



南開大學
Nankai University

人工智能技术实验 实验报告

实验名称：基于遗传算法的旅行商问题

姓名：张恒硕

学号：2212266

专业：智能科学与技术

人工智能学院

2024 年 11 月

目录

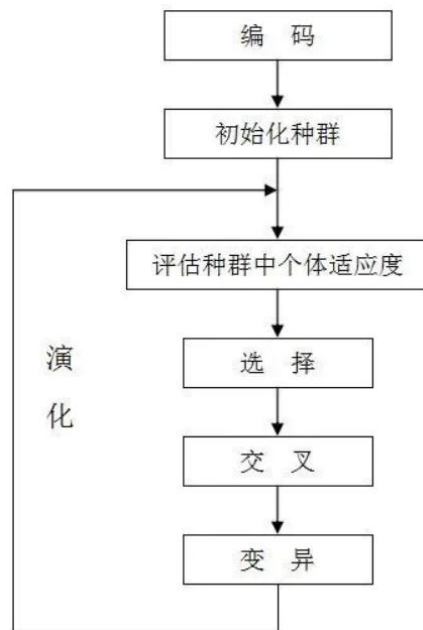
目录	2
一、 问题简述	3
二、 实验目的	4
三、 实验内容	4
四、 编译环境	4
五、 实验步骤与分析	4
1. 参数与设置	4
2. 个体类	5
3. 交叉	6
4. 变异	7
5. 选择	9
6. 展示演化过程	10
7. 绘制路径	10
8. 主函数	12
六、 实验结果	14
1. 暴力遍历	14
2. 旅行商问题调参求解	15
3. 可视化展示（以上述第三组参数为例）	17
4. 经纬度转化	18
七、 分析总结	19
1. 成果总结	19
2. 问题分析与改进思路	19

一、问题简述

旅行商问题：设有 N 个互相可直达的城市，给定每对城市之间的距离，某推销商准备从其中的 A 城出发，周游各城市一遍，最后又回到 A 城，要求访问每一座城市并回到起始城市的最短回路。这是非常经典的组合优化问题中的 NP 难问题（多项式复杂程度的非确定性问题），通过常规的暴力枚举，遍历等方法难以计算出最优解，因此，本实验使用遗传算法进行最优解的计算。

遗传算法是根据大自然中生物体进化规律而设计提出的，是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。该算法通过数学的方式，利用计算机仿真运算，将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异等过程。在求解较为复杂的组合优化问题时，相对一些常规的优化算法，通常能够较快地获得较好的优化结果。

对于本实验，求解我国 34 个城市的旅行商问题，给定 34 个城市的名称和坐标数据，求遍历一次所有城市回到起点期间，所经过路程的较小值。



算法实现流程：

1. 初始化阶段：

初始化城市数据：城市数目、城市位置、城市距离矩阵；

初始化基本参数：种群数目、交叉概率、变异概率、交叉方式、变异方式；

初始化种群：随机生成 n 个城市序列， n 为种群数目。

2. 计算种群适应度：根据城市序列和城市距离矩阵计算种群适应度。

3. 生成下一代种群（选择、交叉、变异）

选择：轮盘赌选择、锦标赛选择等方式；

交叉：单点交叉、两点交叉等方式；

变异：均匀变异、基本位变异等方式。

4. 迭代操作：重复步骤 2 和 3，并记录每一次迭代的最优解
5. 结果输出：输出迭代过程中产生的最短路径长度、最短路径；

二、实验目的

1. 了解旅行商问题的基本概念，理解旅行商问题的求解难度；
2. 了解遗传算法的基本原理及实现流程；
3. 能够利用遗传算法解决旅行商问题；
4. 进一步理解可视化设计。

三、实验内容

1. 实现基于穷举法的 10 个城市（No.1-No.10）的旅行商问题，求解访问每一座城市一次并最终回到起始城市的最短回路，理解穷举法的局限性；
2. 实现基于遗传算法的 34 个城市的旅行商问题，求解访问每一座城市一次并最终回到起始城市的最短回路；
3. 对比遗传算法下不同种群规模、交叉概率、变异概率等参数下，遗传算法对问题求解的效果影响并分析原因。
4. 进行遗传算法可视化，展示搜索过程中每代种群最优路径长度的变化和迭代最优结果。

