中山大学计算机学院

人工智能

本科生实验报告

课程名称：Artificial Intelligence

学号	22336327	姓名	庄云皓

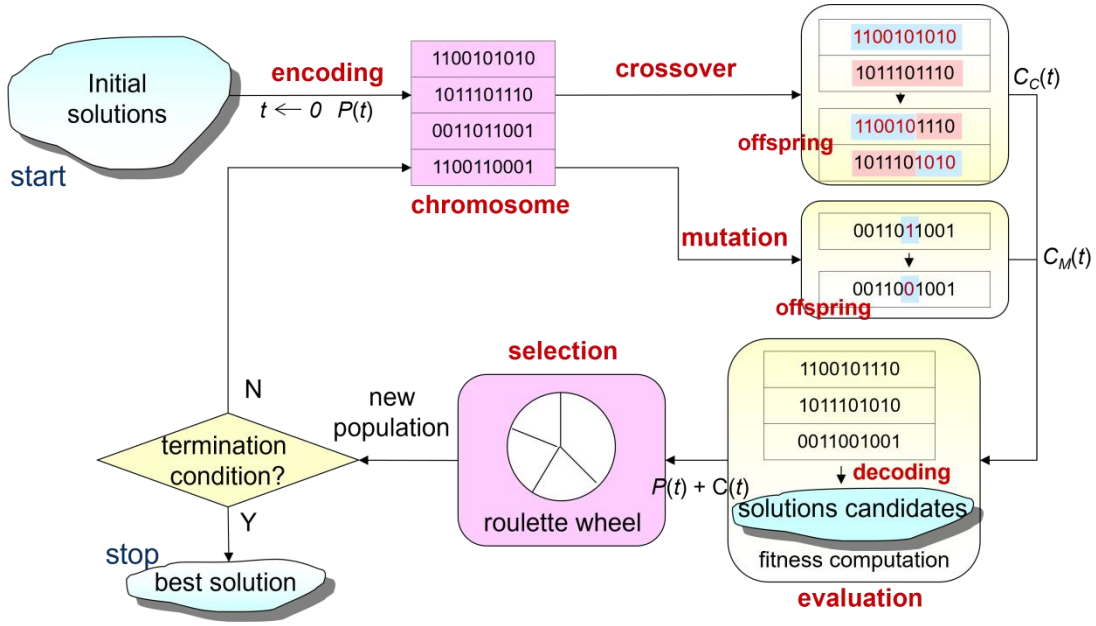
一、 实验题目

用遗传算法求解 TSP 问题

二、 实验内容

1. 算法原理

1.1 遗传算法



1.2 旅行商问题，即 TSP 问题（Traveling Salesman Problem）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，路经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。TSP 问题是一个组合优化问题。该问题可以被证明具有 NPC 计算复杂性。

遗传算法求解 tsp 流程：

算法 遗传算法解决 TSP

输入: n 个城市的坐标

随机生成 10 个不同个体的初始种群 $P(0)$
for $t = 0$ **to** 最大迭代轮数 **do**
  初始化后代种群 $C(t)$
  for $i = 1$ **to** 每一代后代数目 **do**
    依据适应度, 从 $P(t)$ 中选取一对父母 p_1, p_2
    利用交叉操作, 从 p_1, p_2 产生后代 c_1, c_2
    以一定概率对 c_1, c_2 进行变异操作
    $C(t) \leftarrow \{c_1, c_2\}$
  end for
  依据适应度, 从 $P(t) \cup C(t)$ 选取优异个体作为 $P(t+1)$
end for
依据适应度, 从最后一代种群中选择最优个体作为问题的解

初始化编码：设置最大进化代数 T_max、选择概率、交叉概率、变异概率、随机生成 m 个染色体的群体，**每个染色体的编码对于一个可行的路径**（如 6 个城市，[1,3,2,6,4,5]就是一条可行路径）

先使得

在初始种群的赋值之前，先构造一个两倍于种群大小的种群，再从中选择 distance 较小的一半构成种群，以免初始种群分布在远离全局最优解的编码空间，导致遗传算法的搜索范围受到限制，同时也为算法减轻负担。

