基于概率搜索的旅行商问题求解策略分析

摘　要:　本文分析了基于概率搜索的旅行商问题的求解策略。TSP 是一个著名的组合优化问题，需要找到访问地图中所有城市的最短路径。该论文利用遗传算法寻找 TSP 的近优解，并对遗传算法的交叉和下一代步骤提出了两项改进。论文还比较了改进后的遗传算法与原始遗传算法在不同尺度的 TSP 上的收敛效果。论文的结论是改进的遗传算法可以加快收敛速度并提高解的准确性，并提出了一些未来进一步研究的方向。

关键词:　旅行商问题；遗传算法；交叉；概率搜索；近优解

Analysis of solving strategies for traveling salesman problem based on probabilistic search

Abstract:　This paper analyzes the solving strategies for the traveling salesman problem (TSP) based on probabilistic search. TSP is a famous combinatorial optimization problem that requires finding the shortest path that visits all cities in a map. The paper uses genetic algorithm (GA) to find near-optimal solutions for TSP, and proposes two improvements for the crossover and the next generation steps of GA. The paper also compares the convergence effects of the improved GA with the original GA on different scales of TSP. The paper concludes that the improved GA can speed up the convergence and improve the accuracy of the solutions, and suggests some future directions for further research.

Key words: Traveling salesman problem; Genetic algorithm; Crossover; Probabilistic search; Near-optimal solutions

1　引言

旅行商问题（Traveling Salesman Problem, 简称 TSP）是一个著名的组合优化问题：在一幅地图上，给定n 个城市的坐标，要求从起点出发，经过所有城市再回到起点，路径最短的问题[1][2]。在数据结构中用图论来描述便是：给定一全连通的无向带权图$G = (V, E)$，需找出总权值最小的 Hamilton 圈。其中$V = \{v_1, v_2, v_3, \ldots, v_n\}$表示 n 个节点的集合，$E = \{e_{ij} \mid v_i, v_j \in V\}$是集合$G$ 中节点两两连接的集合，每一条边e_{ij} 都存在与之对应的权值d_{ij}，其中d_{ij} 为从i 到j 的度量，可以是距离、费用、时间、油量等[3]。

旅行商问题描述起来虽然简单，解决起来却非常困难。旅行商问题属于 NP-complete 问题，而且是NP(non-deterministic poly-nominal)问题中最难的一类问题，无法在多项式时间内求解。如果用穷举法来列出所有路径以得到精确解，其时间复杂度为$O(n!)$，当 n 比较小的时候，现代计算机能够很快的求解，但是当n 足够大的时候，精确解则在计算上成为不可能。如果 n=20，那么巡回路径有1.2×10^{18}种，如果n=100，巡回路径高达4.6×10^{155}种，而据估计，宇宙中的基本粒子数“仅有”10^{87}个。[4]

旅行商问题具有重要的实际意义，它起初是为了交通运输而提出的，比如飞机的航线安排、送邮件、快递服务等等。实际上其应用范围扩展到了许多其他领域。

印制电路板转孔是 TSP 应用的经典例子，在一块电路板上打上千个孔，转头在这些孔间移动，如何才能保证把所有孔都遍历一遍后，所花费的时间最少。把这个问题转化成 TSP，孔相当于城市，孔到孔之间的移动时间相当于距离；为了避免大气干扰，使光学系统达到其衍射极限分辨率，研究人员提出发展空间
光干涉仪和综合孔径望远镜的计划。美国航空航天局有一个卫星群组成空间天文台(Space-based Observatories)的计划，用来探测宇宙起源和外星智慧生命)。欧洲空间局也有类似的 Darwin 计划。 对天体成像的时候，需要对两颗卫星的位置进行调整，如何控制卫星，使消耗的燃料最少，可以用 TSP 来求解。这里把天体看作城市，距离就是卫星移动消耗的燃料[5]；美国国家卫生协会在人类基因排序工作中用 TSP 方法绘制放射性杂交图。把 DNA 片段作为城市，它们之间的相似程度作为城市间的距离。法国科学家已经

用这种办法作出了老鼠的放射性杂交图。

2 TSP 近似求解方案

2.1 遗传算法求解TSP问题

遗传算法（Genetic Algorithms，简称 GA）是生命科学与工程科学互相交叉、互相渗透的产物，其本质是一种求解问题的高度并行的全局随机化搜索算法，它能在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最优解[6][7][8]。