四、编译环境

Python3.11+numpy+pandas+matplotlib+cartopy

cartopy 用于获取真实投影地图来绘制路径，不使用则 `plot_path` 函数无法直接使用，对其他部分没有影响。

五、实验步骤与分析

1. 参数与设置

● 内容与原理

设置绘图使用的字体，设置遗传算法参数，读取城市经纬距离矩阵和城市经纬度两个文件的数据。

● 代码

```

# 设置中文字体

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode_minus'] = False


# 参数

length = 34          # 基因数目

population_size = 100 # 种群大小

generations = 500     # 迭代代数

Pc = 0.9              # 交叉因子

Pm = 0.01              # 变异因子

elite = 0.1           # 精英保留比例


# 读取数据

file_path_distances = '城市经纬距离矩阵.xlsx'

df_distances = pd.read_excel(file_path_distances, index_col=0)

distance_matrix = df_distances.iloc[:length, :length].values

city_names = df_distances.index[:length]

file_path_cities = '城市经纬度.xlsx'

df_cities = pd.read_excel(file_path_cities, index_col=0)

```

- 两个文件存储的数据分别用作遗传算法的距离矩阵和绘制地图的位置标定。

2. 个体类

- 内容与原理

定义个体类，其包含一个染色体的基因（城市序列）和对应的适应度值（总距离）。

● 代码

```
# 个体类

class Individual:

    def __init__(self, genes):

        self.genes = genes

        self.fitness_value = None

    def fitness(self): # 个体生成后只评估一次，节省时间

        if self.fitness_value is None:

            self.fitness_value = 0

            for i in range(length - 1):

                self.fitness_value += distance_matrix[self.genes[i], self.genes[i + 1]]

            self.fitness_value += distance_matrix[self.genes[-1], self.genes[0]]

        return self.fitness_value
```

- 生成个体时只需给出个体的基因。
- 个体生成后，在第一次被访问时会计算适应度值，之后储存起来，减少评估耗时。适应度值通过计算整个回路的路径获得。

3. 交叉

● 内容与原理

交叉操作使用单点交叉，输入两个亲代，使用交叉因子判断是否交叉，不交叉返回亲代；交叉则选取交叉点，获得子代。

● 代码

```

# 交叉：单点交叉

def crossover(parent1, parent2, pc):

    if np.random.rand() < pc:

        cross_point = np.random.randint(1, length)

        child1_genes = parent1.genes[:cross_point] + [gene for gene in parent2.genes if
gene not in parent1.genes[:cross_point]]

        child2_genes = parent2.genes[:cross_point] + [gene for gene in parent1.genes if
gene not in parent2.genes[:cross_point]]

        return Individual(child1_genes), Individual(child2_genes)

    else:

        return Individual(parent1.genes.copy()), Individual(parent2.genes.copy()) # 避
免修改原始种群个体

```

- 不交叉返回亲代时，为保证不修改原种群中的个体（可能还会被选为亲代或进行其他操作），复制并构建相同个体。
- 执行交叉操作时，进行合法化的单点交叉。以 child1 为例，其基因由 parent1 前半部分（直到交叉点）和 parent2 中未出现在 parent1 前半部分的所有基因组成。

4. 变异

● 内容与原理

设计了四种变异方式，在按变异因子判断染色体上的某一个基因是否变异后，按一定概率选取进行的变异方式。

● 代码

```

# 变异：交换、区间反转、基因移动、片段移动

def mutation(individual, pm):

    # 创建个体的副本，防止修改原始个体

```

```

new_genes = individual.genes.copy()

# 方法选择概率

methods = ['swap', 'inversion', 'insertion', 'shift']

probabilities = [0.7, 0.1, 0.1, 0.1]

for i in range(length):

    if np.random.rand() < pm:

        method = np.random.choice(methods, p=probabilities)

        if method == 'swap':

            j = np.random.choice([x for x in range(length) if x != i])

            new_genes[i], new_genes[j] = new_genes[j], new_genes[i]

        elif method == 'inversion':

            start, end = sorted(random.sample(range(length), 2))

            new_genes[start:end + 1] = reversed(new_genes[start:end + 1])

        elif method == 'insertion':

            j = random.sample(range(length), 1)[0]

            gene = new_genes.pop(i)

            new_genes.insert(j, gene)

        elif method == 'shift':

            start, end = sorted(random.sample(range(1, length-1), 2))

            subsequence = new_genes[start:end + 1]

            sublength = len(subsequence)

            k = random.choice([x for x in range(length - sublength)]) # 保证插入

