DP for the multiple-choice knapsack problem

Outline

A DP algorithm
    Number of steps

Typical mistakes

In summary

Learning objectives

At the end of this chapter you will be able to:

- write the recurrence $v(h, k)$ over the classes and the capacities, with the value $-\infty$ for the subinstances without a feasible solution;
- run the DP algorithm by hand, trying every item of class h that fits;
- fill the matrix s of the chosen items and reconstruct an optimal solution, one item per class;
- count the $\Theta(nc)$ steps of the DP algorithm.

Prerequisites

- From this module, the chapter «DP for the 0/1 knapsack problem»: the recurrence $v(j, k)$ of the 0/1 knapsack problem (KP) over items and capacities, the matrix s of the decisions and the reconstruction of a solution.
- From the module *Combinatorial Optimization Problems*, the chapter «Knapsack problems»: the multiple-choice knapsack problem, its classes $\mathscr{J}_h$ and its feasibility condition.
- From the module *Algorithms*: pseudocode conventions and the size of the input.

The problem of this chapter

The problem of this chapter is the
multiple-choice knapsack problem (MCKP):

its definition and its assumptions are on its page.

- We write a DP algorithm that computes the optimal value of an MCKP instance
- The same algorithm also gives an optimal solution, with exactly one item in each class

A DP algorithm

Introduction: A DP algorithm

- In this section we write a DP algorithm for the MCKP, in two steps: first the optimal value, then an optimal solution
- The stages of the algorithm are the classes, and exactly one item is chosen in each class
- The value $-\infty$ marks the subinstances without a feasible solution
- We run the algorithm by hand on an instance with seven items in three classes, and we count its steps

Example: MCKP instance

We consider a knapsack of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, \ldots, 7\}$ of 7 items ($n = 7$), partitioned into $q = 3$ classes:

$$\mathscr{I}_1 = \{1, 2\}, \qquad \mathscr{I}_2 = \{3, 4, 5\}, \qquad \mathscr{I}_3 = \{6, 7\}.$$

The profits and weights are:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
p_j	4	8	3	5	7	3	7
w_j	1	4	1	2	5	2	3

We use this MCKP instance as the example of the whole chapter.

Feasible solutions and stages

- A feasible solution chooses exactly one item in each class, so it is described by:
  - the selected subset $S \subseteq \mathscr{I}$, with $|S \cap \mathscr{J}_h| = 1$ for every class h, or
  - the binary vector $(x_1, x_2, \ldots, x_n)$ of S.
- The classes are the stages of the DP algorithm:
  - class h is decided after the classes $1, 2, \ldots, h-1$,
  - and its decision is the item chosen in it.

The two steps of the DP algorithm

- Step 1: computation of the optimal value
  - a recursive formula gives the optimal value of each subinstance, over the classes and the capacities
- Step 2: computation of an optimal solution
  - a matrix s stores the item chosen in each class, and an optimal solution is read from it

Step 1: the subinstances

For any $h \in \{0, 1, \ldots, q\}$ and $k \in \{0, 1, \ldots, c\}$, let $v(h, k)$ be the optimal value of the MCKP subinstance given by:

1. a knapsack of capacity k,
2. the classes $\mathscr{J}_1, \mathscr{J}_2, \ldots, \mathscr{J}_h$,
3. for each item $j \in \mathscr{J}_1 \cup \mathscr{J}_2 \cup \cdots \cup \mathscr{J}_h$, a profit p_j and a weight w_j,

with $v(h, k) = -\infty$ if this subinstance has no feasible solution.

Step 1: the recursive formula

The recursive formula to compute $v(h, k)$ reads as follows:

$$v(h,k) = \begin{cases} 0 & \text{if } h = 0, \\ -\infty & \text{if } h \geq 1 \text{ and } \forall j \in \mathscr{J}_h\colon\ w_j > k, \\ \max\limits_{j \in \mathscr{J}_h\colon w_j \leq k} \left\{ p_j + v(h-1, k-w_j) \right\} & \text{if } h \geq 1 \text{ and } \exists j \in \mathscr{J}_h\colon\ w_j \leq k. \end{cases}$$

Reading the recursive formula

- The value $-\infty$ follows the rules

$$p_j + (-\infty) = -\infty \qquad \text{and} \qquad \max\{-\infty, a\} = a :$$

a subinstance without feasible solutions never gives a better value.

