基于模拟退火算法求解旅行商问题和约束优化问题

韩启华

摘要

旅行商问题和约束优化问题是算法设计中的经典数学问题。旅行商问题是组合优化中的NP 困难问题，该问题的可行解是所有顶点的全排列，随着顶点数的增加，会产生组合爆炸；约束优化问题是数学最优化问题，由目标函数以及与目标函数中的变量相关的约束条件两部分组成。多年以来，科学家、数学家和计算机学家等对这两类问题的研究和探索从未停止，产生了针对这两类问题的多种解法，在各个领域发挥了很大的价值。随着人工智能技术的不断发展，局部搜索算法中的模拟退火算法得到广泛的应用，求解出了很多以往难以解决的问题。因此，本文利用模拟退火算法，使用 Python 语言编写程序，研究经典旅行商问题和以工料节省问题为例的约束优化问题的解决办法，并对解决结果进行分析。此外，本文也对模拟退火算法在其他领域的应用进行了分析与展望。

关键词 模拟退火・旅行商问题・约束优化问题・Python・模拟退火的应用

问题简介

旅行商问题的简介

旅行商问题，也称旅行推销员问题（英语：Travelling Salesman Problem, TSP），是组合优化中的一个 NP 难问题，在运筹学和理论计算机科学中非常重要。经典的 TSP 可以描述为：一个商品推销员要到若干个城市推销商商品，该推销员从一个城市出发，需要经过所有城市后，回到出发地，应如何选择行进路线，以使得总的行程最短[1]。

作为计算复杂性理论中的一个典型的判定性问题，TSP 的一个版本是给定一个图和长度L，要求回答图中是否存在比L更短的回路。该问题是 NP 完全问题，已知 TSP 算法最坏情况下的时间复杂度随着城市数量的增多而成超多项式（可能是指数）级别增长。

从图论的角度来看，TSP 问题的实质是在一个加权完全无向图中，找到一个权值最小的 Hamilton 回路。由于该问题的可行解是所有顶点的全排列，会产生组合爆炸，它是一个 NP 完全问题。

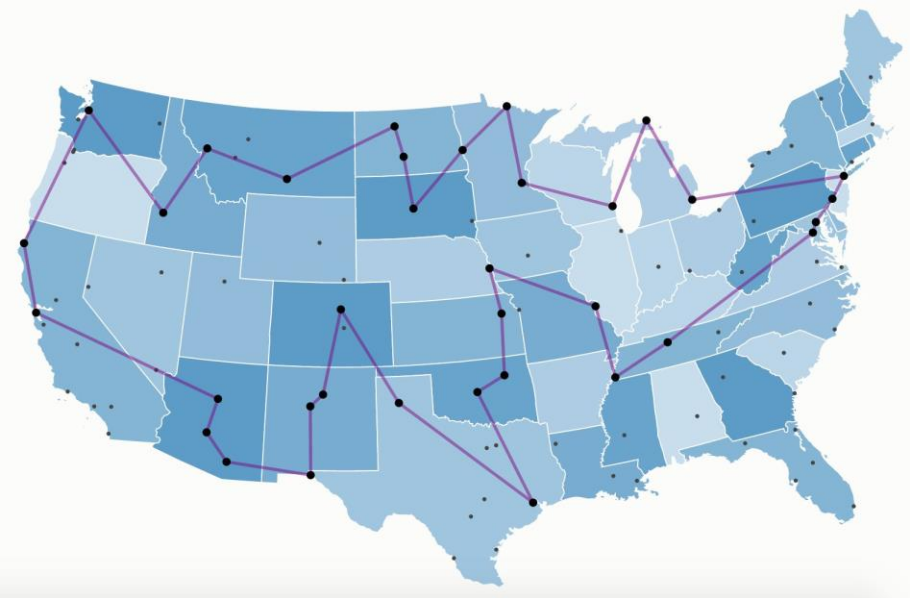

图 1：TSP 问题示例

旅行商问题的研究历史

TSP 的研究历史很久，最早可追溯到 1759 年欧拉研究的骑士环游问题，即对于国际象棋棋盘中的 64 个方格，走访 64 个方格一次且仅一次，并且最终返回到起点。

TSP 问题在 1930 年首次被形式化，并且是在最优化领域中研究最深入的问题之一，许多优化方法都用它作为一个测试基准。

早期的研究者使用精确算法求解该问题，常用的方法包括：分支定界法、线性规划法、动态规划法等。随着科学研究的发展与进步，后来的科学家主要采用近似算法或启发式算法来解决 TSP，主要有遗传算法、蚁群算法、贪心法和神经网络等[2]。

