实验报告

——————模拟退火算法

姓名：蔡梓珩　　学号：16340008　　日期：2018/12/14

摘要：该项目主要是利用局部搜索算法（LS）和模拟退火算法（SA）解决 TSP 问题。先是使用 LS 求解 TSP 问题，再尝试 SA 问题，比较两者，在效率上 SA 更占有。最后再在 LS 的基础上使用 SA，再优化 SA 部分算法，尝试求解 TSP 问题。选用的 TSP 测例为 eil101（有 101 个城市）。代码使用 python 语言编写，因此运算速度因为语言特性比编程语言要低。

1．导言

旅行商问题，即 TSP 问题（Traveling Salesman Problem），是求最短路径的问题，即“已给一个 n 个点的完全图，每条边都有一个长度，求总长度最短的经过每个顶点正好一次的封闭回路”。TSP 是组合优化问题，可以被证明具有 NPC 计算复杂性。如果希望暴力搜索其最佳解，其复杂度将是 O(n!)，其计算量随着 n 的增加将轻易超过目前计算机的可能算力。因此我们需要用更智能的方法求解。

于是我们先考虑局部搜索算法。局部搜索算法是贪心算法，他往往往邻域中最好的状态搜索，因此容易进入局部最优结果，而无法跳出局部最优的区域。

第二部分使用模拟退火算法。模拟退火算法从某一较高初温出发，伴随温度参数的不断下降,结合概率突跳特性在解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。模拟退火算法比起局部搜索算法，赋予了一定跳出局部最优解的能力，但能否跳出局部最优解依然依赖随机性。

2．实验过程

首先使用两种不同的局部搜索算法。

第一种选择邻域的方法是随机交换两个城市在序列中的顺序。每次循环中产生的候选序列为城市数（以下用 Cs 表示）*10，并从中选择一个最优的（距离最短的）作为下一步。

第二种选择邻域的方法是随机交换三个城市在序列中的顺序。每次循环中产生的候选序列为 Cs*10，并从中选择一个最优的（距离最短的）作为下一步。

这两种算法都按以下步骤实现：

1. 录入初始状态，并打乱顺序产生一组随机状态，从这组状态（包括初始状态）中选最佳的状态作为起点；
2. Repeat:
    a) 产生一个集合 S
    b) Repeat 10 * Cs times:
        i. 将当前状态加入 S

ii. 产生 2 个（或 3 个）互不相同的、范围为[1, 城市数-1]的随机数
iii. 以这 2 个（或 3 个）随机数作为下标交换城市在序列中的顺序
iv. 将交换后的序列加入 S 中

c) 从 S 中选择一个最优的序列，作为当前状态
d) 如果当前状态与之前状态一样，则跳出循环。

可以知道，当当前状态与邻域中最佳状态一样时跳出循环，可以理解成到达局部最优解。虽然实际上这个邻域并没有完全覆盖当前状态的所有邻居，但覆盖全部邻居需要(Cs-1) * (Cs-2)（第二种邻域为(Cs-1) * (Cs-2) * (Cs-3)）个数据，将加大每次循环的耗时，而且最终结果同样是会进入局部最优结果而无法跳出。

第二部分在 LS 的基础上加入 SA。
一开始我的 SA 流程如下：

1. 得到初始状态，设定初温 T，降温方式，结束条件
2. 外循环：
   a) 当符合结束条件则跳出循环
   b) 内循环：
      i. 令当前解能量为 D0
      ii. 通过邻域搜索策略得到一组解并取其中最优（不包括当前状态）解能量为 D1
      iii. 令 ΔE = D1 - D0
         1. If ΔE <= 0: 则使 P = 1
            Else: 使 P 为$e^{-\Delta E/T}$（或其他形式，其 P 应随着 T 降低而降低，而且 ΔE 越小则越高）。
         2. 产生一个[0,1)的小数 R，若 R<P 则接受新状态，否则不接受。
   c) 降温

而本次实验使用了非传统的 SA——DSA-CE&MAP[1]

Step 1: Initialize cooling_rate with a small value such as 0.001.
Generate an initial random tour x.

