摘　　要

本文的首要目标是改进模拟退火算法以求解 TSP 问题。从传统模拟退火算法开始，共计进行了 4 轮改进，分别为引入 2-opt 算法进行解变换、采用自适应的循环次数上限、使得多种解变换方式并行、在解变换中尝试调参，并通过重复运行 100 次，比较运行结果中的最小误差、平均误差和最大误差，来衡量改进效果。实验结果证明，每 1 轮改进都是有意义的。最后，将 4 轮改进后的模拟退火算法在 6 个著名 TSP 实例上进行测试，求得最小误差均值为 1.74%。

关键词：模拟退火算法；改进；2-opt；并行；自适应；解变换

目录

1 引言

1.1 模拟退火算法介绍

模拟退火(Simulated Annealing，SA)算法是一种组合优化问题的搜索算法，也是一种流行的迭代元启发式算法，广泛用于解决离散和连续优化问题。SA 算法的关键特征在于通过允许爬山运动找到全局最优解来摆脱局部最优解。SA 的主要缺点之一是它有几个参数需要调整，而且它的性能对这些控制参数的值很敏感。参数整定有两种特殊策略：在线参数整定和离线参数整定。在在线方法中，参数在 SA 算法的整个执行过程中被动态或自适应地控制和更新，而在离线方法中，不同参数的值在 SA 算法执行之前被调整，并且在执行期间是固定的。微调参数值并非微不足道，而且这些参数通常通过反复试验得到非常糟糕的结果。因此，参数较少或对参数设置不敏感的 SA 算法对实际用户非常有吸引力。

1.2 旅行商问题介绍

旅行商问题(traveling salesman problem，TSP)是典型的 NP 完全问题，即其最坏情况下的时间复杂度随着问题规模的增大按指数方式增长，到目前为止还未找到一个多项式时间的有效算法。

TSP 问题可描述为：已知 n 个城市相互之间的距离，某一旅行商从某个城市出发访问每个城市有且仅有一次，最后回到出发城市，如何安排才使其所走路线距离最短。简言之，就是寻找一条最短的遍历 n 个城市的路径，或者说搜索自然子集 X={1,2,…,n}(X 的元素表示对 n 个城市的编号)的一个排列 $\pi(X)=\{V_1,V_2,\cdots,V_n\}$，使得

$$T_d=\sum_{i=1}^{n+1}d(V_i,V_{i+1})+d(V_n,V_1)$$

取得最小值，其中$d(V_i,V_{i+1})$表示城市V_i到城市V_{i+1}的距离。

1.3 研究现状概述

用于解决 TSP 的方法分为三类：精确方法、启发式方法和元启发式方法。精确方法包括动态规划、分支定界、分支切割、切割平面和 LP 松弛等方法。这些方法可以在合理的时间内解决小型 TSP。然而，随着问题规模的增加，精确方法的执行时间也会增加。结果，这些方法不可能获得最佳结果。

启发式方法包括 k-opt、LinKernighan (LK) 、链式 LK (CLK) 、Helsgaun 的 LK (LKH) 、LKH-2 和游览等方法。这些方法可以在合理的时间内为著名的 TSP 基准提供高质量的解决方案。但是，不能保证他们可以提供最佳解决方案。启发式算法通过反复试验寻求解决方案。这些算法通常工作正常，但也有它们无法工作的情况。它们可用于无需求得最佳解决方案的情况。元启发式方法用于结合结合局部搜索和随机化程序的启发式算法。这些方法通常优于简单的启发式方

法，并且可以在较短的运行时间内产生令人满意的结果 。

最近，许多研究已经使用元启发式算法来解决 TSP 问题。Zhan 等人[1]提出了一种基于列表的模拟退火（LBSA）算法。在 LBSA 中创建了一个温度列表，该列表用于决定是否接受候选解决方案。使用不同的更新结构更新温度列表。交换、插入和反转被用作局部搜索运算符。LBSA 在 40 个知名实例上进行了测试。Ilhan 等人[2]提出了一种新的带有交叉算子的模拟退火算法 LBSA-CO。在这种方法中，使用了一种基于列表的温度冷却计划，它可以适应问题的解决方案空间的拓扑结构。通过反转、插入和 2-opt 局部搜索运算符改进了总体中的解决方案。将顺序交叉 (OX1) 和遗传边缘重组交叉 (ER) 算子应用于改进的解决方案以加速收敛。所提出的方法在 65 个著名的 TSP 实例上进行了测试。

除了上述研究之外，还提出了包括几种元启发式算法的混合方法来解决 TSP。Ezugwu 等人[3]提出了一种基于模拟退火的共生生物搜索（SOS-SA）算法。交换运算符被用作 SOS-SA 中的本地搜索运算符。模拟退火(SA)用于确保共生生物搜索 (SOS) 不会陷入局部最小值。同时，SA 在问题搜索空间中寻找最优解的同时提高了种群的多样性水平。SOS-SA 在 40 个众所周知的实例上进行了测试。Wang 等人[4]提出了一种基于实例采样的多智能体 SA 算法（MSA-IBS）。MSA-IBS 有效地利用了基于实例的搜索算法的学习能力。此外，由于 SA 的概率接受标准，它设法避免了过早停滞。MSA-IBS 使用交换、插入和反转作为局部搜索运算符。它在 60 个知名实例上进行了测试。

2 传统模拟退火算法

2.1 算法原理

模拟退火算法是一种模拟物理退火的过程而设计的随机优化算法，结合爬山法和随机行走算法，同时避免算法进入局部最优，早期用于组合优化，后来发展成一种通用的优化算法。它的基本思想最早在 1953 年就被 Metropolis 提出，但直到1983年Kirkpatrick等人才设计出真正意义上的模拟退火算法并进行应用。

该算法采用类似于物理退火的过程，先在一个高温状态下（相当于算法随机搜索），然后逐渐退火，在每个温度下（相当于算法的每一次状态转移），徐徐冷却（相当于算法局部搜索），最终达到物理基态（相当于算法找到最优解）。其物理过程由以下三部分组成：

（1）高温过程：增强粒子的热运动，使其偏离平衡位置，目的是消除系统原先可能存在的非均匀态；

（2）等温过程：退火过程中要让温度慢慢降低，在每一个温度下要达到热平衡状态，对于与环境换热而温度不变的封闭系统满足自由能较少定律，系统状态的自发变化总是朝自由能减少的方向进行，当自由能达到最小时，系统达到平衡态；

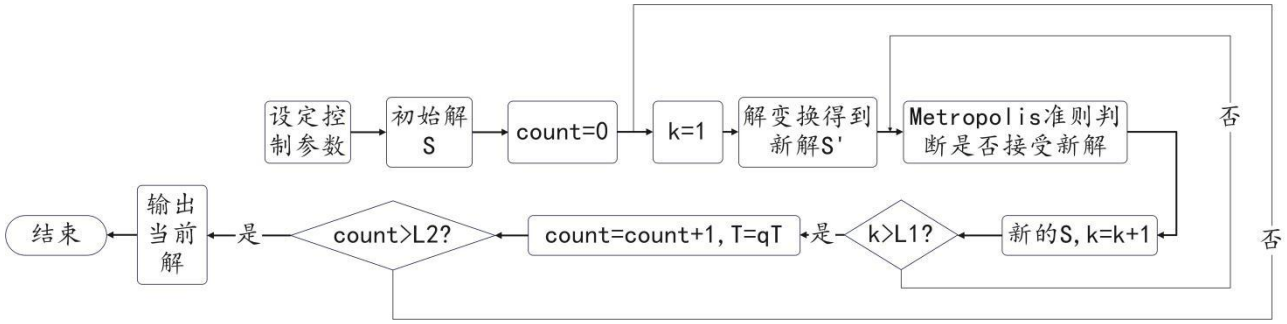
（3）冷却过程：使粒子热运动减弱并渐趋有序，系统能量逐渐下降，从而得到低能的晶体结构。当液体凝固为固体的晶态时退火过程完成。

