

UNIVERSITY OF SCIENCE AND TECHNOLOGY OF
CHINA



OPERATIONS RESEARCH

001251

Course Project Report

Author:

Jiazhi HE: PB19151772

December 21, 2021

Contents

1	问题描述	2
2	算法原理	3
3	数据集 (案例) 说明	4
3.1	背包问题	4
3.2	设备更新问题	6
4	程序输入输出说明	6
4.1	背包问题	6
4.2	设备更新问题	7
5	程序测试结果	8
5.1	背包问题	9
5.2	设备更新问题	9
6	分析总结	9

1 问题描述

动态规划是解决多阶段决策过程最优化问题的一种方法,该方法由美国数学家贝尔曼(R.Bellman)等人在 20 世纪 50 年代初提出.他们针对多阶段决策问题的特点,提出了解决这类问题的最优化原理,并成功地解决了生产管理,工程技术等方面的许多实际问题,从而建立了运筹学的一个新分支.

多阶段决策过程,本意是指这样一类特殊的活动过程,它们可以按时间顺序分解成若干相互联系的阶段,称为“时段”,在每一个时段都要做出决策,全部过程的决策形成一个决策序列,所以多阶段决策问题属序贯决策问题.

多阶段决策过程最优化的目标是要达到整个活动过程的总体效果最优,由于各段决策有机地联系着,本段决策的执行将影响到下一段的决策,以至于影响总体效果,所以决策者在每段决策时不应仅考虑本阶段最优,还应考虑对最终目标的影响,从而做出对全局来讲是最优的决策.动态规划就是符合这种要求的一种决策方法.

在这项作业中,我主要依据杨老师上课说的建立动态规划模型的方法 [1],将问题分解成为可用递推关系式联系起来的若干子问题,从而得到了一个解决动态规划问题的算法.将该算法封装在一个 `Dynamicprogramming` 类中,并对具体问题进行测试.

程序在两个实例上进行了测试,并得到了理想的结果.

- 第一个测试案例是常见的背包问题,具体数据来自于 [2]202 页例 6: 有一辆最大货运量为 10t 的卡车,用以装载 3 种货物,每种货物的单位重量及相应单位价值如表所示. 应如何装载可使总价值最大?

Table 1: 货物重量与价值

货物编号 (i)	1	2	3
单位重量 /t	3	4	5
单位价值 (c_i)	4	5	6

- 第二个测试案例是设备更新模型,即杨老师讲义 [1] 上的 Exercise 3.1: 设现有一台 2 年龄的设备,另规定 5 年龄的设备必须更换. 在规划期购置新设备的成本分别是

$$(p_1, p_2, p_3, p_4, p_5) = (100, 105, 110, 115, 120)$$

试构建如下设备更新的动态规划模型并求其最优更新策略.

Table 2: 五年期设备更新

设备年龄 t	残值 v_t	运行费用 c_t
0	-	30
1	50	40
2	25	50
3	10	75
4	5	90
5	2	-

详细源代码见文件[main.py](#),[Knapsack.py](#)和[Equipmentreplace.py](#).

2 算法原理

动态规划的几个关键要素:

- Stages: 阶段
- Alternatives: 行为选择
- States: 状态
- Recursive Equations: 递归方程

程序的算法主要依赖以上几个要素而展开. 算法类 `Dynamicprogramming` 的初始化如下:

```
class Dynamicprogramming:
def __init__(self, recur_equa, state_choice, stage_final, final_ret):
    """
    recur_equa: a function takes current_stage, current_state, next_stage,
                next-state as input
    """
    self.recur_equa = recur_equa
    self.state_choice = state_choice
```

```
self.stage_final = stage_final
self.final_ret = final_ret
```

其中 `recur_equa` 是一个函数, 输入具体问题的 `currentstage`, `currentstate`, `nextstage` 和 `nextstage`, 返回对应的 reward 或者 cost. 对于 `statechoice` 函数, 输入 `state` 和 `stage`, 输出 `stagenext` 和可供选择的 `state` 组成的列表 `statepos`. 参数 `stagefinal` 表示阶段总数, `finalret` 表示最后一阶段的 reward 或者 cost. (为方便, 其中的下划线均省略)

在类 `Dynamicprogramming` 中定义了一个可供迭代使用的方法如下

```
def __call__(self, init_stage, init_state):
    if init_stage == self.stage_final:
        return ['stage:' + str(init_stage) + ';state:' + str(init_state) + '\n',
                self.final_ret(init_stage,
                                init_state)]

    next_stage, state_list = self.state_choice(init_state, init_stage)
    max_ret = float('-inf')
    state_str = 'stage:' + str(init_stage) + ';state:' + str(init_state) + '\n'

    for next_state in state_list:
        out = self(next_stage, next_state)
        ret = self.recur_equa(init_stage, init_state, next_state) + out[1]
        if ret > max_ret:
            max_ret = ret
            dec_paths = state_str + out[0]
    return [dec_paths, max_ret]
```

主要的想法就是不断的迭代. 用 `maxret` 来记录最大的 reward 或者 cost. 用 `decpaths` 来记录最佳的策略.