遗传算法有几个基本概念：编码、选择、交叉、变异。其基本步骤为：选择合适的编码策略，定义适应度函数，确定种群大小，最大代数，选择方法，交叉方法，变异方法，交叉概率，变异概率等参数。遗传算法流程图如图所示。

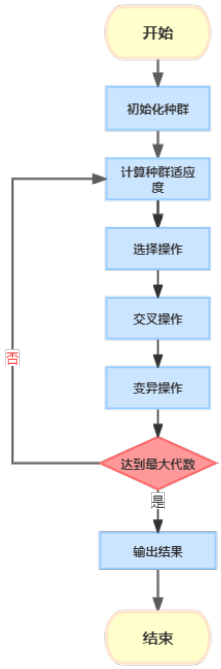

图 2-1 遗传算法流程图

在本文中，对 TSP 问题使用遗传算法求解，基本概念与步骤如下所示。

编码：在本文 TSP 问题中，染色体采用的是节点的顺序作为染色体的编码，其长度固定为 n，并且首位始终为节点：0[9]

选择：本文采用轮盘赌的方法来选择父代，任何一个父代都有可能被选择到，但是每个父代被选择的概率与其适应度成正比。[10]

$$fitness_i = \frac{\text{costs}_{\max} - costs_i}{costs_{max} - costs_{min}} \quad (2-1)$$

$$ps_i = \frac{fitness_i}{\sum fitness_i} \quad (2-2)$$

其中，$fitness_i$表示第 i 个个体的适应度，$costs_{max}, costs_{min}$表示该代种群的巡回一圈消耗值的最大值和最小值，$costs_i$表示第 i 个个体巡回一圈的消耗值，ps_i表示第 i 个个体被选中的概率；

交叉：在选择完父代中的一个个体后，随机生成一个交叉点 idx，不能为 0，交换[1:idx]和[idx+1:end]两段，得到一个新的个体；[11]

变异：染色体序列有小概率会发生变异，即随机生成两个变异点，交换两个变异点的，达到变异的目的。[12]

下一代生成流程：先对上一代进行评估，然后按照适应度从大到小排序，在父代中使用轮盘赌的方法选择一个个体，以一定概率进行交叉，以一定概率进行变异，作为下一代的个体。本文用到的遗传算法，其超参数如表所示：[13]

超参数	值
迭代数	500
种群个体数	1000
变异概率	0.3
交叉概率	0.8
染色体长度	50

2.2 算法中交叉步骤的优化

对于通用的遗传算法，交叉步骤中，常常采用一对父母相互来产生两个后代，这也是参考了有性生殖比无性生殖更能培育出适应性强的个体。而由于本文中染色体的特殊性，如果用通用遗传算法中的交叉方式，就会造成节点的重复到达或漏达。因此，对于本文交叉的操作中，只用到了单体繁殖不能很好的结合两个个体的优点这一问题，本文重新设计了交叉方案，如下所示。

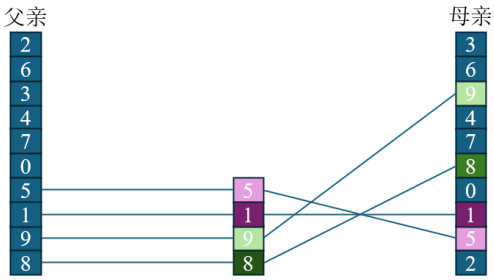

图 2-2 交叉步骤中选择交叉染色体序列

假设选出的两个个体，一个视为父亲，一个视为母亲。随机生成一个交叉点，如图中的为 4，选择父亲染色体中的后 4 个染色体序列值，在母亲中找到对应的染色体序列的位置。

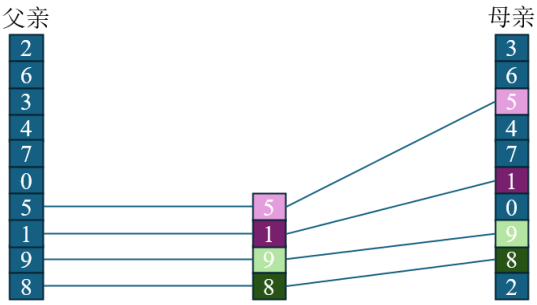

图 2-3 交叉操作中的重新排列染色体序列

按照父亲染色体中的序列，重新对母亲染色体中对应的染色体序列值进行排序，从而生成了新的子代。这样可以充分利用父亲染色体中局部优解序列，将该局部优解序列遗传给下一代。同理，对上述两步反过来对父亲染色体中的部分序列进行重排列。

