上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

图与网络 2020 春

启发式算法编程报告

周四班 xx 0190xxxxxxxx

2020 年 5 月

一、 问题描述

利用启发式算法中的模拟退火算法解决背包问题。

一共有 15 种物品，每个物品的重量为：70, 73, 77, 80, 82, 87, 90, 94, 98, 106, 110, 113, 115, 118, 120，这些物品对应的价值分别为：135, 139, 149, 150, 156, 163, 173, 184, 192, 201, 210, 214, 221, 229, 240，背包总容量为 750，求一种方案满足所选物品的总价值最高且总重量不超过背包的容量。

二、解题思路

背包问题是一个典型的组合优化问题，所有的解决方案都来自一个有限的集合X，称为 universe，其中满足约束条件的$x \in X$便被称为可行解，本题中的约束条件可以表示为：

$$\sum_{i=1}^{15} x_i w_i \leq 750$$

其中，$x_i \in \{0,1\}$表示是否取第 i 个物体，为 1 表示取，w_i是第 i 个物体的重量。该问题的优化目标可以表示为：

$$\max\left(\sum_{i=1}^{15} x_i v_i\right)$$

即所选物体的总价值，其中，v_i是第 i 个物体的价值。

这种背包问题可以用 3 种方法进行求解，分别为穷举遍历法、模拟退火算法、禁忌搜索法。

1、穷举遍历法

穷举遍历法就是把 15 种物品取或者不取一共 2^{15} 种方案都遍历一遍，找出满足约束条件的方案，然后在这些可行方案中选择使目标函数最大的方案，即可获得全局最优解，程序中 *FindOptim()*函数就是实现的这种穷举遍历法。

2、模拟退火法

模拟退火算法是一种基于概率的随机的启发式算法，其实现流程可以分成四个步骤：

① 第一步是由一个产生函数从当前解产生一个位于解空间的新解，为便于后续的计算和接受，减少算法耗时，通常选择由当前新解经过简单地变换即可产生新解的方法，在这个背包问题中，对于每个新解我都采用微调的方式产生，随机选取某一个物品，如果该物品已在背包中，则将其取出并放入另一个不在背包中的物品，如果该物品不在背包中，则一半概率直接在背包中放入该物品，一半概率取出一个已在背包中的物品再放入该物品，如此产生一个微小变动的新解，产生新解的变换方法决定了当前新解的邻域结构，因而对收敛速度有一定的影响。

② 第二步是计算与新解所对应的目标函数差。在本题中，目标函数为$\sum_{i=1}^{15} x_i v_i$，新解与之前解的差异仅在x_i上，所以只要计算两者差值即可。

③ 第三步是判断新解是否被接受，判断的依据是新解获得的总价值是否大于之前解获得的总价值。若大于，则直接接受，若小于，则概率性地选择接受劣解，即将随机产生的 0-1 的数r与$\exp[(P(y)-P(x))/\mathrm{T}]$比大小，如果$r < \exp[(P(y)-P(x))/\mathrm{T}]$，则接受新解。

④ 第四步是当新解被确定接受时，用新解代替当前解，这只需将当前解中对应于产生新解时的变换部分予以实现，同时修正目标函数值即可。此时，当前解实现了一次迭代。可在此基础上开始下一轮试验并要更新温度T。而当新解被判定为舍弃时，则在原当前解的基础上继续下一轮试验。

模拟退火算法具有渐近收敛性，且与初始值无关，算法求得的解与初始解状态 S 无关，收敛速度与退火率以及初始温度都有关。

3、禁忌搜索法

禁忌算法是一种穷举类的启发式算法，它从一个初始可行解出发，选择一系列的特定搜索方向（移动）作为试探，选择实现让特定的目标函数值变化最多的移动。为了避免陷入局部最优解，禁忌搜索中采用了一种灵活的“记忆”技术，对已经进行的优化过程进行记录和选择，指导下一步的搜索方向，这就是 Tabu 表的建立。在本题中，禁忌表记录的就是每次换取物品时所采取的动作，记住一些最近被检查过的解，并使他们成为选取下一个解的禁忌，由此有效地避免了迂回搜索，使得算法在解空间的探索能力增大，性能提高，最终实现全局优化。

三、编程实现

根据模拟退火算法的实现流程，我使用了 Python 语言对该问题进行了编程实现，首先用穷举遍历的方法获得问题的最优解，然后再用模拟退火算法求得迭代解进行比较。本节仅对部分代码进行解析并展示最后结果，完整代码见附件中模拟退火算法代码 Anneal_algorithm.py。

代码中首先设定了模拟退火算法中的参数，比如初始温度 T=0，退火率 Anneal_rate=0.99，迭代次数为 NumIter=200，某一温度下循环次数为 Balance=100，并将所有物品的重量和价值存放到列表 Weight 和 Value 中，Capacity=750 表示背包总容量，NumItem=15 表示物品的个数。

设定好需要的参数后，再通过编写函数来实现模拟退火算法。函数 *cop(A, B)* 的功能是将 B 列表中的所有元素赋值给 A 列表，因为 Python 语言中的地址问题，不能将列表直接进行赋值，所以编写了该函数方便之后调用，输入时两个列表，

