解的概率小，近似于暴力搜索。

遗传算法：

遗传算法模拟了优胜劣汰、适者生存的生物规律。由随机初始化的种群开始优化：首先，将最优秀的一批直接移植到下一代种群。接下来，通过轮盘赌的方式生成后代，轮盘赌保证优秀的个体更容易被选中产生后代，但也不绝对，没被选到下一代的个体其实也可以产生后代进入下一代（这一点我一开始没想通）。crossover的过程产生后代，使得后代获得父母的"优良基因"，突变保证种群可以得到优化。（一开始，由于代码错误，我的突变实际上没起作用，所以导致种群根本优化不动，基本上卡在很低的得分，也算让我体会了一把突变的作用）。

2 .关键代码展示

一.模拟退火

```
    def get_new_state(self):
        new_x = self.coordinate_x.copy()
        new_y = self.coordinate_y.copy()
        if np.random.rand() > 0.5:
            while True:#产生两个不同的随机数
                loc1 = np.int(np.ceil(np.random.rand()*(self.problem_size-1)))
                loc2 = np.int(np.ceil(np.random.rand()*(self.problem_size-1)))
                if loc1 != loc2:
                    break
            new_x[loc1],new_x[loc2] = new_x[loc2],new_x[loc1]
            new_y[loc1],new_y[loc2] = new_y[loc2],new_y[loc1]
        else:
            while True:
                loc1 = np.int(np.ceil(np.random.rand()*(self.problem_size-1)))
                loc2 = np.int(np.ceil(np.random.rand()*(self.problem_size-1)))
                loc3 = np.int(np.ceil(np.random.rand()*(self.problem_size-1)))

                if((loc1 != loc2)&(loc2 != loc3)&(loc1 != loc3)):
                    break

            # 下面的三个判断语句使得loc1<loc2<loc3
            if loc1 > loc2:
                loc1,loc2 = loc2,loc1
            if loc2 > loc3:
                loc2,loc3 = loc3,loc2
            if loc1 > loc2:
                loc1,loc2 = loc2,loc1

            #下面的三行代码将[loc1,loc2)区间的数据插入到loc3之后
            tmplist = new_x[loc1:loc2].copy()
            new_x[loc1:loc3-loc2+1+loc1] = new_x[loc2:loc3+1].copy()
            new_x[loc3-loc2+1+loc1:loc3+1] = tmplist.copy()

            tmplist = new_y[loc1:loc2].copy()
            new_y[loc1:loc3-loc2+1+loc1] = new_y[loc2:loc3+1].copy()
            new_y[loc3-loc2+1+loc1:loc3+1] = tmplist.copy()


        new_path_count = 0
        for i in range(self.problem_size - 1):
            new_path_count += self.get_distance(i , i+1 , new_x , new_y)

        return new_path_count,new_x,new_y
```

```
    def solve(self):
        self.show1(title = "start", k = 0)
        x.append(0)
        y.append(self.path_count)
        while self.temperature >0.05:
            self.temperature = self.max_temperature / (1 + self.k)
            if self.k % 100 == 0:
                x.append(self.k)
                y.append(self.path_count)
            if self.k % 500 == 0:
                title  = "temperature = " + str(self.temperature) + " score = "
+ str(int(self.path_count))
                self.show1(title,self.k)
            for i in range(100):
                print("使用模拟退火 , 温度为" , self.temperature  , ", 第 " , i , "
轮.")
                print(self.path_count)
                new_cost,new_x,new_y = self.get_new_state()
                if new_cost < self.path_count:
                    self.accept(new_cost,new_x,new_y)
                else:
                    if np.random.rand() < np.exp(-(new_cost-
self.path_count)/self.temperature):
                        self.accept(new_cost,new_x,new_y)
            self.k = self.k + 1
        self.show2()
```