- The base cases are

$$v(0, k) = 0 \quad \text{for every } k :$$

with no class, the empty subset is feasible and has profit 0, whatever the capacity.

Optimal substructure

- The recursive formula generates at most $|\mathscr{J}_h|$ **independent subproblems**, one for each item j of class h that fits in capacity k:
  - the residual subproblem has the classes $1, 2, \ldots, h-1$ and the capacity $k - w_j$.
- Hence, an optimal solution for (h, k) is built by combining the best item j of class h with an optimal solution of its residual subproblem.

$\rightsquigarrow$ **Optimal Substructure verified!**

Overlapping subproblems

- Different choices in the classes $h+1, h+2, \ldots, q$ can leave the same residual capacity.
- Therefore, the same value $v(h,k)$ may be needed several times during the recursive evaluation of the formula.

$\rightsquigarrow$ **Overlapping Subproblems verified!**

The bottom-up approach

- The value $v(h, k)$ depends only on values $v(h-1, k')$ with $k' < k$.
- Therefore, the **bottom-up approach** computes the rows $h = 0, 1, \ldots, q$ in this order, each one for increasing k, and stores every value in a **lookup table**.
- The optimal value of the MCKP instance is $v(q, c)$, and the instance is infeasible if and only if

$$v(q, c) = -\infty.$$

Step 2: computation of an optimal solution

- For each class h and capacity k, we store $s[h][k]$:
  - the item of class h chosen in an optimal solution of the subinstance (h, k),
  - that is, the item that maximizes the right-hand side of the recursive formula.
- If $v[h][k] = -\infty$, no item is chosen and $s[h][k] = 0$.

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**,
which stores the matrix s together with the matrix v.

Algorithm: MCKP_DP (1/2)

Input : An MCKP instance given by q, the classes $\mathscr{J}_1, \ldots, \mathscr{J}_q$, arrays $p[1..n]$, $w[1..n]$, and capacity c

Output : The optimal value of the MCKP instance and the matrix $s[0..q][0..c]$

```
1 function MCKP_DP(q, J, p, w, c)
2 |   v := new array[0..q][0..c]; s := new array[0..q][0..c]
3 |   for k := 0 to c do
4 |   |   v[0][k] ← 0; s[0][k] ← 0
5 |   end
```

Algorithm: MCKP_DP (2/2)

function *MCKP_DP*$(q, \mathscr{J}, p, w, c)$ (continued)

```
6     for h := 1 to q do
7         for k := 0 to c do
8             m := −∞; t := 0
9             foreach j ∈ J_h do
10                if w[j] ≤ k and p[j] + v[h − 1][k − w[j]] > m
                  then
11                    m ← p[j] + v[h − 1][k − w[j]]; t ← j
12                end
13            end
14            v[h][k] ← m; s[h][k] ← t
15        end
16    end
17    return v[q][c], s
18 end
```

How MCKP_DP works

- The variable m starts from $-\infty$, not from 0: exactly one item of class h must be chosen, so the value of an empty choice is not available.
- The innermost loop tries every item j of class h that fits, keeps in m the largest value found so far, and in t the item that gives it.
- If no item of class h gives a value larger than $-\infty$, the cell keeps $v[h][k] = -\infty$ and $s[h][k] = 0$.
- In case of ties the item met first in $\mathscr{J}_h$ is kept, because m is updated only when the value is strictly larger.

Example: execution of the DP algorithm (1/10)

We consider the MCKP instance of the example above, with capacity $c = 10$ and classes $\mathscr{J}_1 = \{1, 2\}$, $\mathscr{J}_2 = \{3, 4, 5\}$, $\mathscr{J}_3 = \{6, 7\}$:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
p_j	4	8	3	5	7	3	7
w_j	1	4	1	2	5	2	3

- For each step of the DP algorithm with $h \geq 1$, the next tables show the values of h and k, the cells $v[h][k]$ and $s[h][k]$, and the operation performed.
- The row $h = 0$ is $v[0][k] = 0$ and $s[0][k] = 0$ for every k.