2.3 算法中下一代生成的优化

遗传算法存在两个极为重要的超参数——变异概率和交叉概率，这两个概率对算法收敛的准确性以及收敛速度都有很大的影响。又因为，一旦换了节点数或者节点位置，重新使用遗传算法计算的时候，两个概率都需要重新调整才能到最合适的大小。其次，鉴于本文中一个染色体上的值数量只有不到 100，而一代中有 1000 个个体，因此，重新调整下一代的生成算法为：下一代个体的 1/4 来自上一代的前 1/4 的优秀个体（这保证了最优解随着代数是单调不减的），下一代个体的 1/4 来自上一代中经过交叉和变异的个体，下一代个体的 1/4 来自上一代中仅经过交叉的个体，下一代个体的 1/4 来自上一代中仅经过变异的个体。

3 遗传算法求解 TSP 近优解的实验

Python 是一种广泛使用的解释型、高级和通用的编程语言，提供了丰富的图形与数学库，比如 matplotlib 和 numpy 等，本文采用 Python 来作为实现遗传算法求解 TSP 的近优解的解决方案。[14][15][16]

3.1 随机节点生成

在$n \times n$的地图上，随机生成 n-1 个点以及原点(0, 0)，并依次计算两两坐标的欧几里得距离，公式如(3-1)所示。得到一张$n \times n$的距离矩阵，该距离矩阵为对称矩阵，其中对角线上的值为 0。

$$d_{ij} = d_{ji} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3-1}$$

下面以$n = 10$为例，图 3.1 为节点图片，表 3.1 为距离矩阵。

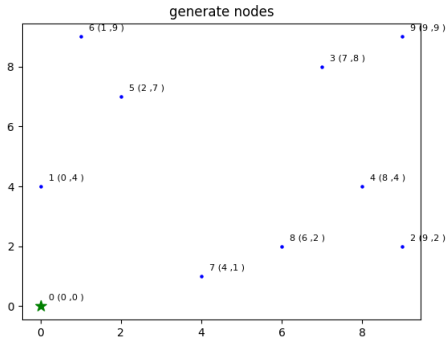

图 3.1 随机生成节点数为 10 的地图

表 3.1 节点数为 10 的距离矩阵

0	4.00	9.22	10.63	8.94	7.28	9.06	4.12	6.32	12.73
4.00	0	9.22	8.06	8.00	3.61	5.10	5.00	6.32	10.30
9.22	9.22	0	6.32	2.24	8.60	10.63	5.10	3.00	7.00
10.63	8.06	6.32	0	4.12	5.10	6.08	7.62	6.08	2.24
8.94	8.00	2.24	4.12	0	6.71	8.60	5.00	2.83	5.10
7.28	3.61	8.60	5.10	6.71	0	2.24	6.32	6.40	7.28
9.06	5.10	10.63	6.08	8.60	2.24	0	8.54	8.60	8.00
4.12	5.00	5.10	7.62	5.00	6.32	8.54	0	2.24	9.43
6.32	6.32	3.00	6.08	2.83	6.40	8.60	2.24	0	7.62
12.73	10.30	7.00	2.24	5.10	7.28	8.00	9.43	7.62	0