因此模拟退火算法从某一高温出发，在高温状态下计算初始解，然后以预设

的邻域函数产生一个扰动量，从而得到新的状态，即模拟粒子的无序运动，比较新旧状态下的能量，即目标函数的解。如果新状态的能量小于旧状态，则状态发生转化；如果新状态的能量大于旧状态，则以 Metropolis 接受准则发生转化。当状态稳定后，便可以看作达到了当前状态的最优解，便可以开始降温，在下一个温度继续迭代，最终达到低温的稳定状态，便得到了模拟退火算法产生的结果。

2.2 算法流程图

使用软件“亿图图示”绘制出传统模拟退火算法的流程图如下：



其中，L1 为常数，表示同一温度下搜索次数上限/内循环次数上限；L2 为常数，表示外循环次数上限。

2.3 具体实现方式

实现代码详见附录源代码中的 SA1.py。

2.3.1 初始解

对于 n 个城市的 TSP 问题，得到的初始解是对 0~n-1 的一个随机排序，其中每个数字对应城市的编号。

2.3.2 解变换得到新解

解变换使用的方法是“简单交换两个城市”。假设在交换前，城市序列为[1,2,…,i-1,i,i+1,…,j-1,j,j+1,…,n-1,n]。设随机到的需交换的两个城市所在位置分别为 i 和 j，则交换后得到的城市序列为[1,2,…,i-1,**j**,i+1,…,j-1,**i**,j+1,…,n-1,n]。

2.3.3 Metropolis 准则

若路径长度函数为$f(S)$，则当前解的路径为$f(S_1)$，新解的路径为$f(S_2)$，路径差为$df=f(S_2)-f(S_1)$，则 Netropolis 准则为

$$P=\begin{cases}1, & df<0\\ e^{-\frac{df}{T}}, & df\geq 0\end{cases}$$

如果$df<0$，则以概率 1 接受新的路径；否则以概率$e^{-\frac{df}{T}}$接受新的路径。

2.3.4 超参数设定

超参数	含义	预设值
L1	内循环次数上限	200
L2	外循环次数上限	200
T_0	温度初始值	1200
q	温度衰减系数	0.9

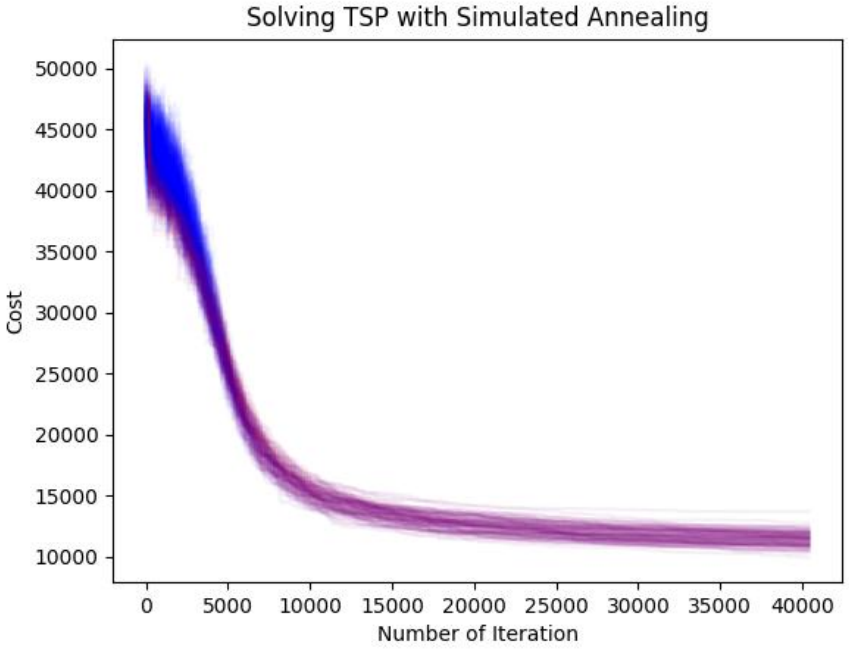
2.4 运行结果展示

将传统模拟退火算法 SA1 重复运行 100 次，得到的运行结果如下表所示。

数据集	已知最优解	最小误差	平均误差	最大误差	误差标准差
eil76	538	26.03%	39.40%	56.46%	6.68%
ch130	6110	62.61%	87.58%	124.40%	9.79%

可以发现，运行得到的平均误差相当大，甚至能达到 87.58%。说明传统的模拟退火算法需要有效改进才能适应大规模 TSP 问题的要求。

下图是以 ch130 为数据集时 SA1 算法的最优解迭代示意图，包括了 100 次的运行结果。其中，红线表示迄今为止已求得的最优解，蓝线表示当前解。



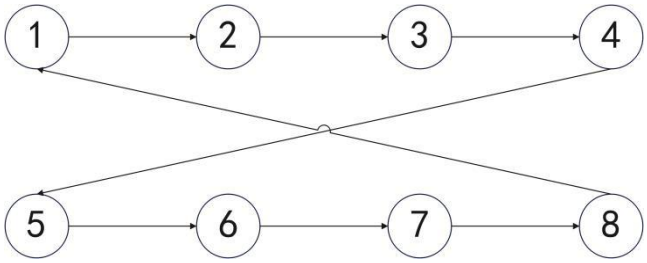
3 逐步改进模拟退火算法

3.1 引入 2-opt 算法进行解变换

3.1.1 改进原理

崔昱欣同学分享蚁群算法复现时，提到了一种较为新奇的解变换方式：2-opt 算法。假设在变换前，城市序列为[1, 2, …, i-1, i, i+1, …, j-1, j, j+1, …, n-1, n]。设随机到的两个城市所在位置分别为 i 和 j，则变换后得到的城市序列为[1, 2, …, i-1, **j, j-1, …, i+1, i**, j+1, …, n-1, n]。

这种变换方式十分适用于 TSP 问题，它可以快速解决路线交叉导致解不优的问题，提高解的质量。



在上图中，当前解为[1, 2, 3, 4, 5, 6, 7, 8]。显然的是，[1, 2, 3, 4, 8, 7, 6, 5]这一解更优。如果使用 2-opt 算法，只需变换 1 次；而使用交换两座城市的算法，则需交换 2 次，这期间还额外消耗了随机到正确城市所需的时间。而随着城市数的增加，这两种解变换方式的差距只会更大。

3.1.2 具体实现

在 common.py 中实现了 2-opt 算法，并在 SA2.py 中调用。

实现代码详见附录源代码中的 SA2.py，以及 common.py 中的 get_new_sol_2opt 函数。

3.1.3 运行结果展示

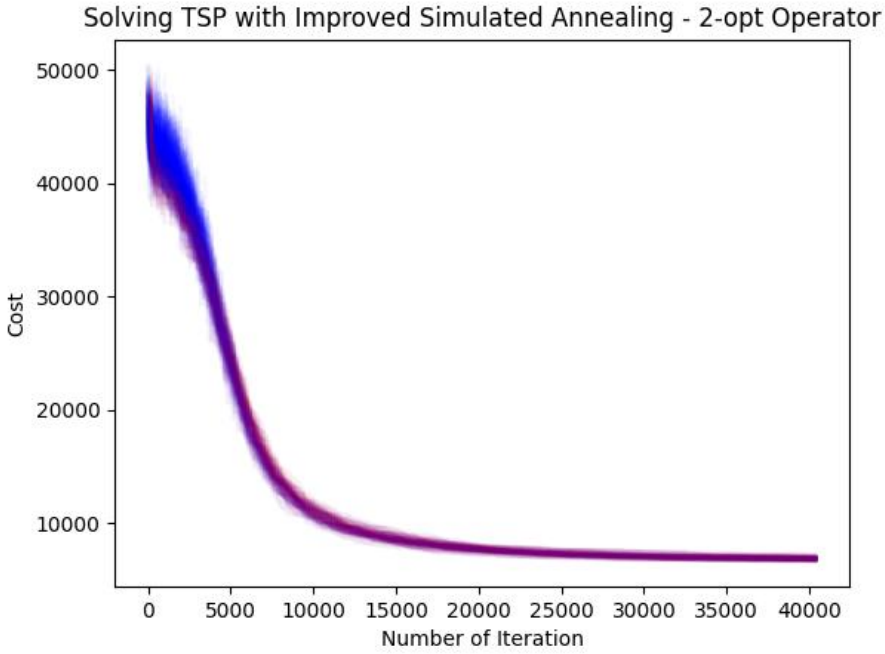
将改进后的模拟退火算法 SA2 重复运行 100 次，得到的运行结果如下表所示。