Example: execution of the DP algorithm (2/10)

h	k	$v[h][k]$	$s[h][k]$	Action
1	0	$-\infty$	0	no item of $\mathscr{J}_1$ fits: $-\infty$
1	1	4	1	$\max\{p_1+v[0][0]\}$ $= \max\{4\} = 4$
1	2	4	1	$\max\{p_1+v[0][1]\}$ $= \max\{4\} = 4$
1	3	4	1	$\max\{p_1+v[0][2]\}$ $= \max\{4\} = 4$
1	4	8	2	$\max\{p_1+v[0][3],\ p_2+v[0][0]\}$ $= \max\{4, 8\} = 8$
1	5	8	2	$\max\{p_1+v[0][4],\ p_2+v[0][1]\}$ $= \max\{4, 8\} = 8$

Example: execution of the DP algorithm (3/10)

h	k	$v[h][k]$	$s[h][k]$	Action
1	6	8	2	$\max\{p_1+v[0][5],\ p_2+v[0][2]\}$ $= \max\{4, 8\} = 8$
1	7	8	2	$\max\{p_1+v[0][6],\ p_2+v[0][3]\}$ $= \max\{4, 8\} = 8$
1	8	8	2	$\max\{p_1+v[0][7],\ p_2+v[0][4]\}$ $= \max\{4, 8\} = 8$
1	9	8	2	$\max\{p_1+v[0][8],\ p_2+v[0][5]\}$ $= \max\{4, 8\} = 8$
1	10	8	2	$\max\{p_1+v[0][9],\ p_2+v[0][6]\}$ $= \max\{4, 8\} = 8$

Example: execution of the DP algorithm (4/10)

h	k	$v[h][k]$	$s[h][k]$	Action
2	0	$-\infty$	0	no item of $\mathscr{J}_2$ fits: $-\infty$
2	1	$-\infty$	0	$\max\{p_3+v[1][0]\}$ $= \max\{-\infty\} = -\infty$
2	2	7	3	$\max\{p_3+v[1][1],\ p_4+v[1][0]\}$ $= \max\{7, -\infty\} = 7$
2	3	9	4	$\max\{p_3+v[1][2],\ p_4+v[1][1]\}$ $= \max\{7, 9\} = 9$
2	4	9	4	$\max\{p_3+v[1][3],\ p_4+v[1][2]\}$ $= \max\{7, 9\} = 9$
2	5	11	3	$\max\{p_3+v[1][4],\ p_4+v[1][3],\ p_5+v[1][0]\}$ $= \max\{11, 9, -\infty\} = 11$

Example: execution of the DP algorithm (5/10)

h	k	$v[h][k]$	$s[h][k]$	Action
2	6	13	4	$\max\{p_3+v[1][5],\ p_4+v[1][4],\ p_5+v[1][1]\}$ $= \max\{11, 13, 11\} = 13$
2	7	13	4	$\max\{p_3+v[1][6],\ p_4+v[1][5],\ p_5+v[1][2]\}$ $= \max\{11, 13, 11\} = 13$
2	8	13	4	$\max\{p_3+v[1][7],\ p_4+v[1][6],\ p_5+v[1][3]\}$ $= \max\{11, 13, 11\} = 13$
2	9	15	5	$\max\{p_3+v[1][8],\ p_4+v[1][7],\ p_5+v[1][4]\}$ $= \max\{11, 13, 15\} = 15$
2	10	15	5	$\max\{p_3+v[1][9],\ p_4+v[1][8],\ p_5+v[1][5]\}$ $= \max\{11, 13, 15\} = 15$

Example: execution of the DP algorithm (6/10)

h	k	$v[h][k]$	$s[h][k]$	Action
3	0	$-\infty$	0	no item of $\mathscr{J}_3$ fits: $-\infty$
3	1	$-\infty$	0	no item of $\mathscr{J}_3$ fits: $-\infty$
3	2	$-\infty$	0	$\max\{p_6+v[2][0]\}$ $= \max\{-\infty\} = -\infty$
3	3	$-\infty$	0	$\max\{p_6+v[2][1],\ p_7+v[2][0]\}$ $= \max\{-\infty, -\infty\} = -\infty$
3	4	10	6	$\max\{p_6+v[2][2],\ p_7+v[2][1]\}$ $= \max\{10, -\infty\} = 10$
3	5	14	7	$\max\{p_6+v[2][3],\ p_7+v[2][2]\}$ $= \max\{12, 14\} = 14$