适应度函数：对每一个染色体 x_k，其个体适应度函数发 f(x_k)设置为 1 / d(x_k)，其中 d(x_k)表示该条路径的总长度。

选取父母：由适应度函数确定的概率$f(x_k)/\sum_{k=0}^{n} f(x_k)$，采用轮盘赌的方式进行选择。

交叉：在交叉概率的控制下，对选择群体中的个体进行两两交叉:

```
start_index = random.randint(0, num_cities - 2)
        end_index = random.randint(start_index, num_cities - 1)
```

随机选取 i,j,把下标[i,j]范围内的 parent1 的基因给 child1,其余部分在从 parent2 中获得，也就是说，parent2 中的基因如果在 parent1[i,j]中出现则不给自代，否则则依次把 parent2 中基因放进子代下标[i,j]范围外的位置。Child2 同理

比如：

```
rossover([4,5,7,1,6,3,2],[3,5,4,6,1,2,7])
```

i = 2;j = 5

结果 child1 为[5, 4, 7, 1, 6, 3, 2]

变异：在不同变异概率下发生不同变异

50%可能随机将染色体分为四段，其中 25%可能交换中间两段，25%可能交换首尾两段。

25%随机将两个城市访问顺序进行交换。

25%分成三段，中间一段反转。

重插：选择的子代与父代形成新的种群（先选出前 10%,再用锦标赛算法从剩下个体中进行选择出新一代种群:每次从中随机选取一个 group，把 group 中适应度最好个体的加入种群，直至种群个体数量达到 population_size）

2. 关键代码展示

初始化种群

```
    def initialize_population(self, pop_size):
        # 初始化种群的实现
        population = []
        for _ in range(2*pop_size):
            chromosome = list(range(self.num_cities))
            random.shuffle(chromosome)
            population.append(chromosome)
            #print(population)
        #self.population = self.rank(population)
        return population[0:pop_size//2]
```

计算适应度

```
    def fitness(self, chromosome):
        # 计算适应度的实现
        total_distance = 0
        for i in range(self.num_cities):
            city_a = chromosome[i]
            city_b = chromosome[(i + 1) % self.num_cities]#注意最后要形成回路
            total_distance += self.distance_map[city_a][city_b]
            if(total_distance==0):#防止除 0 错误
                return 10
        return 1 / total_distance
```

选择父母

```
    def select_parents(self):
        # 选择父母的实现
        fitness_scores = [self.fitness(chromosome) for chromosome in
self.population]#轮盘赌算法
        total_fitness = sum(fitness_scores)
        probabilities = [f / total_fitness for f in fitness_scores]
        #print("probabilities = ",probabilities)
        parents = []
        for _ in range(self.population_size):
            parent = random.choices(self.population, probabilities)[0]
            #print(random.choices(self.population, probabilities))
            parents.append(parent)
        return parents
```

交叉操作

```
    def crossover(self, parent1, parent2):
        # 交叉操作的实现
        start_index = random.randint(0, self.num_cities - 2)
        end_index = random.randint(start_index, self.num_cities - 1)
        child1 = [None] * self.num_cities
```

```
#子代 child1 的 start[start_index:end_index+1]与 parent1 相同
        child1[start_index:end_index+1] = parent1[start_index:end_index+1]
        for gene in parent2:
            if gene not in child1:
                for i, c in enumerate(child1):
                    if c is None:
                        child1[i] = gene
                        break
        child2 = [None] * self.num_cities
        child2[start_index:end_index+1] = parent2[start_index:end_index+1]
        for gene in parent2:
            if gene not in child2:
                for i, c in enumerate(child2):
                    if c is None:
                        child2[i] = gene
                        break
        return [child1,child2]
```

