# 计算机模拟 project1

**汪奕晨 3180105843**

**数学科学学院 数学与应用数学专业**

# 1 问题描述

1. 用简单遗传算法求解 TSP 问题
2. 比较简单遗传算法与模拟退火算法在各方面的优劣
3. 提高遗传算法的收敛性与收敛率，并验证想法
4. 结合遗传算法和模拟退火算法构建出更加先进的算法

# 2 Notations

| 记号 | 含义 |
|---|---|
| $N$ | TSP 实例中城市的数目 |
| $G$ | 使用 GA 算法时种群的大小 |
| $M$ | GA 算法中迭代次数，在 SA 算法中指降温次数 |
| $I$ | SA 算法中每个温度的迭代次数 |
| $p_r, p_c, p_m$ | GA 算法中生成子代时复制 (repeat), 交叉 (cross), 突变 (mutate) 的概率 |

Table 1: Notations

# 3 设计思路

## 3.1 简单遗传算法的设计思路

### 3.1.1 适合度计算

由于 TSP 算法的目的是要**最小化**环游距离，而遗传算法中需要**最大化**适合度，我们使用种群中最大环游距离减去各样本的环游距离作为各样本的适合度。尽管这会使得不同代数的种群适合度无法比较（因为种群的最大距离也在不断变化），但在算法中并不需要我们比较不属于同一子代个体的适合度，因此该方法不会造成逻辑上的错误。

#### 3.1.2 遗传计算

根据 $p_r, p_c, p_m$ 的比例，决定自带的每个个体时进行轮盘赌抽取决定要进行复制、交叉或突变。在具体的实现上，我们总是先通过抽取得到子代分别由复制、交叉、突变而来的个体数目，而后再批量生成各遗传方式得到的子代。

**选择算子：**

在最初的尝试中，我们直接使用轮盘赌抽取实现选择算子，但这并不能导致最短距离随着迭代次数收敛。

于是我们优化为采用 2-锦标赛进行选择复制。根据上代各亲本适合度的比例通过随机数抽取 $n_r$ 个子代作为一组，为了保证子代的多样性，同一父本不会被重复抽取。共抽取两组。选择两组中对应位置表现最好 (有更短的距离/更高的适合度) 的复制到子代。

相比于轮盘赌抽取，锦标赛抽取可以更好地选出优质的个体，同时也保留了跳出局部最优的机会。在应用中有更好的表现。

**交叉算子：**

分别实现讲义中'partial-mapped', 'order', 'position-based' 三种交叉方法，在之后对各交叉方法的性能进行对比。

**变异算子：**

随机抽取 $n$-排列中的两个位置，对其之间的环游进行反向。

#### 3.1.3 $p_r, p_c, p_m$ 比例的自适应修改方法

一种朴素的想法是，当交叉/突变效果较好时则增大交叉/突变概率。我们采用 Equation (1) 中的方式自适应调整比例 [1]。

$$
\begin{aligned}
p_c &= \begin{cases} K_1 \frac{F_{max} - F'}{F_{max} - F_{avg}} & F' > F_{avg} \\ K_2 & F' \le F_{avg} \end{cases} \\
p_m &= \begin{cases} K_3 \frac{F_{max} - F''}{F_{max} - F_{avg}} & F'' > F_{avg} \\ K_4 & F'' \le F_{avg} \end{cases}
\end{aligned} \tag{1}
$$

$F_{max}, F_{avg}$ 为种群中适应度的最大值与平均值，$F', F''$ 分别为交叉得到子代与突变得到子代的平均适应度，$K_i, i = 1, 2, 3, 4$ 为超参数，本文中设定为 $(0.5, 0.3, 0.25, 0.15)$。

### 3.2 代码设计思路

这里给出一些实现难点的伪代码以及实现中采用的技巧。对三种交叉算法我们仅介绍最复杂的 partial mapped 方法，剩余两个方法可以通过 np.in1d 方法快速实现，具体参考附录中的代码。

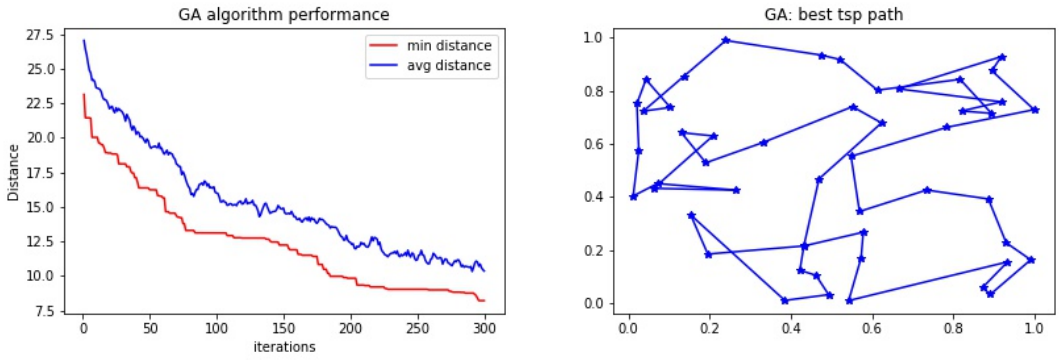
```
def cross_partial_mapped(path_1, path_2, node_1, node_2):
    '''
    输入：path_1,path_2 分别为两个不同的 n - 排列，node_1,node_2 为交换的节点位置
    输出：path_1,path_2 为可变对象， 更改path_1 为交叉后的值
    '''
    cross_part = path_2 中需要交叉到path_1 的点集
    for 遍历path_1 中的每个位置:
        if 该位置不在cross_part 中:
            x = 该位置上的值
            while 当x 在cross_part 中时:
                找到path_2 中对应位置对应的path_1 中的值 y
                x = y

    将path_1 中需要交换的位置赋值为cross_part
```

Listing 1: partial-mapped 方法实现交叉

# 4 模拟结果与分析

## 4.1 GA 算法的有效性

对 $N = 50, G = 200, M = 300$ 的参数进行测试，结果如 Figure (1)



