模拟退火实验报告

姓名：欧穗新
学号：16340173
日期：2019 年 1 月 12

摘要：

本次实验在 TSPLIB 中选了 ch130（130 个城市）的 TSP 问题，然后采用多种邻域操作的局部搜索 local search 策略求解（包括使用交换城市、逆转路径两种邻域操作），并在局部搜索策略的基础上，加入模拟退火 simulated annealing 策略，最终模拟退火求得的解不超过最优值的 5%，第一种局部搜索求得的解超过最优值的 10%，第二种局部搜索求得的解不超过最优值的 10%。得出结论局部搜索能够在很快的时间内得到比较好的解，但是极易陷入局部最优无法跳出，并且合适的邻域操作能够改进搜索结果。而模拟退火算法能够有一定跳出局部最优的能力，稳定得出全局较优解，但是缺点是用时更长、求得的解不是整个过程遇到的最优解等。除此之外，代码能够提供可视化，观察路径的变化和交叉程度。

1．导言

旅行推销员问题（英语：Travelling salesman problem, TSP）是这样一个问题：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。它是组合优化中的一个 NP 困难问题，在运筹学和理论计算机科学中非常重要。
最早的旅行商问题的数学规划是由 Dantzig（1959）等人提出，并且是在最优化领域中进行了深入研究。许多优化方法都用它作为一个测试基准。尽管问题在计算上很困难，但已经有了大量的启发式算法和精确方法来求解数量上万的实例，并且能将误差控制在 1%内

使用的方法：

（1）局部搜索，局部搜索是解决最优化问题的一种元启发式算法。 局部搜索从一个初始解出发，然后搜索解的邻域，如有更优的解则移动至该解并继续执行搜索，否则返回当前解。

（2）模拟退火，模拟退火是对热力学退火过程的模拟,在某一给定初温下，通过缓慢下降温度参数，使算法能够在多项式时间内给出一个近似最优解。本质上也是蒙特卡洛算法。作为一种比较简单的智能算法，能以较高的效率解决优化问题，如求最值，tsp 问题等。

2．实验过程

算法概述：

爬山法（局部搜索算法）：

（1）设置随机产生的初始解

（2）产生临近解，如果临近解更好就接受，并更新当前解，否则丢弃生成的邻居解

产生解的方法（产生邻域的操作）：

（i）开始使用的是交换路径上的两个点，以此产生邻居：

当前解：A->B->C->D->E->F

交换路径上的 B、E 两个节点

得到邻居解：A->E->C->D->B->F

（ii）后来使用的方法是“逆转两个节点之间的路径”，以此产生邻居：

当前解：A->B->C->D->E->F

逆转子路径 B->C->D->E

得到邻居解：A->E->D->C->B->F

（3）一直执行（2）10 万次，将最后的解作为结果输出

模拟退火：

（1）设置初始温度 T=100

（2）温度固定情况下执行：

（i）产生临近解，如果临近解更好就接受，并更新当前解，否则以 exp(-ΔE/T) 的概率接受该解

产生解的方法使用爬山法中的（i）（ii），其中（i）效果很差（有时能限制与最优解的差距到 10%以内，大多数时候不行）。

所以最终采用（ii），能够稳定跑到 10%之内，调整好参数可以跑到 5%之内

（ii）执行（i）1000 次，然后跳到（3）

（3）将温度下降：T = T*0.99，如果温度 T 小于最低温度 T_min=0.5，结束，输出当前解

实现算法的程序主要流程，功能说明

（1）generate_random_list 函数产生初始随机解，输入问题规模（有多少个城市），返回一种随机的走法：

```
def generate_random_list(point_num):
    point_list = [0]*(point_num - 1)
    for i in range(0, point_num - 1):
        point_list[i] = i+1
    random.shuffle(point_list)
    point_list.insert(0, 0)
    point_list.append(0)
    return point_list
```

（2）get_distance 获取两点间的距离，将两个城市的位置作为参数，返回距离，用于计算整个 tsp 路径长度

```
def get_distance(point_a, point_b):
    temp1 = math.pow(coordinate_x[point_a] - coordinate_x[point_b], 2)
    temp2 = math.pow(coordinate_y[point_a] - coordinate_y[point_b], 2)
    return math.pow(temp1 + temp2, 0.5)
```