3.2 遗传算法求解最短巡回路径

分别在$n=10, n=20, n=30, n=40, n=50$的情况下，对 TSP 问题应用交叉优化及下一代生成优化后的遗传算法，求解结果如下所示。

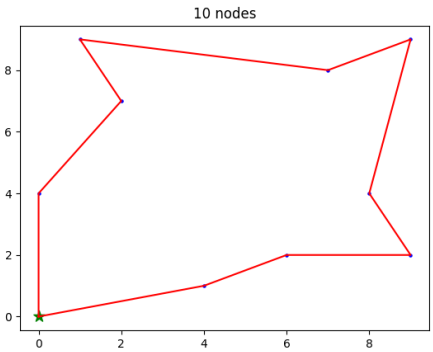

图 3.2 节点数为 10 的求解结果

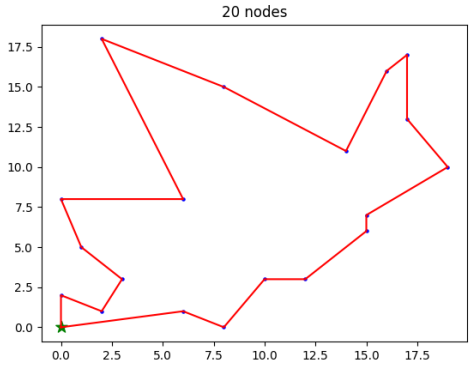

图 3.3 节点数为 20 的求解结果

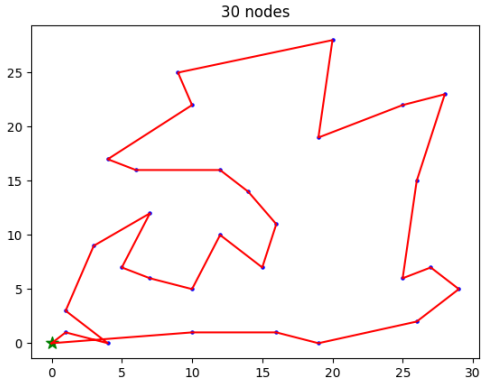

图 3.4 节点数为 30 的求解结果

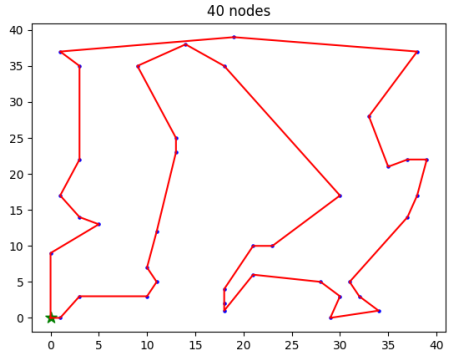

图 3.5 节点数为 40 的求解结果

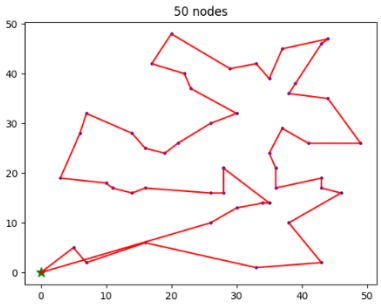

图 3.6 节点数为 50 的求解结果

可以看到，这五个结果分别求出了对应的最优或次优的解。具体结果如表 3.2 所示

表 3.2 $n=10,20,30,40,50$的求解结果

节点数	10	20	30	40	50
结果	34.85	81.72	153.75	248.82	312.24

3.3 优化前后收敛效果

在$n=10,n=20,n=30,n=40,n=50$的每种情况下分别进行，有无交叉优化及有无下一代生成优化四组实验，并对比四组实验收敛效果。

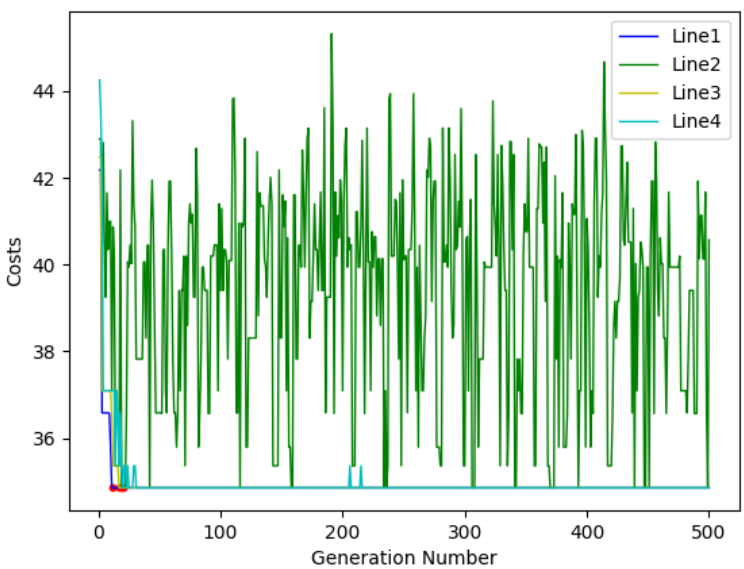