数据集	已知最优解	最小误差	平均误差	最大误差	误差标准差
eil76	538	4.72%	8.71%	14.67%	1.94%
ch130	6110	7.16%	11.89%	19.32%	2.45%

可以发现，改进解变换方式为模拟退火算法带来的优化是空前的。在 eil76 上，平均误差由 39.40%降到 9.70%；在 ch130 上，平均误差由 87.58%降到 11.89%。所有的误差指标都在减小。

但是，注意到最小误差仍然在 5%左右，说明还存在一定的优化空间。

下图是以 ch130 为数据集时 SA2 算法的最优解迭代示意图，包括了 100 次的运行结果。与 SA1 图相比，线条更细，说明改进后算法的鲁棒性有所增强。



3.2 自适应的循环次数上限

3.2.1 改进原理

虽然传统的 SA 方法设置了内循环，在同一温度值 T 执行多次 Metropolis 算法，保证全局搜索能力。但是实践表明，将温度 T 下执行 Metropolis 算法的次数和温度衰减次数（即内循环次数和外循环次数）设置为定值往往不够灵活，次数过多造成时间的浪费，次数过少则损失解的质量。

改进之处即设置采样稳定条件，使得采样次数可以根据当前所得解的情况灵活处理。如果最优解频繁更新，则应当多循环几次。这样即可在保证解质量的前提下，尽量避免时间的浪费。

3.2.2 具体实现

在完成一次内循环/外循环后，对比当前解与最优解时，若发现当前解更优，在更新最优解时，额外将循环计数器置为 0；若发现当前解不优，才能将循环计数器加 1。

同时，考虑到运行时间的限制，将内循环/外循环的次数上限均由 200 降为 100，以节省运行时间。

实现代码详见附录源代码中的 SA3.py。

3.2.3 运行结果展示

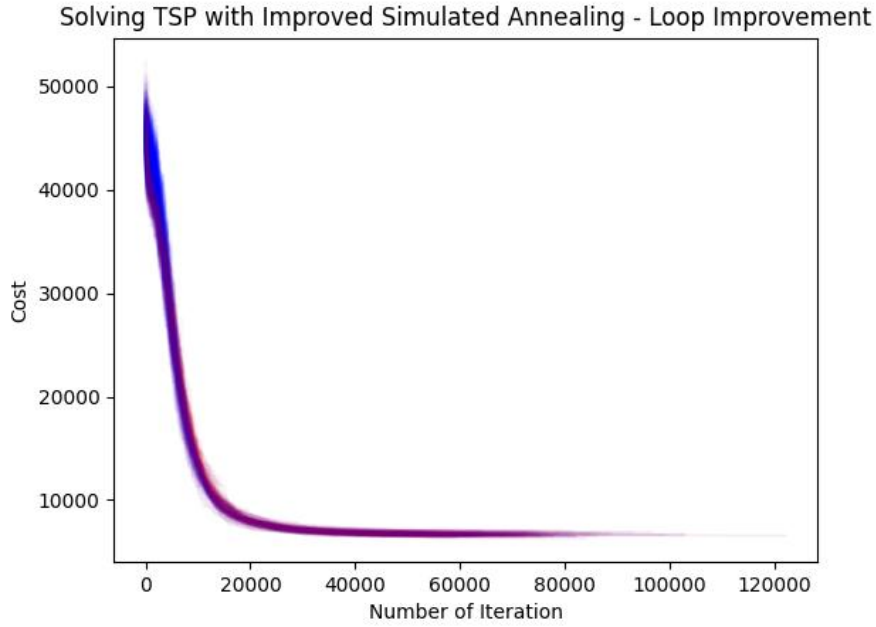
将改进后的模拟退火算法 SA3 重复运行 100 次，得到的运行结果如下表所示。

数据集	已知最优解	最小误差	平均误差	最大误差	误差标准差
eil76	538	5.41%	9.71%	14.42%	1.44%

ch130	6110	3.24%	9.57%	15.56%	1.56%

可以发现，与 3.1 中的 SA2 算法相比，尽管降低了内循环和外循环的次数上限，ch130 的各项误差指标仍然全面减小，eil76 虽在部分指标上有反弹但幅度不大。这充分说明了自适应的循环次数的优越性。

下图是以 ch130 为数据集时 SA3 算法的最优解迭代示意图，包括了 100 次的运行结果。与 SA2 图相比，尽管循环次数上限减少，但迭代次数仍从 40000+增长到 120000+，体现了自适应策略的有效性；线条末端大幅收窄，也说明了自适应策略对解改良的促进作用。



3.3 多种解变换方式的并行

3.3.1 改进原理

传统的 SA 算法选用单一的解变换方式，而且不能证明这种单一的扰动机制就是最优的。在解变换上，笔者认为其最终目的就是**尽可能保证产生的候选解遍布全部解空间。**因此，参考一篇 ccf-c 会论文[2]的做法，我额外添加了一种解变换方式：移位。

假设在变换前，城市序列为[1, 2, …, i-1, i, i+1, …, j-1, j, j+1, …, n-1, n]。设随机到的两个城市所在位置分别为 i 和 j，则变换后得到的城市序列为[1, 2, …, i-1, i, **j,** i+1, …, j-1, j+1, …, n-1, n], 即仅将 j 移动到 i 的后方。

加入这种变换方式是有益处的。2-opt 算法虽好，但是一定要改变 2 个城市的次序；而移位仅会改变 1 个城市的次序。这为产生解提供了更多的可能。

然后，在需要进行解变换时，算法以均等的概率，随机选择”2-opt”和“移位”这两种方式中的一种进行变换。这样综合了两种算子的优点，对二者产生的解空间取得并集。

3.3.2 具体实现

在 common.py 中实现了移位算法，并在 SA4.py 中调用。

实现代码详见附录源代码中的 SA4.py，以及 common.py 中的 get_new_sol_move 函数。

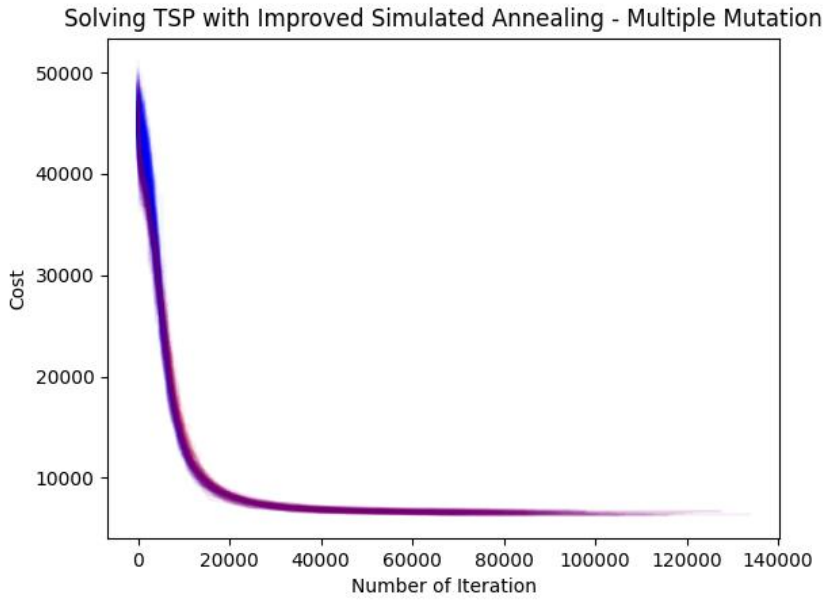
3.3.3 运行结果展示

将改进后的模拟退火算法 SA4 重复运行 100 次，得到的运行结果如下表所示。

数据集	已知最优解	最小误差	平均误差	最大误差	误差标准差
eil76	538	1.48%	6.30%	12.41%	1.49%
ch130	6110	2.17%	7.28%	13.96%	2.43%