变异

```
def mutate(self,way): #变异函数
        p=random.random()
        if p<0.5:#分成四段
            a,b,c=random.sample(range(1, len(way)-1), 3)
            point=sorted([a,b,c]) #升序
            a,b,c=point[0],point[1],point[2]
            if p<0.25: #交换中间两段
                ans=way[0:a]+way[b:c]+way[a:b]+way[c:len(way)]
            else: #交换首尾两段
                ans=way[c:len(way)]+way[a:b]+way[b:c]+way[0:a]

        elif p<0.75: #选择其中两个点
            i=random.randint(0,len(way)-2)
            j=random.randint(1,len(way)-1)
            while True:
                if i!=j: #如果不相同 则交换两个城市
                    way[i],way[j]=way[j],way[i]
                    ans=way[:]
                    way[i],way[j]=way[j],way[i]
                    break
                else:
                    i=random.randint(0,len(way)-2)
                    j=random.randint(1,len(way)-1)
        else:#分成三段
            a,b=random.sample(range(1, len(way)-1), 2)
            if a>b: a,b=b,a
```

```
            ans=way[0:a]+way[a:b][::-1]+way[b:len(way)] #中间反转
        return ans
```

锦标赛算法选择

```
    def select(self,father_and_son):
        # 锦标赛
        top_n = self.population_size//10
        #self.population = self.get_smallest_elements(top_n, self.population)
        group_size = 10  # 每小组人数
        group_winner = self.num_cities // group_num  # 每小组获胜人数
        winners = self.get_smallest_elements(top_n, father_and_son)  # 先选取前 10%
的个体
        for i in range(self.population_size-top_n):
            group = []
            for j in range(group_size):
                group.append(random.choice(father_and_son[top_n:]))
            winners.append(self.best(group))

        self.population = winners
```

3. 创新点&优化

1.在每轮从父子代集合选取子代的 select 函数中先选取了适应度前 10%的个体，采用上述设定的原因是尽量让适应度更强的物种活下来，同时防止适应性最强的物种因随机性而被轮盘转赌法淘汰。

2.两种选择方法，在选择父母是采用轮盘赌的方法，在最后从采用锦标赛方法。因为一个种群内 distance 的相差不大，计算出的 fitness 以及被选择的概率相差也不大。所以轮盘赌算法效果不佳。