图 3.7 节点数为 10 的情况，四组实验收敛速度对比

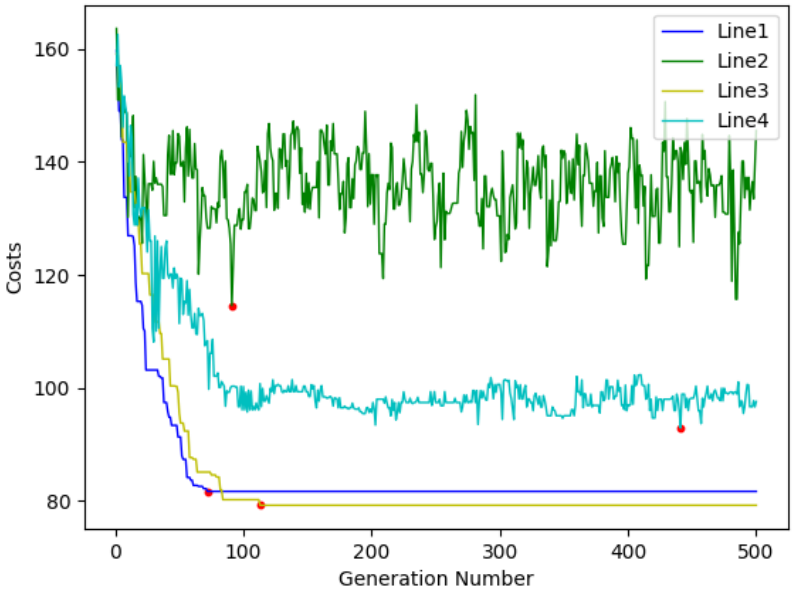

图 3.8 节点数为 20 的情况，四组实验收敛速度对比

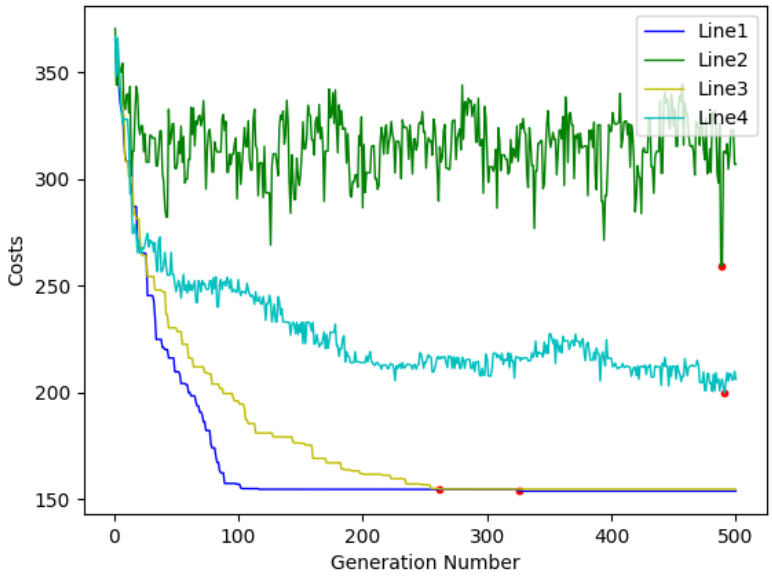

图 3.9 节点数为 30 的情况，四组实验收敛速度对比

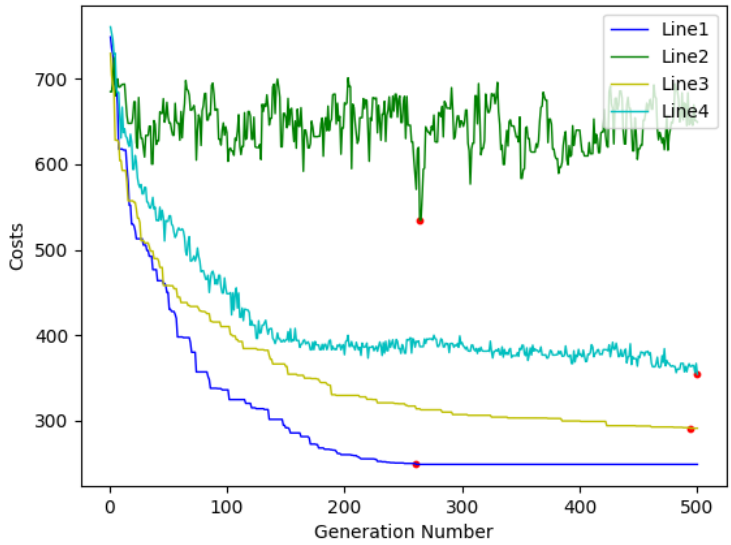

图 3.10 节点数为 40 的情况，四组实验收敛速度对比

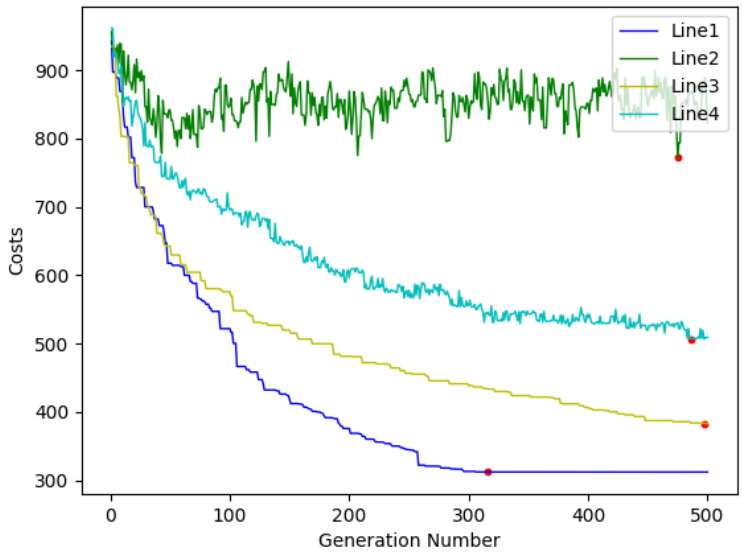

图 3.11 节点数为 50 的情况，四组实验收敛速度对比

其中 Line1 表示优化交叉算法且优化下一代生成算法，Line2 表示优化交叉算法但不优化下一代生成算法，Line3 不优化交叉算法但优化下一代生成算法，Line4 表示不优化交叉算法且不优化下一代生成算法。其具体结果如表 3.3 所示。

表 3.3 $n = 10, 20, 30, 40, 50$的五组实验中是否优化交叉算法及下一代生成算法结果

节点数量	优化情况	最小 costs 的代数	最小 costs
10	两者都优化	11	34.85
	仅优化交叉	19	34.85
	仅优化下一代生成	17	34.85
	两者都不优化	20	34.85
20	两者都优化	72	81.72
	仅优化交叉	91	114.60
	仅优化下一代生成	113	79.27
	两者都不优化	441	92.94
30	两者都优化	326	153.75