可以发现，与 3.2 中的 SA3 算法相比，除了误差标准差外，eil76 和 ch130 的各项误差指标再次全面减小。eil76 的平均误差从 9.71%降到了 6.30%，ch130 的平均误差从 9.57%降到了 7.28%。这充分说明了产生多样化解的重要性和学习 c 会论文的重要性。

下图是以 ch130 为数据集时 SA4 算法的最优解迭代示意图，包括了 100 次的运行结果。



3.4 在解变换中尝试调参

3.4.1 改进原理

考虑到解变换方式中无论是”2-opt”还是“移位”，这两种变换方式都需要随机 2 个数 i 和 j 作为参数。为了让能使解更优的 i 和 j 有更大的概率被随机选取到，可以对当前解进行多次解变换尝试。在多次尝试后，对比得出能使结果最优的 i 和 j，再进行有效的解变换。

这一改进本质上源自调参思想：将 i 和 j 视作参数，尝试多种不同的参数并运行程序，最后选出最好的参数。而这里的改进，也不过是让电脑帮笔者调参罢了。

3.4.2 具体实现

在 common.py 中实现了调参算法，并在 SA5.py 中调用。本文将解变换尝试的次数定为 50。

实现代码详见附录源代码中的 SA5.py，以及 common.py 中的 get_new_sol_move 函数。

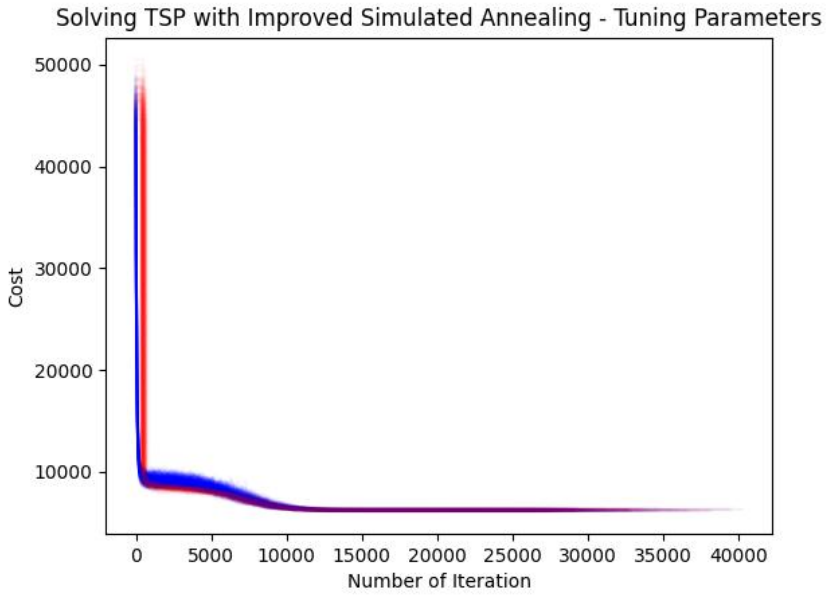
3.4.3 运行结果展示

将改进后的模拟退火算法 SA5 重复运行 100 次，得到的运行结果如下表所示。

数据集	已知最优解	最小误差	平均误差	最大误差	误差标准差
eil76	538	1.18%	2.83%	4.94%	0.76%
ch130	6110	0.00%	2.30%	5.43%	1.06%

可以发现，与 3.3 中的 SA4 算法相比，eil76 和 ch130 的各项误差指标无一例外地减小。eil76 的平均误差从 6.30%降到了 2.83%，ch130 的平均误差从 7.28%降到了 2.30%。更重要的是，在 ch130 上，该算法已经可以求出最优解。在启发式算法中调参十分重要，有时甚至比改进算法更重要，此观点在这里又一次得到了验证。

下图是以 ch130 为数据集时 SA5 算法的最优解迭代示意图，包括了 100 次的运行结果。与 SA4 图相比，可以发现解的优化速度大大加快，总的迭代次数也大幅减小，由 140000+降至 40000+。



4 实验

在第 3 节对 SA 算法的逐步改进中，通过重复运行程序并比较各误差指标，可以发现各版本算法之间存在明显的优劣关系。在解的质量上，SA5>SA4>SA3>SA2>SA1 几乎是毫无争议的。

因此，本节中将使用 SA5 算法，采用 c 会论文[2]中使用的部分数据集进行测试，来检测本文改进的模拟退火算法与学术前沿水平的差距。

4.1 算法参数分析与优化

在经过多轮改进后，当前模拟退火算法的超参数及其预设值如下。

超参数	含义	预设值
L1	内循环解不变次数上限	100
L2	外循环解不变次数上限	100
T_0	温度初始值	1200
q	温度衰减系数	0.9
TP	解变换尝试次数	50

增大上述参数均可使得解更优，误差更小，但是会导致运行时间更长。考虑到时间因素，不再对参数进行调整。

4.2 实验结果与分析

将改进后的模拟退火算法 SA5 重复运行 100 次，得到的运行结果如下表所示。

数据集	已知最优解	平均耗时	最小误差	平均误差	最大误差	误差标准差
eil101	629	5.04	1.78%	3.94%	6.01%	0.95%
kroA150	26524	4.40	0.38%	2.38%	5.32%	1.11%
d198	15780	7.05	0.45%	1.41%	3.29%	0.55%
gil262	2378	6.87	2.39%	5.16%	7.32%	1.15%
a280	2579	8.08	2.18%	6.17%	10.26%	1.72%
d493	35002	14.04	3.27%	5.37%	8.05%	0.97%
平均值			1.74%	4.07%	6.71%	1.07%

而该 c 会论文中的测试结果表如下图所示。

Table 14 The performance comparison of the LBSA-CO, SA and LBSA methods in the same running times on nine instances

Instance	BKS	Time	SA				LBSA				LBSA-CO			
			Best	Mean	Worst	StdDev	Best	Mean	Worst	StdDev	Best	Mean	Worst	StdDev
eil101	629	8.54	2.54	4.59	6.84	7.96	0.32	2.95	5.09	7.76	0.00	0.85	2.23	4.80
kroA150	26,524	12.06	3.12	5.46	8.51	400.09	0.81	3.34	6.61	403.96	0.00	0.60	1.56	114.29
d198	15,780	18.52	1.16	2.82	3.69	112.51	0.48	1.40	3.33	124.16	0.05	0.29	0.61	24.30
gil262	2378	31.64	5.05	7.29	11.02	39.23	1.72	3.62	6.01	29.16	0.21	1.59	3.20	16.22
a280	2579	30.64	4.81	8.65	12.18	55.77	2.91	5.28	8.72	40.85	0.00	0.76	2.02	16.19
gr431	171,414	168.04	3.83	6.53	11.36	3336.93	2.91	4.73	7.56	2564.49	0.91	1.82	2.89	746.11
d493	35,002	209.42	4.95	6.71	9.54	477.63	2.96	4.38	5.38	205.37	1.46	2.12	2.94	155.97
p654	34,643	396.86	0.87	6.27	9.73	834.24	2.31	4.69	7.95	534.30	0.18	0.38	0.85	64.40
u724	41,910	690.92	6.08	7.72	9.63	379.05	4.90	6.26	7.61	280.67	2.60	3.02	3.82	160.26
Average			3.60	6.23	9.16	627.05	2.15	4.07	6.47	465.64	0.60	1.27	2.24	144.73