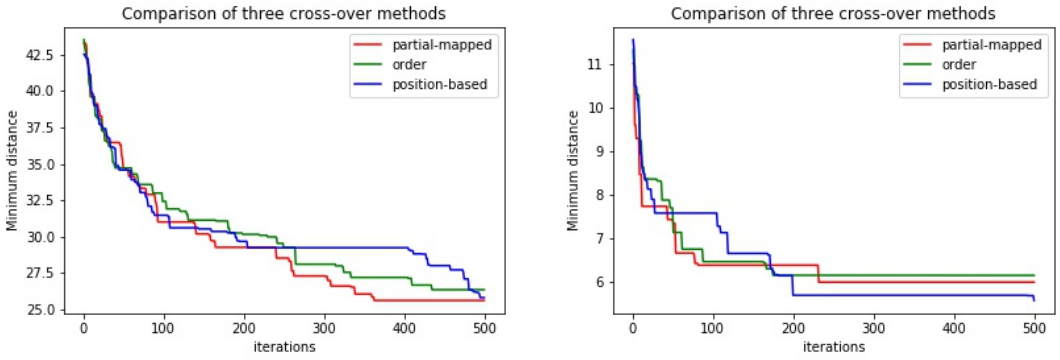

(a) Performance $N = 50, G = 200, M = 300$ (b) Result $N = 50, G = 200, M = 300$

Figure 1: Sample Performance and Result

可以看到随着迭代不断进行，最短距离确实在不断地降低，且解也在向最优逼近。此外，最短距离和种群平均距离以相同的趋势下降，在后文中我们将仅适用最短距离的变化来衡量下降趋势。

## 4.2 交叉方法的选择

分别使用三种交叉策略对不同的参数进行计算，得到的结果如 Figure (2(a),2(b))




(a) $N = 50, G = 150, M = 750$ (b) $N = 25, G = 50, M = 500$

Figure 2: Cross Methods Comparison

可以看出在不同的参数下，三种方法最终均会收敛到相近的结果，相对优劣也有偶然因素，我们可以认为三种交叉方法并没有显著的优劣之分。在后文中我们将默认使用 partial-mapped 方法。

## 4.3 自适应方法的测试

为了说明自适应方法的有效性，我们选取固定 $p_{c,const} = 0.4, p_{m,const} = 0.2$ 与 $K_1 = 0.5, K_2 = 0.3, K_3 = 0.3, K_4 = 0.15$ 两组参数进行对比。对比 Equation (1)，目的是让自适应的 $p_c$ 在交叉子代效果较好时会变大，在子代效果不佳时会变小，$(K_1 + K_2)/2 = p_{c,const}$ 则保证了 $p_c$ 在 $p_{c,const}$ 附近波动，$p_m$ 同理。这样在前期种群较为随机时可以加强交叉、变异迅速筛选优势个体，后期种群表现较优异时可以通过降低 $p_c, p_m$ 让解减小波动，迅速收敛。

选取 $N = 100, G = 200, M = 1000$ 分别对同一初始化种群采用自适应与固定值求解，结果如 Figure (3). 可以看出自适应的方法确实可以加快 GA 算法的收敛速度，后文默认采用自适应的方法,$K_i$ 的取值分别为 $(0.5, 0.3, 0.25, 0.15)$

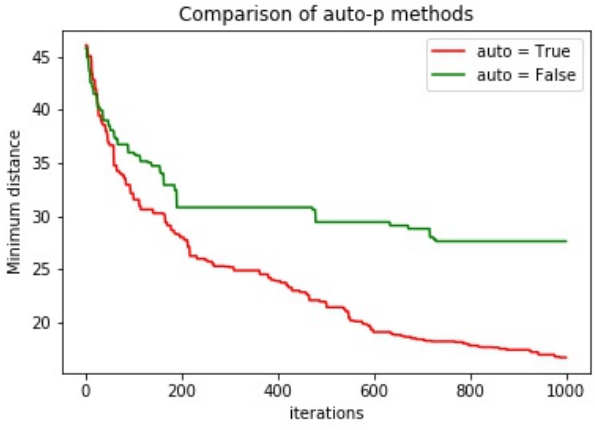
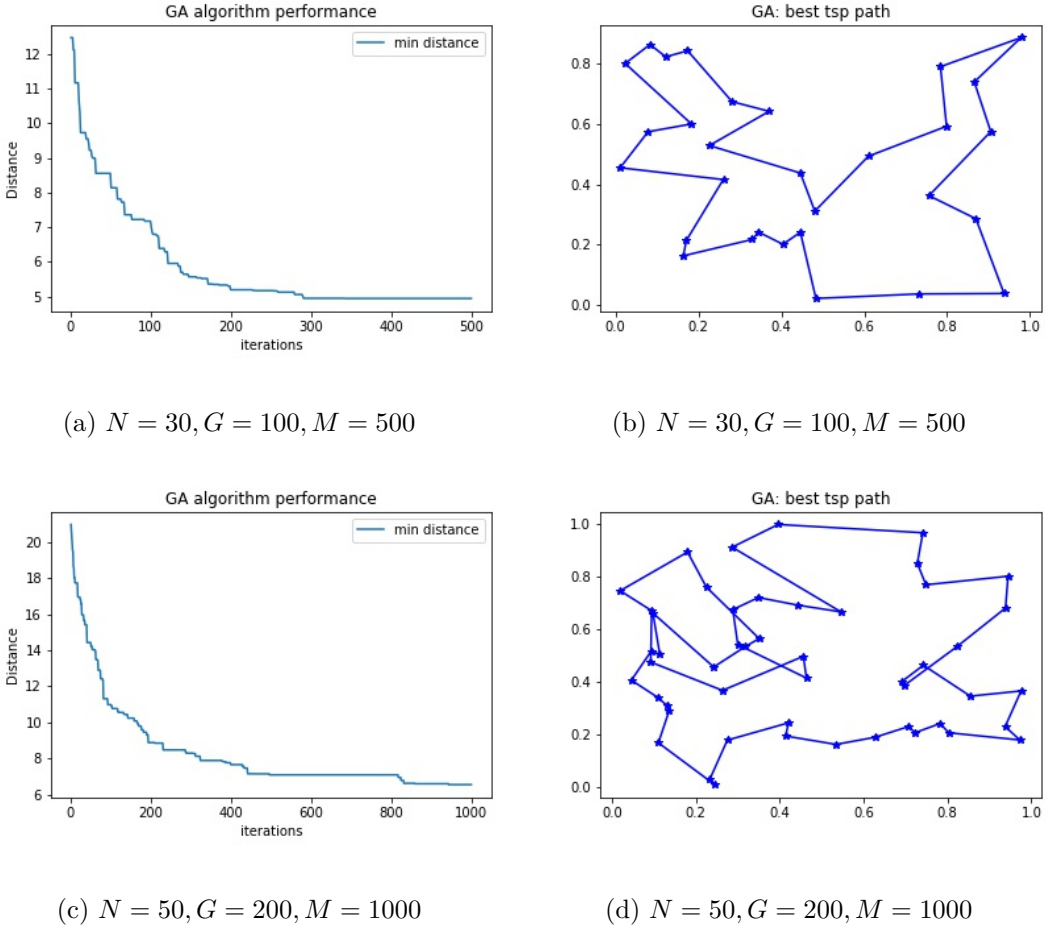



