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	22336326	姓名	朱禹溪

一、 实验题目

高级搜索

1.遗传算法解决TSP问题

为了方便批改作业，我们统一用类 `GeneticAlgTSP` 来编写遗传算法的各个模块，并分析算法性能. 该类需包含以下方法:

- 构造函数 `__init__()`，输入为TSP数据集文件名 `filename`，数据类型 `str`. 例如 `"dj38.tsp"` 是Djibouti的38个城市坐标数据文件; `"ch71009.tsp"` 是China的71009个城市坐标数据文件. 我们需要在构造函数中读取该文件中的数据，存储到类成员 `self.cities` 中(数据类型自定，建议存储为 `numpy` 数组). 同时在构造函数中初始化种群，存储到类成员 `self.population` 中(数据类型自定).
- 求解方法 `iterate()`，输入为算法迭代的轮数 `num_iterations`，数据类型 `int`. 该方法是基于当前种群 `self.population` 进行迭代(不是从头开始)，返回迭代后种群中的一个较优解，数据类型 `list`，格式为1-n个城市编号的排列. 例如，对于n=5的TSP问题，迭代后返回的较优解形如 `[1,3,4,5,2]`，表示当前较好的游览城市次序为1-3-4-5-2-1.

可以在类中编写其他方法以方便编写并分析遗传算法的性能. 请在代码注释或实验报告中说明每个方法/模块的功能.

二、 实验内容

1. 算法原理

遗传算法：

初始化（Initialization）：

从给定文件中读取城市信息，将城市坐标存储在 citys 中，并初始化一个空的种群 population 用于容纳城市路径。

对于种群的初始化，先随机生成一半个体（50），然后用贪心算法生成另一半（）50。每个个体代表了一种城市的访问路径。

评估函数（Evaluation Function）：

定义了计算路径总距离的函数 distance，用于评估城市路径的适应度。

通过 evaluate 方法计算给定路径的总距离，即为该路径的适应度评价。

选择：

使用锦标赛选择 tournament_selection 方法，从当前种群中选择一定数量的个体参与比赛，然后选择适应度最佳的个体作为下一代的父代。

交叉：

采用顺序交叉（OX）操作 reproduce_ox，从两个父代个体中随机选择一段基因片段，并将另一父代中不包含该片段的基因按序填充生成两个子代。

变异：
定义了三种不同的变异方法根据一定的概率进行基因的变异操作，以增加种群的多样性。
迭代：
在 iterate 方法中，进行多轮迭代优化处理。
每轮迭代中，选择父代、进行交叉、变异操作，更新种群，直到迭代次数达到指定值。
每隔一定迭代次数，绘制出当前最佳路径对应的城市访问顺序的动态图，并显示当前最短路径长度。
最终找出种群中适应度最佳的个体作为最优解，并输出最短路径长度和城市访问顺序。
可视化：
使用 matplotlib.pyplot 库绘制动态图展示迭代过程中的优化情况，以及绘制费用变化的图表。