Step 2: Initialize T with a large value such as 100000.

Step 3: if totalCities < 30, then set coolingEnhancer = 0.5.
else if totalCities < 150 then set coolingEnhancer = 0.05.
else if totalCities < 750 then set coolingEnhancer = 0.005.
Otherwise, set coolingEnhancer = 0.0005.

Step 3: Repeat:
i. Generate next tour (x + Δx) by applying some operations on the current tour x.
ii. Evaluate ΔE(x) = E(x + Δx) − E(x), (i.e. neighborTourCost - currentTourCost):
if ΔE(x) < 0, keep the new state (i.e. new path distance less than current distance);
otherwise, evalutate ΔE` = $E_{bestSoFar}$(x) - E(x + Δx), (i.e. bestTourCost - neighborTourCost) and then accept the new state with
acceptance probability, $P = \frac{e-\Delta E/T}{e-\Delta E`/T}$
iii. If $E_{bestSoFar}$(x) > E(x + Δx), then set $E_{bestSoFar}$(x) = E(x + Δx).
iv. Set T = T − ΔT , ∓ ΔT= T x coolingEnhancer x cooling_rate.
until T is small enough.

（以上为 DSA-CE&MAP 论文中描述的过程）
使用该种策略能在经典 SA 的基础上更合理的降温且更合理的得到选择概率。
观察概率函数可以发现，新解不仅与当前解比较，还与最佳解比较。用到概率函数的前

提是当前解比新解好。当新解与当前解差距大的时候，分子会减小，P 减小，符合策略。当新解与最优解差距大的时候（注意这里是最优解 – 新解），分母会增大，P 减小，符合策略。即，一个新解不仅考虑与当前解的差距，还考虑与曾到达的最优解的差距。这样每次升温将考虑到更多因素，使每次升温更慎重。

这里还引入了一个新的参数 coolingEnhancer 来影响降温策略。当城市越多的时候，因为每个状态将更复杂，引入 coolingEnhancer 使其降温速度更慢，使外循环迭代次数增加，增强算法的适应能力。

在 DSA-CE&MAP 的基础上，邻域搜索策略我再作了修正，由于前两种局部搜索策略效果不佳，使用了第三种局部搜索策略（2-OPT）：

若 W(I, I+1) + W(J, J+1) > W(I, J) + W(I+1, J+1)

则用边(I,J)和(I+1,J+1)替换(I,I+1),(J,J+1)

其中 I，J 为某两座城市的下标，W(a,b)表示城市 a 到下一座城市 b 的距离。这种策略能很好的解决路线交叉的问题，而上面两种交换城市的方法很难处理路线交叉。这种方法可以理解成用凸四边形的两条对边代替两条对角线（好的效果）。

这种边的替代依赖于该问题中城市之间的距离是对称的（即交换两个序列中相邻的城市的顺序不会影响两城市之间的距离）。

假设原本的顺序是 i,i+1,s[n],j,j+1，则边替换后则变成 i,j,s[n].r,i+1,j+1，其中 i+1 与 j 之间的路线将会因为先到达 j 再到达 i+1 而反转。我们观察可以发现 i 和 j+1 是没有变动的。S2 = (i+1) + s[n] + (j)是整个反转了。因此我们只需要获得两个随机下标并将其中的城市序列反转即可得到新状态。