```


位置在剩余序列内

```
del new_genes[start:end + 1]

for item in reversed(subsequence):

    new_genes.insert(k, item)

return Individual(new_genes)
```

- 交换（swap）：选取一个位置，将当前位与该位置交换。
- 区间反转（inversion）：另选取两个位置，将其构成的区间反转。
- 基因移动（insertion）：选取一个位置，将当前位移动到该位之后。
- 片段移动（shift）：另选取三个位置，将前两个位置构成的区间移动到第三个位置后。为保障移动的合法性，不能选取整个染色体序列为移动区间，且第三个位置要在去除移动区间后选取。
- 四种变异方式的图解如下图 5。第一种方式对染色体整体的改变远小于后三者，所以设置其被选取的概率最大。

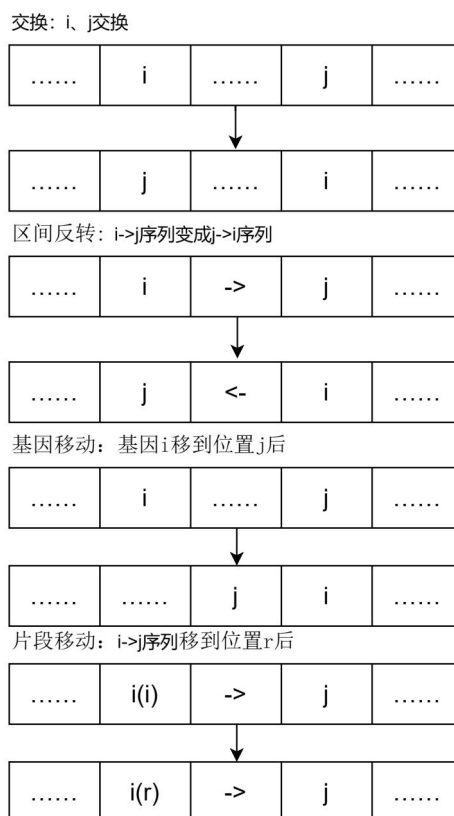


图 5 四种变异方式图解

5. 选择

● 内容与原理

选取轮盘赌选择，每个个体被选择的概率等于它的适应值与所有适应值和的比，如下式：

$$p_i = \frac{f_i}{\sum_{j=1}^{\text{population}} f_j}$$

- 代码

```
# 选择：轮盘赌选择

def select(pop, fitness):

    probabilities = 1 / np.array(fitness)

    probabilities /= probabilities.sum() # 归一化

    selected_indices = np.random.choice(np.arange(population_size), size=2,
replace=True, p=probabilities)

    return pop[selected_indices[0]], pop[selected_indices[1]]
```

6. 展示演化过程

- 内容与原理

使用世代最优或全局最优距离数据绘制迭代演化过程。

- 代码

```
def evolution(fitness, title):

    plt.figure(figsize=(10, 6))

    plt.plot(range(1, generations + 1), fitness, label=title)

    plt.xlabel('迭代次数')

    plt.ylabel('最小距离')

    plt.title('演化过程')

    plt.legend()

    plt.show()
```

- 为绘制两种演化过程，图的标题也作为参数传入。

7. 绘制路径

- 内容与原理

使用 `cfeature` 获取地图投影，标记城市坐标和城市名，绘制路径。

- 代码

```
# 绘制路径

def plot_path(best_individual, city_names, distance_matrix):

    # 模拟地图

    fig = plt.figure(figsize=(10, 8))

    ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())

    ax.add_feature(cfeature.COASTLINE)

    ax.add_feature(cfeature.LAND, edgecolor='black')

    ax.add_feature(cfeature.OCEAN)

    # 绘制城市点并标注城市名

    path = [city_names[i] for i in best_individual.genes]

    lats = distance_matrix.loc[path, 'Latitude'].values

    lons = distance_matrix.loc[path, 'Longitude'].values

    ax.scatter(lons, lats, 60, marker='o', color='k', zorder=5,
transform=ccrs.PlateCarree(), label='城市')

    for i, name in enumerate(path):

        ax.text(lons[i], lats[i], name, fontsize=10, ha='right', va='bottom',
transform=ccrs.PlateCarree())

    # 绘制路径（回到起点）

    ax.plot(lons, lats, 'r-', linewidth=2, label='路径', transform=ccrs.PlateCarree())

    ax.plot([lons[-1], lons[0]], [lats[-1], lats[0]], 'r-', linewidth=2,
```

```
transform=ccrs.PlateCarree())
```

```
plt.show()
```