Example: execution of the DP algorithm (7/10)

h	k	$v[h][k]$	$s[h][k]$	Action
3	6	16	7	$\max\{p_6+v[2][4],\ p_7+v[2][3]\}$ $= \max\{12, 16\} = 16$
3	7	16	7	$\max\{p_6+v[2][5],\ p_7+v[2][4]\}$ $= \max\{14, 16\} = 16$
3	8	18	7	$\max\{p_6+v[2][6],\ p_7+v[2][5]\}$ $= \max\{16, 18\} = 18$
3	9	20	7	$\max\{p_6+v[2][7],\ p_7+v[2][6]\}$ $= \max\{16, 20\} = 20$
3	10	20	7	$\max\{p_6+v[2][8],\ p_7+v[2][7]\}$ $= \max\{16, 20\} = 20$

Example: execution of the DP algorithm (8/10)

This is the matrix v after the execution of the DP algorithm:

$v[h][k]$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$h=0$	0	0	0	0	0	0
$h=1$	$-\infty$	4	4	4	8	8
$h=2$	$-\infty$	$-\infty$	7	9	9	11
$h=3$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	10	14

$v[h][k]$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$h=0$	0	0	0	0	0
$h=1$	8	8	8	8	8
$h=2$	13	13	13	15	15
$h=3$	16	16	18	20	20

Example: execution of the DP algorithm (9/10)

This is the matrix s after the execution of the DP algorithm:

$s[h][k]$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$h=0$	0	0	0	0	0	0
$h=1$	0	1	1	1	2	2
$h=2$	0	0	3	4	4	3
$h=3$	0	0	0	0	6	7

$s[h][k]$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$h=0$	0	0	0	0	0
$h=1$	2	2	2	2	2
$h=2$	4	4	4	5	5
$h=3$	7	7	7	7	7

Example: execution of the DP algorithm (10/10)

- For example, $v[2][1] = -\infty$: every choice of one item in $\mathscr{J}_1$ and one in $\mathscr{J}_2$ weighs at least

$$w_1 + w_3 = 2.$$

- The optimal value of the instance is

$$v[3][10] = 20,$$

attained with item $s[3][10] = 7$ of class 3.

Reconstruction of an optimal solution

- Once the matrix s has been computed and $v[q][c] > -\infty$, an optimal solution is reconstructed:
  - scanning the classes from $h = q$ down to $h = 1$,
  - starting from $k = c$ and $x_j = 0$ for every item j:

$$j \leftarrow s[h][k], \qquad x_j \leftarrow 1, \qquad k \leftarrow k - w_j.$$

- Every class gives exactly one item, so the reconstruction selects q items.

Algorithm: MCKP_Reconstruct

Input : The number q of classes, the capacity c, the array $w[1..n]$, and the matrix $s[0..q][0..c]$, with $v[q][c] > -\infty$

Output: An optimal solution of the MCKP instance, as a binary vector

```
1 function MCKP_Reconstruct(q, c, w, s)
2 |   x := new array[1..n], with x[j] = 0 for every j
3 |   k := c
4 |   for h := q downto 1 do
5 |   |   j ← s[h][k]; x[j] ← 1
6 |   |   k ← k − w[j]
7 |   end
8 |   return x
9 end
```

Example: computing an optimal solution (1/2)

After the execution of the DP algorithm on the instance of the example above, we start from $h = 3$ and $k = 10$:

Iteration	h	k	$j = s[h][k]$	Selection	Capacity
–	–	–	–	–	$k = 10$
1	3	10	7	$x[7] \leftarrow 1$	$k \leftarrow 10 - 3 = 7$
2	2	7	4	$x[4] \leftarrow 1$	$k \leftarrow 7 - 2 = 5$
3	1	5	2	$x[2] \leftarrow 1$	$k \leftarrow 5 - 4 = 1$

Example: computing an optimal solution (2/2)

- An optimal solution of the MCKP instance of the example is

$$S = \{2, 4, 7\},$$

that is, the vector

$$x = (0, 1, 0, 1, 0, 0, 1).$$

- Its total weight is $4 + 2 + 3 = 9 \leq c$.
- Its optimal value is $v(3, 10) = 20$.

How many steps does the DP algorithm execute?

We count how many times each line
of the pseudo-code of MCKP_DP is executed.

Number of steps (1/3)

- Let $S(n, q, c)$ denote the number of steps executed by `MCKP_DP` in the worst case.
- The innermost loop runs over the items of class h, so over all the classes it runs

$$|\mathscr{J}_1| + |\mathscr{J}_2| + \cdots + |\mathscr{J}_q| = n$$

times for each capacity k.