旅行商问题的应用

TSP 具有重要的实际应用价值。它一开始就是为交通运输而提出的，因此在飞机航线安排、邮件寄送、物流配送中具有极高的价值。此外，TSP 在印制电路板转孔、卫星位置调整、车流量调整、互联网结点设置中都发挥了重要作用。

约束优化问题

约束优化问题，也称受约束的最优化问题（英语：Constrained Optimization Problem, COP），是一类数学最优化问题，是经典约束满足问题（英语：Constraint Satisfaction

Problem, CSP）的推广。它由目标函数以及与目标函数中的变量相关的约束条件两部分组成，优化过程则为在约束条件下最优化目标函数。

算法简介

模拟退火算法的历史

模拟退火算法（Simulated annealing, SA）最早由 N.Metropolis[3]等人于 1953 年提出。1983 年，S.Kirkpatrick 等人成功将退火思想引入到组合优化领域。它是基于 Monte-Carlo 迭代求解策略的一种随机寻优算法，其出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。

模拟退火算法的核心思想

SA 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[4]。

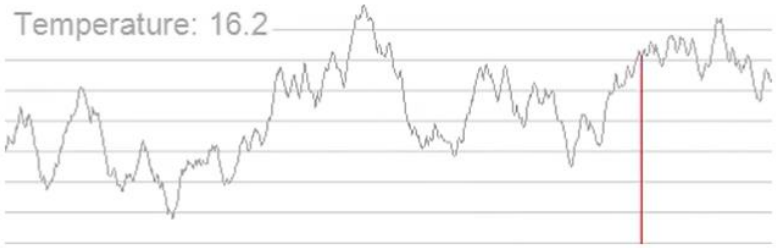

图 2：模拟退火初态示例

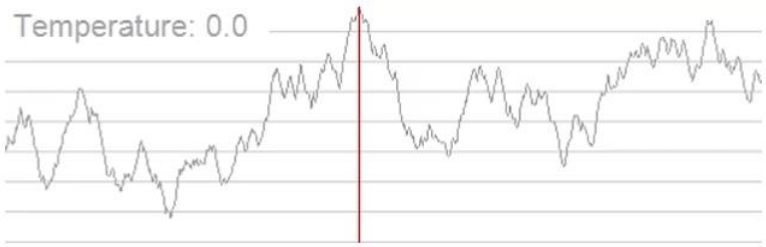

图 3：模拟退火终态示例

这里的“概率”的计算参考了金属冶炼的退火过程，这也是 SA 名称的由来。将温度T当作控制参数，目标函数值f视为内能E，而固体在某温度T时的一个状态对应一个解，然后算法试图随着控制参数T的降低，使目标函数f（内能E）也逐渐降低，直至趋于全局最小值（退火中低温时的最低能量状态），就像金属退火过程一样。

SA 是通过赋予搜索过程一种时变且最终趋于零的概率突跳性，从而可有效避免陷入局部极小并最终趋于全局最优的串行结构的优化算法。

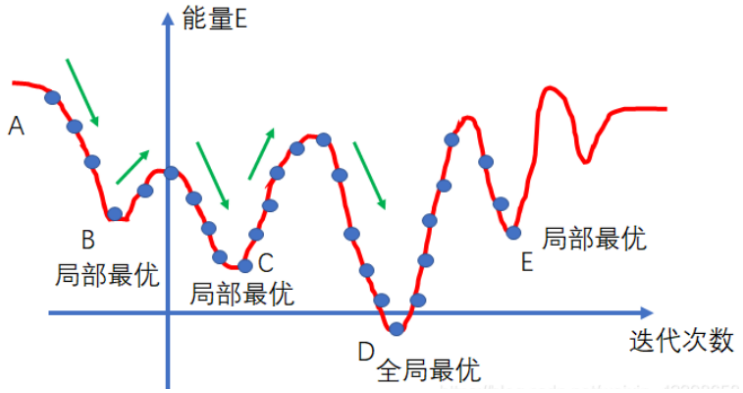

图 4：局部最优与全局最优

模拟退火算法的数学原理

SA 接收新解的原理是：如果新解比当前解更优，则接受新解，否则基于 Metropolis 准则判断是否接受新解，接受概率为：

$$P = \begin{cases} 1, & E_{x_{new}} < E_{x_{old}} \\ e^{\frac{-(E_{x_{new}} - E_{x_{old}})}{kT}}, & E_{x_{new}} \geq E_{x_{old}} \end{cases}$$