3.开始时计算了邻接矩阵来表示距离，避免每次计算适应度时都要计算距离，减少复杂度。

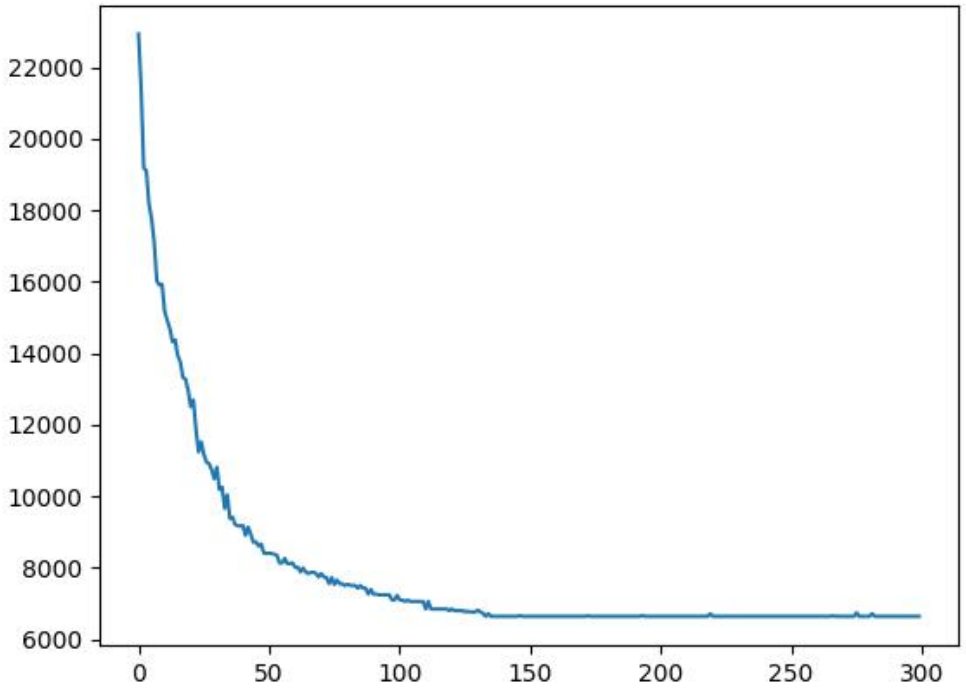
4.采用依据产生随机数大小进行不同的变异。

三、 实验结果及分析

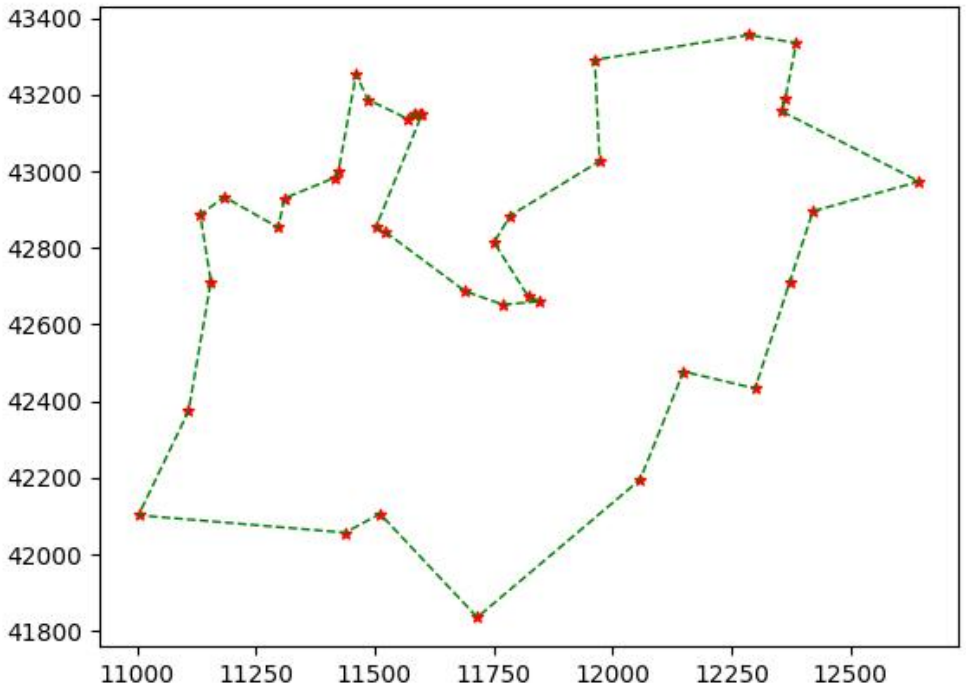
1. 实验结果展示示例（可图可表可文字，尽量可视化）

1.1 Dj38 数据集

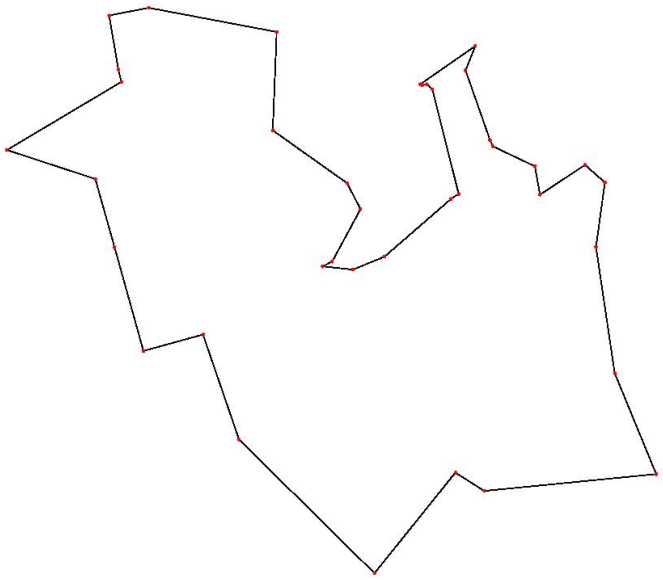