（3）

ExchangeLocalSearch 类，其 solve 函数使用交换两个节点作为邻域操作，使用局部搜索爬山法求解 tsp 最短路

InverseLocalSearch 类，其 solve 函数使用逆转两个节点之间的路径作为邻域操作（来产生新解），使用局部搜索爬山法求解 tsp 最短路

SimulateAnneal 类，其 solve 函数使用交换两个节点/逆转两个节点之间的路径作为领域操作，使用模拟退火进行求解

```
class ExchangeLocalSearch:
class InverseLocalSearch:
class SimulateAnneal:
```

（4）

show_figure 函数用于将 tsp 问题可视化，显示出；路径求解的具体过程

store_result 函数用于将求解的结果存入文件中

```
def show_figure(self):
def store_result(self):
```

（5）

swap_two_point()，邻域操作函数，交换两个节点来产生新解

swap_sub_path()，邻域操作函数，逆转两个节点之间的路径来产生新解

```
def swap_two_point(self):
def swap_sub_path(self):
```

以及三个解题类的 solve 函数：

（6）ExchangeLocalSearch 的 solve 函数，遍历 100*1000 次，不断使用 swap_two_point 邻域操作产生更好的解，显示求解过程，并将最终的解存储：

```
def solve(self):
    # 进行迭代求解
    for i in range(0, 100*MARKOV_LENGTH):
        if self.swap_two_point():
            print('使用交换点策略的局部搜索 ', ' 第', i, '轮 ', '路径长度更新为:
', self.path_count)

            self.path_count_set.append(self.path_count)
            if IF_SHOW_FIGURE:
                self.show_figure()
    self.store_result()
```

（7）InverseLocalSearch 的 solve 函数，遍历 100*1000 次，不断使用 swap_sub_path 邻域操作产生更好的解，显示求解过程，并将最终的解存储：

```
def solve(self):
    # 进行迭代求解
    for i in range(0, 100*MARKOV_LENGTH):
        if self.swap_sub_path():
            print('使用交换路径策略的局部搜索 ', ' 第', i, '轮 ', '路径长度更新为：
', self.path_count)

            self.path_count_set.append(self.path_count)
            if IF_SHOW_FIGURE:
                self.show_figure()
    self.store_result()
```

（8）SimulateAnneal 的 solve 函数，外层遍历由温度控制，从初温 0.4 到末温 0.01，降温系数 0.99，内层循环固定温度做 1000 次，内层循环中不断使用 swap_sub_path 邻域操作产生更好的解，并显示求解过程，在温度降到 0.01 之下，跳出外层循环，将最终的解存储：

```
def solve(self):
    # 进行迭代求解
    while self.temperature > MIN_TEMPERATURE:
        for i in range(0, MARKOV_LENGTH):
            if self.swap_sub_path():
                print('使用模拟退火', ' 当前温度：', self.temperature, ' 第', i,
'轮 ', '路径长度更新为：', self.path_count)
                self.path_count_set.append(self.path_count)
                if IF_SHOW_FIGURE:
                    self.show_figure()
        self.temperature *= ATTENUATION_QUOTIENT
    self.store_result()
```

由于逆转子路径的邻域操作效果好很多，所以最终使用逆转子路径作为邻域操作

3．结果分析

实验环境：

该实验在 Windows 下使用 python3 编码实现

参数说明：

```
MAX_TEMPERATURE = 0.4    # 如果是超过250个点就改成1
MIN_TEMPERATURE = 0.01     # 如果是超过250个点就改成0.01
MARKOV_LENGTH = 2000         # 越大越好，写文件的时候设置大一点，显示图片的时候设置小一点
ATTENUATION_QUOTIENT = 0.99
```

图片中显示

局部搜索迭代次数 100*1000 次

模拟退火算法初温 0.4，末温 0.01，降温系数 0.99，内层循环迭代次数 1000 次

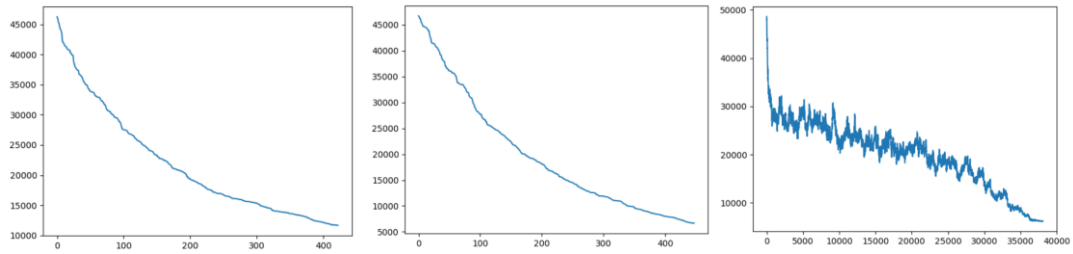
实验结果：（ch130 问题）

（1）表格 1

指标：（运行 10 次之后）	局部搜索 1	局部搜索 2	模拟退火
最好解	10876	6669	6203
最差解	11674	6850	6382
平均值	11319.2	6730	6305
标准差	337.09005	125.29276	65.50159
平均偏差值	85.25%	10.14%	3.19%