计算可得，在上文中用于测试的 6 个数据集上，LBSA 的最小误差平均值为 1.53%，平均误差平均值为 3.50%，最大误差平均值为 5.86%，均较 SA5 更优。

而且，LBSA 仅是此论文提出算法 LBSA-CO 的基础版，若将 SA5 与 LBSA-CO 相比，差距更大。这说明本文提出的算法与学术前沿论文提出的算法存在较大差距，仍然具有一定的改进空间。

但是，值得肯定的是，SA5 算法的最小误差平均值已经降到了 1.74%，甚至在 KroA150 数据集（150 座城市）上能将误差缩小到 0.38%，与原 SA 算法相比，取得的进步也是显而易见的。

4.3 算法优缺点分析

该算法的优点有：

- 引入 2-opt 算法进行解变换，能够快速优化存在路径交叉的解。
- 采用自适应的内循环/外循环次数上限值，能够在解频繁更新时循环更多次，提高最终解的质量。
- 多种解变换方式并行，使得产生的解更加多样化，尽可能保证产生的候选解遍布全部解空间。
- 在解变换过程中应用调参思想，多次随机选出最优的 i 值和 j 值，使得解变换更有可能产生更优解。

该算法的缺点与改进方向有：

- 对超参数的调参不够充分。实验过程中未形成系统的调参方法论，可能导致更佳的超参数值（如初始温度等）未被找到。
- 初始解仍是随机生成，可以进一步优化。不过，随机生成也能保证解的多样性，如何在多样性与最优化之间权衡也是一个难题。
- 并行的解变换方式只有 2 个，仍可增加。可以尝试引入学术界常用的“简单交换”、3-opt 算法等更多种类解变换方式，并在多次实验中选出最佳的执行概率。

5 心得体会

感觉这次期末报告与第 2 次作业相比，还是进步了许多的。

仍记得在“翻转课堂”上分享第 2 次作业（改进的遗传算法）时，老师指出的几个问题：没有重复运行程序，求出算法求解的平均值、最劣值和标准差；实验所用数据集较少，无法证明算法改进的有效性；调参方法有待改进等。该算法在 ch130 上的运行结果与已知最优解的误差也有 1.84%之多。

而在本文中，每一版本的算法都重复运行了 100 次，减少了随机算法运行结果的偶然性；改进过程中用 2 个数据集，最后实验时用 6 个数据集测试并对比，充分证明了算法改进的有效性；在解变换上实现了自动调参。而改进模拟退火算法在 ch130 上已经能跑出最优解，误差缩小到 0%。

老师在课堂上多次强调要进行有意义的改进。笔者的 4 次改进想法，部分来自翻转课堂，部分来自论文，部分来自自己的灵感，都是为尽可能保证产生的候选解遍布全部解空间。而每次改进后，当前算法与上一版本算法相比，在最小误差、平均误差、最大误差上都是有显著改进的。这样，从理论上和实践上，都诠释了改进的意义。

报告写到最后，感觉模拟退火算法与遗传算法是存在许多相通之处的。SA 中的解变换方式，其实对应的就是 GA 中的变异算子；学术界也有论文[2]将 GA 中的交叉算子融入 SA 中并取得更好的求解效果。虽然时常听同学说用 A+B 的方式（将两篇论文的成果结合起来）写论文属于 lack of novelty，但是不可否认，不同方法相互借鉴，取长补短，肯定是有意义、有价值的。

本学期的 2 次工作，总的目标是尽可能求出 TSP 问题的最优解，在第 2 次作业中我尝试改进了遗传算法，在期末报告中我尝试改进了模拟退火算法，都取得了一定的进展。这 2 次工作，客观上起到了科研启蒙的作用。在第 2 次工作中，笔者查阅的论文更多了，灵感更多了，算法改进能力也更强了（改进失败的次数也更多了）。虽然，不可否认，第 2 次的工作相比当前学术前沿工作还是 naive 的，与其作比较也有班门弄斧的嫌疑。但是，笔者也是不断成长的，总有一天，自己工作在经历更多轮的改进后，有资格发表在 ccf 认可的会议期刊上！

参考文献

[1] Zhan S, Lin J, Zhang Z, et al. List-based simulated annealing algorithm for traveling salesman problem[J]. Computational intelligence and neuroscience, 2016, 2016.

[2] İlhan İ, Gökmen G. A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem[J]. Neural Computing and Applications, 2022, 34(10): 7627-7652.

[3] Ezugwu A E S, Adewumi A O, Frîncu M E. Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem[J]. Expert Systems with Applications, 2017, 77: 189-210.

[4] Wang C Y, Lin M, Zhong Y W, et al. Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling[J]. International Journal of Computing Science and Mathematics, 2015, 6(4): 336-353.