Number of steps (2/3)

Assuming that the condition at line 10 is always true:

- Lines 2 and 17 are executed 1 time each.
- Lines 3 and 4 are executed $c + 1$ times each.
- Line 6 is executed q times.
- Lines 7, 8 and 14 are executed $q(c + 1)$ times each.
- Lines 9, 10 and 11 are executed $n(c + 1)$ times each.

Number of steps (3/3)

Therefore, the total number of steps in the worst case is

$$S(n, q, c) = 3n(c + 1) + 3q(c + 1) + q + 2c + 4.$$

Since every class contains at least one item, $1 \leq q \leq n$, and

$$3n(c + 1) \leq S(n, q, c) \leq 6n(c + 1) + n + 2c + 4.$$

Both bounds are $\Theta(nc)$, hence $S(n, q, c) = \Theta(nc)$.

The number of steps of the DP algorithm

The DP algorithm for the MCKP executes

$$S(n, q, c) = \Theta\big(c\,(|\mathscr{J}_1| + |\mathscr{J}_2| + \cdots + |\mathscr{J}_q|)\big) = \Theta(nc)$$

steps in the worst case.

- The reconstruction executes $\Theta(q)$ steps, one for each class.

Summary (1/2)

Step 1: the optimal value:

- the stages are the classes:
  $v(h, k) = \max_{j \in \mathscr{J}_h : w_j \leq k} \{p_j + v(h-1, k-w_j)\}$
- base cases $v(0, k) = 0$; $v(h, k) = -\infty$ if no item of class h fits
- optimal substructure and overlapping subproblems:
  bottom-up, row by row, in a lookup table
- the optimal value is $v(q, c)$; the instance is infeasible if and only if $v(q, c) = -\infty$

Summary (2/2)

Step 2: an optimal solution:

- $s[h][k]$ is the item of class h that maximizes the recursive formula, 0 if $v[h][k] = -\infty$
- the solution is read backwards from (q, c): $j \leftarrow s[h][k]$, $x_j \leftarrow 1$, $k \leftarrow k - w_j$
- on the example: $S = \{2, 4, 7\}$, with total weight 9 and optimal value 20

Number of steps:

- $S(n, q, c) = \Theta(nc)$ for MCKP_DP, $\Theta(q)$ for the reconstruction

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the DP algorithm for the MCKP
- Two of them concern the value $-\infty$ and the rule «exactly one item per class», two the base cases and the reconstruction
- Check them before running the DP algorithm by hand

Typical mistakes (1/2)

Observation 1: Starting m from 0 instead of $-\infty$

With $m = 0$ a class with no item that fits would be skipped, and the solution would not contain exactly one item per class.

Observation 2: Adding the option «no item of class h»

The term $v(h-1, k)$ of the KP recurrence has no place here: every class gives exactly one item.

Typical mistakes (2/2)

Observation 3: Setting $v(0, k) = -\infty$ for $k \geq 1$

With no class the empty subset is feasible for every capacity: the capacity does not have to be filled.

Observation 4: Reconstructing an infeasible instance

If $v(q, c) = -\infty$ the instance has no feasible solution, and $s[q][c] = 0$ gives no item to select.

Summary

Exactly one item per class:

- m starts from $-\infty$, not from 0
- no option «no item of class h»: the term $v(h-1,k)$ of the KP has no place here

Base cases and reconstruction:

- $v(0,k) = 0$ for every k: the capacity does not have to be filled
- reconstruct a solution only when $v(q,c) > -\infty$

In summary

Introduction: In summary

- We collect the key ideas of the chapter on the DP algorithm for the MCKP
- The dashboard gathers the result on the example of the chapter

Summary (1/2)

- The stages of the DP algorithm are the classes:

$$v(h,k) = \max_{j \in \mathscr{J}_h:\, w_j \le k} \{p_j + v(h-1, k-w_j)\}, \quad \text{with } v(0,k) = 0.$$

- The value $-\infty$ marks the subinstances without a feasible solution, and the instance is infeasible if and only if $v(q,c) = -\infty$.
- The matrix s stores the item chosen in each class, and an optimal solution is read backwards from (q,c), one item per class.
- The DP algorithm executes $\Theta(c \sum_{h=1}^{q} |\mathscr{J}_h|) = \Theta(nc)$ steps.

Summary (2/2)

Dashboard of the MCKP example ($c = 10$, three classes):

Method	Lower bound	Upper bound	Origin
DP algorithm	20	20	optimal value, solution $S = \{2, 4, 7\}$