2. 关键代码展示（可选）

```
    def evaluate(self, path):
        s = 0
        for i in range (1,len(path)):
            s += self.distance(path[i-1], path[i])
        s += self.distance(1,path[len(path)-1])
        return s

    #交叉下一代
    def reproduce_ox(self, parent1, parent2):
        length = len(parent1)
        sta,end = random.sample(range(1, length), 2)
        if end < sta:
            sta,end = end,sta

        child = parent1[sta:end]
        child1 = []
        child2 = []
        for i in range(0, length):
            if parent2[i] not in child:
                if i<sta:
                    child1.append(parent2[i])
                else:
                    child2.append(parent2[i])
        child = child1 + child + child2
        return child
```

```python
    def mutate(self, path):
        lenth = len(path)
        charge = random.random()
        if charge < 0.4:
            a1,a2,a3 = random.sample(range(1, lenth-1), 3)
            if a1>a2:
                a1,a2 = 2,a1
            if a2>a3:
                a2,a3 = a3,a2
            if a1>a2:
                a1,a2 = a2,a1
            tmp = path[0:a1]+path[a2:a3]+path[a1:a2]+path[a3:lenth]
        elif charge<0.6:
            i = random.randint(1,lenth-2)
            j = random.randint(2,lenth-1)
            #print(i,"<->",j)
            #print(path)
            if i != j:
                path[i],path[j] = path[j],path[i]
                tmp = path[:]
                path[i],path[j] = path[j],path[i]
            else:
                tmp = path[:]
        else:
            k1,k2 = random.sample(range(1, lenth-1), 2)
            if k1 > k2:
                k1,k2 = k2,k1
            tmp = path[0:k1]+path[k1:k2][::-1]+path[k2:lenth]
            #print(len(tmp))
        return tmp
```

```python
9   class GeneticAlgTSP:
126     def iterate(self, num):
127         start_time = time.time()
128         min = 99999999
129         for var in self.population:
130             if min > self.evaluate(var):
131                 min = self.evaluate(var)
132         print(f'种群最小路径: {min}')
133         print("----------------------------------------")
134         ims = []
135         fig1 = plt.figure(1)#用于生成动态图
136         dis_change = []
137         iteration = 0 #迭代次数
138         while iteration < num:
139             new_population = [] #新种群
140             for count in range(0,10): #杂交10次
141                 #选择亲代
142                 ch1 = self.tournament_selection(self.population, 5)
143                 ch2 = self.tournament_selection(self.population, 5)
144                 #杂交出两个子代
145                 child1 = self.reproduce_ox(ch1, ch2)
146                 child2 = self.reproduce_ox(ch2, ch1)
147                 #有一定概率变异
148                 if random.random() < self.mutation :
149                     child1 == self.mutate(child1)
150                     child2 == self.mutate(child2)
151                 #为了防止子代全都一样，当父代相等时进行变异
152                 if child1 == child2:
153                     child1 = self.mutate(child1)
154                     child2 = self.mutate(child2)
```

```
                child2 = self.mutate(child2)
            #并入新种群
            new_population.append(child1)
            new_population.append(child2)
        flag=-1
        max=9999999
        #选取最优良的个体保存到下一代
        for i2 in range(0, len(self.population)):
            if self.evaluate(self.population[i2]) < max:
                max = self.evaluate(self.population[i2])
                flag=i2
        temp_one = self.population[flag][:]
        #显示当前迭代次数和最优值
        print(iteration, self.evaluate(temp_one))
        dis_change.append(max)
        self.population.clear()
        #更新
        self.population = new_population[0:19]
        new_population.clear()
        self.population.append(temp_one)
        iteration += 1
        #每隔一段时间进行采样
        if iteration%10==0:
            x1=[]
            y1=[]
            x1.append(self.citys[0][1])
            y1.append(self.citys[0][2])
            for var in temp_one:
```

```
                x1.append(self.citys[var-1][1])
                y1.append(self.citys[var-1][2])
            x1.append(self.citys[0][1])
            y1.append(self.citys[0][2])
            im = plt.plot(x1, y1, marker = '.', color = 'red',linewidth=1)
            ims.append(im)
    elapsed_time = time.time() - start_time
    print(f'After {iteration} iterations, elapsed time: {elapsed_time/60:.2f} minutes')
    #结束杂交
    print("loop end")
    #找出最优解
    flag=-1
    max=9999999
    for i in range(0, len(self.population)):
        if self.evaluate(self.population[i]) < max:
            min = self.evaluate(self.population[i])
            flag = i
    curr = self.population[flag]
    xo=[]
    yo=[]
    xo.append(self.citys[0][1])
    yo.append(self.citys[0][2])
    for var in curr:
        xo.append(self.citys[var-1][1])
        yo.append(self.citys[var-1][2])
    xo.append(self.citys[0][1])
    yo.append(self.citys[0][2])
    print(f'最短路径长度:{self.evaluate(curr)}')
```

```
        print(f'城市的访问次序：{curr}')

        fig2 = plt.figure(2)#用于显示最终的解
        plt.plot(xo, yo, marker = '.', color = 'red',linewidth=1)
        #保存动态图
        ani = animation.ArtistAnimation(fig1, ims, interval=200, repeat_delay=1000)
        ani.save("TSP.gif", writer='pillow')

        fig3 = plt.figure(3)#用于显示总共费用的降低过程
        plt.title('the evolution of the cost')
        x_=[i for i in range(len(dis_change))]
        plt.plot(x_,dis_change)
        plt.show()

        print("End of all, and you got the best answer!")
        return self.evaluate(curr),elapsed_time
```