附录：源代码

main.py：用于选择数据集、模拟退火算法版本，调整超参数，输出结果并绘图的代码

```
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
import matplotlib.animation as animation
import numpy as np
from pprint import pprint
import time
import math
from tqdm import tqdm

from common import *
from GA import GA

import SA1
import SA2
import SA3
import SA4
import SA5
import SA6
import SA6

import os
if not os.path.exists('results'):
    os.makedirs('results')

fileName = ["a280.tsp","d198.tsp","d493.tsp","eil101.tsp","gil262.tsp","gr431.tsp","kroA150.tsp","p654.tsp","u724.tsp","ch130.tsp","eil76.tsp"]
optDis = [2579 , 15780 , 35002 , 629 , 2378 , 171414 , 26524 , 34643 , 41910 , 6110 , 538]

print("1.a280")
print("2.d198")
print("3.d493")
print("4.eil101")
print("5.gil262")
print("6.gr431")
print("7.kroA150")
print("8.p654")
print("9.u724")
print("10.ch130")
print("11.eil76")
```

```
print("请输入您选择的测试集序号",end=":")
choose=int(input())
#choose=1
choose-=1
while choose>=len(fileName):
    print("序号不在 1~11 以内，请重新输入")

# load data
with open('data/'+fileName[choose]) as fp:
    lines=fp.readlines()
    pos = [[float(x) for x in s.split()[1:]] + [int(x) for x in s.split()[:1]] for s i
n lines]
    dim=len(lines)
    graph=np.zeros((dim,3))  # 0:city index 1:x 2:y
    distmat=np.zeros((dim,dim))
    for i in range(dim):
        for j,pox in enumerate(filter(lambda x: x and x.strip(),lines[i].split(' '))):

            graph[i][j]=float(pox)
    for i in range(dim):
        for j in range(i,dim):
            if i==j:
                distmat[i][j]=float('inf')
            else:
                distmat[i][j]=distmat[j][i]=np.linalg.norm(graph[i,1:]-graph[j,1:])

opt_cost = optDis[choose]  # get result from tsp_gurobi.py
num_tests = 100 # number of iid tests
result = {'best_sol': [], 'best_cost': math.inf, 'best_gap': math.inf,
          'cost': [0] * num_tests, 'time': [0] * num_tests,
          'avg_cost': math.inf, 'avg_gap': math.inf, 'cost_std': math.inf,
          'avg_time': math.inf, 'time_std': math.inf}
best_cost = math.inf
best_sol = []
data = {}

# set method

# method = 'GA'  # genetic algorithm
# method = 'SA'  # simulated annealing
# method = 'ISA-2-opt' #Use 2-opt operator
# method = 'ISA-LOOP' #Inner Loop
# method = 'ISA-MM' # Multiple Mutation
method = 'ISA-TP' # Tuning Parameters
```

```
# method = 'ISA-OIS' # Optimized Initial Solution

# run and visualization

method_name = ''
for _ in tqdm(range(num_tests)):
    start = time.time()
    if method == 'GA':
        method_name = 'Genetic Algorithm'
        best_sol, best_cost, data = GA(pos, distmat,
                                       n_pop=200,
                                       r_cross=0.8,
                                       r_mut=0.3,
                                       max_tnm=3,
                                       term_count=5000).search()
    elif method == 'SA':
        method_name = 'Simulated Annealing'
        best_sol, best_cost, data = SA1.SA1(dim, distmat,
                                            term_count_1=200,  # inner loop termination flag
                                            term_count_2=200,  # outer loop termination flag
                                            t_0=1200,  # starting temperature, calculated by init_temp.py
                                            alpha=0.9  # cooling parameter
                                            )
    elif method == 'ISA-2-opt':
        method_name = 'Improved Simulated Annealing - 2-opt Operator'
        best_sol, best_cost, data = SA2.SA2(dim, distmat,
                                            term_count_1=200,  # inner loop termination flag
                                            term_count_2=200,  # outer loop termination flag
                                            t_0=1200,  # starting temperature, calculated by init_temp.py
                                            alpha=0.9  # cooling parameter
                                            )
    elif method == 'ISA-LOOP':
        method_name = 'Improved Simulated Annealing - Loop Improvement'
        best_sol, best_cost, data = SA3.SA3(dim, distmat,
                                            term_count_1=100,  # inner loop termination flag
                                            term_count_2=100,  # outer loop termination flag
```

```
                                        t_0=1200,  # starting temperature, calculated by init_temp.py
                                        alpha=0.9  # cooling parameter
                                        )
    elif method == 'ISA-MM':
        method_name = 'Improved Simulated Annealing - Multiple Mutation'
        best_sol, best_cost, data = SA4.SA4(dim, distmat,
                                        term_count_1=100,  # inner loop termination flag
                                        term_count_2=100,  # outer loop termination flag
                                        t_0=1200,  # starting temperature, calculated by init_temp.py
                                        alpha=0.9  # cooling parameter
                                        )

    elif method == 'ISA-TP':
        method_name = 'Improved Simulated Annealing - Tuning Parameters'
        best_sol, best_cost, data = SA5.SA5(dim, distmat,
                                        max_tnm=50,  # how many candidates picked in tournament selection
                                        term_count_1=100,  # inner loop termination flag
                                        term_count_2=100,  # outer loop termination flag
                                        t_0=1200,  # starting temperature, calculated by init_temp.py
                                        alpha=0.9  # cooling parameter
                                        )
    elif method == 'ISA-OIS':
        method_name = 'Improved Simulated Annealing - Optimized Initial Solution'
        best_sol, best_cost, data = SA6.SA6(dim, pos, distmat,
                                        max_tnm=50,  # how many candidates picked in tournament selection
                                        term_count_1=100,  # inner loop termination flag
                                        term_count_2=100,  # outer loop termination flag
                                        t_0=1200,  # starting temperature, calculated by init_temp.py
                                        alpha=0.9  # cooling parameter
                                        )
    end = time.time()
    result['time'][_] = end - start
```

```
149.     result['cost'][_] = best_cost
150.     if best_cost < result['best_cost']:
151.         result['best_sol'] = best_sol
152.         result['best_cost'] = best_cost
153.         result['best_gap'] = best_cost / opt_cost - 1
154.     plt.plot(range(len(data['cost'])), data['cost'], color='b', alpha=math.pow(num_tes
    ts, -0.75))
155.     plt.plot(range(len(data['cost'])), data['best_cost'], color='r', alpha=math.pow(nu
    m_tests, -0.75))
156.
157. plt.title('Solving TSP with {}'.format(method_name))
158. plt.xlabel('Number of Iteration')
159. plt.ylabel('Cost')
160. plt.savefig('results/{}.png'.format(method))
161.
162. # print results
163. result['avg_cost'] = np.mean(result['cost'])
164. result['avg_gap'] = result['avg_cost'] / opt_cost - 1
165. result['worst_cost'] = np.max(result['cost'])
166. result['worst_gap'] = result['worst_cost'] / opt_cost - 1
167. result['cost_std'] = np.std(result['cost'])
168. result['cost_std_gap'] = result['cost_std'] / opt_cost
169. result['avg_time'] = np.mean(result['time'])
170. result['time_std'] = np.std(result['time'])
171. del result['best_sol']
172. del result['cost']
173. del result['time']
174. pprint(result)
```