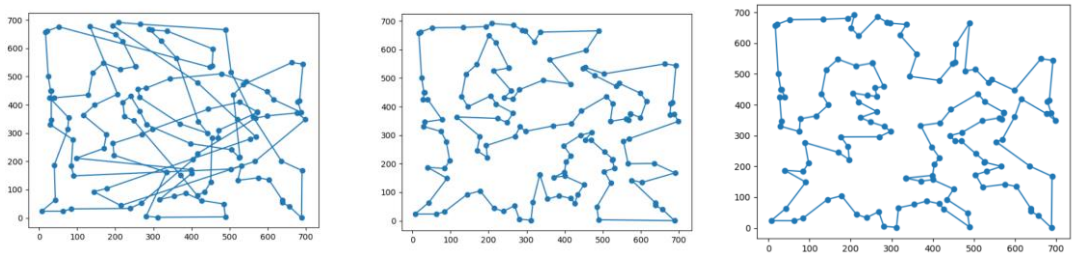
（2）图像 1（取 10 次运行其中的一组）

求解问题时随迭代次数增加，tsp 解对应路径长度变化曲线：（从左往右依次对应局部搜索 1，局部搜索 2，模拟退火算法求解时 tsp 对应路径长度变化曲线）



（3）图像 2（取 10 次运行其中的一组）

最终 tsp 路径连线图：（从左往右依次对应局部搜索 1，局部搜索 2，模拟退火算法求解所得最终 tsp 路径连线图）



性能分析：

（1） 表格 1 对比分析

（i）使用局部搜索 1，每次得到的解有好有坏，**标准差非常大**，这是因为初始解的随机选择，如果初始解选的好，最后得到的局部最优解比较好，否则会较差，从**平均值以及平均偏差值**看来，该方法也非常差，偏离最优解 85%之多

（ii）使用第二种局部搜索时也有类似的缺点（得到的解时好时坏，与初始随机解有关，随机解设置的好，就能够得到较好的局部最优解），但是相比第一种好很多。从**标准差**来看，这种方法求得的解还算比较稳定，标准差不大，从**平均值以及平均偏差值**，这种局部搜索仅仅偏离最优值 10%左右，可见对于邻域操作的改进能大幅提升算法效率

（iii）使用**模拟退火能够稳定得到比较好的解，从表一的标准差可以看出**，模拟退火算法求得结果的标准差相比局部搜索 2 更加小，因此算法性能更加稳定，而不像局部搜索，时好时坏（差的时候非常差），而模拟退火即使差的时候也能距离最优解 5%之内（从**平均值以及平均偏差值**看来）

（2） 图像 1 对比分析

（i）使用局部搜索求解过程中随迭代次数增加，tsp 解对应路径长度一直变小，这是因为爬山法只接受更好的解，所以长度只会不断减小

（ii）**求解时下降过程非常快这是爬山法的优点**，仅仅用了 400-500 多步就收敛

（iii）使用模拟退火求解过程中随迭代次数增加，tsp 解对应路径**长度不断波动**，并且在求解的前期波动剧烈，求解后期波动平缓甚至没有波动。这是因为模拟退火算法能够以一定的概率接受差解，所以随着迭代次数增加，tsp 路径长度能够变大或者变小，温度高时（前期）接受差解概率大，所以波动大，温度低时（后期）接受差解的概率小所以波动小，趋于平缓；

（iv）使用模拟退火方法求解时解的值（路径总长度）**下降非常慢（比前面的局部搜索慢几倍），这是模拟退火算法的缺点**，用了 40000 多步才完成整个求解过程，前期波动下降，后期稳定下降；