Figure 3: Automatic $p$ versus Constant $p$

## 4.4 不同规模 TSP 问题的 GA 算法求解结果




(a) $N = 30, G = 100, M = 500$

(b) $N = 30, G = 100, M = 500$

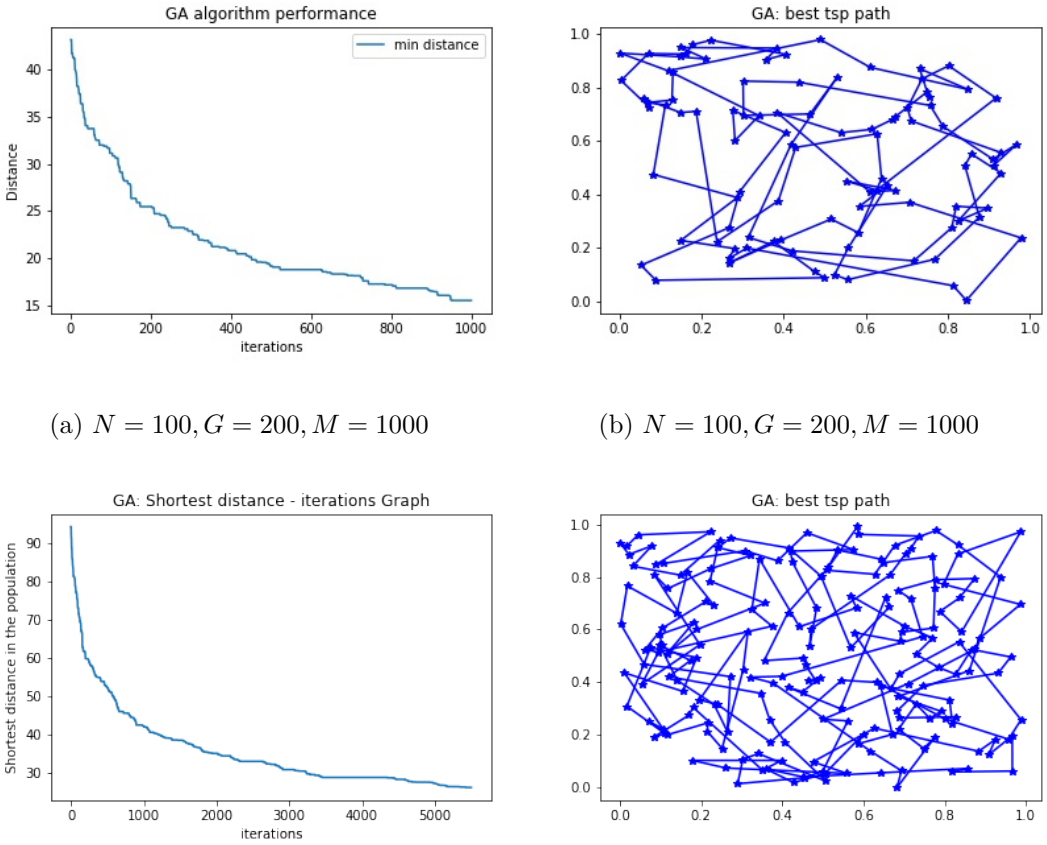
(c) $N = 50, G = 200, M = 1000$

(d) $N = 50, G = 200, M = 1000$

Figure 4: GA result for small $N$

如 Figure (4), 当 $N$ 较小时，GA 算法可以给出比较好的结果。

如 Figure (5)，当 $N$ 比较大时，GA 算法很容易陷入局部最优，给出一个不太好的结果。且从距离下降的趋势来看，增加迭代次数也并不能使算法更加接近最优解。如对 $N = 200, G = 200$ 迭代 5800 次，仍然不能给出一个较好的解




(a) $N = 100, G = 200, M = 1000$

(b) $N = 100, G = 200, M = 1000$

(c) $N = 200, G = 200, M = 5800$

(d) $N = 200, G = 200, M = 5800$

Figure 5: GA result for large $N$

## 4.5 GA 算法与 SA 算法的优劣对比

选取 $N = 100, G = 200, I = 600, M = 1000$，对较大的参数，可以认为具有一定的普遍性，两种算法运行结果如 Figure (6)

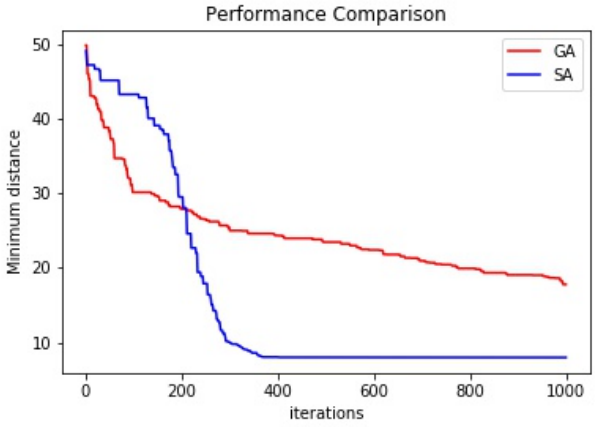



Figure 6: Comparison of GA and SA

可见，在前期 GA 算法有较快的收敛速度，但当达到足够的迭代次数后，SA 算法相比于 GA 算法会更快地接近最优值。此外，在本例的参数下，SA 算法也比 GA 算法有更快的运行速度。尽管这可能有代码优化的原因，但我们仍然可以认为 SA 算法是优于 GA 算法的。

# 5 GA 算法引入 Boltzman 生存机制的改进

为了改进 GA 算法的表现，我们尝试在其中融入 Boltzman 生存机制，即产生交叉、突变的子代时，根据适合度计算 $h = \min(1, e^{(new-old)/t})$, 其中 $new, old$ 分别为子代与亲代的适合度，$t$ 为当前温度，每次迭代时按照 $t = t * (1 - 1/(50 + \ln(iter + 1)))$,$iter$ 为迭代次数。思想即来源于 SA 算法。

但仅仅做出以上改变，会使得 GA 算法中的种群早熟，种群多样性减小，如 Figure (7)