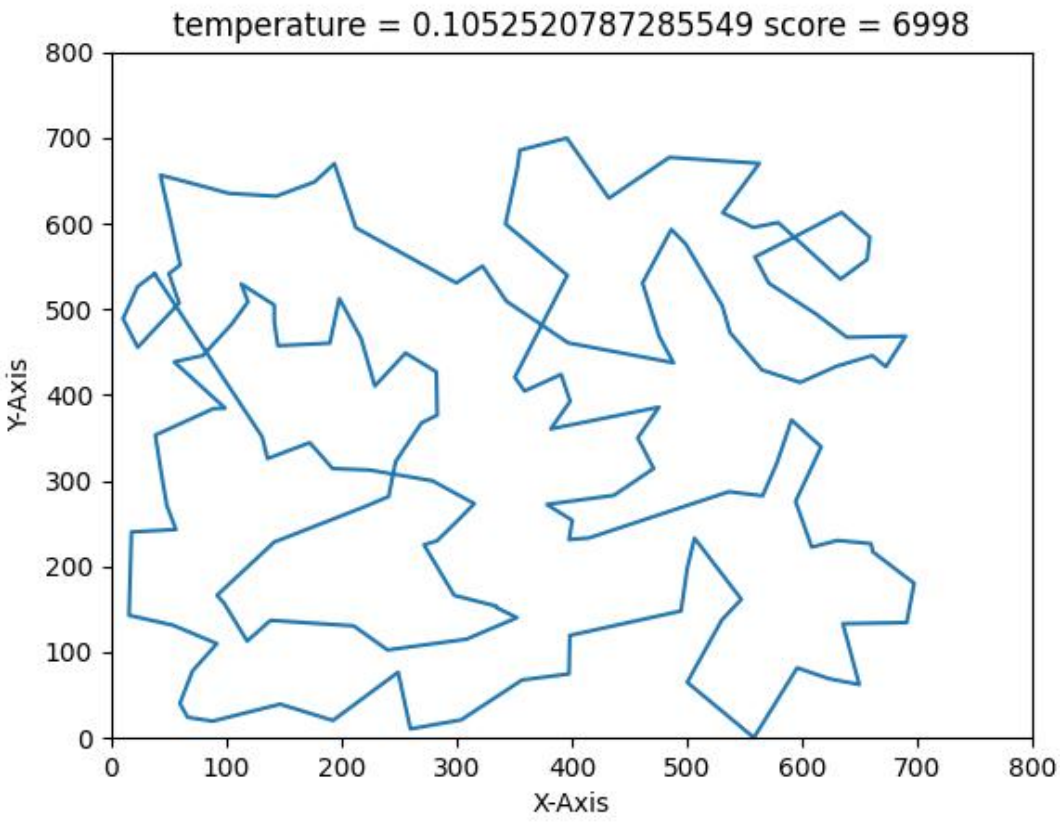
二.遗传算法

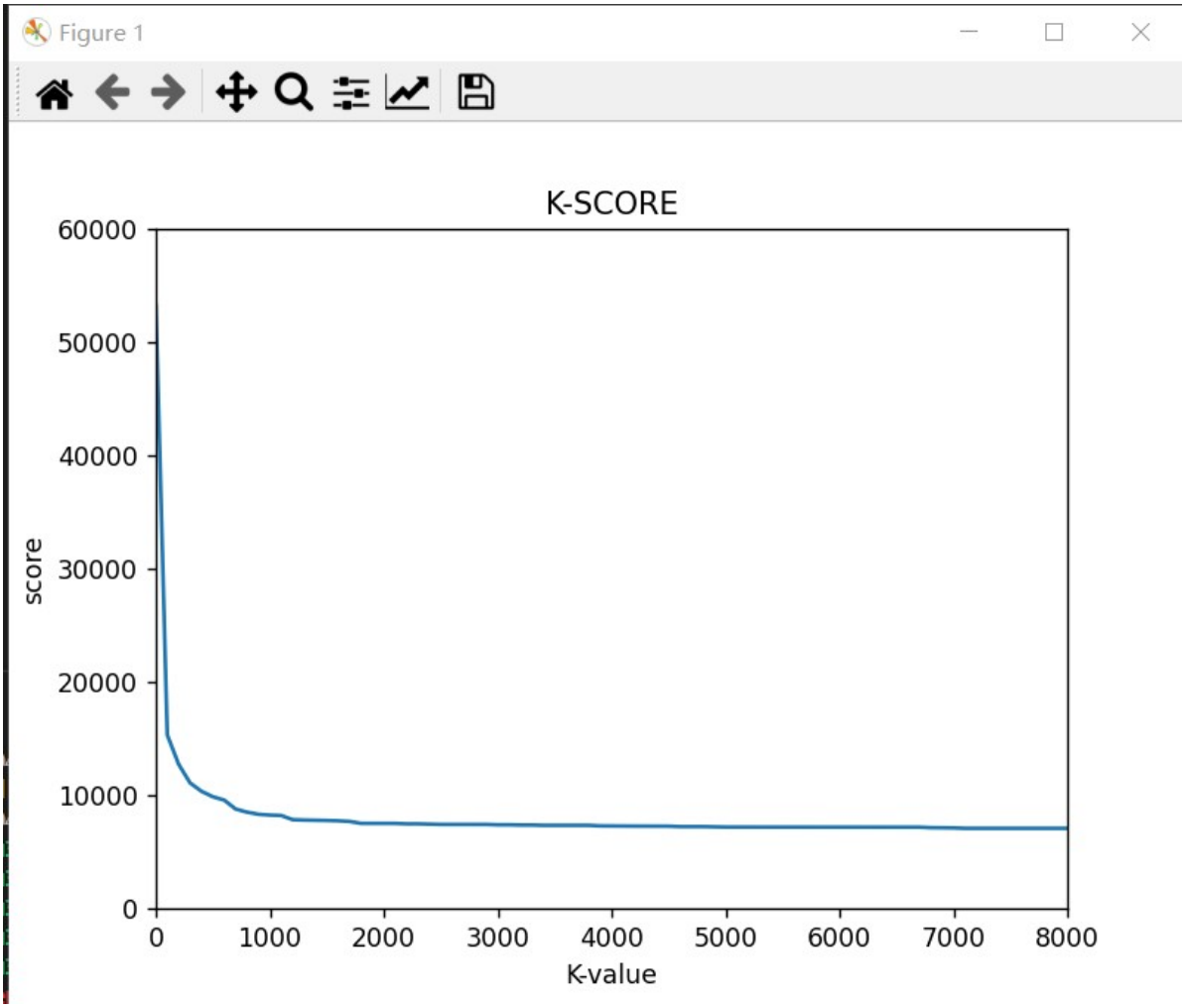
```
    def get_children(self):
        parent1,parent2 = self.get_two_parents()
        new_child1,new_child2 = self.crossover2(parent1,parent2)
        new_child1.mutation(0.25)
        new_child2.mutation(0.25)
        return new_child1,new_child2

    def crossover2(self,parent1,parent2):
        ord1 = random.randint(0, self.problem_size)
        ord2 = random.randint(0, self.problem_size)
        son1 = self.ox(parent1, parent2, ord1, ord2)
        son2 = self.ox(parent1, parent2, ord2, ord1)
        return son1,son2

    def ox(self,parent1, parent2, ord1, ord2):  # 顺序交叉
        son1 = self.population[parent1].state[ord1:ord2]
        son2 = []
        for gene in  self.population[parent2].state:
            if gene not in son1:
                son2.append(gene)  # 继承父代2
        for i in range(len(son1)):  # 插入
            son2.insert(ord1, son1[len(son1) - i - 1])
        return Individual(son2,self.problem_size)

    def get_next_genaration(self):
        new_population = []
        new_population_size = 0
```

```
        while(new_population_size < self.max_population*0.3):
            new_population.append(self.population[new_population_size])
            new_population_size += 1
        while(len(new_population) < self.max_population):
            new_c1,new_c2 = self.get_children()
            if new_c1 not in new_population:
                new_population.append(new_c1)
            if new_c2 not in new_population:
                new_population.append(new_c2)
        tmp = []
        for v in new_population:
            heapq.heappush(tmp,v)
        new_population = [heapq.heappop(tmp) for i in range(len(tmp))]
        self.population = new_population
```