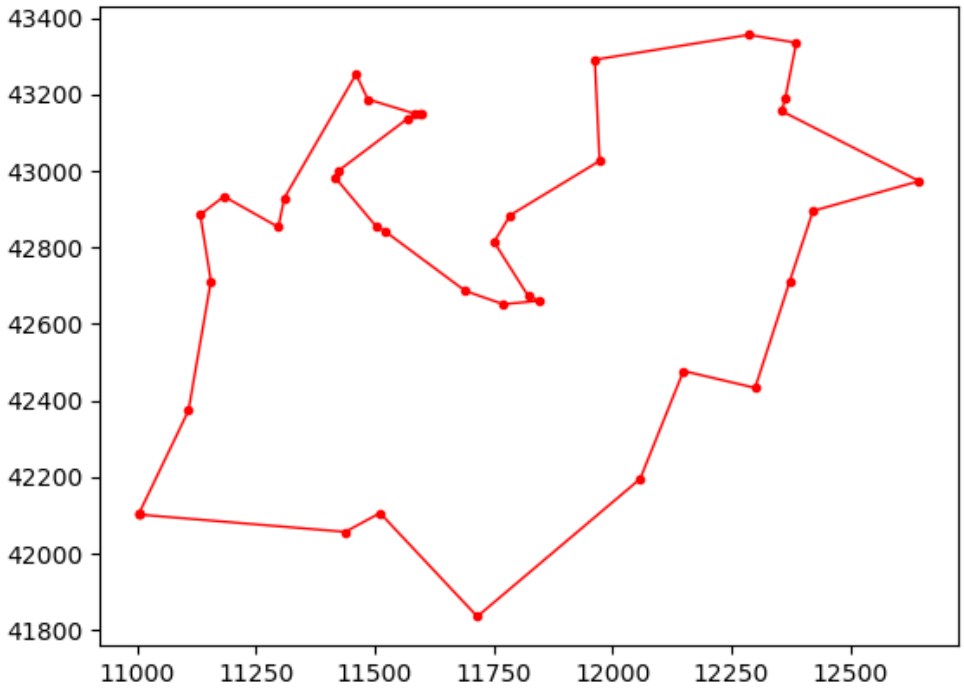
三、 实验结果及分析

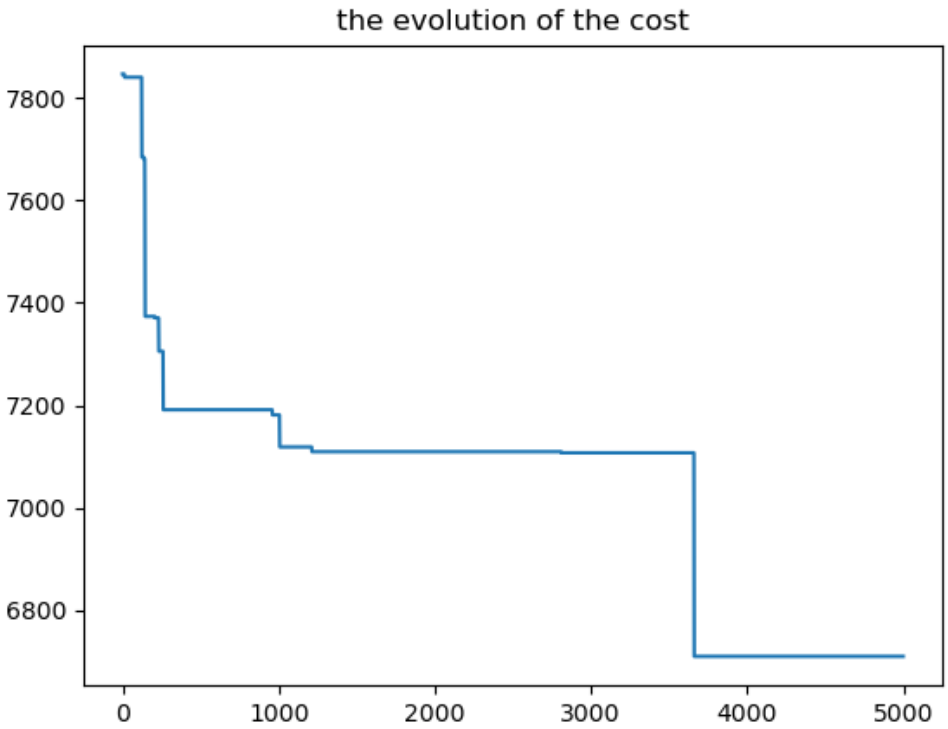
1. 实验结果展示示例（可图可表可文字，尽量可视化）

dj38 网页最好结果 6656 个人输出最好结果 6659 由于时间较短故没有进行平均时间的计算。

```
main.py > main
def main():
    filename = 'qa194.txt'

    num_iterations = 50000
    result=[]
    result_time=[]
    for i in range(30):
        TSP = GeneticAlgTSP(filename)
        num,time=TSP.iterate(num_iterations)
        result.append(num)
        result_time.append(time)
    print("The interation  is",num_iterations)
    print("eacch case distance is")
    print(result)
    print("each case time is\n",result_time)
    print("The average distance is",sum(result)/len(result))
    print("The average time is",sum(result_time)/len(result_time))
    # a=[69.05011367797852, 65.33138966560364, 66.65562510490417, 69.88337349891663, 65.39635181427002, 65.6269884109497, 65.77681827545166, 67.2089295387268,
    # print(sum(a)/len(a))
if __name__ == '__main__':
    main()
```

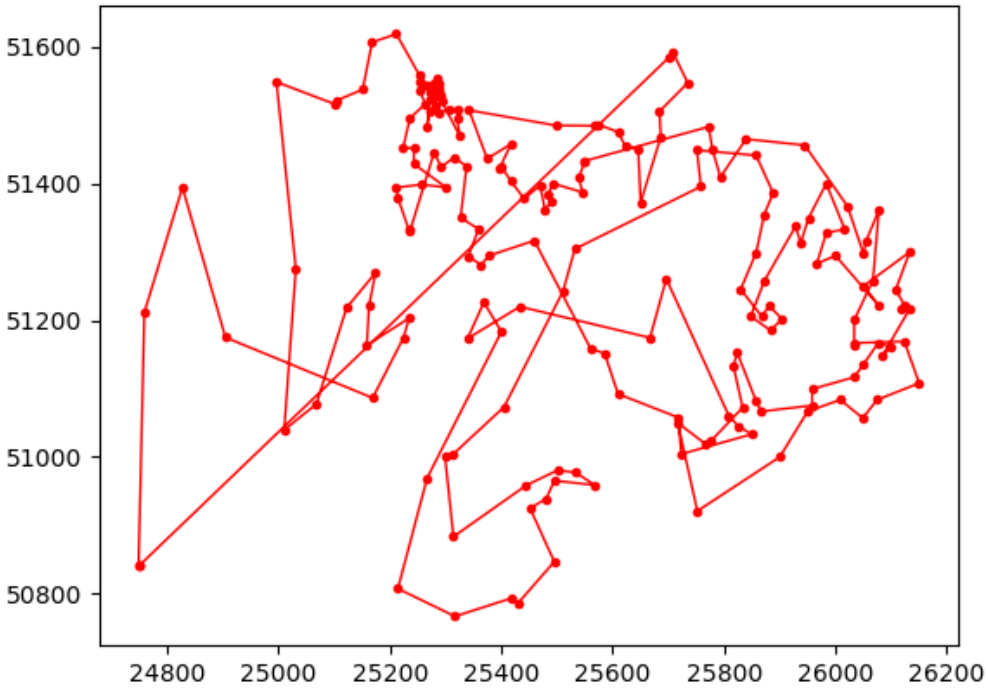
```
问题  输出  调试控制台  终端  端口