其中，$E_{x_{old}}$为当前搜索的解为x_{old}时对应的系统能量，$E_{x_{new}}$为下一时刻搜索解为x_{new}时对应的系统能量。可以看出，随着温度的下降，在新解不比当前解更优时，接受新解的概率逐渐降低。

模拟退火算法的流程

算法实际为两层循环，温度的逐渐降低为外层循环，迭代次数为内层循环。在任意温度情况下，根据迭代次数，对当前状态进行干扰，计算目标函数值的变化。刚开始温度较高时，有较大概率接受非更优的目标函数值，随着温度的下降，接受非更优目标函数值的概率降低，因此，算法最终可能逐渐收敛到全局最优解。

具体流程为[5]：

1. 令$T = T_0$，表示开始退火时的初始温度，随机产生一个初始解x_0，并计算对应的目标函数值$E(x_0)$；
2. 令$T = kT$，其中k取值 0 到 1 之间，为温度下降速率；
3. 对当前解x_t施加随机扰动，对其邻域内产生一个新解x_{t+1}，并计算对应的目标函数值Ex_{t+1}，计算$\Delta E = Ex_{t+1} - Ex_t$；

4. 若$\Delta E < 0$，接收新解作为当前解，否则按照概率$e^{-\Delta E/kT}$判断是否接受新解；
5. 在温度为T下，重复L次步骤 3 和步骤 4；
6. 判断温度是否达到终止温度水平，若是，则终止算法，否则，返回步骤 2。

注意，参数的选择应满足以下标准：

1. 温度初始值T_0的设定：初始值T_0越大，搜索次数越多，搜索到全局最优解的可能性也越大，但是消耗时间增加，导致性能下降；反之，消耗时间减少，导致性能提高，但是搜索到全局最优解的可能性降低；因此，初始温度T的设定需要不断调整测试，取合适值；
2. 内循环次数L的选取：在衰减函数已选定的前提下，L的选取应遵循在控制参数的每一取值上都能恢复准平衡的原则。

具体流程如下图所示：

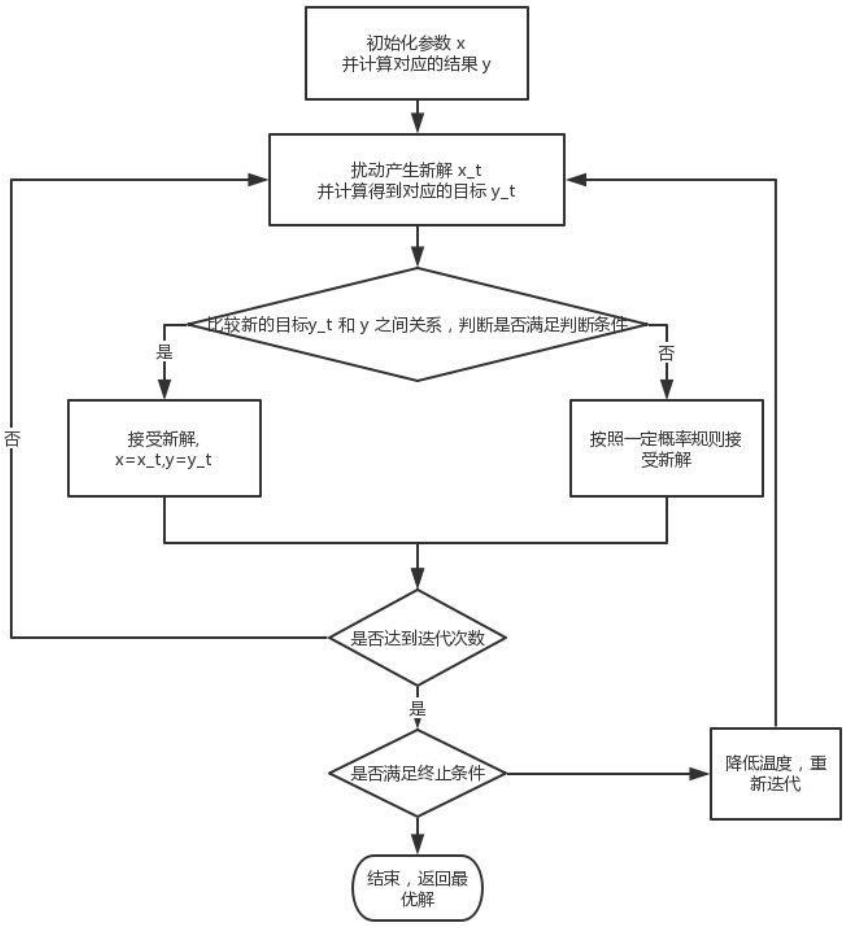

图 5：模拟退火流程图

模拟退火算法的应用

SA 在超大规模集成电路（Very Large Scale Integration, VLSI）设计中具有很大的作用，用 SA 几乎可以很好地完成所有优化 VLSI 设计工作，如全局布线、布板、布局和逻辑最小化等[6]；