- 模拟地图的部分并不必须，可以去掉。
- 在绘制路径的时候，需要保证回到起点。

8. 主函数

● 内容与原理

调用前面的函数，进行精英保留、选择、交叉、变异的迭代过程，记录世代最优和全局最优，并可视化展示。

● 代码

```
# 主程序

start_time = time.time()

generation_best_fitness = [] # 世代最优

best_fitness = []           # 全局最优

population = [Individual(random.sample(range(length), length)) for _ in
range(population_size)] # 种群初始化

best_individual = min(population, key=lambda x: x.fitness()) # 保留迭代过程中的最优个体

for generation in range(generations):

    sorted_population = sorted(population, key=lambda x: x.fitness())

    new_population = sorted_population[:int(elite * population_size)]

    while len(new_population) < population_size:

        parent1, parent2 = select(population, [ind.fitness() for ind in population])

        child1, child2 = crossover(parent1, parent2, Pc)

        child1 = mutation(child1, Pm)
```

```

    child2 = mutation(child2, Pm)

    new_population.extend([child1, child2])

    population = new_population[:population_size]

    # 最佳个体

    generation_best_individual = min(population, key=lambda x: x.fitness())

    generation_best_fitness.append(generation_best_individual.fitness())

    if generation_best_individual.fitness() < best_individual.fitness():

        best_individual = generation_best_individual

    best_fitness.append(best_individual.fitness())

    if (generation + 1) % 100 == 0:

        print(f'第{generation + 1}轮: 当前最优方案{best_individual.genes}的花费为
{best_individual.fitness()}')

# 输出最终结果

print(f'最短路径: {best_individual.genes}')

print(f'最短距离: {best_individual.fitness():.5f}经纬距离")

end_time = time.time()

print(f'运行时间: {end_time - start_time:.5f}s")

evolution(generation_best_fitness, '世代最优')

evolution(best_fitness, '全局最优')

plot_path(best_individual, city_names, df_cities)

```

- 每一代由上一代保留的精英个体和产生的新个体组成，要保证其总数在

迭代过程中保持不变。

- 没有精英保留策略的前提下，每一代记录的世代最优和全局最优不一定相同，前者记录的是当前种群中的最优个体，而后者记录的是到目前为止产生的所有个体中的最优个体。

六、实验结果

1. 暴力遍历 (Brute Force.py)

通过遍历所有方案，暴力算法必定能获得最优解。暴力算法遍历的方案数是城市数的阶乘，其随城市数的增加快速增加，及其不利于处理复杂问题。

以城市列表前 10 个城市为例进行求解，结果如下：

最短路径:

['天津', '哈尔滨', '上海', '南宁', '重庆', '拉萨', '乌鲁木齐', '银川', '呼和浩特', '北京']

最短距离: 110.34594 经纬距离

运行时间: 7.50780s

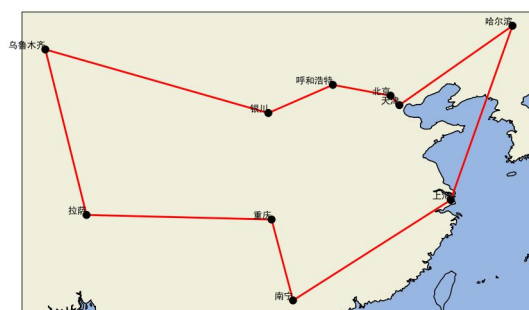


图 6-1 暴力遍历 10 个城市最优线路

面对 10 个城市的旅行商问题，暴力遍历方法就要遍历 $10!$ 个方案，耗时 7 秒多。下图图 6-2 给出了使用暴力遍历方法，解决 3 到 12 个城市的 TSP 问题的耗时，可以发现，在 10 个城市之后，耗时开始爆炸式增长。当扩展到 34 个城市时，其耗时将是令人绝望的，这里不做展示。与暴力遍历相比，遗传算法的搜索具有随机性和启发性，更具效率。