城市的访问次序：[29, 30, 32, 35, 37, 38, 33, 34, 36, 31, 27, 28, 24, 22, 25, 26, 23, 20, 15, 13, 16, 17, 18, 19, 11, 12, 9, 8, 7, 6, 5, 3, 4, 2, 14, 21, 10, 1]
The interation  is 20000
eacch case distance is
[6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.4315329314
67, 7221.756502411423, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 7080.949899191818, 6659.4315329
31467, 6773.75378145367, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6659.431532931467, 6810.967444875683, 7373.29241
4355641]
each case time is
 [62.12092137336731, 62.07112169265747, 62.24548387527466, 63.28740429878235, 62.017418384552, 62.53011775016785, 62.87351655960083, 62.606407165527344, 61.900236845016
48, 60.87083911895752, 60.54590153694153, 61.3172824382782, 61.43539309501648, 62.481812715530396, 63.12489628791809, 63.161805868148804, 62.983837604522705, 62.2763702
8694153, 62.378233909606934, 64.21692442893982, 63.467591524124146, 65.64322757720947, 104.34567904472351, 106.27296924591064, 108.23647475242615, 106.69325280189514, 1
05.33081150054932]
The average distance is 6732.156065436314
The average time is 70.60873821046617

(base) C:\Users\朱禹溪\Desktop\lab6>
```