SA 可以在图像处理工作中用于进行图像恢复，即把一幅被污染的图像恢复成清晰的原图，滤掉其中被畸变的部分，因此 SA 在图像处理中的应用前景是广阔的；

SA 具有跳出局部最优解的能力，因此在神经网计算机中具有很高的价值，在 Boltzmann 机中，即使系统落入了局部最优的陷阱，经过一段时间后，仍可以跳出陷阱，系统最终往全局最优值的方向收敛。

SA 在电网无功优化中也具有很大作用，传统的线性规划法和非线性规划法不能很好地处理整形变量问题，采用记忆指导搜索，对 SA 进行改造，并采用模式法修正最优解，可以很好地进行电网无功优化[7]。

旅行商问题的求解

求解思路

1. 城市的描述与记录：使用坐标(x, y)表示每个城市的位置，将所有城市存于一位列表中记录下来；
2. 解空间与初始解：解空间为恰好遍历每个城市一次的城市序列集合；初始解为解空间中任意一个解，方便起见，可选择序号为 0，1，2…的顺序序号为初始解；
3. 代价函数：代价函数即按照解中城市序列顺序依次访问所有城市所经过的路程的总和，TSP 的目标是求出一解使得代价函数最小；
4. 新解产生：可随机生成两个不相等的城市编号，以此二数为序号的城市相交互顺序，生成的新的城市序列作为新解；
5. 代价函数差值：以新解与原解的路径代价之差来表示；
6. Metropolis 接受准则：根据$e^{-\Delta E/kT}$的概率接受非更优解。

方便起见，城市坐标选择为一边长为 40 的正方形顶点及各边中点，同时由于初始解为序号为 0，1，2…的顺序序号，为了避免初始解就是最优解的问题，可随机打乱城市坐标，即：(0, 0)，(20, 20)，(0, 20)，(20, 0)，(0, 10)，(10, 0)，(20, 10)，(10, 20)。

不断调整初始温度、温度衰减系数和内循环次数，分别记录下不同组合数据情况下，停止循环所需要的时间，同时观察停止循环时，当前解是否是最优解。

求解过程

1. 引入 Python 下 numpy 库用于记录和操作数据，引入 matplotlib 库用于绘制图形，引入 datetime 库用于计算循环停止所需时间；
2. 使用二维数组记录城市坐标，记录为 citysCoordinates，citysCoordinates[n][0]表示第 n 个城市的横坐标，citysCoordinates[n][1]表示第 n 个城市的纵坐标；
3. 编写 get_distance_array 函数，用于计算每两个城市之间的距离，并将其记录到 distance 二维数组中；
4. 使用 count 记录城市数量，使用 distance_array 记录城市间距，使用 solution_initial 记录初始解，使用 temperature_initial 记录初始温度，使用 alpha 记录温度衰减系数，使用 frequency_per_circulation 记录内循环次数；
5. 为了能够展示并记录图形变化，在每次绘制图形后，使用 plt.pause(0.01)暂停 0.01s。
6. 在循环开始前后，都使用 datetime.now()获取当前时间，以两者相减作为时间差，为了消除机器误差的影响，执行三次取平均值；
7. 使用循环实现算法：

```
# 进入循环，终止条件为当前温度小于等于1
while temperature_current > 1:
    for i in np.arange(frequency_per_circulation):
        # 产生两个不相等的随机数，用于交换路径中这两个城市的位置
        while True:
            loc1 = np.int(np.ceil(np.random.rand() * (count - 1)))
            loc2 = np.int(np.ceil(np.random.rand() * (count - 1)))
            if loc1 != loc2:
                break
        # 交换路径中城市loc1, loc2的位置
        solution_new[loc1], solution_new[loc2] = solution_new[loc2], solution_new[loc1]
        # 计算交换后的路径长度
        value_new = 0
        for j in range(count - 1):
            value_new += distance_array[solution_new[j]][solution_new[j + 1]]
        value_new += distance_array[solution_new[0]][solution_new[count - 1]]
        # 判断是否接受新路径长度
        if value_new < value: # 如果路径长度减小，接受新路径
            value = value_new
            solution_current = solution_new.copy()
        else: # 否则，有一定概率接受新路径
            if np.random.rand() < np.exp(-(value_new - value) / temperature_current):
                value = value_new
                solution_current = solution_new.copy()
            else: # 放弃新路径
                solution_new = solution_current.copy()
    temperature_current = alpha * temperature_current
```