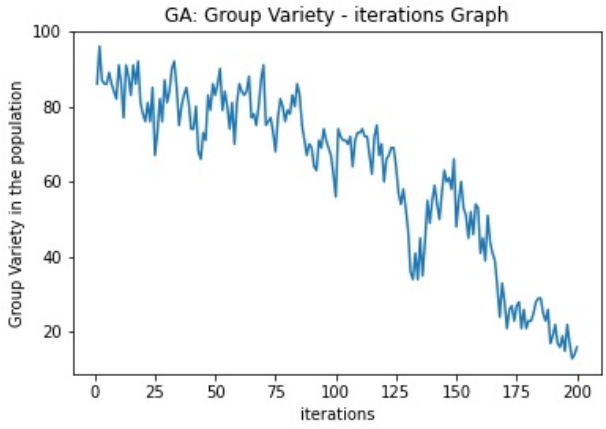
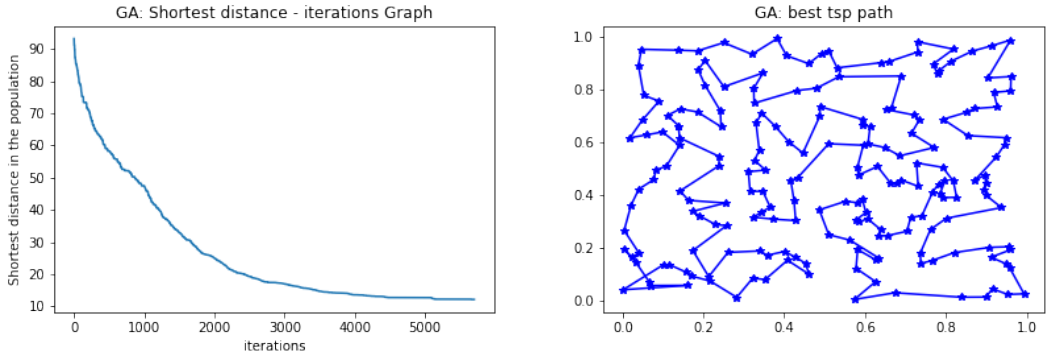



Figure 7: Group Variety in Improved GA Algorithm

这时需要我们引入防早熟的机制，这里采用最粗暴的方法，每次迭代时剔除种群中的重复个体，使用随机生成的个体去补全使得种群个数保持恒定。采用这个方法对 $N = 200, G = 200, M = 5800$ 得到的结果如 Figure (8)，对比 Figure (5(d)), 已经有较好的结果，但相比于 SA 算法给出的结果，仍然较劣。




(a) $N = 200, G = 200, M = 5800$ Improved GA (b) $N = 200, G = 200, M = 5800$ Improved GA

Figure 8: GA result for large $N$

# 6 结论与改进

## 6.1 GA 算法的改进

1. 在 GA 算法迭代过程中，总是执行固定的迭代次数。使得对不同的 $N$ 与 $G$ 需要人工调整 $M$ 以得到更好的解。如果设定提前终止策略，可以节约计算资源，并且可以察到不同 $N$,$G$ 对收敛到最优解的 $M$ 取值的影响。

## 6.2 遗传算法与退火算法的结合思路

在 Section (5) 中我们已经基于 SA 算法的想法对简单的 GA 算法进行了一些改进，此外仍有一些可考虑的改进方法 [3]

如**引入退火算子**：

在种群进行复制、交叉、变异前，对每一个个体以 SA 算法在全局中搜索（可以采用 SA 算法中的领域，在本文中实际上与一次变异算子效果相同），降温若干次后再进行复制、交叉。变异操作。每次退火会在上次的温度基础上继续退火。但在实现中这样的算法运行缓慢，且距离下降曲线更接近 SA 算法，主要是 SA 算法在主导，GA 算法似乎起不到较好的作用。

# 7 总结

**撰写本文过程中遇到的问题：**

1. 在按照讲义思路测试简单 GA 算法时，始终得不到收敛的结果，即种群的最短距离不会降低。这时尝试了使用自适应的 $p_r, p_c, p_m$，使用不同的交叉算子都无法解决，直到在选择算子中将轮盘赌抽取优化为锦标赛抽取才能得到一个收敛的结果。

2. 相比于 SA 算法，GA 算法在代码设计上更加复杂。在编写代码时前期没有规划好类的设计，导致后期匆忙增加了一些属性，如实例属性中记录每次迭代最优解的向量，记录迭代中种群多样性的向量，使得代码的鲁棒性较差，也较为臃肿

3. 向量化实现可以优化运行效率也让代码更加优雅，这里主要学习了 from python to numpy 这一网站，学习到了许多向量化的思想，优化了一些地方的向量化实现。如使用 numpy 中 apple_along_axis 函数，repeat 函数，使得代码更加简洁。

**收获：**

1. 熟练使用面向对象思想编写 GA 算法，SA 算法，以及测试模块。

2. 了解到常用的加快收敛的设计思路：如使用锦标赛法抽取，在抽取时使用退火的思想等。

3. 学习 from python to numpy, 更深刻体悟了向量化编程思想

**总结：**

我觉得本次作业是一次很好的机会来让数学系的同学们提升代码能力（我是 CS 双学位自然没问题）。毕竟在一些交叉方法的设计和整体逻辑的设计上还是略微复杂的，需要一些技巧。在先前的数院代码课中我感觉数院同学的 debug 能力十分欠缺，而在 GA 算法中，可能会出各种各样的问题，这样就需要我们更多的记录一些中间过程量打印出来以观察到底哪个环节出了问题。如记录每次迭代种群的多样性，种群的最低距离，若不去记录这些观察，我想抓破头也想不出究竟是算法收敛的慢，压根不收敛或甚至反向收敛（由于我们要最小化距离，在 boltzman 生存机制中若以距离为度量则需要调换 $new, old$ 的顺序，如果疏漏了就会导致反向收敛到距离最大值，我就犯过这样的错，但一观察最短距离变化曲线就很容易找到症结）综上，很多同学应该都在本次训练中学会自己解决问题，甚至更好地践行模块化编程的思想（否则改动起来十分麻烦）。

此外在本次作业中，对比之下更觉 SA 算法的优越性，在实现难度，运行时长和运行结果上都优于 GA 算法。也认识到近似算法的一些通病，如收敛的太慢，难跳出局部最优。

# References

[1] 易伟民, 申群泰. 整流机组效率优化中遗传算法的研究与应用 [J]. 贵州工业大学学报 (自然科学版), 2003.

[2] 朱凤龙. 遗传算法“早熟”现象的探究及改进策略 [D]. 西南大学,2010.

[3] 张晖, 吴斌, 余张国. 引入模拟退火机制的新型遗传算法 [J]. 电子科技大学学报,2003(01):39-42.