common.py：存放各解变换函数的代码

```
1.  import random
2.  import math
3.  def get_cost(n, adj_mat, sol):
4.      """
5.      :param n: number of vertices, e.g. 2
6.      :param adj_mat: adjacency matrix, e.g. [[0,1], [1,0]]
7.      :param sol: solution, e.g. [1,0]
8.      """
9.      return sum([adj_mat[sol[_]][sol[(_ + 1) % n]] for _ in range(n)])
10.
11. def get_new_sol_swap(sol, i, j):
12.     new_sol = sol.copy()
13.     new_sol[i], new_sol[j] = new_sol[j], new_sol[i]
14.     return new_sol
```

```
15.
16. def get_delta_swap(n, adj_mat, sol, i, j):
17.     # bef: [..., i-1, i, i+1, ..., j-1, j, j+1] / [...,i-1, i, j, j+1, ...]
18.     # aft: [..., i-1, j, i+1, ..., j-1, i, j+1] / [...,i-1, j, i, j+1, ...]
19.     # the latter case, 2 * adj_mat(i, j) is extra deducted!
20.     delta = adj_mat[sol[i - 1]][sol[j]] + adj_mat[sol[j]][sol[(i + 1) % n]] + \
21.             adj_mat[sol[j - 1]][sol[i]] + adj_mat[sol[i]][sol[(j + 1) % n]] - \
22.             adj_mat[sol[i - 1]][sol[i]] - adj_mat[sol[i]][sol[(i + 1) % n]] - \
23.             adj_mat[sol[j - 1]][sol[j]] - adj_mat[sol[j]][sol[(j + 1) % n]]
24.     if j - i == 1 or i == 0 and j == n - 1:
25.         delta += 2 * adj_mat[sol[i]][sol[j]]  # symmetrical TSP
26.     return delta
27.
28. def get_new_sol_2opt(sol, i, j):
29.     new_sol = sol.copy()
30.     new_sol[i:j+1] = new_sol[i:j+1][::-1]  # notice index + 1 !
31.     return new_sol
32.
33. def get_delta_2opt(n, adj_mat, sol, i, j):
34.     # bef: [..., i-1, i, i+1, ..., j-1, j, j+1] / [...,i-1, i, j, j+1, ...] / [i, i+1,
    ..., j-1, j]
35.     # aft: [..., i-1, j, j-1, ..., i+1, i, j+1] / [...,i-1, j, i, j+1, ...] / [j, i+1,
    ..., j-1, i]
36.     # the latter case, 2 * adj_mat(i, j) is extra deducted!
37.     delta = adj_mat[sol[i - 1]][sol[j]] + adj_mat[sol[i]][sol[(j + 1) % n]] - \
38.             adj_mat[sol[i - 1]][sol[i]] - adj_mat[sol[j]][sol[(j + 1) % n]]
39.     if i == 0 and j == n - 1:  # the first two value == 0, while others < 0
40.         delta = 0
41.     return delta
42.
43. def get_new_sol_move(sol, i, j):
44.     new_sol = sol.copy()
45.     new_sol[i:j+1] = new_sol[i:i+1] + new_sol[j:j+1] + new_sol[i+1:j]  # notice index
    + 1 !
46.     return new_sol
47.
48. def get_delta_move(n, adj_mat, sol, i, j):
49.     # bef: [..., i-1, i, i+1, ..., j-1, j, j+1]
50.     # aft: [..., i-1, i, j, i+1, ..., j-1,j+1]
51.     # the latter case, 2 * adj_mat(i, j) is extra deducted!
52.     delta = adj_mat[sol[i]][sol[j]] + adj_mat[sol[j]][sol[(i + 1) % n]] + adj_mat[sol[
    j - 1]][sol[(j + 1) % n]] - \
53.             adj_mat[sol[i]][sol[(i + 1) % n]] - adj_mat[sol[j - 1]][sol[j]] - adj_mat[
    sol[j]][sol[(j + 1) % n]]
```

```
    if i == n - 1 and j == 0:  # the first two value == 0, while others < 0
        delta = 0
    return delta

def get_new_sol_CIM(sol, i, j):
    new_sol = sol.copy()
    new_sol = new_sol[:i][::-1] + new_sol[i:][::-1]
    return new_sol

def get_delta_CIM(n, adj_mat, sol, i, j):
    # bef: [0 , 1 , ... , i-1 , i , i+1 , ... , n-1]
    # aft: [i-1 , i-2 , ... , 0 , i , n-1 , n-2 , ... , i+1]
    # the latter case, 2 * adj_mat(i, j) is extra deducted!
    delta = adj_mat[sol[0]][sol[i]] + adj_mat[sol[i]][sol[n - 1]] + adj_mat[sol[i - 1]][sol[(i + 1) % n]] - \
            adj_mat[sol[i - 1]][sol[i]] - adj_mat[sol[i]][sol[(i + 1) % n]] - adj_mat[sol[0]][sol[n - 1]]
    if i == 0 and i == n - 1:  # the first two value == 0, while others < 0
        delta = 0
    return delta

#种群初始化
class Init:
    def __init__(self, points, dim, og, distmat):
        self.points = points
        self.dim = dim
        self.og = og
        self.distmat = distmat
        self.group = []

    # 计算两个向量之间的叉积。返回三点之间的关系：
    def ccw(self,a, b, c):
        return ((b[1] - a[1]) * (c[0] - b[0]))   -    ((c[1] - b[1]) * (b[0] - a[0]) )


# 分别求出后面 n-1 个点与出发点的斜率，借助 sorted，按斜率完成从小到大排序
    def compute(self,next):
        start = self.points[0]  # 第一个点
        if start[0] == next[0]:
            return 99999
        slope = (start[1] - next[1]) / (start[0] - next[0])
        return slope

    def Graham_Scan(self,points):
```

```
95.       # # 找到最左边且最下面的点作为出发点，和第一位互换
96.           Min=999999
97.           for i in range(len(points)):
98.           # 寻找最左边的点
99.               if points[i][0]<Min:
100.                  Min = points[i][0]
101.                  index = i
102.          # 如果同在最左边，可取 y 值更小的
103.              elif points[i][0]==Min:
104.                  if points[i][1]<=points[index][1]:
105.                      Min = points[i][0]
106.                      index = i
107.          # 和第一位互换位置
108.          temp = points[0]
109.          points[0] = points[index]
110.          points[index] = temp
111.          # 排序：从第二个元素开始，按与第一个元素的斜率排序
112.          points = points[:1] + sorted(points[1:], key=self.compute)   # 前半部分是出发点；
    后半部分是经过按斜率排序之后的 n-1 个坐标点
113.          #注意： “+”是拼接的含义，不是数值相加
114.          # 用列表模拟一个栈。（最先加入的是前两个点，前两次 while 必定不成立，从而将点加进去）
115.          convex_hull = []
116.          for p in points:
117.              while len(convex_hull) > 1 and self.ccw(convex_hull[-2], convex_hull[-1],
    p) >= 0:
118.                  convex_hull.pop()
119.              convex_hull.append(p)
120.          person=[x[2] for x in convex_hull]
121.          return person
122.
123.      #优化后的初始种群产生方法
124.      def newCreate(self):
125.          stp=self.Graham_Scan(self.points)
126.          origin=[i for i in range(self.dim)]
127.          for x in stp:
128.                  origin.remove(x)
129.          sto=origin
130.          for i in range(self.og):
131.              person=stp.copy()#不加.copy()是赋了指针！！！
132.              #print("st",person)
133.              origin=sto.copy()
134.              toto,totp=len(sto),len(stp)
135.              #print(toto,totp)
136.              #print(origin)
```

```
            while toto>0:
                x=origin[random.randint(0,toto-1)]
                mn,posmn=999999,0
                for j in range(totp):
                    if j<totp-1:Dis=self.distmat[person[j],x]+self.distmat[x,person[j+1]]-self.distmat[person[j],person[j+1]]
                    else:Dis=self.distmat[person[j],x]+self.distmat[x,person[0]]-self.distmat[person[j],person[0]]
                    if Dis<mn:
                        mn=Dis
                        posmn=j
                person.insert(posmn+1,x)
                totp+=1
                origin.remove(x)
                toto-=1
            self.group.append(person)  # 产生随机个体
            #print(len(person),self.dim)
            #print(person)
        if self.og == 1:return self.group[0]
        return self.group

#Tuning parameters
def tnm_selection(n, adj_mat, sol, max_tnm, mut_md):
    """
    :param n: number of vertices
    :param adj_mat: adjacency matrix
    :param sol: solution where the neighbours are chosen from
    :param max_tnm: how many candidates picked in tournament selection
    :param mut_md: [get_sol, get delta], method of mutation, e.g. swap, 2-opt
    """

    get_new_sol = mut_md[0]
    get_delta = mut_md[1]

    cost = get_cost(n, adj_mat, sol)

    best_delta  = math.inf
    best_i = best_j = -1

    for _ in range(max_tnm):
        i, j = random.sample(range(n), 2)  # randomly select two indexes
        i, j = (i, j) if i < j else (j, i)  # let i < j
        delta = get_delta(n, adj_mat, sol, i, j)
        if delta < best_delta:
```

```
179.                 best_delta = delta
180.                 best_i = i
181.                 best_j = j
182.     new_sol = get_new_sol(sol, best_i, best_j)
183.     new_cost = cost + best_delta
184.     # assert abs(new_cost - get_cost(n, adj_mat, new_sol)) < 1e-9, 'new_sol does not m
    atch new_cost'
185.     return new_sol, new_cost
186.
187. def Search(n, adj_mat, sol, mut_md):
188.     """
189.     :param n: number of vertices
190.     :param adj_mat: adjacency matrix
191.     :param sol: solution where the neighbours are chosen from
192.     :param mut_md: [get_sol, get delta], method of mutation, e.g. swap, 2-opt
193.     """
194.
195.     get_new_sol = mut_md[0]
196.     get_delta = mut_md[1]
197.
198.     cost = get_cost(n, adj_mat, sol)
199.
200.     i, j = random.sample(range(n), 2)  # randomly select two indexes
201.     i, j = (i, j) if i < j else (j, i)  # let i < j
202.     delta = get_delta(n, adj_mat, sol, i, j)
203.
204.     new_sol = get_new_sol(sol, i, j)
205.     new_cost = cost + delta
206.     # assert abs(new_cost - get_cost(n, adj_mat, new_sol)) < 1e-9, 'new_sol does not m
    atch new_cost'
207.     return new_sol, new_cost
```

SA1.py：传统模拟退火算法代码

```
1.  import random
2.  import math
3.  from collections import deque
4.  from common import *
5.
6.  def SA1(n, adj_mat, term_count_1, term_count_2, t_0, alpha):
7.
8.      # initialization
9.      sol = list(range(n))
10.     random.shuffle(sol)  # e.g. [0,1,...,n]
11.
```

```
    cost = get_cost(n, adj_mat, sol)

    best_sol = sol.copy()
    best_cost = cost

    t = t_0

    data = {'cost': deque([]), 'best_cost': deque([]),
            'sol': deque([]), 'best_sol': deque([])}
    count_2 = 0  # outer loop count
    while True:
        count_1 = 0  # inner loop count
        best_inner_sol = sol
        best_inner_cost = cost
        while True:
            last_sol = sol
            last_cost = cost

            mut_md = [get_new_sol_swap, get_delta_swap]

            sol, cost = Search(n, adj_mat, sol, mut_md)
            # mention the iteratively variable 'sol'!
            if cost > last_cost and math.exp((last_cost - cost) / t) < random.random():

                sol = last_sol
                cost = last_cost
            if cost < best_inner_cost:
                best_inner_sol = sol
                best_inner_cost = cost
            count_1 += 1
            data['cost'].append(cost)
            data['best_cost'].append(best_cost)
            data['sol'].append(sol)
            data['best_sol'].append(best_sol)
            if count_1 > term_count_1:
                break
        # end of inner loop
        # get best_inner_sol < sol
        t = alpha * t
        if best_inner_cost < best_cost:
            best_sol = best_inner_sol
            best_cost = best_inner_cost
        count_2 += 1
        if count_2 > term_count_2:
```

```
            break
    return best_sol, best_cost, data
```