图 6：算法的代码实现

求解结果

在初始温度 temperature_initial 为 10，温度衰减系数 alpha 为 0.9，内循环次数 frequency_per_circulation 为 5 的情况下，循环开始时，初始解和终止解如下图所示：

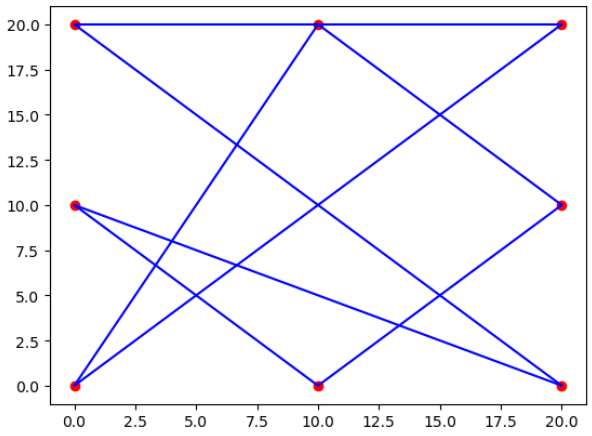

图 7：默认情况下初始解

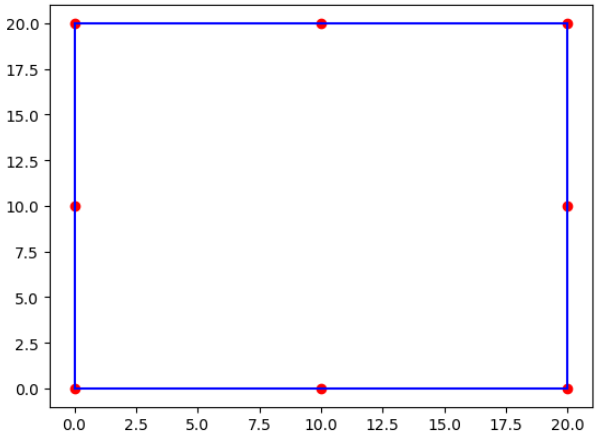

图 8：默认情况下终止解

同时，执行三次获取平均执行时间为 1.641996s。

为了探究初始温度、温度衰减系数、内循环次数对循环终止所需时间以及是否能找到最优解的影响，分别改变这三个数值，重复实验，每组重复执行三次以消除机器误差，得到结果如下表所示：

表 1：不同初始温度、温度衰减系数、内循环次数对结果的影响

初始温度	温度衰减系数	内循环次数	循环终止时间	三次是否找到最优解
10	0.9	5	1.641996s	是，是，是
10	0.7	5	0.348026s	是，是，否
10	0.5	5	0.229011s	否，否，否
10	0.9	4	1.089020s	否，是，是
10	0.9	3	1.038019s	否，否，是
6	0.9	5	0.889025s	否，是，是
6	0.9	5	0.351018s	否，否，否

总结

通过代码执行结果可以明显看出，使用 SA，在初始温度、温度衰减系数、内循环次数设置合理的情况下，可以有效地找出最优解。同时，对不同初始温度、温度衰减系数、内循环次数对结果的影响的探究也表名，初始温度越低、温度衰减系数越低、内循环次数越低，均使得循环终止时间缩短，但是找到最优解的可能性也会降低。

因此，在解决实际问题时，既要考虑算法执行效率，适当降低初始温度、温度衰减系数、内循环次数，也要保证三个数据不能过低，要在多次执行可找到最优解的前提下，适当降低三个数据，以提高算法执行效率。

约束优化问题的求解

问题描述

以工料节省问题作为约束优化问题的示例，工料节省问题描述如下：

集合 A 是工料结合，集合 B 是部件集合，其中的数值代表工料和部件长度，问题是用集合 A 里面的工料生产集合 B 里面的部件，目标是节省工料；

给定两个集合 A 和 B，集合的元素都是大于 0 的实数，假设 A 有 m 个元素，B 有 n 个元素，将 B 的每个元素映射到 A 的某个元素，可以多对一映射，即集合 A 中的一个工料可以生产多个集合 B 中的部件，但剩余部分要丢弃；

