**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:
   1. 产生一个集合S
   2. Repeat 10 \* Cs times:
      1. 将当前状态加入S
      2. 产生2个（或3个）互不相同的、范围为[1, 城市数-1]的随机数
      3. 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
      4. 将交换后的序列加入S中
   3. 从S中选择一个最优的序列，作为当前状态
   4. 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。

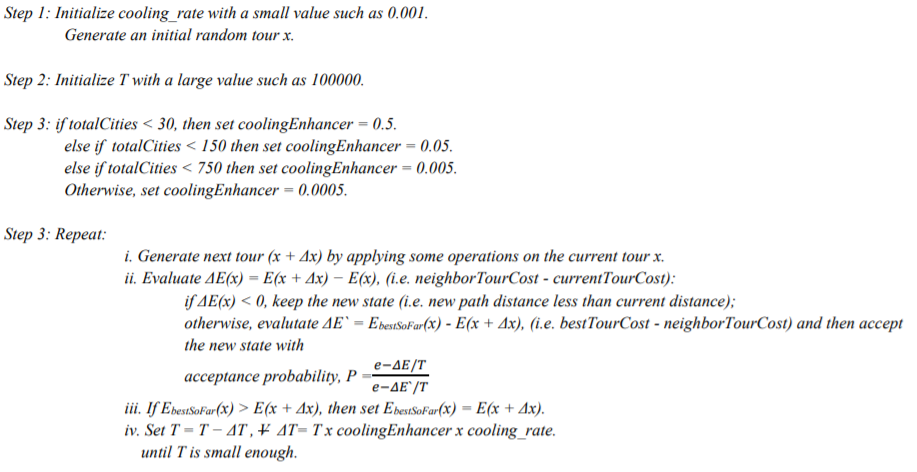
一开始我的SA流程如下：

1. 得到初始状态，设定初温T，降温方式，结束条件
2. 外循环：
   1. 当符合结束条件则跳出循环
   2. 内循环：
      1. 令当前解能量为D0
      2. 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
      3. 令ΔE = D1–D0
         1. If ΔE <= 0: 则使P = 1

Else: 使P为（或其他形式，其P应随着T降低而降低，而且ΔE越小则越高）。

* + - 1. 产生一个[0,1)的小数R，若R<P则接受新状态，否则不接受。
  1. 降温

而本次实验使用了非传统的SA——DSA-CE&MAP[1]



（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。

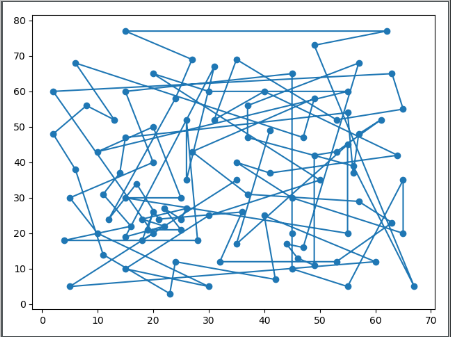
由于DSA-CE&MAP中给出的初温过高，因此将初温降低为1000，并将结束条件设置为T<1（试运行发现T<1后基本到达局部最优解），以进一步提升速率。

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%。

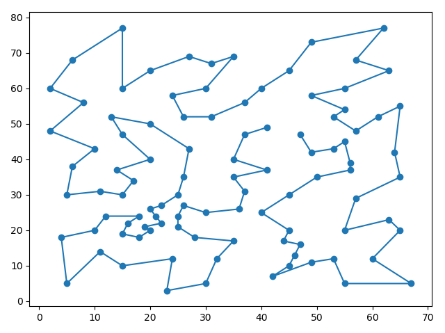
**3．结果分析**

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

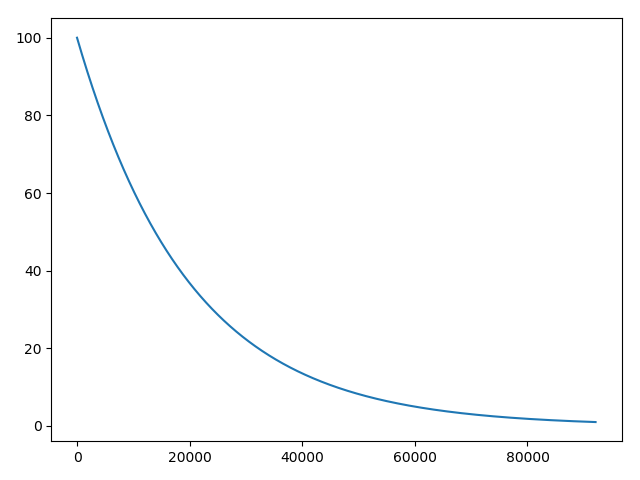
END Distance: 672.5236373155444

Times: 92102

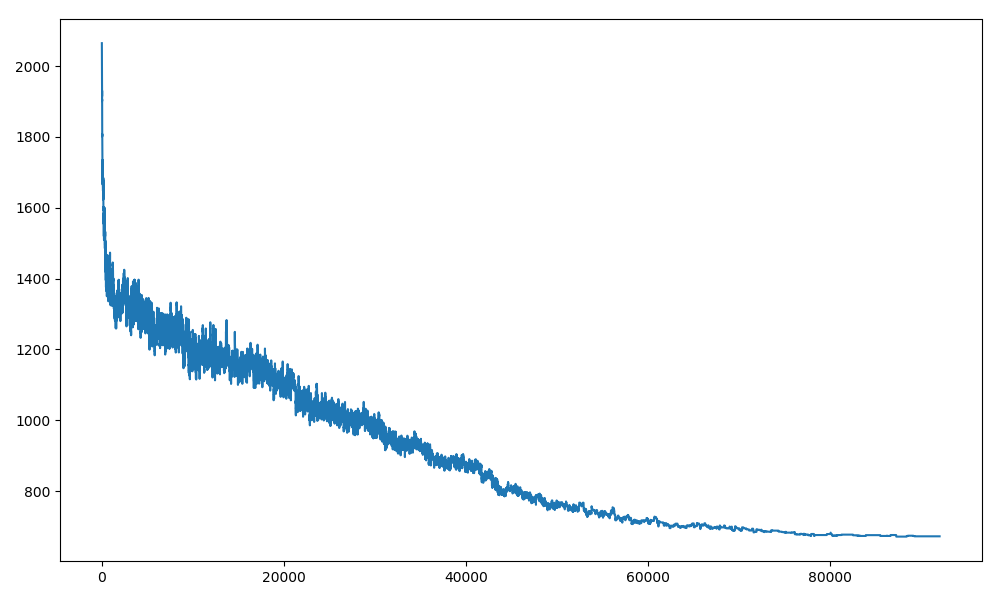
Excess: 6.92%

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》