	仅优化交叉	488	259.33
	仅优化下一代生成	262	154.70
	两者都不优化	491	199.73
40	两者都优化	261	248.82
	仅优化交叉	264	534.73
	仅优化下一代生成	494	291.12
	两者都不优化	500	354.45
50	两者都优化	316	312.24
	仅优化交叉	475	772.42
	仅优化下一代生成	497	383.23
	两者都不优化	486	507.05

由上述结果得出结论，综合收敛速度与求解巡回路径最短值来看，优化交叉方法且优化下一代生成算法的情况会更好一下。而下一代生成算法不优化，其可能不收敛，并且如果不优化下一代生成算法，那么优化了交叉反而会适得其反，由图 3.8-3.11 看出，Line4 表现得比 Line2 效果更好。由图 3.9-3.11 中 Line1 和 Line3 的，Line1 的收敛速度快于 Line3，最小值也小于 Line3。总之，既优化下一代生成算法又优化交叉算法的情况下，则遗传算法在求解 TSP 的近优解的方面表现得较好。

4 结论与下一步工作

本文使用遗传算法来求解 TSP 问题，并对遗传算法中的交叉方法与下一代生成算法进行了优化，其中交叉方法对遗传算法应用于排列组合提供了解决方案，不同于以往遗传算法中的简单交叉父母染色体的后半部分，该优化能够很好地传承父母亲中染色体序列值的先后顺序，加快收敛速度。下一代生成算法的优化，能够保证适应度的单调不减性质，保证遗传算法的收敛性，能够提高交叉率与变异率的自适应性，提高鲁棒性。