没有返回值。函数 *CalValueWeight(solution)*的作用是计算当前方案的总价值和总重量，输入 solution 时当前方案，返回值 TotalValue 是当前方案的总价值，TotalWeight 是当前方案的总重量。函数 *InitSolution()*的作用是产生一个满足约束条件的随机初始解，返回的是一个初始解 initsolution 和其对应的总价值 profits。函数 *OutItem(solution)*的功能是随机将背包中已经存在的物品取出，而函数 *InItem(solution)*的功能是随机放入背包中不存在的物品，两个函数的输入 solution 都是迭代中的当前方案。函数 *GenerateSolution(initsolution, profits, T)*是模拟退火算法中最重要的函数，该函数的获取在某一种温度下迭代 Balance 次后的方案，输入是迭代前的方案及其总价值，返回的是迭代后的方案及其总价值，函数中体现了模拟退火算法随机性的特点，对当前的方案做微小的改动，如果所获增益增加，则直接替换为新的方案，如果所获增益减少，那么概率地接受劣解，迭代 Balance 次后返回迭代后的方案及其对应的总价值。函数 *FindOptim()*的作用是通过穷举，遍历所有的方案，在满足约束条件的方案中选择收益最大的方案，是一种贪婪算法，该函数的返回值就是最优方案 optimsolution 及其对应的总价值 optimprofit。最后，函数 *Annealing(T)*将之前的函数进行整合，循环迭代 NumIter 次，若中途获得最优解便提前跳出循环，若循环结束后仍未获得最优解，则输出当前获得的次优解，该函数将最后将迭代次数、模拟退火找到的最佳收益以及对应的方案都打印出来。

根据上述函数，可以得到运行结果如下：

```
通过穷举法，可以获得问题的最优解为1458
最优的方案为[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
模拟退火算法迭代62次后，找到最优解:1458,
模拟退火算法得到的方案为： [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
背包使用了749/750
```

图 1. 模拟退火算法的运行结果

由于模拟退火算法是一种随机的搜索方法，多运行几次，得到的最优解时迭代次数还会不一样：

```
通过穷举法，可以获得问题的最优解为1458
最优的方案为[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
模拟退火算法迭代1次后，找到最优解:1458,
模拟退火算法得到的方案为： [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
背包使用了749/750
```

```
通过穷举法，可以获得问题的最优解为1458
最优的方案为[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
模拟退火算法迭代54次后，找到最优解:1458,
模拟退火算法得到的方案为： [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
背包使用了749/750
```

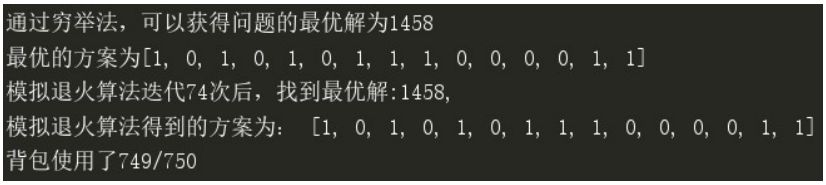
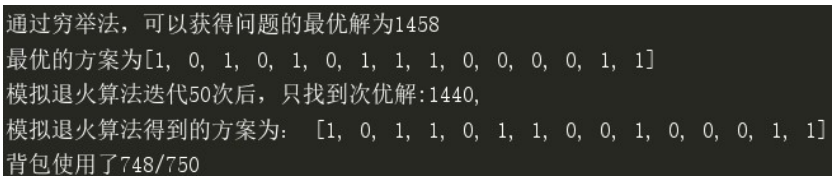


图 2. 模拟退火算法运行多次的结果

减少迭代次数，将 NumIter 设为 50，运行多次可得到次优解的结果：



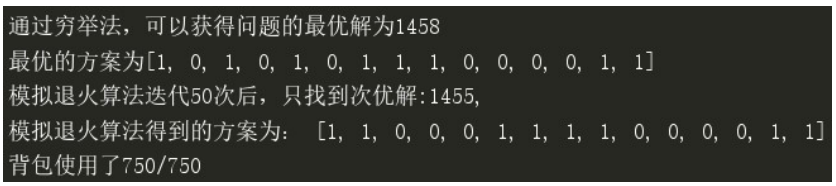


图 3. 迭代次数改为 50 后模拟退火算法得到的结果

若修改初始温度，将 T 改为 1000，从运行结果可以看出迭代 200 次后仍然为找到最优解，只找到次优解：

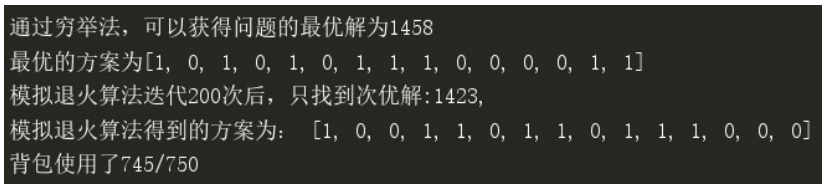


图 4. 初始温度改为 1000 后模拟退火算法得到的结果

从结果可以看出，使用模拟退火算法能否达到最优值和很多因素都有关系，比如总的迭代次数，初始的温度，此外与退火率也有一定的关系。

经过这一次的编程作业，加深了我对启发式算法的理解，并且通过对背包问题这个简单例子的解答也让我更加清晰地了解了模拟退火算法的原理和实现步骤。