在此前提下求出使用工料最少的方法。

求解思路

1. 工料和部件的表示与记录：工料和部件各自都有长度，因此可以用两个一维列表分别记录工料和部件，同时再使用一个二维列表，用于记录每个工料生产哪些部件；
2. 解空间与初始解：解空间为生产所有部件所需要的工料的集合；初始解为遍历部件列表，依次遍历二维列表，如果某工料剩余量满足生产该部件，则放入，否则，继续遍历；
3. 代价函数：每个工料只有用和不用两种状态，因此总效用即所有被使用的工料的长度和；
4. 新解产生：随机选择一个工料，如果它的列表为空，表明没有使用该工料生产任何产品，继续随机选择，直到该工料列表不为空，从中随机抽取一个产品，放入另一个可以容纳的工料的列表中；
5. 代价函数差值：以新解与原解的长度代价之差来表示；
6. Metropolis 接受准则：根据$e^{-\Delta E/kT}$的概率接受非更优解。

求解过程

1. 引入 Python 下 random 库实现生产随机数，引入 math 库实现数学运算，引入 matplotlib 库进行图像绘制；
2. 使用一维列表 A 和 B 分别记录工料和部件的长度，使用二维列表 a2b 记录工料使用情况，a2b[i][j]表示下标为 j 的部件由下标为 i 的工料生产；
3. 根据求解思路初始化 a2b 列表，使用双重循环进行遍历，如下图所示：

```
def Generatea2b():
    global a2b
    i = 0 # B下标
    while (i < bLen):
        for j in range(aLen): # A下标
            if (sum(a2b[j]) + B[i]) <= A[j]: # 如果已经用于生产的j工料加上生产B[i]所需的工料少于或等于j工料的总数
                a2b[j].append(B[i]) # 将生产B[i]所需工料加入已经用于生产的j工料列表
                i += 1 # 继续遍历
                break
```

图 9：初始化 a2b 列表

4. 根据求解思路更新新解，随机选择一个工料列表中的一个产品，将其加入另一个工料，如下图所示：

```
def Generatestatus(a2b):
    i = randint(0, aLen - 1) # 随机选择一个工料i
    while (a2b[i] == []): # 如果工料i的列表为空
        i = randint(0, aLen - 1) # 继续选择i
    temp = a2b[i].pop(randint(0, len(a2b[i]) - 1)) # 随机从工料i的列表中选择一个产品
    j = randint(0, aLen - 1) # 随机选择另一个工料j
    while ((sum(a2b[j]) + temp) > A[j]): # 如果将该产品放入工料j的列表超过工料j的大小
        j = randint(0, aLen - 1) # 继续选择j
    a2b[j].append(temp) # 将该产品加入工料j的列表
```

图 10：产生新解函数

5. 根据求解思路设计代价函数和接收函数，如下图所示：

```
def Effect(a2b):
    sum = 0 # 初始化总和
    for i in range(aLen): # 进行遍历
        if a2b[i] != []: # 如果工料已经被使用
            sum += A[i] # 总和加上整个工料i的大小
    return sum # 返回总和
```

```
def Accept(a2bnew):
    global a2b
    ds = Effect(a2bnew) - Effect(a2b) # 计算新状态与旧状态的效用差
    if ds <= 0: # 如果新状态效用小
        a2b = a2bnew # 接受新状态
    else: # 否则
        if (exp(-ds / temperature) > random()): # 以模拟退火规定的概率
            a2b = a2bnew # 接受新状态
```

图 11：代价函数和接收函数

6. 使用循环实现算法：

```
while (temperature > T_min): # 当温度大于最小值时
    for i in range(bLen):
        a2bnew = deepcopy(a2b) # 拷贝当前状态到新状态
        Generatestatus(a2bnew) # 产生新状态
        Accept(a2bnew) # 是否接受新状态函数
    print("%.1f" % Effect(a2b))
    x.append(n) # x轴添加坐标
    effect.append(Effect(a2b)) # 添加效用值
    n += 1 # 更新x坐标
    temperature = temperature * ALPHA # 更新温度
```

图 12：算法的代码实现

求解结果

设定初始温度为所有部件的长度总和，温度衰减系数为 0.98，结束温度为 1，循环次数为 1；将输入写入 input.txt，运行程序，程序将输出写入 output.txt，如下图所示：