Distance 随迭代轮数的变化



找到的路径 Distance = 6656



实际最优解：

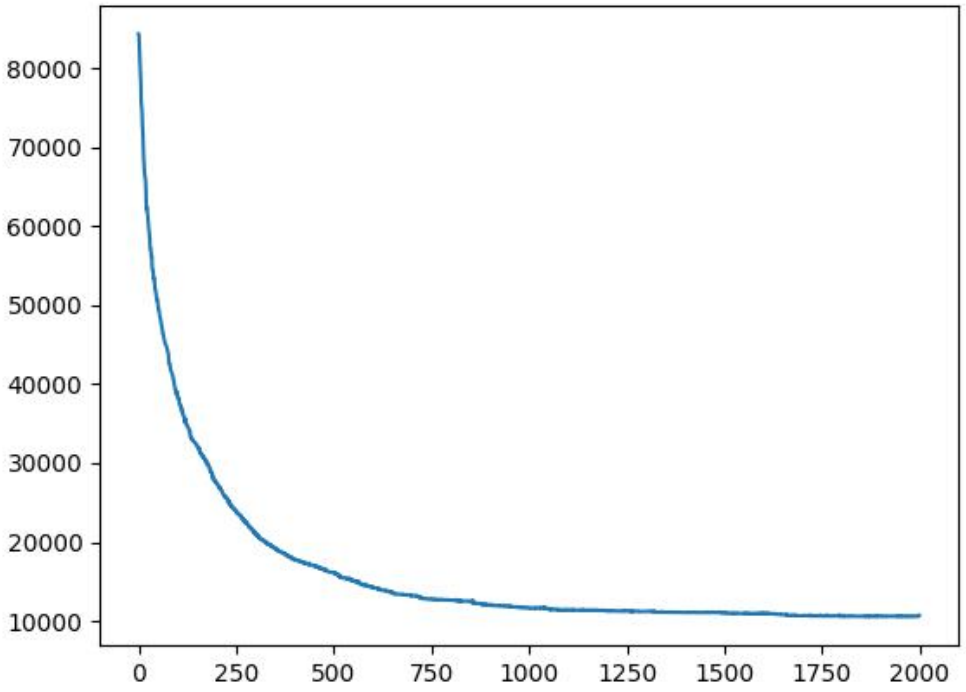


solution [16, 17, 18, 12, 14, 19, 22, 25, 24, 21, 23, 27, 26, 30, 35, 33, 32, 37, 36, 34, 31, 29, 28, 20, 13, 9, 0, 1, 3, 2, 4, 5, 6, 7, 8, 10, 11, 15]