用与其他搜索同样的方法得到一个关于序列的集合，并挑最优解。

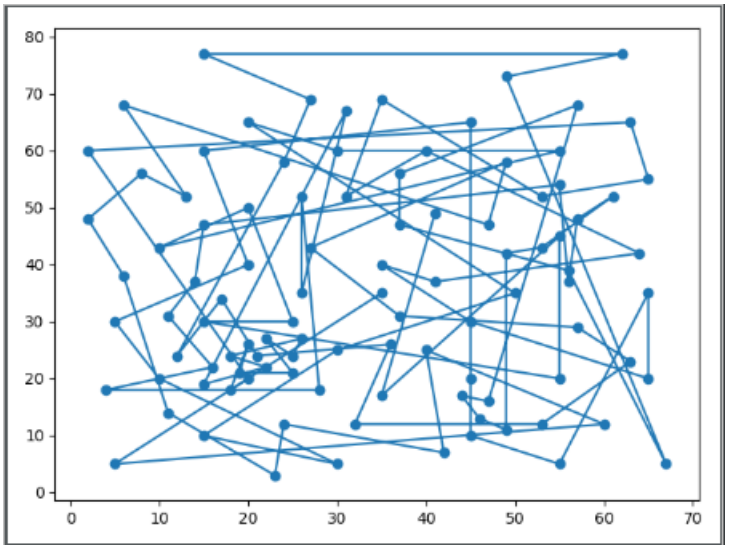
由于 DSA-CE&MAP 中给出的初温过高，因此将初温降低为 1000，并将结束条件设置为 T<1（试运行发现 T<1 后基本到达局部最优解），以进一步提升速率。

在结合 DSA-CE&MAP（改良模拟退火）与 2-OPT（局部搜索）后，达到了实验目标的 10%。

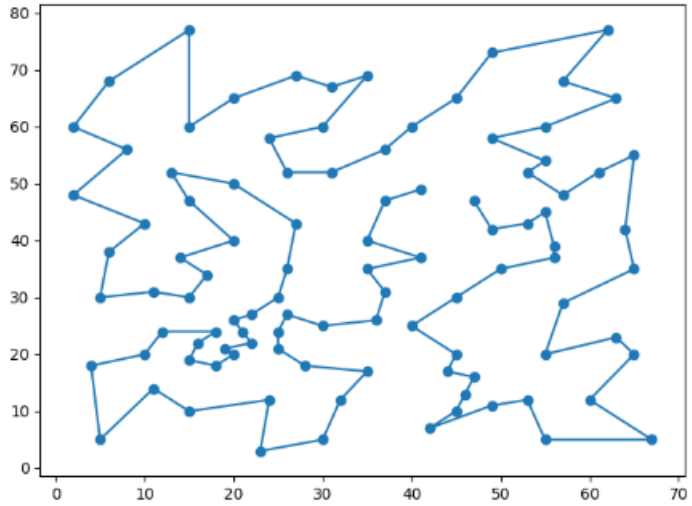
3．结果分析

测例：eil101，最优解 629

初始状态：



在使用 T = 1000，邻域大小为 10，降温速率为 0.001 时得到的一个最优解：



可以看到最优解已经没有交叉路线（这是 2-OPT 的功劳，实际上即使没有模拟退火，只有 2-OPT 也能轻易达到没有交叉路线的结果），而且路线尽可能的圆润。

运行环境为 windows10，Intel Core i5-8400 2.81GHz，RAM 2667MHz 16GB
编译器 PyCharm，语言 Python3