3 数据集 (案例) 说明

3.1 背包问题

来源已经在问题描述中提到, 详细可见下图 (摘自书中)

例 6 有一辆最大货运量为 10t 的卡车,用以装载 3 种货物,每种货物的单位重量及相应单位价值如表 7-4 所示。应如何装载可使总价值最大?

表 7-4

货物编号(i)	1	2	3
单位重量/t	3	4	5
单位价值(c_i)	4	5	6

设第 i 种货物装载的件数为 x_i ($i=1,2,3$),则问题可表示为

$$\begin{aligned} \max z &= 4x_1 + 5x_2 + 6x_3 \\ \text{s. t. } &\begin{cases} 3x_1 + 4x_2 + 5x_3 \leq 10 \\ x_i \geq 0 \text{ 且为整数} \end{cases} \quad (i=1,2,3) \end{aligned}$$

可按前述方式建立动态规划模型,由于决策变量取离散值,所以可以用列表法求解。

$$\text{当 } k=1 \text{ 时, } f_1(s_2) = \max_{\substack{0 \leq 3x_1 \leq s_2 \\ x_1 \text{ 为整数}}} \{4x_1\} \text{ 或 } f_1(s_2) = \max_{\substack{0 \leq x_1 \leq s_2/3 \\ x_1 \text{ 为整数}}} \{4x_1\} = 4[s_2/3]$$

计算结果见表 7-5。

表 7-5

s_2	0	1	2	3	4	5	6	7	8	9	10
$f_1(s_2)$	0	0	0	4	4	4	8	8	8	12	12
x_1^*	0	0	0	1	1	1	2	2	2	3	3

$$\text{当 } k=2 \text{ 时, } f_2(s_3) = \max_{\substack{0 \leq x_2 \leq s_3/4 \\ x_2 \text{ 为整数}}} \{5x_2 + f_1(s_3 - 4x_2)\}$$

计算结果见表 7-6。

Figure 1: 背包问题

表 7-6

s_3	0	1	2	3	4	5	6	7	8	9	10
x_2	0	0	0	0	0	1	0	1	0	1	2
$c_2 + f_2$	0	0	0	4	4	5	4	5	8	9	10
$f_2(s_3)$	0	0	0	4	5	5	8	9	10	12	13
x_2^*	0	0	0	0	1	1	0	1	2	0	1

当 $k=3$ 时,

$$\begin{aligned} f_3(10) &= \max_{x_3=0,1,2} \{6x_3 + f_2(10 - 5x_3)\} \\ &= \max\{f_2(10), 6 + f_2(5), 12 + f_2(0)\} \\ &= \max\{13, 6 + 5, 12 + 0\} \\ &= 13 \end{aligned}$$

此时 $x_3^* = 0$,逆推可得全部策略为:

$x_1^* = 2, x_2^* = 1, x_3^* = 0$,最大价值为 13。

Figure 2: 背包问题

3.2 设备更新问题

Exercise 3.1: 设现有一台2年龄的设备，另规定5年龄的设备必须更换。在规划期购置新设备的成本分别是

$$(p_1, p_2, p_3, p_4, p_5) = (100, 105, 110, 115, 120).$$

试构建如下设备更新的动态规划模型并求其最优更新策略。

Table: 五年期设备更新

设备年龄 t	残值 v_t	运行费用 c_t
0	-	30
1	50	40
2	25	50
3	10	75
4	5	90
5	2	-

Figure 3: 设备更新问题

4 程序输入输出说明

4.1 背包问题

结合具体问题背景写出递归方程如下

$$\begin{cases} f_i(x_i) = \max_{m_i=0,1,\dots, \lfloor \frac{W}{w_i} \rfloor, x_i \leq W} \{r_i m_i + f_{i+1}(x_{i+1})\}, i = n, \dots, 1 \\ f_{n+1}(x_{n+1}) = 0 \end{cases} \quad (1)$$

其中 r_i 表示物体单位价值, w_i 表示物体所占体积. 因此, 可以写出如下 python 代码, 调用 `Dynamicprogramming` 类即可.

```
from main import *
weight_group = [3, 4, 5]
value_group = [4, 5, 6]

def knapsack_alternatives(state, stage):
    stage_next = stage + 1
```

```

state_pos = [state]
while state_pos[-1] - weight_group[stage] >= 0:
    state_pos.append(state_pos[-1] - weight_group[stage])
return [stage_next, state_pos]

def knapsack_recurequa(init_stage, init_state, next_state):
    return value_group[init_stage] * int((init_state - next_state) /
                                         weight_group[init_stage])

def final_ret(init_stage, init_state):
    return value_group[init_stage - 1] * int(init_state / weight_group[
                                         init_stage - 1])

bag_fill = Dynamicprogramming(knapsack_recurequa,
                              knapsack_alternatives, 3,
                              final_ret) # stage_final == 3
print(bag_fill(0, 10)[0]) # 10 is the backpack capacity
print(bag_fill(0, 10)[1])

```

代码中具体数据即为图 (1) 中数据. 程序会输出最佳策略的每个 stage 对应的 state, 以及最大的 reward.