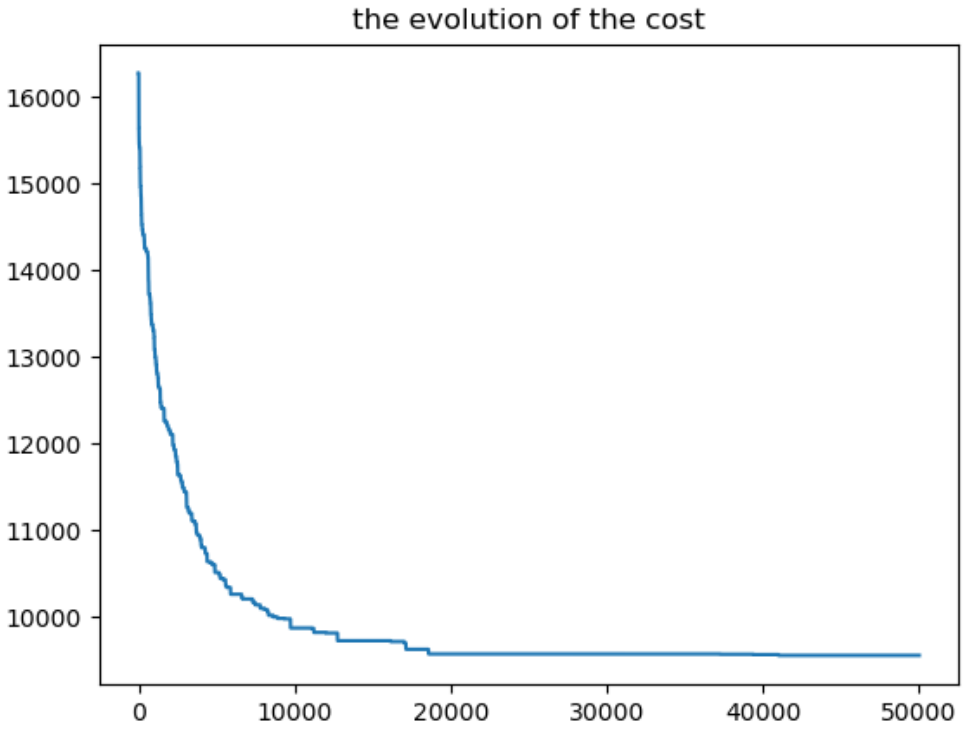

43400
43200
43000
42800
42600
42400
42200
42000
41800
11000
11250
11500
11750
12000
12250
12500


the evolution of the cost
7800
7600
7400
7200
7000
6800
0
1000
2000
3000
4000
5000