以下为不同参数下的运行结果：
以下为不同策略得到的结果，每组测试 10 次。

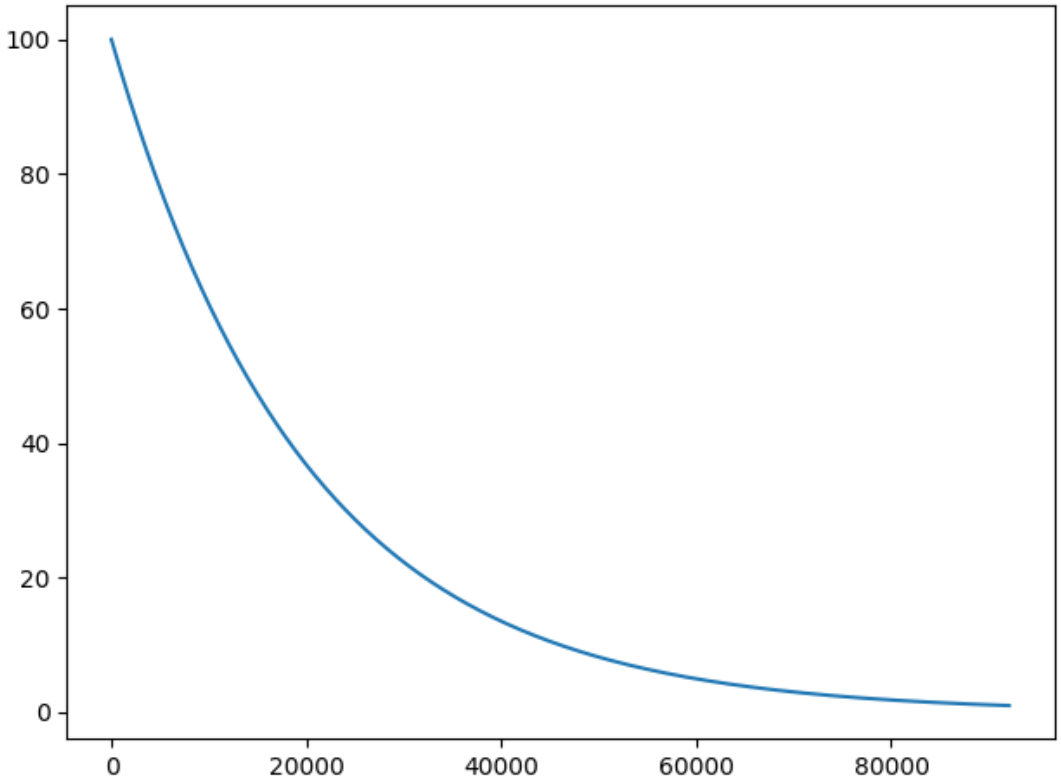
局部搜索	Best	Excess	Worst	Excess	Adv	Adv Time(s)
策略一	994.6	58.00%	1104.4	75.6%	1057.5	27.0
策略二	1256.3	99.7%	1362.0	116.5%	1311.1	11.2
策略三	695.1	10.5%	767.4	22%	727.6	18.2
模拟退火	Best	Excess	Worst	Excess	Adv	Adv Time(s)
T = 1000 Range = 10 Coolrate = 0.001	651.0	3.5%	673.7	7.1%	661.5	308.9
T = 100 Range = 10 Coolrate = 0.001	653.4	3.9%	676.4	7.5%	663.8	207.1
T = 100 Range = 5 Coolrate = 0.001	659.1	4.8%	677.3	7.68%	667.6	140.2
T = 100 Range = 1 Coolrate = 0.001	664.0	5.57%	698.5	11.05%	682.6	82.4

其中策略一为交换两个城市，策略二为交换三个城市，策略三为 2-OPT（部分逆转）
T 为初温，Range 为内循环中邻域大小（样本个数），Coolrate 为降温速率。
可以看到 2-OPT 的比起单纯交换城市有好很多的效果。而对比模拟退火，能看