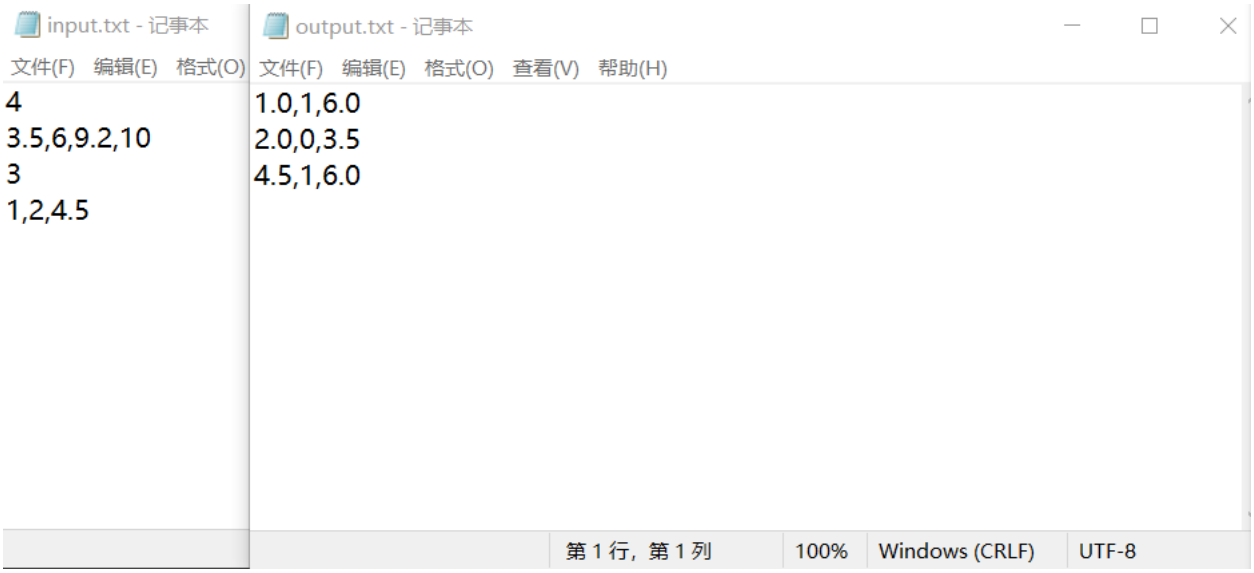

图 13：输入示例及输出结果

input.txt 解读：第 1 行表示有 4 个工料，第 2 行表示工料长度分别为 3.5、6、9.2、10，第 3 行表示有 3 个部件，第 4 行表示部件长度分别为 1、2、4.5。

output.txt 解读：第 1 行表示长度为 1.0 的部件交由第 1 个工料生产，第 1 个工料长度为 6.0，第 2 行表示长度为 2.0 的部件交由第 0 个工料生产，第 0 个工料长度为 3.5，第 3 行表示长度为 4.5 的部件交由第 1 个工料生产，第 1 个工料长度为 6.0。

同时，记录程序执行中代价函数（所需工料总长度）的变化如下图：

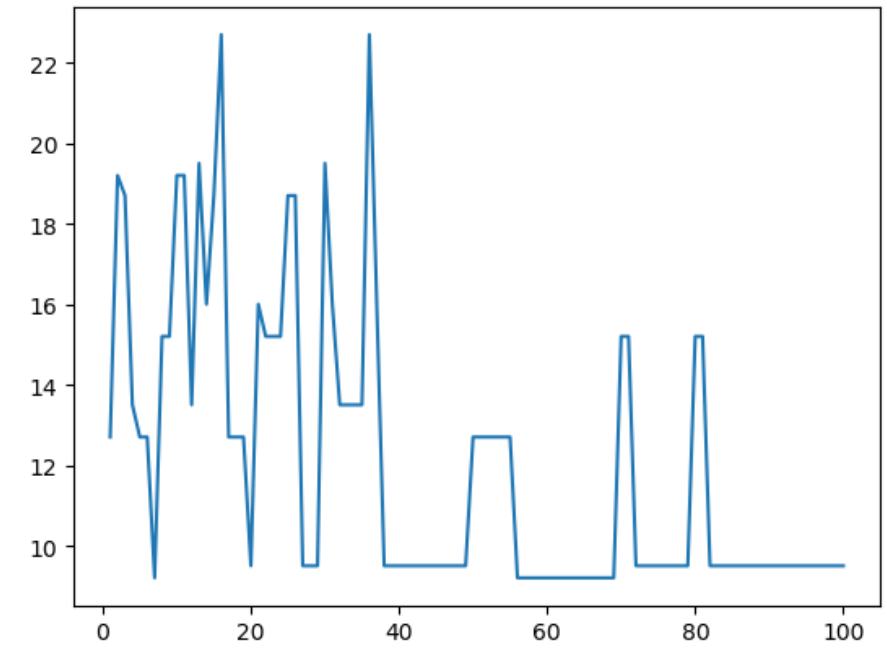

图 14：程序执行中代价函数变化

可以看出，程序刚开始执行时，代价函数波动较大，且有多次的上升，说明温度较高时，接收非更优解的概率更大；随着程序的执行，温度逐渐降低，得到的最优解也趋近于稳定，最终稳定在 9.5。