SA2.py：改进模拟退火算法代码——引入 2-opt 算法进行解变换

```
import random
import math
from collections import deque
from common import *

def SA2(n, adj_mat, term_count_1, term_count_2, t_0, alpha):

    # initialization
    sol = list(range(n))
    random.shuffle(sol)  # e.g. [0,1,...,n]

    cost = get_cost(n, adj_mat, sol)

    best_sol = sol.copy()
    best_cost = cost

    t = t_0

    data = {'cost': deque([]), 'best_cost': deque([]),
            'sol': deque([]), 'best_sol': deque([])}
    count_2 = 0  # outer loop count
    while True:
        count_1 = 0  # inner loop count
        best_inner_sol = sol
        best_inner_cost = cost
        while True:
            last_sol = sol
            last_cost = cost

            mut_md = [get_new_sol_2opt, get_delta_2opt]

            sol, cost = Search(n, adj_mat, sol, mut_md)
            # mention the iteratively variable 'sol'!
            if cost > last_cost and math.exp((last_cost - cost) / t) < random.random():

                sol = last_sol
                cost = last_cost
            if cost < best_inner_cost:
                best_inner_sol = sol
                best_inner_cost = cost
```

```
            count_1 += 1
            data['cost'].append(cost)
            data['best_cost'].append(best_cost)
            data['sol'].append(sol)
            data['best_sol'].append(best_sol)
            if count_1 > term_count_1:
                break
        # end of inner loop
        # get best_inner_sol < sol
        t = alpha * t
        if best_inner_cost < best_cost:
            best_sol = best_inner_sol
            best_cost = best_inner_cost
        count_2 += 1
        if count_2 > term_count_2:
            break
    return best_sol, best_cost, data
```

SA3.py：改进模拟退火算法代码——自适应的循环次数上限

```
import random
import math
from collections import deque
from common import *

def SA3(n, adj_mat, term_count_1, term_count_2, t_0, alpha):

    # initialization
    sol = list(range(n))
    random.shuffle(sol)  # e.g. [0,1,...,n]

    cost = get_cost(n, adj_mat, sol)

    best_sol = sol.copy()
    best_cost = cost

    t = t_0

    data = {'cost': deque([]), 'best_cost': deque([]),
            'sol': deque([]), 'best_sol': deque([])}
    count_2 = 0  # outer loop count
    while True:
        count_1 = 0  # inner loop count
        best_inner_sol = sol
        best_inner_cost = cost
```

```
        while True:
            last_sol = sol
            last_cost = cost

            mut_md = [get_new_sol_2opt, get_delta_2opt]

            sol, cost = Search(n, adj_mat, sol, mut_md)
            # mention the iteratively variable 'sol'!
            if cost > last_cost and math.exp((last_cost - cost) / t) < random.random():

                sol = last_sol
                cost = last_cost
            if cost < best_inner_cost:
                best_inner_sol = sol
                best_inner_cost = cost
                count_1 = 0
            else:
                count_1 += 1
            data['cost'].append(cost)
            data['best_cost'].append(best_cost)
            data['sol'].append(sol)
            data['best_sol'].append(best_sol)
            if count_1 > term_count_1:
                break
        # end of inner loop
        # get best_inner_sol < sol
        t = alpha * t
        if best_inner_cost < best_cost:
            best_sol = best_inner_sol
            best_cost = best_inner_cost
            count_2 = 0
        else:
            count_2 += 1
        if count_2 > term_count_2:
            break
    return best_sol, best_cost, data
```

SA4.py：改进模拟退火算法代码——多种解变换方式的并行

```
import random
import math
from collections import deque
from common import get_cost
from common import *

```

```
def SA4(n, adj_mat, term_count_1, term_count_2, t_0, alpha):

    # initialization
    sol = list(range(n))
    random.shuffle(sol)  # e.g. [0,1,...,n]

    cost = get_cost(n, adj_mat, sol)

    best_sol = sol.copy()
    best_cost = cost

    t = t_0

    data = {'cost': deque([]), 'best_cost': deque([]),
            'sol': deque([]), 'best_sol': deque([])}
    count_2 = 0  # outer loop count
    while True:
        count_1 = 0  # inner loop count
        best_inner_sol = sol
        best_inner_cost = cost
        while True:
            last_sol = sol
            last_cost = cost

            Select = random.randint(1,4)
            if Select == 1 or Select == 2: mut_md = [get_new_sol_2opt, get_delta_2opt]

            # if Select == 2: mut_md = [get_new_sol_swap, get_delta_swap]
            if Select == 3 or Select == 4: mut_md = [get_new_sol_move, get_delta_move]

            sol, cost = Search(n, adj_mat, sol, mut_md)
            # mention the iteratively variable 'sol'!
            if cost > last_cost and math.exp((last_cost - cost) / t) < random.random():

                sol = last_sol
                cost = last_cost
            if cost < best_inner_cost:
                best_inner_sol = sol
                best_inner_cost = cost
                count_1 = 0
            else:
                count_1 += 1
            data['cost'].append(cost)
```

```
            data['best_cost'].append(best_cost)
            data['sol'].append(sol)
            data['best_sol'].append(best_sol)
            if count_1 > term_count_1:
                break
        # end of inner loop
        # get best_inner_sol < sol
        t = alpha * t
        if best_inner_cost < best_cost:
            best_sol = best_inner_sol
            best_cost = best_inner_cost
            count_2 = 0
        else:
            count_2 += 1
        if count_2 > term_count_2:
            break
    return best_sol, best_cost, data
```

SA5.py：改进模拟退火算法代码——在解变换中尝试调参

```
import random
import math
from collections import deque
from common import get_cost
from common import *

def SA5(n, adj_mat, max_tnm, term_count_1, term_count_2, t_0, alpha):

    # initialization
    sol = list(range(n))
    random.shuffle(sol)  # e.g. [0,1,...,n]

    #sol = Init(points, n, 1, adj_mat).newCreate()
    cost = get_cost(n, adj_mat, sol)

    best_sol = sol.copy()
    best_cost = cost

    t = t_0

    data = {'cost': deque([]), 'best_cost': deque([]),
            'sol': deque([]), 'best_sol': deque([])}
    count_2 = 0  # outer loop count
    while True:
        count_1 = 0  # inner loop count
```

```
        best_inner_sol = sol
        best_inner_cost = cost
        while True:
            last_sol = sol
            last_cost = cost

            Select = random.randint(1,4)
            if Select == 1 or Select == 2: mut_md = [get_new_sol_2opt, get_delta_2opt]

            # if Select == 2: mut_md = [get_new_sol_swap, get_delta_swap]
            if Select == 3 or Select == 4: mut_md = [get_new_sol_move, get_delta_move]


            sol, cost = tnm_selection(n, adj_mat, sol, max_tnm, mut_md)
            # mention the iteratively variable 'sol'!
            if cost > last_cost and math.exp((last_cost - cost) / t) < random.random():

                sol = last_sol
                cost = last_cost
            if cost < best_inner_cost:
                best_inner_sol = sol
                best_inner_cost = cost
                count_1 = 0
            else:
                count_1 += 1
            data['cost'].append(cost)
            data['best_cost'].append(best_cost)
            data['sol'].append(sol)
            data['best_sol'].append(best_sol)
            if count_1 > term_count_1:
                break
        # end of inner loop
        # get best_inner_sol < sol
        t = alpha * t
        if best_inner_cost < best_cost:
            best_sol = best_inner_sol
            best_cost = best_inner_cost
            count_2 = 0
        else:
            count_2 += 1
        if count_2 > term_count_2:
            break
    return best_sol, best_cost, data
```