# 附录

## 代码

**由于在代码中中英文交杂在 Latex 中会有顺序错误出现，故建议直接观看 jupyter notebook 中的代码**

```
class tsp_genetic():
  def __init__(self, Dots, group_size = 1000, k = (0.5, 0.3, 0.25, 0.15), p_tuple
   =(0.4,0.4,0.2)):
      if Dots.shape[0] != 2:
          raise ValueError('输入的点并非二维! ')
      self.Dots = Dots # 点阵
      self.n = Dots.shape[1] # 点的数目
      # 三个初值 其实作用不大
      self.set_p(p_tuple)
      self.k = k # 自适应方法的参数
      self.min_rec = np.array([]) # 记录每一次迭代的最短距离
      self.mean_rec = np.array([]) # 记录每一次迭代的平均距离
      self.best_ans = np.array([]) # 记录最优解

      # 融入算法SA
      self.iter_per_t = 10

      self.iter = 1
      self.t = 10

      self.Graph = np.zeros((self.n,self.n)) # 距离矩阵
      # 生成距离矩阵 由于只需要在初始化时计算，就采用更直观的循环而非向量化实现了
      for i in range(self.n):
          for j in range(self.n):
              if i != j:
                  self.Graph[i][j] = self.Graph[j][i] =  np.sqrt( (Dots[0,i] -
   Dots[0,j])**2 + (Dots[1,i] - Dots[1,j])**2 )

      self.reset_group(group_size)

  # 重置概率
  def set_p(self, p_tuple):
      if np.sum(p_tuple) != 1:
          print(p_tuple)
          raise ValueError('三个概率的和不为1 ! ')
```

```
34
35          self.pr = p_tuple[0]
36          self.pc = p_tuple[1]
37          self.pm = p_tuple[2]
38
39      # 重置种群
40      def reset_group(self, group_size=None, const=False):
41          '''
42          group_size: number of groups
43          const: back to original group if True, else reset randomly and renew
        const_init_group
44          '''
45          if const:
46              self.Group = self.const_init_group
47          else:
48              if group_size:
49                  self.group_size = group_size
50              # 向量化实现生成种群
51              # 生成种群每行为一个,n- 排列
52              x = np.arange(self.n)
53              x = x.reshape(1,-1) # 变至二维的
54              self.Group = np.repeat(x, self.group_size, axis=0) # 复制若干次
55              np.apply_along_axis(np.random.shuffle, 1, self.Group) # 对每一行打乱
56
57              # 以下注释掉的为非向量化的方法
58              # self.Group = np.zeros((self.group_size, self.n)) # 每行为一个个体
59              # for i in range(self.group_size):
60              #     np.random.shuffle(x)
61              #     self.Group[i] = x
62              self.const_init_group = self.Group
63
64          self.min_rec = np.array([])
65          self.mean_rec = np.array([])
66          self.best_ans = np.array([])
67          self.group_variety = np.array([])
68
69          self.iter = 1
70          self.t = 10
71
72          self.distance = self.gen_distance(self.Group) # 记录当前种群每个个体对应的距离
73          self.fitness = self.gen_fitness(self.Group) # 记录当前种群每个个体的适合度
```

```
74
75
76  # 随机抽取
77  def rand_choose(self, pdf, times = 1):
78      if len(pdf) != self.group_size: # 需抽取的总数
79          raise ValueError('pdf 的长度与group_size 不一致')
80
81      pdf_normed = pdf/np.sum(pdf)
82      n = self.group_size
83      c = np.arange(n) # 抽取的值列表
84      # choice 的第三个参数为False 表明抽取不放回
85      return np.random.choice(c,times, False, p=pdf_normed.ravel())
86
87  # 计算距离
88  def gen_distance(self, paths):
89      if paths.shape[1] != self.n:
90          raise ValueError('输入的路径与矩阵维度不一致')
91
92      d = np.zeros(paths.shape[0])
93      for j in range(paths.shape[0]):
94          path = paths[j]
95          for i in range(self.n -1):
96              d[j] += self.Graph[int(path[i])][int(path[i+1])]
97          d[j] += self.Graph[int(path[-1])][int(path[0])]
98      return d
99
100
101 # 计算适合度
102 def gen_fitness(self, paths):
103     d = self.gen_distance(paths)
104     return np.max(d) - d
105
106
107 # 变异方法
108 def gen_mutation(self, path):
109     if len(path) != self.n:
110         raise ValueError('path 的长度与节点数量不同')
111     nodes = np.random.choice(self.n, 2, False)
112     node_1 = min(nodes)
113     node_2 = max(nodes)
```

```
114         return np.hstack((path[0: node_1], path[node_2:node_1:-1], path[node_1:
    node_1 + 1], path[node_2 + 1:]))
115 
116 
117     def mutation_over(self, times, SA):
118         # times 为变异的数量
119         idx = self.rand_choose(self.fitness ,times = times)
120         individuals = self.Group[idx]
121         individuals_after_mutation = np.zeros((times, self.n))
122         # 这里可以采用Apply 方法向量化实现，但我比较懒
123         for i in range(times):
124             individuals_after_mutation[i] = self.gen_mutation(individuals[i])
125 
126         if SA:
127             old_dist = self.gen_distance(individuals)
128             new_dist = self.gen_distance(individuals_after_mutation)
129             c = self.boltzman_choose(-old_dist, -new_dist, self.t) # 符号表明最小化距
    离
130             individuals_after_mutation[~c] = individuals[~c]
131 
132         return individuals_after_mutation
133 
134 
135     # 交叉方法
136     def gen_children(self, paths, method):
137         if paths.shape[0] != 2:
138             raise ValueError('超过两个路径的输入! ' + str(paths.shape[0]))
139         elif paths.shape[1] != self.n:
140             raise ValueError('路径长度与节点数目不同! ')
141 
142         nodes = np.random.choice(self.n, 2, False)
143         node_1 = min(nodes)
144         node_2 = max(nodes)
145         crossed_path = paths.copy()
146         # 三种交叉方法
147         if method == 'partial-mapped':
148             tsp_genetic.cross_partial_mapped(crossed_path[0], paths[1], node_1,
    node_2)
149             tsp_genetic.cross_partial_mapped(crossed_path[1], paths[0], node_1,
    node_2)
150 
```

```
151         elif method == 'order':
152             tsp_genetic.cross_order(crossed_path[0], paths[1], node_1, node_2)
153             tsp_genetic.cross_order(crossed_path[1], paths[0], node_1, node_2)
154 
155         elif method == 'position-based':
156             L = np.arange(self.n)
157             nodes = np.random.choice(L, self.n//2, False)
158             mask_1 = np.in1d(paths[0], nodes)
159             mask_2 = np.in1d(paths[1], nodes)
160             crossed_path[0][~mask_1] = paths[1][~mask_2]
161             crossed_path[1][~mask_2] = paths[0][~mask_1]
162 
163         else:
164             raise ValueError('wrong method!')
165 
166         return crossed_path
167 
168 
169     def cross_over(self, times, method, SA):
170         # 交叉得到的下一代
171         individuals_after_cross = np.zeros((times, self.n))
172         k = 0
173         parents_all = self.Group[self.rand_choose(self.fitness, times = times+1 if
     times%2 else times)]
174         while k < times:
175             # idx = self.rand_choose(self.fitness ,times = 2)
176             # parents = self.Group[idx]
177             parents = parents_all[k:k+2]
178             children = self.gen_children(parents, method=method)
179             individuals_after_cross[k] = children[0]
180             k += 1
181             if k == times:
182                 break
183             individuals_after_cross[k] = children[1]
184             k += 1
185         # 引入退火选择
186         if SA:
187             parents_all = parents_all[:times]
188             old_dist = self.gen_distance(parents_all)
189             new_dist = self.gen_distance(individuals_after_cross)
190             c = self.boltzman_choose(-old_dist, -new_dist, self.t)
```

```
            individuals_after_cross[~c] = parents_all[~c]

        return individuals_after_cross

    # 选择复制算子/
    def select_over(self, nr):
        # 2- 锦标赛法 两组进行竞赛选取表现最优的个体
        group1 = self.Group[self.rand_choose(self.fitness ,times = nr)]
        group2 = self.Group[self.rand_choose(self.fitness ,times = nr)]
        dist1 = self.gen_distance(group1)
        dist2 = self.gen_distance(group2)
        left = dist1 < dist2 # 选择距离较小，表现较好的
        return np.vstack((group1[left],group2[~left]))
        # return self.Group[self.rand_choose(self.fitness ,times = nr)] # 弃用的轮盘赌选择

    # # 在复制交叉突变前先进行退火
    # def SA_in_GA(self, SA=False):
    #     if SA:
    #         np.apply_along_axis(self.SA_per, 1, self.Group)
    #         self.distance = self.gen_distance(self.Group) # 记录当前种群每个个体对应的距离
    #         self.fitness = self.gen_fitness(self.Group) # 记录当前种群每个个体的适合度
    #     self.iter += 1