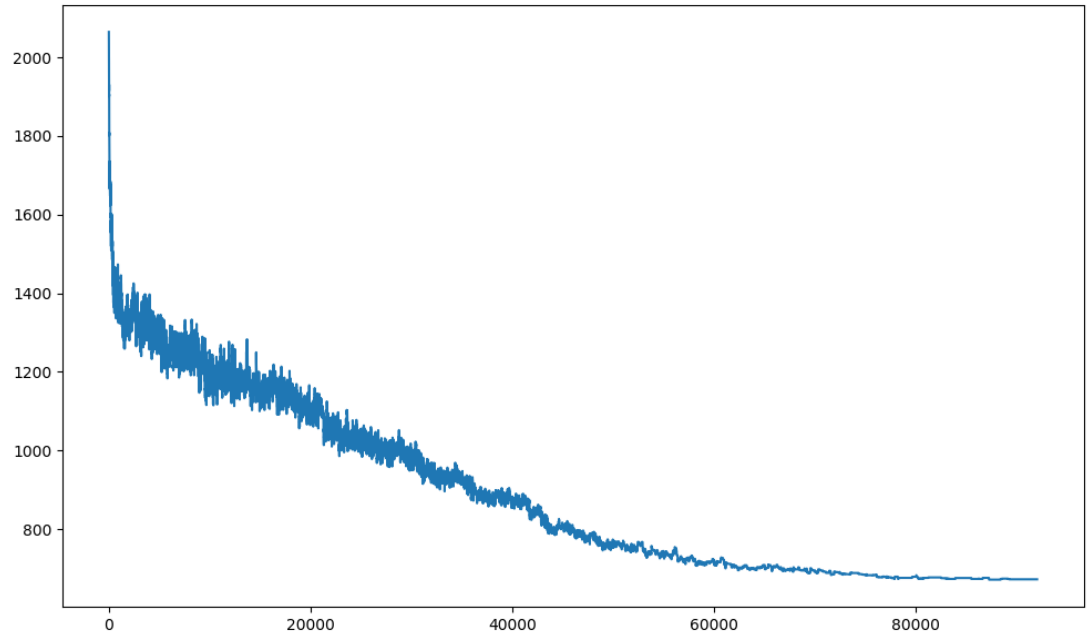
到当温度减少或邻域范围减小，最终解都会变差。但是减少初温或减少内循环的邻域大小能明显减少时间消耗，其中第一行是第三行（模拟退火内）的时间的两倍，而 Excess 相差仅 1%。

其中选取了一个数据如下的样本
END Distance: 672.5236373155444
Times: 92102
Excess: 6.92%
totally cost 150.7281141281128 s

其中 T 的曲线如图：



而每次外循环迭代时状态的距离如图：



可以看到温度的下降是非线性且平稳的单调递减的，而状态的值则有起伏，在越早期欺负越多，越到后期则越趋于平稳，这都是符合 SA 的规律的

算法中其实还可以加入类似升温、更好初始解等方法提高最终解的质量，但是升温会显著延长搜索时间。若升温条件苛刻，则每次升温前置时间过长；若升温条件简易，则容易频繁升温难以收敛。升温的程度也是需要调试的部分。
而对于初始解，可以考虑不一定在随机初始组中选最优者作为起点。
对于需要大量迭代的如 SA 的算法，实际上使用 java 等语言配合 matlab 或 python 作图会更有效率，因为 python 的运行速度确实不如 java 或 C 等编程语言快。

4．结论

实际一开始使用的 SA 是局部搜索策略一、或策略一结合策略二得到的邻域配合经典 SA。由于该局部搜索策略过于简单，导致即使调高邻域范围、调高初温、调低降温速率，结果耗费大量时间（一个小时）迭代也无法进入 10%的解。后来即使改进了 SA 的策略，提升是有，但也无法进入 10%，而且还增加了时间消耗。最终以上方案轻易被 2-OPT 这个局部搜索解超过。因此选择一个正确的局部搜索策略十分重要。可以看到 SA 对 LS 的提升是有的，但响应的，也会耗费更多的时间，耗费的时间主要取决与邻域的范围、初温、降温策略。因此需要多此调试得到最佳参数。

主要参考文献(三五个即可)

[1] Akshay Vyas, Dashmeet Kaur Chawla, Dr. Urjita Thakar, “Dynamic Simulated Annealing for solving the Traveling Salesman Problem with Cooling Enhancer and Modified Acceptance Probability”
[2] TSPLIB https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
[3] 赵赫，杜端甫，《TSP 的邻域搜索算法的分析和改进》