qa194 网页最好结果 9352 个人最好结果 9455 平均运行时间 13 分钟（30 次，5 万次迭代）

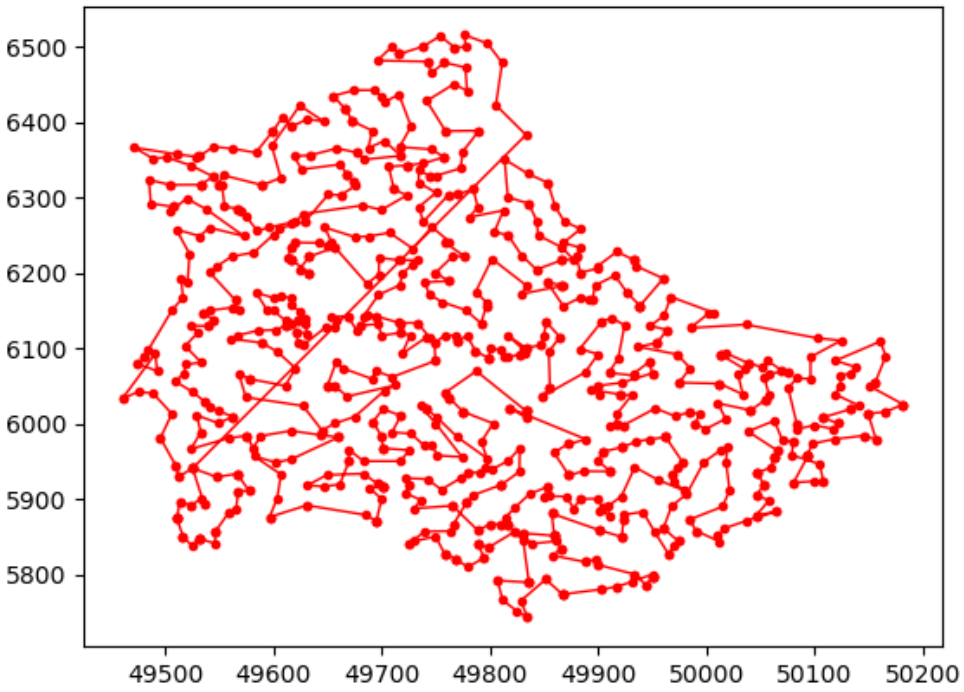
```
问题  输出  调试控制台  终端  窗口

1, 9727.256105400076, 10112.10790482233, 9698.950217663834, 9592.862191436236, 9949.094413246343, 10131.604385130688, 9935.491979177694, 9455.124703935926, 9877.4709402
3281, 9850.685889834962, 9860.162234388845, 9490.030445109942]
each case time is
 [769.2060782909393, 851.9982528686523, 769.7087507247925, 762.066339969635, 770.180469751358, 768.2608637809753, 773.6228218078613, 766.771312713623, 768.3511571884155
, 764.3019578456879, 757.9616260528564, 753.4453837871552, 750.2614254951477, 755.2078754901886, 755.7822394371033, 754.0350277423859, 764.4806640148163, 765.0359389781
952, 757.8049452304 84, 742.0400335788727, 689.8170981407166, 690.4293167591095, 697.8398067951202, 746.583410024643, 856.2770857810974, 856.9985098838806, 824.712430477
1423, 841.612245798111, 865.6628425121307, 864.5610814094543]
The average distance is 9709.255550321825
The average time is 775.1672330776851

(base) C:\Users\朱勇派\Desktop\lab6>
```