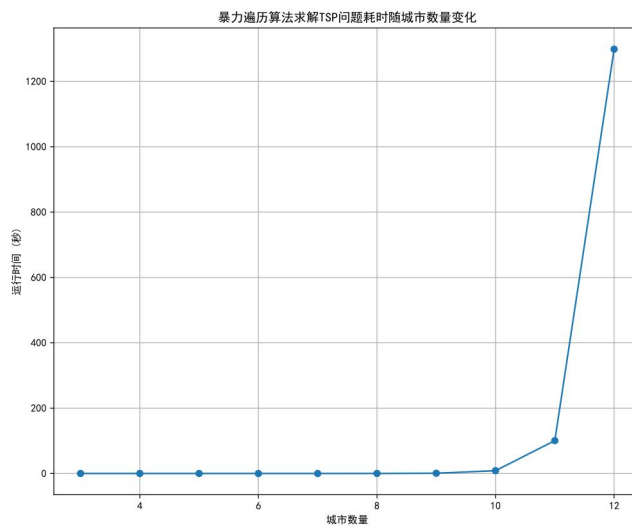


图 6-2 暴力遍历 3-12 个城市 TSP 问题耗时

2. 旅行商问题调参求解

表 1 TSP 问题参数组合结果

	第 1 组	第 2 组	第 3 组	第 4 组	第 5 组	第 6 组	第 7 组	第 8 组
种群规模	50	100	50	50	50	50	50	100
迭代次数	200	200	500	200	200	200	饱和 (连续 100 代 全局最 优不 变)	500
交叉概率	0.6	0.6	0.6	0.9	0.6	0.6	0.6	0.9
变异概率	0.02	0.02	0.02	0.02	0.05	0.02	0.02	0.05
精英保留比	0.1	0.1	0.1	0.1	0.1	0.25	0.1	0.25

例								
最小开销 (经纬距离)	235.71 441	195.81 773	174.73 183	217.47 703	232.42 951	177.82 148	171.62 364	174.52 277
耗时 (s)	0.6656 3	1.1733 7	1.6433 9	0.6164 1	0.9608 8	0.4818 7	1.5084 4 (443代)	4.3853 1

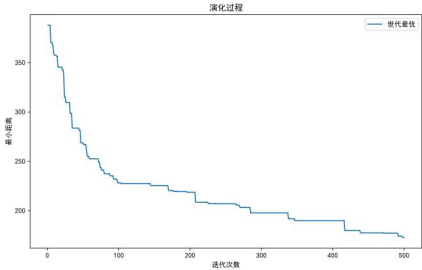
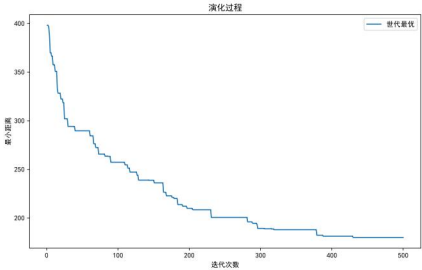
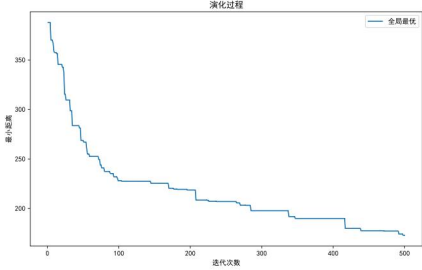
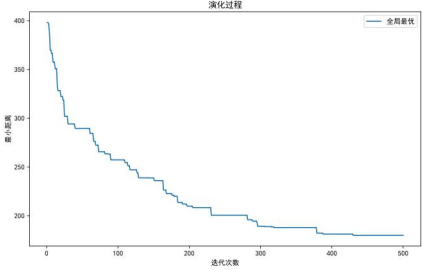
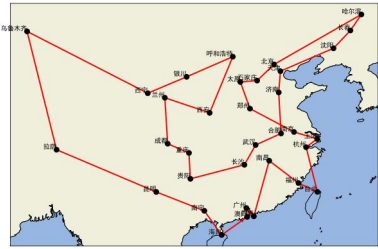
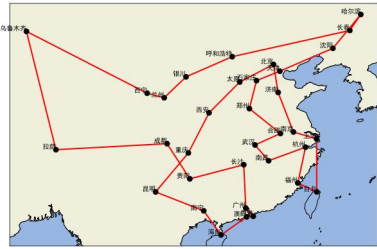
结合最小开销和耗时，选取第三组（第七组）参数为最优解。第 8 组参数的耗时是第三/七组的 2.5 倍左右；第 6 组虽然兼具优秀的最小开销和耗时，但是并不能稳定获得这种优秀解。以下对各参数对结果的影响作以分析：