4.2 设备更新问题

相应的递归方程为

$$f_i(t) = \max \begin{cases} -c_t + f_{i+1}(t+1) & K \\ -c_0 + v_t - p_i + f_{i+1}(1) & R \end{cases} \quad (2)$$

因此, 可以写出如下 python 代码, 调用 Dynamicprogramming 类即可.

```

from main import *

def equip_replace_state_choice(state, stage):
    stage_next = stage + 1

```



```

state_pos = [1]
if state < 5:
    state_pos.append(state + 1)
return [stage_next, state_pos]

def equip_replace_recur_equa(init_stage, init_state, next_state):
    s = [None, 50, 25, 10, 5, 2]
    c = [30, 40, 50, 75, 90]
    new_cost = [100, 105, 110, 115, 120]
    if next_state == 1:
        return -new_cost[init_stage-1] + s[init_state] - c[0]
    else:
        # print(init_state)
        return -c[init_state]

def final_ret(init_stage, init_state):
    return 0

replace_eq = Dynamicprogramming(equip_replace_recur_equa,
                                equip_replace_state_choice, 6,
                                final_ret)

print(replace_eq(1, 2)[0])
print(replace_eq(1, 2)[1])

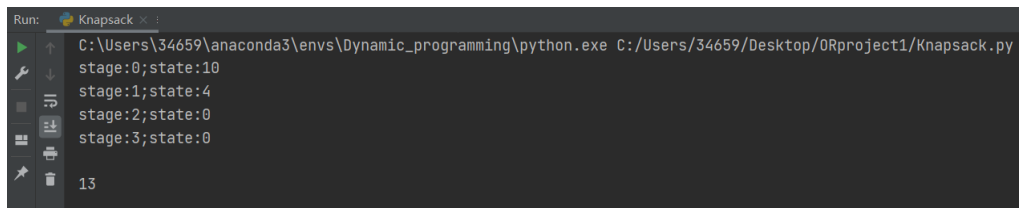
```

代码中具体数据即为图(3)中数据. 程序会输出最佳策略的每个 stage 对应的 state, 以及最小的 cost.

5 程序测试结果

测试结果如图

5.1 背包问题



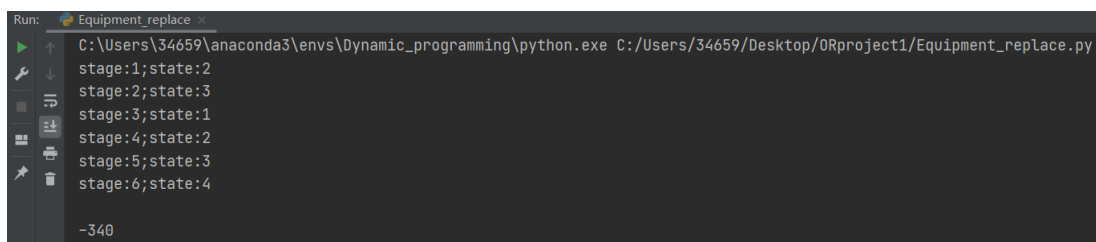
```
Run: Knapsack x
C:\Users\34659\anaconda3\envs\Dynamic_programming\python.exe C:/Users/34659/Desktop/0Rproject1/Knapsack.py
stage:0;state:10
stage:1;state:4
stage:2;state:0
stage:3;state:0
13
```

Figure 4: 背包问题测试结果

与图 (1)和 (2)中的结果吻合, 即最优策略为 $x_1^* = 2, x_2^* = 1, x_3^* = 0$, 最大价值为 13.

5.2 设备更新问题

测试结果如图



```
Run: Equipment_replace x
C:\Users\34659\anaconda3\envs\Dynamic_programming\python.exe C:/Users/34659/Desktop/0Rproject1/Equipment_replace.py
stage:1;state:2
stage:2;state:3
stage:3;state:1
stage:4;state:2
stage:5;state:3
stage:6;state:4
-340
```

Figure 5: 设备更新问题测试结果

与作业中计算的结果吻合, 即最优更新策略为 KRKKK, 收益为 -340 .

如果你想测试其他问题, 结合具体问题写出递归方程, 类似定义对应的函数, 调用 `Dynamicprogramming` 类即可.

6 分析总结

考虑到算法的普适性, 我把算法封装在一个类中, 但对于每一个具体的问题, 都需要给出对应的递归方程, 并写出对应的代码, 可能操作起来有些麻烦, 这在以后时间充足的情况下是可以考虑改进的地方.

程序在两个具体问题上测试,都得到了满意的结果,以后在时间充足的情况下可以考虑进行更多测试,提高程序的鲁棒性.

动态规划算法的手动实现,加深了我对算法深层次的理解,也锻炼了我的 debug 能力.通过此次实验也掌握了很多 python 和 L^AT_EX 的用法,收获很大.

References

- [1] 杨周旺: 运筹学讲义;
- [2] 胡运权, 郭耀煌: 运筹学教程, 第 5 版, 清华大学出版社.