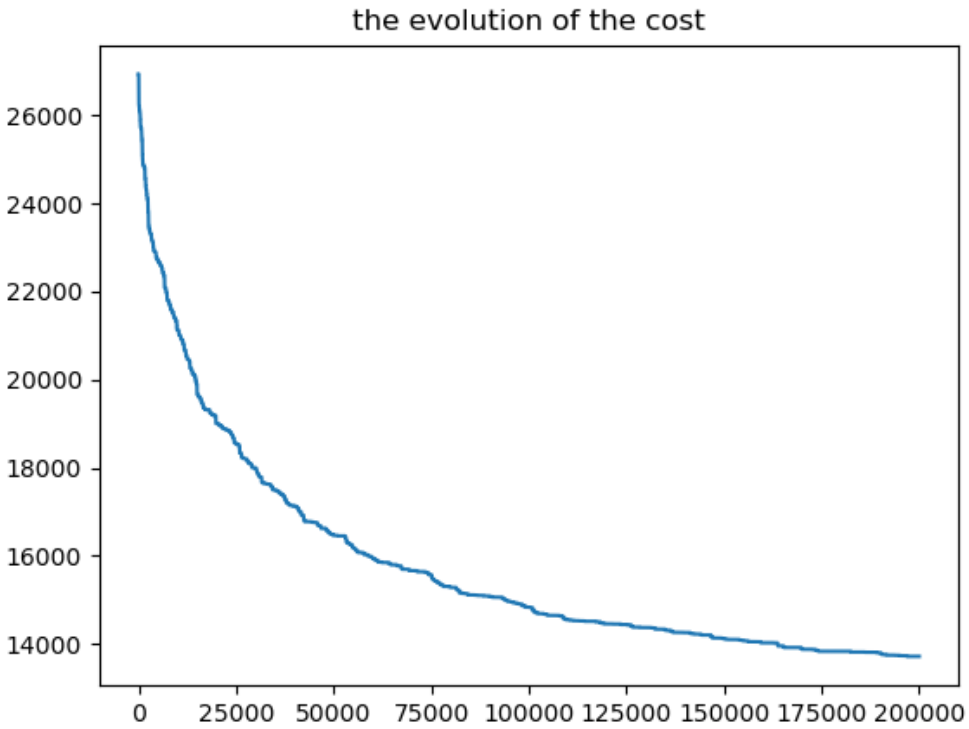




Lu980 网页最好结果 11340 个人最好结果 13721 平均时间 8.5h(迭代 20 万次)

```
After 200000 iterations, elapsed time: 412.99 minutes
loop end
最短路径长度:13721.479356370719
城市的访问次序: [278, 927, 921, 730, 731, 925, 254, 650, 634, 631, 924, 30, 70, 377, 419, 900, 411, 926, 187, 637, 653, 919, 191, 418, 417, 178, 237, 238, 393, 896, 279
, 255, 538, 549, 374, 291, 508, 153, 155, 1, 111, 2, 880, 878, 879, 684, 673, 668, 144, 450, 88, 162, 161, 783, 785, 467, 923, 711, 710, 939, 521, 370, 607, 540, 906, 9
12, 629, 743, 341, 37, 698, 312, 651, 742, 691, 780, 693, 695, 9, 208, 638, 12, 680, 677, 36, 13, 200, 148, 125, 126, 887, 940, 441, 199, 107, 570, 704, 473, 688, 4, 76
8, 715, 281, 259, 920, 402, 774, 147, 942, 146, 807, 179, 177, 905, 62, 184, 181, 182, 82, 675, 306, 248, 247, 129, 75, 188, 373, 882, 666, 664, 796, 790, 791, 789, 3,
949, 7, 360, 886, 463, 464, 877, 406, 357, 87, 301, 562, 380, 379, 605, 606, 646, 222, 221, 561, 51, 548, 839, 814, 815, 856, 867, 871, 786, 642, 455, 577, 855, 618, 36
7, 245, 899, 946, 446, 246, 167, 171, 339, 299, 945, 131, 347, 749, 92, 541, 45, 911, 941, 514, 654, 180, 748, 426, 849, 841, 875, 876, 240, 390, 858, 922, 656, 477, 26
7, 868, 104, 223, 27, 708, 212, 209, 928, 228, 615, 311, 313, 707, 395, 929, 862, 861, 108, 726, 498, 773, 457, 576, 583, 582, 601, 744, 132, 511, 627, 420, 421, 759, 5
73, 574, 701, 603, 157, 625, 159, 289, 510, 424, 271, 934, 364, 425, 933, 100, 532, 547, 460, 443, 102, 431, 427, 588, 135, 703, 775, 114, 116, 112, 916, 368, 168, 189,
 812, 76, 77, 571, 158, 453, 127, 192, 659, 599, 792, 437, 439, 935, 778, 71, 202, 151, 68, 827, 805, 824, 804, 563, 564, 559, 626, 936, 843, 277, 253, 892, 534, 326, 3
25, 320, 321, 322, 530, 329, 331, 739, 513, 550, 516, 671, 667, 672, 674, 288, 960, 612, 529, 580, 579, 584, 745, 270, 335, 338, 336, 334, 681, 678, 49, 19, 24, 272, 26
8, 163, 784, 452, 383, 397, 470, 820, 818, 552, 895, 348, 286, 422, 799, 195, 568, 567, 67, 670, 613, 106, 608, 337, 484, 486, 359, 58, 297, 266, 206, 205, 204, 885, 47
4, 482, 694, 692, 124, 269, 276, 203, 190, 369, 225, 860, 220, 465, 557, 777, 365, 494, 493, 495, 243, 696, 458, 697, 910, 870, 699, 398, 729, 734, 252, 372, 282, 753,
893, 260, 537, 752, 236, 169, 170, 414, 363, 362, 937, 413, 376, 585, 215, 617, 619, 620, 621, 500, 499, 239, 535, 174, 954, 6, 819, 853, 957, 173, 352, 640, 172, 94, 9
3, 85, 84, 542, 483, 89, 468, 469, 669, 523, 59, 55, 648, 645, 616, 958, 894, 813, 890, 851, 319, 639, 318, 907, 430, 122, 123, 386, 956, 385, 788, 641, 546, 522, 524,
811, 781, 959, 826, 889, 825, 346, 344, 962, 263, 865, 109, 110, 722, 963, 822, 207, 224, 83, 15, 787, 964, 665, 438, 442, 119, 802, 737, 555, 556, 138, 141, 273, 143,
249, 918, 81, 78, 968, 898, 80, 747, 526, 544, 14, 16, 31, 969, 828, 830, 829, 384, 724, 970, 720, 721, 740, 971, 50, 663, 401, 196, 160, 98, 47, 551, 976, 506, 972, 28