下一步工作可以重新优化交叉部分以及变异部分的算法，以使得模型能够在大规模的 TSP 问题中也有好的效果，在其他的解空间为排列问题也有较好的表现，通过引入自适应选择算法来提高交叉算法的鲁棒性[17][18]；第二，可以引入机器学习或者强化学习来增强遗传算法，加速其收敛且提高其求解准确性。[19]最后，将问题扩展到多目标优化，而不仅仅是单个“权值和最下化”的目标，比如带时间窗的旅行商问题。[20]

对于 TSP 问题，未来可以采取更强高效的智能算法来求解，以提高求解速度来增加节点数，比如量子退火算法[21]、差分信息启发式平滑蚁群算法[22]等等。

References:

[1] 陈文兰, 戴树贵. 旅行商问题算法研究综述[J]. 滁州学院学报, 2006, 8(3): 1-6.

[2] 郭靖扬. 旅行商问题概述[J]. 大众科技, 2006 (8): 229-230.

[3] Hoffman K L, Padberg M, Rinaldi G. Traveling salesman problem[J]. Encyclopedia of operations research and management science, 2013, 1: 1573-1578.

[4] 王敏. TSP 问题及几种常见算法的比较研究[J]. 长春理工大学学报: 自然科学版, 2010 (5): 184-185.
[5] 文建国, 金声震, 宁书年. 光综合孔径望远镜直线阵最优排列方案[J]. 科学通报, 2002, 47(15): 1140-1144.
[6] Lambora A, Gupta K, Chopra K. Genetic algorithm-A literature review[C]//2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon). IEEE, 2019: 380-384.
[7] Mathew T V. Genetic algorithm[J]. Report submitted at IIT Bombay, 2012: 53.
[8] Mirjalili S, Mirjalili S. Genetic algorithm[J]. Evolutionary Algorithms and Neural Networks: Theory and Applications, 2019: 43-55.
[9] 高经纬, 张煦, 李峰, 等. 求解 TSP 问题的遗传算法实现[J]. 计算机时代, 2004 (2): 19-21.
[10] 谢胜利, 唐敏, 董金祥. 求解 TSP 问题的一种改进的遗传算法[J]. 计算机工程与应用, 2002, 38(8): 58-60.
[11] 何庆, 吴意乐, 徐同伟. 改进遗传模拟退火算法在 TSP 优化中的应用[J]. 控制与决策, 2018 (2): 219-225.
[12] Razali N M, Geraghty J. Genetic algorithm performance with different selection strategies in solving TSP[C]//Proceedings of the world congress on engineering. Hong Kong, China: International Association of Engineers, 2011, 2(1): 1-6.
[13] Khan F H, Khan N, Inayatullah S, et al. Solving TSP problem by using genetic algorithm[J]. International Journal of Basic & Applied Sciences, 2009, 9(10): 79-88.
[14] Van Der Walt S, Colbert S C, Varoquaux G. The NumPy array: a structure for efficient numerical computation[J]. Computing in science & engineering, 2011, 13(2): 22-30.
[15] Hunter J, Dale D. The matplotlib user's guide[J]. Matplotlib 0.90. 0 user's guide, 2007.
[16] Python W. Python[J]. Python Releases for Windows, 2021, 24.
[17] Ilavarasi K, Joseph K S. Variants of travelling salesman problem: A survey[C]//International conference on information communication and embedded systems (ICICES2014). IEEE, 2014: 1-7.
[18] 左翔, 赵杏杏, 叶瑞禄, 等. 基于改进自适应遗传算法的 新安江模型参数率定研究[J]. China Rural Water & Hydropower, 2023 (11).
[19] Sohail A. Genetic algorithms in the fields of artificial intelligence and data sciences[J]. Annals of Data Science, 2023, 10(4): 1007-1018.
[20] Ban H B, Pham D H. Solving optimization problems simultaneously: the variants of the traveling salesman problem with time windows using multifactorial evolutionary algorithm[J]. PeerJ Computer Science, 2023, 9: e1192.
[21] Le T V, Nguyen M V, Khandavilli S, et al. Quantum annealing approach for selective traveling salesman problem[C]//ICC 2023-IEEE International Conference on Communications. IEEE, 2023: 2686-2691.
[22] Li W, Wang C, Huang Y, et al. Heuristic smoothing ant colony optimization with differential information for the traveling salesman problem[J]. Applied Soft Computing, 2023, 133: 109943.