- 种群规模（1、2）：耗时与种群规模基本成正比，这是因为形成新种群、评估的耗时都与种群规模成正比例。增加种群规模能改善结果，这是因为每次搜索的解空间变大了。
- 迭代次数（1、3、7）：耗时与迭代次数基本成正比，这是因为形成新种群、评估的次数都与迭代次数成正比例。增加迭代次数，在一定范围内，能改善结果。在结果收敛后（本问题约为 400 轮左右），继续增加迭代次数，基本不会改善最终结果。这是因为已经基本收敛到多峰问题的一个次优解，在交叉变异时很难跳转到更优解。
- 交叉概率（1、4）：在不同的问题中，有相应的优秀的交叉概率。本问题由于搜索解空间庞大，需要交叉提供多样性，因此交叉概率较大为佳。这也是因为交叉能在序列维度上集合两个优质个体的优点。由于交叉操作比较简单，增加交叉概率引起的更多次交叉并不会带来多少耗时上的增加。
- 变异概率（1、5）：在不同的问题中，有相应的优秀的变异概率。本问题是明显的多峰问题，各优质解序列的差异较大，变异操作虽然提供了多样性，但由于其改动的方向并不确定，其概率的增加并不会显著改善结果，反而会因为增加复杂操作而增加耗时。本问题适宜使用较小的变异因子，并搭配精英保留策略，因为这能提供必须的多样性，跳出局部最优解，并通过精英保留策略确保不丢失已获得的优秀子序列。

- 精英保留比例（1、6）：在不同的问题中，有相应的优秀的精英保留比例。如果没有精英保留策略，交叉、变异可能破坏世代中的优质个体的优质片段，无法进一步在迭代过程中进行改善。而由于搜索解空间较大，保留过多的精英会减弱每一代的搜索能力，降低收敛速度，甚至可能加大收敛到局部最优解的可能新。

3. 可视化展示（以上述第三组参数为例）

以下展示两次运行的可视化结果。

	第一次	第二次
世代最优		
全局最优		
最优路径		

世代最优和全局最优展示了遗传算法的演化过程，因为精英保留策略的加入，确保了不会出现倒退现象，两张图片是一致的。以下展示一组没有精英保留策略的情况，来说明二者可能出现的差别。

最终结果	329.60022 经纬距离
------	----------------

世代最优	
全局最优	
最优路线	

4. 经纬度转化

使用经纬度表示距离可能与实际情况出现偏差，以下转换经纬度距离为实际距离以展示环游我国主要城市的近似最小距离。

最终结果	13310.74335km
------	---------------

[illegible]

七、分析总结

1. 成果总结

本次实验成功实现了 TSP 问题的遗传算法求解和相应的可视化，并且对参数进行了调试。

2. 问题分析与改进思路

- **精英保留策略的使用**：开始编写代码时，没有使用精英保留策略，因为在之前其他问题的遗传算法代码编写中，由于可能加大收敛到次优解的概率，所以并未使用该策略。而在本实验中，这会导致程序一直无法收敛。经测试，即使迭代 100000 次，也无法将最终路径长度降到 300 以下。这是因为之前处理的问题和本问题在解域规模上有着本质区别。本问题的解域过于庞大，每轮 100 的个体数显得过小，这就需要加速收敛的精英保留策略。与之前较小解域寻求最优解的问题不同，本体只得到优秀解即可。
- **局部搜索**：对每一个个体，遍历内部自交叉结果，用获得的最优解替代这个个体，相当于每个个体都寻求局部最优解。这样会极大增加耗时，但可以在

更少的迭代次数内获得更优解。本次实验按照推荐的参数进行，并未融入局部搜索机制。