    #     self.t *= (1 - 1 / (50 + np.log(self.iter + 1))) # 更新温度


    # def SA_per(self, path):
    #     k = 0
    #     dist = self.gen_distance(path.reshape(1,-1))[0]
    #     while k < self.iter_per_t:
    #         new_path = self.gen_mutation(path)
    #         new_dist = self.gen_distance(new_path.reshape(1,-1))[0]
    #         if new_dist <= dist:
    #             path[:] = new_path[:]
    #         else:
    #             h = np.exp((dist - new_dist)/self.t)
    #             U = np.random.rand()
    #             if (U < h):
    #             # x = y.copy()
```

```
#                     path[:] = new_path[:]
#                     dist = new_dist # 这里优化了更新dx 的方法
#             k = k + 1
#         # self.t *= (1 - 1 / (50 + np.log(i + 1))) # 更新温度

    @staticmethod
    def boltzman_choose(old, new, t):
        if len(old) != len(new):
            raise ValueError('比较的向量长度不同 {}, {}'.format(len(old), len(new)))
        L = len(old)
        h = np.exp(-(old-new)/t)
        u = np.random.rand(L)
        return u<h

    # 生成下一代
    def reproduce(self, method, auto, SA, UPD):
        pt = np.array([self.pr, self.pc, self.pm])
        choose = np.random.choice(np.arange(3), self.group_size, True, p=pt)

        nr = np.sum(choose == 0)
        nc = np.sum(choose == 1)
        nm = np.sum(choose == 2)
        new_group = self.Group.copy()
        # self.SA_in_GA(SA=SA)
        # 复制得到的下一代
        new_group[:nr] = self.select_over(nr)
        # 交叉得到的
        new_group[nr: nr+nc] = self.cross_over(nc, method=method, SA=SA)
        # 突变得到的
        new_group[nr+nc:] = self.mutation_over(nm, SA=SA)

        new_distance = self.gen_distance(new_group)
        new_fitness = self.gen_fitness(new_group)

        if auto:
            # 自适应更改pc pm pr
            d_max = np.max(new_distance)
            d_min = np.min(new_distance)
            d_mean = np.mean(new_distance)

            dc_new = np.sum(new_distance[nr: nr+nc]) / nc if nc != 0 else d_mean
```

```
            dm_new = np.sum(new_distance[nr+nc: ]) / nm if nm != 0 else d_mean
            # print(dc_new, dm_new, d_max, d_min, d_mean)
            self.pc = self.k[0] if dc_new >= d_mean else self.k[0]-self.k[1]*(dc_new - d_min)/(d_mean - d_min)
            self.pm = self.k[2] if dm_new >= d_mean else self.k[2]-self.k[3]*(dm_new - d_min)/(d_mean - d_min)
            self.pr = 1 - self.pc - self.pm

        # update
        # 强制更新种群 防早熟
        if UPD:
            unique_group = np.unique(new_group, axis=0)
            # mask = np.in1d(self.Group.view(dtype=','.join(['i']*self.n)), unique_group.view(dtype=','.join(['i']*self.n)))
            # temp_fit = self.fitness.copy()
            # print(temp_fit)
            # temp_fit[mask] = 0
            # print(mask)
            # print(temp_fit)
            # add_group = self.Group[self.rand_choose(temp_fit, self.group_size - b.shape[0])]
            x = np.arange(self.n)
            x = x.reshape(1,-1) # 变至二维的
            if self.group_size == unique_group.shape[0]:
                self.Group = unique_group
            else:
                add_group = np.repeat(x, self.group_size - unique_group.shape[0], axis=0) # 复制若干次
                np.apply_along_axis(np.random.shuffle , 1, add_group) # 对每一行打乱
                self.Group = np.vstack((unique_group, add_group))
        else:
            self.Group = new_group

        self.fitness = self.gen_fitness(self.Group)
        self.distance = self.gen_distance(self.Group)


# 遗传算法主体
# 由于会更新实例内的Group 故可以通过不断evolute 在原有基础上进行再次迭代
```

```
304 def evolute(self, generations, method = 'partial-mapped', auto=True, SA=False,
     UPD=True):
305     start_time = time.time()
306     record = np.zeros(generations)
307     mean_dist = np.zeros(generations)
308     group_varieties = np.zeros(generations)
309     dist_best = self.min_rec[-1] if self.min_rec.shape[0] != 0 else 65535
310     individual_best = self.best_ans if self.best_ans.shape[0] else np.zeros(
    self.n)
311     percentage = 0.05
312     for i in range(generations):
313         self.reproduce(method = method, auto=auto, SA=SA, UPD=UPD)
314         if np.min(self.distance) < dist_best:
315             dist_best = np.min(self.distance)
316             individual_best = self.Group[np.where(self.distance == dist_best)
    [0][0]]
317         record[i] = dist_best
318         mean_dist[i] = np.mean(self.distance)
319         b = np.unique(self.Group, axis=0)
320         group_varieties[i] = b.shape[0] # 更新种群多样性
321 
322         if i >= percentage * generations:
323             print('{0:02.0f}%'.format(percentage*100), end=' ')
324             percentage += 0.05
325 
326         # 更新退火温度
327         if SA and i%self.iter_per_t:
328             self.iter += 1
329             self.t *= (1 - 1 / (50 + np.log(self.iter + 1))) # 更新温度
330 
331         # 防早熟策略在reproduce 中
332 
333 
334     end_time = time.time()
335     self.min_rec = np.append(self.min_rec, record)
336     self.mean_rec = np.append(self.mean_rec, mean_dist)
337     self.group_variety = np.append(self.group_variety, group_varieties)
338     self.best_ans = individual_best
339     print('\nrun time = {}'.format(end_time - start_time))
340     return record, mean_dist, individual_best.astype(int)
341 
```

```
342
343 @staticmethod
344 def cross_partial_mapped(path_1, path_2, node_1, node_2):
345     # 传入可变对象直接修改
346     if node_1 > node_2:
347         raise ValueError('节点顺序反啦！')
348     if len(path_1) != len(path_2):
349         raise ValueError('path 长度不一致！')
350
351     # cross_part_1 = path_1[node_1 : node_2 + 1].copy()
352     cross_part_2 = path_2[node_1 : node_2 + 1].copy()
353     # node_len = node_2 - node_1 + 1
354
355     for i in range(len(path_1)):
356         if not (node_1 <= i <= node_2):
357             this_node = path_1[i]
358             while this_node in cross_part_2:
359                 idx = np.where(path_2 == this_node)[0][0]
360                 this_node = path_1[idx]
361             path_1[i] = this_node
362
363     path_1[node_1: node_2 + 1] = path_2[node_1: node_2 + 1]
364
365
366 @staticmethod
367 def cross_order(path_1, path_2, node_1, node_2):
368     # 传入可变对象直接修改
369     if node_1 > node_2:
370         raise ValueError('节点顺序反啦！')
371     if len(path_1) != len(path_2):
372         raise ValueError('长度不一致！path')
373
374     cross_part_1 = path_1[node_1 : node_2 + 1].copy()
375     # setdiff1d 会自动排序
376     # left_part_1 = np.setdiff1d(path_2, cross_part_1)
377     # 使用掩码解决自动排序的问题
378     left_part_1 = path_2[~np.in1d(path_2,cross_part_1)]
379     path_1[:node_1] = left_part_1[:node_1]
380     path_1[node_2 + 1 :] = left_part_1[node_1 :]
381
382
```

```
383  # 绘图函数
384  def plot_cities(self, path=None, savepath=None):
385      if path is None:
386          path = self.best_ans
387      if len(path) != self.n:
388          raise ValueError('路径长度错误! ')
389      plt.figure()
390      for i in range(self.n - 1):
391          plt.plot( self.Dots[0][[path[i], path[i+1]]], self.Dots[1][[path[i],
      path[i+1]]], 'b*-')
392      plt.plot(self.Dots[0][[path[-1], path[0]]], self.Dots[1][[path[-1], path
     [0]]], 'b*-')
393      plt.title('GA: best tsp path')
394      if savepath:
395          plt.savefig(savepath)
396      plt.show()
397
398  def plot_record(self, type=1, savepath=None):
399      if type == 1:
400          y, title, ylabel = (self.min_rec, 'GA: Shortest distance - iterations
      Graph', 'Shortest distance in the population')
401      elif type == 2:
402          y, title, ylabel = (self.mean_rec, 'GA: Average distance - iterations
      Graph', 'Average distance in the population')
403      else:
404          y, title, ylabel = (self.group_variety, 'GA: Group Variety - iterations
       Graph', 'Group Variety in the population')
405
406      n = y.shape[0]
407      plt.figure()
408      plt.plot(np.arange(1, n+1), y)
409      plt.title(title)
410      plt.xlabel('iterations')
411      plt.ylabel(ylabel)
412      if savepath:
413          plt.savefig(savepath)
414      plt.show()
```