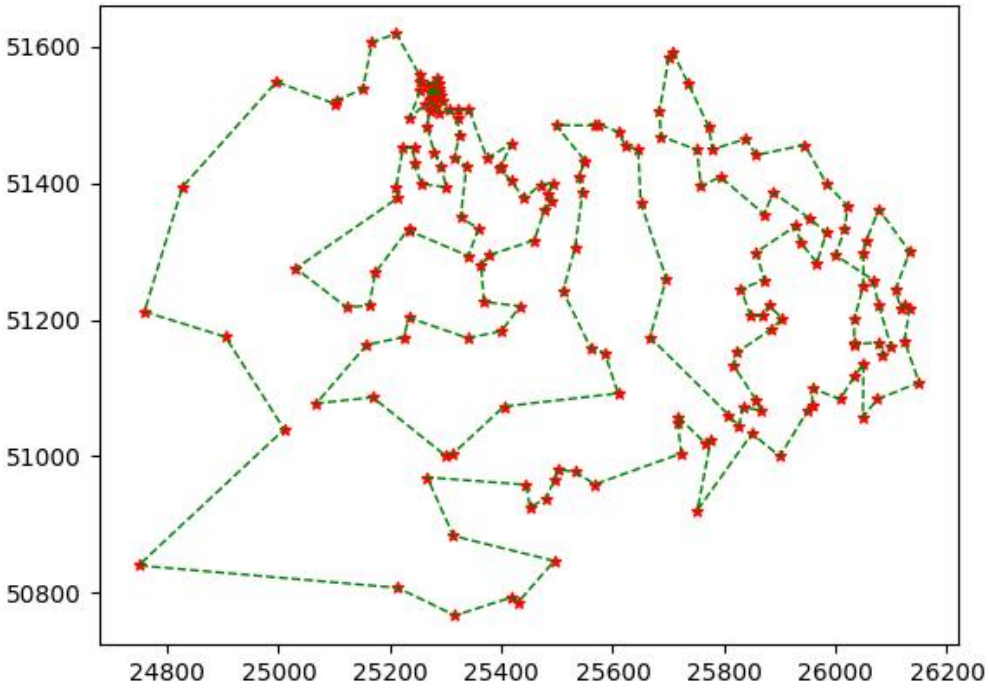
diatance 6656.0

1.2 Qa194 数据集

Distance 随迭代轮数的变化



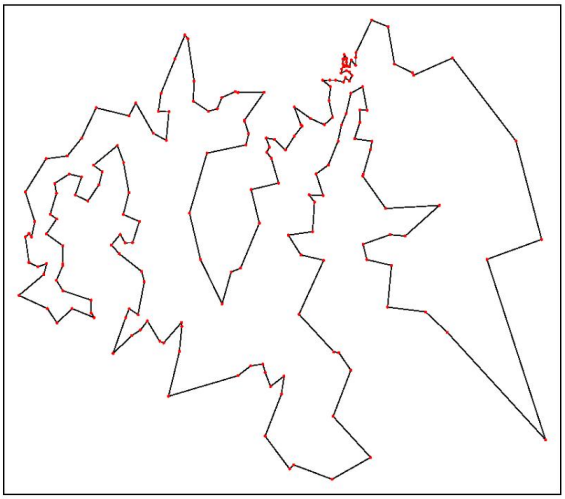
找到的路径
diatance 10592.0



solution [46, 39, 37, 40, 43, 45, 47, 42, 34, 41, 49, 48, 54, 53, 51, 52, 55, 57, 60, 65, 67, 63, 69, 68,

73, 71, 25, 23, 16, 13, 10, 6, 20, 17, 21, 28, 27, 32, 59, 56, 44, 36, 38, 33, 26, 30, 31, 29, 18, 14, 11, 9, 8, 4, 2, 1, 3, 5, 0, 19, 64, 84, 85, 97, 62, 35, 88, 89, 93, 98, 100, 103, 110, 126, 124, 125, 131, 133, 129, 144, 155, 160, 162, 163, 168, 171, 178, 175, 181, 193, 189, 191, 190, 188, 187, 192, 184, 179, 177, 176, 174, 172, 173, 182, 185, 186, 183, 180, 167, 169, 170, 165, 159, 147, 142, 134, 132, 127, 123, 122, 119, 120, 128, 130, 135, 150, 154, 161, 166, 164, 158, 157, 146, 151, 140, 143, 149, 152, 156, 153, 138, 137, 145, 148, 141, 139, 136, 118, 121, 117, 116, 115, 114, 111, 109, 99, 107, 106, 104, 105, 102, 101, 108, 112, 113, 81, 61, 58, 15, 7, 12, 22, 24, 70, 79, 86, 75, 74, 77, 90, 92, 95, 94, 96, 91, 87, 82, 78, 80, 83, 76, 72, 66, 50]

实际最优解 distance = 9352



2. 评测指标展示及分析

测评指标：(1)找到的解与最优解的 distance 差值的比值 Q = (distance(find)-distance(opt))/distance(opt)。(2)迭代过程运行总时间 T

Dj38:

q = 0(找到的解为最优解)

T=0.0 小时 1.0 分钟 20.712100744247437 秒

Qa194:

Q = (10592-9352)/9352 = 0.132

T = 3.0 小时 27.0 分钟 3.582146644592285 秒

四、 参考资料

人工智能导论——遗传算法求解 TSP 问题实验_遗传算法求 tsp 问题实验-CSDN 博客

变异操作 mutate 函数参考了 https://github.com/cantian114514/Artificial-intelligence/blob/main/%E5%AE%9E%E9%AA%8C/%E6%BA%90%E7%A0%81/lab6/GA_TSP.py