7, 705, 973, 741, 689, 706, 690, 903, 838, 283, 897, 261, 539, 587, 591, 586, 478, 480, 975, 136, 610, 479, 649, 974, 32, 33, 881, 407, 410, 38, 481, 798, 502, 850, 22,
 714, 713, 712, 604, 764, 241, 74, 79, 349, 154, 803, 496, 844, 72, 560, 475, 528, 396, 459, 185, 65, 66, 73, 258, 97, 96, 216, 298, 48, 44, 328, 353, 330, 575, 736, 49
1, 497, 750, 429, 832, 834, 833, 300, 295, 719, 901, 655, 965, 230, 488, 718, 967, 717, 356, 186, 435, 434, 623, 622, 821, 354, 197, 757, 758, 440, 343, 345, 966, 115,
716, 687, 20, 21, 229, 403, 405, 917, 451, 835, 831, 91, 234, 232, 231, 194, 350, 738, 732, 725, 150, 448, 904, 902, 589, 533, 250, 504, 492, 274, 137, 809, 779, 391, 4
15, 800, 445, 961, 782, 323, 35, 257, 536, 280, 630, 296, 57, 679, 676, 955, 471, 26, 660, 661, 275, 251, 635, 145, 569, 769, 845, 34, 869, 754, 408, 938, 636, 404, 164
, 166, 662, 227, 226, 235, 316, 866, 310, 332, 210, 211, 355, 447, 340, 428, 794, 23, 18, 632, 801, 932, 449, 503, 908, 264, 244, 456, 772, 358, 183, 399, 140, 930, 139
, 863, 95, 333, 771, 931, 433, 39, 600, 501, 382, 400, 381, 233, 554, 760, 41, 817, 466, 602, 543, 217, 293, 444, 723, 294, 54, 17, 40, 702, 304, 416, 394, 175, 113, 11
7, 883, 873, 60, 519, 525, 527, 99, 303, 305, 476, 308, 219, 218, 578, 134, 176, 864, 806, 366, 121, 142, 392, 387, 388, 520, 823, 837, 361, 29, 432, 307, 128, 915, 810
, 472, 700, 214, 213, 509, 944, 505, 628, 566, 644, 565, 848, 652, 847, 846, 156, 11, 290, 292, 943, 118, 487, 485, 262, 285, 683, 581, 461, 590, 198, 531, 852, 948, 65
7, 412, 682, 633, 28, 766, 324, 624, 133, 909, 256, 746, 423, 952, 5, 130, 265, 342, 378, 105, 597, 763, 761, 762, 120, 793, 733, 643, 891, 978, 86, 90, 609, 979, 61, 9
80, 489, 25, 611, 490, 840, 842, 69, 658, 767, 770, 859, 351, 389, 462, 284, 977, 558, 371, 315, 507, 201, 854, 727, 728, 553, 874, 64, 888, 884, 872, 686, 454, 857, 43
, 42, 103, 614, 647, 685, 53, 52, 152, 545, 765, 63, 951, 302, 797, 795, 756, 755, 836, 572, 436, 193, 518, 913, 914, 515, 751, 149, 409, 953, 950, 598, 517, 375, 242,
735, 776, 808, 314, 593, 592, 327, 56, 8, 947, 10, 317, 309, 816, 594, 596, 595, 512, 165, 101, 709, 46]
End of all, and you got the best answer!
The interation  is 200000
eacch case distance is
[13721.479356370719]
each case time is
 [24779.44728088379]
The average distance is 13721.479356370719
The average time is 24779.44728088379
```


6500
6400
6300
6200
6100
6000
5900
5800
49500
49600
49700
49800
49900
50000
50100
50200


the evolution of the cost
26000
24000
22000
20000
18000
16000
14000
0
25000
50000
75000
100000
125000
150000
175000
200000

3. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

由于在初始化种群时使用了贪心算法导致种群数量较大时初始化时间很长，当使用数据 194 的时候可以感受到明显卡顿，当使用数据 980 时等待时间较长因而更大的数据级此法并不适用需要使用类似于随机进行种群的初始化，当然在数据量较小时此法的速度以及初始的最短路径都是要远小于纯随机进行的。对于数据更多时，此代码的运行效果较差，例如数据集 980，初始化加上运行时间大致在 8.5h，更多的数据本次实验就没有进行测试了。

在迭代函数中我一开始使用轮盘赌算法进行亲代的选择，但是效果不太好，194 数据的最佳值仅为 12000，修改后改为锦标赛算法，答案得到显著优化。

就结果而言，当输入数据集不变时，迭代次数越多结果越理想，对于 980 数据迭代 15 万次答案为 16952，而当 迭代次数达到 20 万时结果就达到了 13721。当种群量增加时，结果也会得到一定改善。

四、 参考资料

PPT