（3） 图像 2 对比分析

（i）使用局部搜索 1 所得**最终的解看起来虽然比随机产生路径好，但是还是非常差，路径交错**很多，长度为 10000 以上，而真正的最优解是 6110，超出接近 100%，这是这个算法的局限性，由于不能接受差解，所以无法跳过局部最优，达到全局最优（或者其它更优的局部最优）

（ii）使用局部搜索 2 所得最终的解看起来比局部搜索 1 所得最终解路径更好（几乎没有太多路径交叉），但是还是比较差（个别顺序还可以优化），解路径长度 6669，而真正的最优解是 6110，超出接近 10%，**虽然比前面的 ExchangeLocalSearch 类的解法更好，但是后面马上就能看到，这种方法比模拟退火差不少，而且得到的解时好时坏，这也是缺点**，这是这个算法的局限性，由于不能接受差解，所以无法跳过局部最优，达到全局最优（或者其它更优的局部最优）

（iii）使用模拟退火算法所得**解路径长度 6203，而真正的最优解是 6110，偏差最优解仅仅 1%~2%，完全符合老师要求**，这已经非常接近最优解了，比其前面两种局部搜索优秀太多，我们看到最终 tsp 路径连线图中，不仅没有了路径交

叉，相比之前的 InverseLocalSearch 局部搜索方法求出的解，在一些路径的选择上更好，这是**因为模拟退火为了防止陷入局部最优，会以一定的概率接受差解，从而跳出局部最优，接受差解的概率会随着温度下降、差解劣质程度上升而降低，因此在遇到比较好的解时则跳出概率较小（遇到最优解或者较优解时不会放弃该解），从而跳过一些较差的局部最优，最终搜索到较优解，这是模拟退火的优点** ；

算法优缺点：

（1）优点，个人认为本次实验，自己所得的模拟退火实现 tsp 算法已经相当不错了，能够稳定得到较优解（偏离最优解 3%左右），得益于参数调整（初温较高末温较低，降温系数接近 1，内层循环 1000 次等），和较好的邻域操作（逆转子路径）。

（2）缺点&改进，可以改进提高的部分：该算法所设置的参数针对 ch130 问题能够稳定得到最优解 3%以内的解，但是对于其它 tsp 问题（比如更大规模的问题 a280 之类），效果就有些折扣，所以可能有什么方法能够设计出普适的参数（可能是变量，根据问题规模、特点动态变化的参数），以此增进模拟退火算法的适应性。

4．结论

（1） 使用局部搜索（爬山法）求解 tsp 问题用时很短，收敛快速，模拟退火用时较长波动到趋于稳定；

（2） 虽然局部搜索（爬山法）求解分配方案用时短，但是求得的解效果很差，在上述的两种爬山法中，第一种爬山法得出的方案距离最优解超出 100%之多，完全无法接受，第二种爬山法比第一种好了不少，但是仍然距离最优解在 10%左右，无法稳定达到 10%之内，达不到老师的要求，足以看出**爬山法的特点**：搜索过程中快速接近最优解，但是极易陷入局部极小点而停止更新解，最终解的好坏与初始解的选择有很大关系（因此只有很小概率得到比较好的解）

（3） 改进局部搜索的邻域操作能够有效改进局部搜索/模拟退火的最终解

（4） 而模拟退火算法则在该问题中有比较好的表现，虽然用时较长，但是能够稳定得出距离最优解 5%之内的解，完全符合老师的要求，甚至有时候能够到达 2%之内（比如上面的截图给出的解法） 。**模拟退火算法的特点概括为**：（1）具有一定的跳过局部最优解的能力，能够以较快的速度得到问题的近似最优解（较优解），如果参数（初温、末温、降温系数）设置的很好，模拟退火能够表现出相当好的效果（2）但是正因为它会有概率接受差解，所以最终的解有可能比搜索过程中的某些解更差（3）对参数非常敏感，如果降温系数、初温、末温稍微设置不当，就会表现出较差的效果（初温要较高，否则一开始就很快收敛，末温要设置较低，否则无法充分冷却收敛，降温系数越接近 1 越好，能够充分退火，并且内层循环迭代次数至少要 1000 次才能有比较好的效果）

主要参考文献(三五个即可)

[1] 遗传模拟退火算法——黑龙江 TSP 问题[J]. 姚君. 价值工程. 2016(36)
[2] 改进遗传模拟退火算法求解 TSP[J]. 张雁翔,祁育仙. 智能计算机与应用. 2017(03)
[3] 贪心算法在 TSP 问题中的应用[J]. 来学伟. 许昌学院学报. 2017(02)