修改目标温度为 0.5，再次执行程序，得到结果为：

图 15：修改目标温度后输入示例及输出结果

同时，程序执行中代价函数的变化如下图所示：

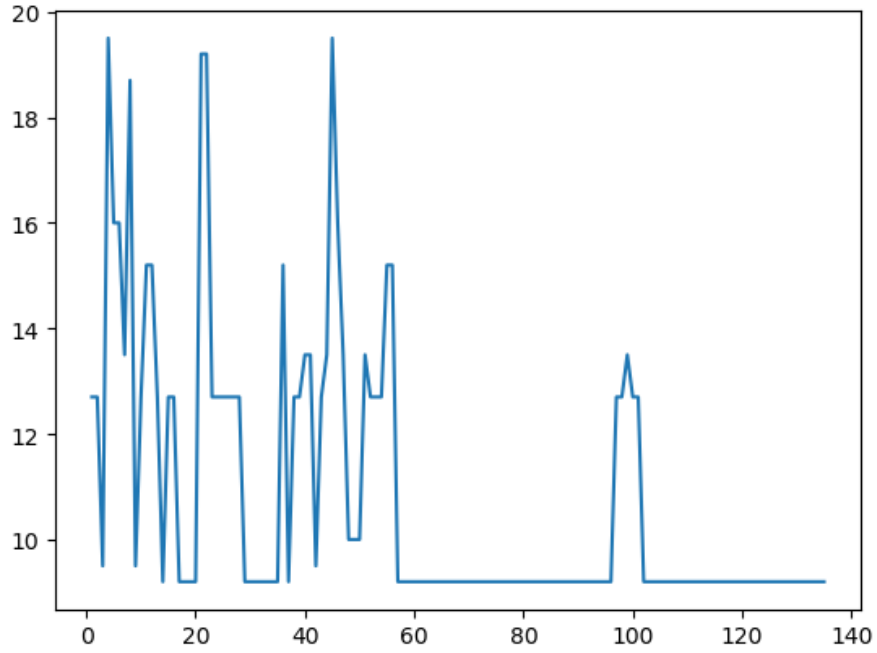

图 16：修改目标温度后程序执行中代价函数变化

总结

从图 14 中可以看出，在最终稳定在 9.5 之前，多次出现了比 9.5 更优的解，通过分析数据可知，如果所有部件都由第 3 个工料生产，仅需 9.2 的长度代价即可，这也说明了，SA 算法可以避免陷入局部最优解，但是并不一定能得到全局最优解，即便最后有相当长的时间保持稳定，该数值也不一定是全局最优解，降低目标温度后，得到了全局最优解。

总结与展望

旅行商问题和约束优化问题都是经典的数学问题，多年以来，人们对它们的热情依旧不减。随着人工智能的发展，许多新的算法逐渐产生，帮助人们对以往难以解决的问题产生新的思路。通过此论文的研究，模拟退火算法在解决旅行商问题和约束优化问题（以工料节省问题为例）中有着较好的表现，可以较为快速地求出最优解；并且发现，初始温度越低、温度衰减系数越低、内循环次数越少，模拟退火算法停止便越快，但是求出最优解的概率就越低，并且即便最后阶段在相当长时间内保持稳定，也不一定是全局最优解，而降低目标温度有助于求出全局最优解。

在实际科学研究和工业生产中，为了能够尽可能高效地找出全局最优解，需要根据实际数值规模大小、数据类型、期望的精度，多次实验设计出合理的模拟退火的初始温度、温度衰减系数、内循环次数、目标温度，以达到时间生产作业的要求。

参考文献

[1] 郭靖扬. 旅行商问题概述[J]. 大众科技, 2006 (8): 229-230.
[2] 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.
[3] Steinbrunn M ,Moerkotte G, Kemper A. Heuristic and Randomized Optimization for the Join Ordering Problem[J] . The VLDB Journal , 1997 , 6 (3) :8 - 17.
[4] Nikolaev A G, Jacobson S H. Simulated annealing[M]//Handbook of metaheuristics. Springer, Boston, MA, 2010: 1-39.
[5] 庞峰. 模拟退火算法的原理及算法在优化问题上的应用. 吉林大学, 2006.
[6] Wong D F, Leong H W, Liu H W. Simulated annealing for VLSI design[M]. Springer Science & Business Media, 2012.
[7] 贾德香, 唐国庆, 韩净. 基于改进模拟退火算法的电网无功优化[J]. 电力系统保护与控制, 2004, 32(4):32-35.