附录

本文中所有代码均可在 github 上面找到，链接为：https://github.com/MinzhiYoyo/TSPGAsolution。下面将展示主要代码：

```
class TSPGA:
    def __init__(self, node_num, file_path=None, save_path=None):
        pass
    def evaluate(self, route: np.ndarray):
        costs = np.sum([self.distance_table[route[i], route[i + 1]] for i in range(self.node_num - 1)])
        costs += self.distance_table[route[self.node_num - 1], route[0]]
        return costs
    def generate_node_random(self):
        self.nodes = np.vstack((np.array([0, 0]),np.random.randint([0, 0], [self.node_num, self.node_num],size=(self.node_num - 1, 2))))
```

```
10.
11.class GA:
12.    def __init__(self, generation_num, population_num, mutation_rate,
   crossover_rate, node_num, file_path=None,
13.                 save_path=None):
14.        pass
15.    def next_population(self, parents_num=2, separate=False):
16.        next_population = np.zeros(shape=(self.population.shape[0], se
   lf.population.shape[1]), dtype=np.int64)
17.        i = 0
18.        s = self.population_num//4 if separate else self.population_nu
   m
19.        while i < s:
20.            parents = self.select(num=parents_num)
21.            if np.random.rand() < self.crossover_rate:
22.                next_population[i:i + parents_num] = self.crossover(pa
   rents)
23.            else:
24.                next_population[i:i + parents_num] = self.population[p
   arents].copy()
25.            for j in range(parents_num):
26.                if np.random.rand() < self.mutation_rate:
27.                    next_population[i+j] = self.mutation(next_populati
   on[i+j])
28.            i += parents_num
29.        if separate:
30.            while i < self.population_num*2//4:
31.                parents = self.select(num=parents_num)
32.                if np.random.rand() < self.crossover_rate:
33.                    next_population[i:i + parents_num] = self.crossove
   r(parents)
34.                else:
35.                    next_population[i:i + parents_num] = self.populati
   on[parents].copy()
36.                i += parents_num
37.            while i < self.population_num*3//4:
38.                parents = self.select(num=1)
39.                next_population[i] = self.mutation(self.population[par
   ents[0]].copy())
40.                i += 1
41.        next_population[i:] = self.population[self.idx[:len(next_popul
   ation)-i]].copy()
```

```
        self.population = next_population
        self.evaluate()
    def select(self, num=2):
        return np.random.choice(np.arange(self.population_num), num, r
eplace=False,
                                p=self.fitness / np.sum(self.fitness))
    def crossover(self, parents):
        if len(parents) == 1:
            idx = np.random.randint(2, self.node_num - 1)
            child = np.hstack((self.population[parents[0]][0:1], self.
population[parents[0]][idx:],
                               self.population[parents[0]][1:idx]))
            return child
        elif len(parents) == 2:
            idx = np.random.randint(2, self.node_num - 1)
            mother = self.population[parents[0]].copy()
            father = self.population[parents[1]].copy()
            mother_X = mother[idx:].copy()
            father_X = father[idx:].copy()
            mother_idxs = np.array([np.where(mother == i)[0][0] for i
in father_X])
            father_idxs = np.array([np.where(father == i)[0][0] for i
in mother_X])
            mother_idxs.sort()
            father_idxs.sort()
            mother[mother_idxs] = father_X
            father[father_idxs] = mother_X
            return np.vstack((mother, father))
    def mutation(self, child):
        idx1, idx2 = np.random.choice(np.arange(1, self.node_num), 2,
replace=False)
        child[idx1], child[idx2] = child[idx2], child[idx1]
    def evaluate(self):
        for i in range(self.population_num):
            self.costs[i] = self.tsp.evaluate(self.population[i])
        min_val, max_val = np.min(self.costs), np.min(self.costs)
        self.fitness = (max_val - self.costs) * 100 / (max_val - min_v
al)
```