Listing 2: GA 算法计算 TSP 的类

```
1  class tsp_anneal():
2  def __init__(self, city, t0=20, maxit=2000, iter_time=600):
```

```
        self.city = city
        self.n = dots.shape[1]
        self.set_params(t0, maxit, iter_time)
        # self.ans

    def set_params(self, t0, maxit, iter_time):
        self.t0 = t0
        self.maxit = maxit
        self.iter_time = iter_time


    def tspsa(self, params=None):
        # 在有些不需要copy 的地方，只传地址会更快
        if params:
            self.set_params(*params)
        k = 0
        t = self.t0
        x = self.city
        dx = self.distance(x)
        ds = dx
        min_dist = np.zeros(self.iter_time)
        # run_time = np.zeros(self.iter_time)
        start_time = time.time()
        percent = 0.05
        for i in range(self.iter_time):
            while (k < self.maxit):
                if (dx < ds):
                    xs = x
                    ds = dx
                y = self.swapcities(x)
                dy = self.distance(y)
                h = min(1, np.exp(-(dy - dx)/t))
                U = np.random.rand()
                if (U < h):
                    # x = y.copy()
                    x = y
                    dx = dy # 这里优化了更新dx 的方法
                k = k + 1
            t *= (1 - 1 / (50 + np.log(i + 1)))
            k = 0
            min_dist[i] = ds
```

```
            if i > percent * self.iter_time:
                print('{0:02.0f}%'.format(percent*100), end=' ') # 打印以追踪进度
                percent += 0.05
            # mid_time = time.time()
            # run_time[i] = mid_time - start_time

        end_time = time.time()
        print('\nrun time: {0:}'.format(end_time - start_time))
        self.best_ans = xs
        self.min_dist = min_dist
        # self.run_time = run_time
        return min_dist, xs


    def swapcities(self, cityXY):
        nodes = np.random.choice(self.n, 2, False)
        city_1 = min(nodes)
        city_2 = max(nodes)

        s = np.hstack((cityXY[:,0: city_1], cityXY[:,city_2:city_1:-1], cityXY[:,city_1:city_1 + 1], cityXY[:,city_2 + 1:]))
        # 这里使用了加一使得得到的矩阵的列宽为1, 否则使用.shape 会输出空的列宽，可以防止hstack 报错 'dimention 不一致'
        return s


    @staticmethod
    def dis2(x, y):
        return np.sqrt( (x-y)[0] * (x-y)[0] + (x-y)[1] * (x-y)[1]  )


    @staticmethod
    def distance(city):
        dd = tsp_anneal.dis2(city[:,:-1], city[:, 1:])
        d = np.sum(dd)
        #使用向量化可以极大地提高计算速度
        d += tsp_anneal.dis2(city[:, -1], city[:, 0])
        return d


    def plotcities(self, city=None, savepath=None):
```

```
    cityXY = city if city else self.best_ans
    try:
        plt.figure()
        plt.plot(cityXY[0, :], cityXY[1, :], 'b*')
        plt.plot(cityXY[0, :], cityXY[1, :], 'b')
        plt.plot([cityXY[0, -1:], cityXY[0, 0]], [cityXY[1, -1:], cityXY[1, 0]], 'b')
        plt.title('SA: best tsp path')
        if savepath:
            plt.savefig(savepath)
        plt.show()
    except:
        raise ValueError('no figure to plot')

def plot_record(self, savepath=None):
    y = self.min_dist
    n = y.shape[0]
    x = np.arange(1, n+1)
    plt.figure()
    plt.plot(x, y, 'r-')
    plt.xlabel('iterations')
    plt.ylabel('Minimum distance')
    plt.title('SA: Minimum distance - iterations Graph')
    if savepath:
        plt.savefig(savepath)
    plt.show()
```