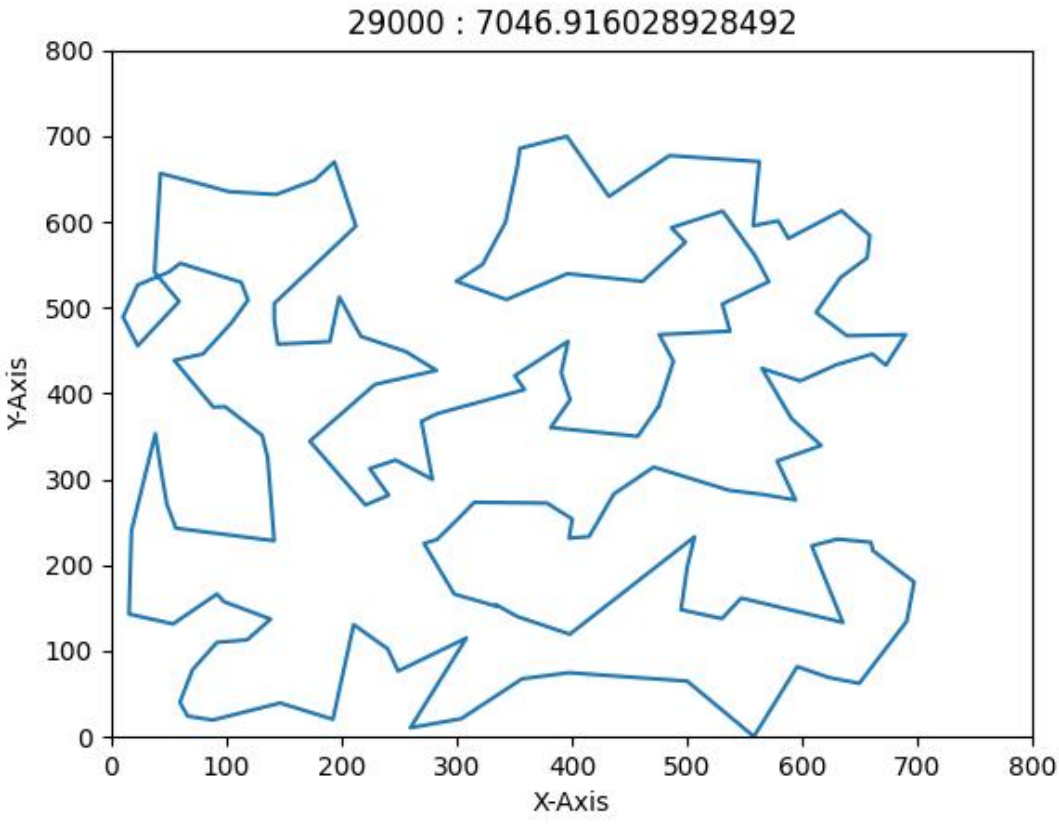
3.实验结果：

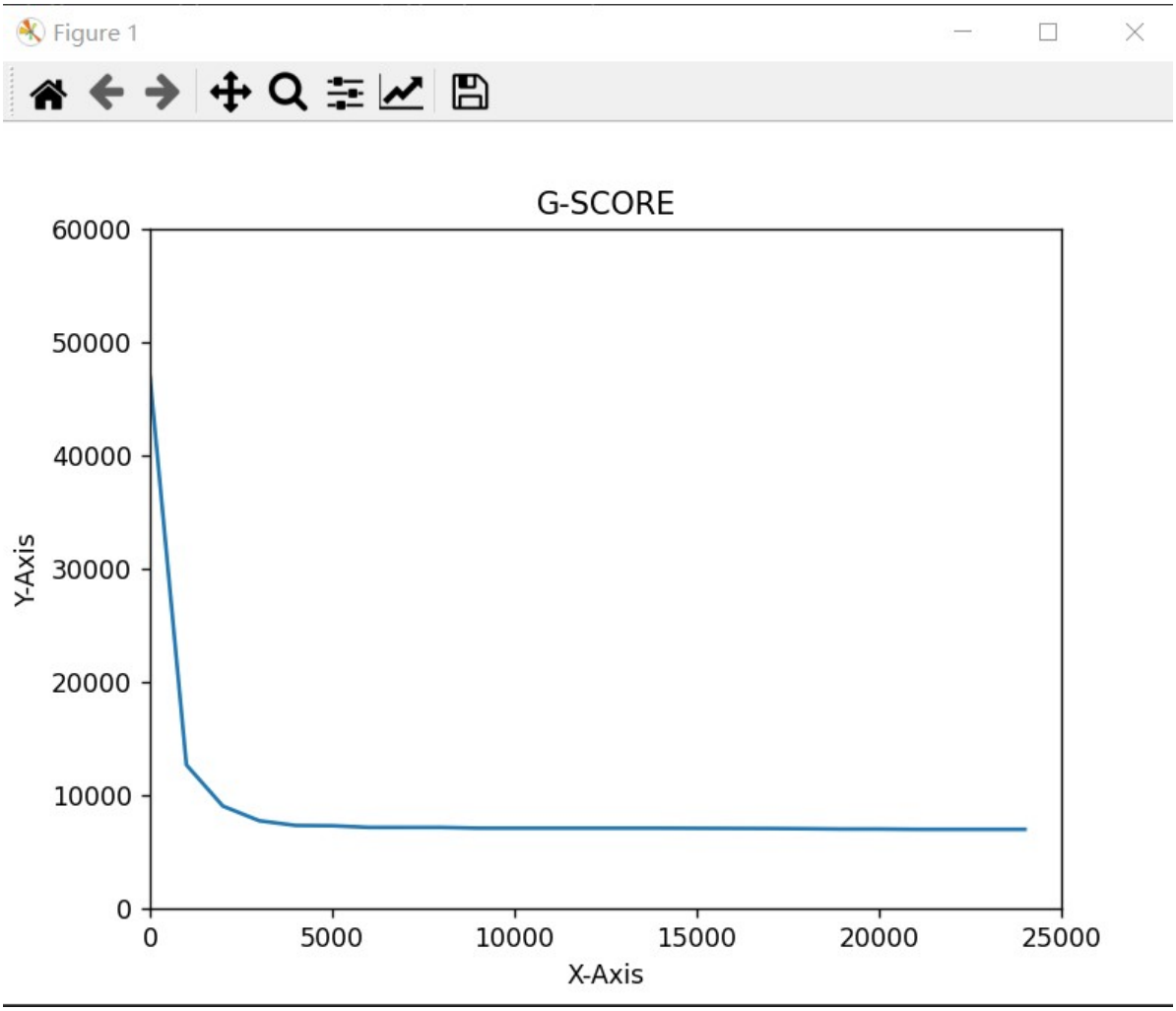
1.模拟退火: (ch150: 6998/6528 停止温度：0.1度) ：





2.遗传算法：(ch150: 7046/6528 29000代) ：





4. 创新点&优化

1. 模拟退火：采用两交换和三交换，产生新的搜索结点。

2. 遗传算法：尝试网络上各种变异方法(见mutation.py)， 最后选择滑动变异的方法。

3. 代码风格上，参考TA的代码，采用全类封装，可读性更强。

5.实验感想

这次实验我感觉和15-puzzle一样，又是收获非常大的一次作业。早就听说咱们学院的人工智能课程很硬核，几乎几周一个小项目，现在来看果真名不虚传。不仅大大地巩固了课上学过的内容，而且对代码能力非常有提高。同时，我也不免赞叹，这些前辈们是怎样聪明，居然把物理和生物的现象融合到算法里面。如今连接主义的人工智能还是比较强势，但我觉得把人类的智慧传递给机器，和想着如何去让模型"学习"，前者还是更酷一些。