Listing 3: SA 算法计算 TSP 问题的类

```
class testmodule():
def __init__(self, dots=None, size=None):
    '''
    size has a higher priviledge than dots
    '''
    self.tsp_GA = None
    self.tsp_SA = None
    if size:
        self.set_bysize(size)
    elif dots:
        self.set_bydots(dots)

def set_bydots(self, dots, obj='ALL'):
```

```
14      if obj == 'ALL' or obj == 'GA':
15          self.tsp_GA = tsp_genetic(dots, group_size = 1)
16      if obj == 'ALL' or obj == 'SA':
17          self.tsp_SA = tsp_anneal(dots)
18
19  def set_bysize(self, tsp_size, obj='ALL'):
20      dots = np.random.rand(2, tsp_size)
21      if obj == 'ALL' or obj == 'GA':
22          self.tsp_GA = tsp_genetic(dots, group_size = 1)
23      if obj == 'ALL' or obj == 'SA':
24          self.tsp_SA = tsp_anneal(dots)
25
26  def GA_cross_method_test(self, group_size, iterations, savepath=None):
27      methods = ['partial-mapped', 'order', 'position-based']
28      record = np.zeros((3, iterations))
29      self.tsp_GA.reset_group(group_size)
30
31      for i in range(3):
32          self.tsp_GA.reset_group(const=True)
33          record[i] = self.tsp_GA.evolute(iterations, method=methods[0])[0]
34      x = np.arange(1, iterations+1)
35      plt.figure()
36      plt.plot(x, record[0], 'r-', label=methods[0])
37      plt.plot(x, record[1], 'g-', label=methods[1])
38      plt.plot(x, record[2], 'b-', label=methods[2])
39      plt.legend()
40      plt.xlabel('iterations')
41      plt.ylabel('Minimum distance')
42      plt.title('Comparison of three cross-over methods')
43      if savepath:
44          plt.savefig(savepath)
45      plt.show()
46
47
48  def GA_auto_test(self, group_size, iterations, savepath=None):
49      self.tsp_GA.reset_group(group_size)
50
51      record1 = self.tsp_GA.evolute(iterations, auto=True)[0]
52      self.tsp_GA.reset_group(const=True)
53      record2 = self.tsp_GA.evolute(iterations, auto=False)[0]
54
```

```
55      x = np.arange(1, iterations+1)
56      plt.figure()
57      plt.plot(x, record1, 'r-', label='auto = True')
58      plt.plot(x, record2, 'g-', label='auto = False')
59      plt.legend()
60      plt.xlabel('iterations')
61      plt.ylabel('Minimum distance')
62      plt.title('Comparison of auto-p methods')
63      if savepath:
64          plt.savefig(savepath)
65      plt.show()
66
67  def GA_perf_test(self, group_size, iterations, tsp_size=None, savefolder=None,
     merge=False, mergesave=None, SA=False):
68      '''
69      input params must be lists
70      '''
71      if len(group_size) != len(iterations):
72          raise ValueError('dimention does not match!')
73      if tsp_size:
74          if len(group_size) != len(tsp_size):
75              raise ValueError('dimention for tsp_size does not match!')
76      for i in range(len(group_size)):
77          if tsp_size is not None:
78              if self.tsp_GA is None:
79                  self.set_bysize(tsp_size[i], 'GA')
80              elif tsp_size[i] != self.tsp_GA.n:
81                  self.set_bysize(tsp_size[i], 'GA')
82
83          if group_size[i] != self.tsp_GA.group_size:
84              self.tsp_GA.reset_group(group_size[i])
85          else:
86              self.tsp_GA.reset_group(const=True)
87
88          min_rec, mean_rec, res = self.tsp_GA.evolute(iterations[i], SA=SA)
89
90          postfix = str(i+1)
91          SAfix = 'SA' if SA else ''
92
93          x = np.arange(1, iterations[i]+1)
94          if not merge:
```

```
95              postfix = ''
96              plt.figure()
97          plt.plot(x, min_rec, label='min distance' + postfix)
98          # 注释掉下行可以取消输出mean_rec
99          # plt.plot(x, mean_rec, label='avg distance' + postfix)
100         plt.xlabel('iterations')
101         plt.ylabel('Distance')
102         plt.legend()
103         plt.title('GA ' + SAfix + ' algorithm performance')
104         if not merge:
105             if savefolder:
106                 plt.savefig(savefolder + 'GA' + SAfix + '_perf_{}_{}_{}.jpg'.
    format(tsp_size[i], group_size[i], iterations[i]))
107             plt.show()
108             self.tsp_GA.plot_cities(path=res, savepath=savefolder + 'GA' +
    SAfix + '_res_{}_{}_{}.jpg'.format(tsp_size[i], group_size[i], iterations[i])
     if savefolder else None)
109
110     if merge:
111         if mergesave:
112             plt.savefig(mergesave)
113         plt.show()
114
115
116 def compare_GASA_test(self, group_size, iterations, savepath=None):
117     self.tsp_GA.reset_group(group_size)
118     record_GA, mean_rec_GA, res_GA = self.tsp_GA.evolute(iterations)
119     record_SA, res_SA = self.tsp_SA.tspsa(params=(10, 600, iterations))
120     self.tsp_GA.reset_group(const=True)
121     record_GASA, mean_rec_GASA, res_GASA = self.tsp_GA.evolute(iterations, SA=
    True)
122     x = np.arange(1, iterations+1)
123
124     plt.figure()
125     plt.plot(x, record_GA, 'r.-', label='GA')
126     plt.plot(x, record_SA, 'b.-', label='SA')
127     plt.plot(x, record_GASA, 'g.-', label='GA promoted by SA')
128     plt.xlabel('iterations')
129     plt.ylabel('Minimum distance')
130     plt.legend()
131     plt.title('Performance Comparison')
```

```
132         if savepath:
133             plt.savefig(savepath)
134         plt.show()
```

Listing 4: 测试模块代码