DP for the knapsack problem with setups

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 of the KPS: for each class, the best value without opening it and the best value after paying its setup;
- run the DP algorithm by hand, class by class, with a KP recurrence inside each opened class;
- reconstruct an optimal solution from the choice tables s and o;
- 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 knapsack problem with setups, its classes, setup costs f_h and setup weights s_h.
- From the module *Algorithms*: pseudocode conventions and the size of the input.

The problem of this chapter

The problem of this chapter is the knapsack problem with setups (KPS): its definition and its assumptions are on its page.

A DP algorithm

Introduction: a DP algorithm

- We build a DP algorithm for the knapsack problem with setups (KPS)
- The classes are the stages: each class either stays closed, or is opened by paying its setup once
- Inside an open class, the choice of the items is a 0/1 knapsack problem (KP): we reuse the DP algorithm of the KP
- We compute the optimal value, then an optimal solution from two choice tables, and finally we count the number of steps

Items, classes and solutions

- We number the items class by class, so that

$$\mathscr{J}_h = \{\ell_{h-1} + 1, \ldots, \ell_h\}$$

  for every class h, with $\ell_0 = 0$ and $\ell_q = n$.

- Class h has $t_h = |\mathscr{J}_h| = \ell_h - \ell_{h-1}$ items, and its i-th item is $j = \ell_{h-1} + i$.

- A solution is described by the binary vector $(x_1, x_2, \ldots, x_n)$.

- Class h is **open** if at least one of its items is selected: then its setup weight s_h is added to the total weight, and its setup cost f_h is subtracted from the total revenue.

Example: KPS instance

We consider a knapsack of capacity $c = 10$ and a set $\mathscr{I} = \{1, 2, \ldots, 5\}$ of 5 items ($n = 5$), partitioned into $q = 2$ classes:

- $\mathscr{J}_1 = \{1, 2\}$, with setup cost $f_1 = 2$ and setup weight $s_1 = 2$;
- $\mathscr{J}_2 = \{3, 4, 5\}$, with setup cost $f_2 = 3$ and setup weight $s_2 = 1$.

The revenues and weights are:

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

We use this KPS instance throughout the chapter.

Step 1: computation of the optimal value

- The classes are the stages of the DP algorithm.
- For each class h there are two options:
  - class h stays closed, or
  - class h is opened, its setup is paid, and some of its items are selected.
- Inside an open class, the choice of the items is a KP: the DP algorithm of the KP runs over the items of the class, starting from the value left after paying the setup.

The values $v(h, k)$ and $u(h, i, k)$

- 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 KPS subinstance with capacity k and the classes $\mathscr{J}_1, \ldots, \mathscr{J}_h$.
- For any $h \in \{1, \ldots, q\}$ and $i \in \{0, 1, \ldots, t_h\}$, let $u(h, i, k)$ be the optimal value of the same subinstance when class h is open (its setup is paid) and only its first i items are available, with $u(h, i, k) = -\infty$ if the setup weight does not fit.

Next: the recursive formulas to compute $u(h, i, k)$ and $v(h, k)$.

The recurrence for $u(h, i, k)$

The recursive formula to compute $u(h, i, k)$ reads as follows, where $j = \ell_{h-1} + i$ is the i-th item of class h:

$$u(h,i,k) = \begin{cases} -\infty & \text{if } i = 0 \text{ and } k < s_h, \\ v(h-1, k-s_h) - f_h & \text{if } i = 0 \text{ and } k \geq s_h, \\ u(h, i-1, k) & \text{if } i \geq 1 \text{ and } w_j > k, \\ \max \left\{ u(h, i-1, k), \right. & \text{if } i \geq 1 \text{ and } w_j \leq k. \\ \qquad \left. p_j + u(h, i-1, k-w_j) \right\} & \end{cases} \tag{1}$$

The recurrence for $v(h, k)$

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

$$v(h,k) = \begin{cases} 0 & \text{if } h = 0, \\ \max\left\{v(h-1,k),\ u(h,t_h,k)\right\} & \text{if } h \geq 1. \end{cases} \tag{2}$$

How the recurrences work

- The row $u(h, 0, \cdot)$ **pays the setup** of class h: the setup weight s_h leaves the capacity $k - s_h$ to the classes $1, 2, \ldots, h-1$, and the setup cost f_h is subtracted.
- The rows $u(h, 1, \cdot), \ldots, u(h, t_h, \cdot)$ are the recurrence of the KP over the items of class h, with $-\infty + p_j = -\infty$.
- The formula (2) compares the two options for class h: closed, with value $v(h-1, k)$, or open, with value $u(h, t_h, k)$.

An open class with no selected item

- The row $u(h, 0, \cdot)$ allows an open class with no selected item, which is not a solution of the KPS.
- This value is never chosen in (2), since $f_h \geq 1$ gives

$$v(h-1, k-s_h) - f_h \; < \; v(h-1, k).$$

- Hence, when the maximum in (2) is attained only by $u(h, t_h, k)$, at least one item of class h is selected.

Optimal substructure and overlapping subproblems

- An optimal solution for (h, k) is built from an optimal solution of a smaller subinstance: $(h-1, k)$ if class h is closed, and, inside an open class, $(h, i-1, \cdot)$ with or without item j.

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

- Different decisions on the classes $h+1, \ldots, q$ can leave the same capacity k, and therefore the same value $v(h, k)$ may be needed several times.

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

The bottom-up approach

- The **bottom-up approach** computes, for $h = 1, 2, \ldots, q$, first the rows $u(h, i, \cdot)$ for $i = 0, 1, \ldots, t_h$, and then the row $v(h, \cdot)$, each one for increasing k.

$$
\begin{array}{rl}
 & v(0,\cdot) \\
h = 1: & u(1,0,\cdot),\ u(1,1,\cdot),\ \ldots,\ u(1,t_1,\cdot) \ \rightarrow\ v(1,\cdot) \\
 & \vdots \\
h = q: & u(q,0,\cdot),\ u(q,1,\cdot),\ \ldots,\ u(q,t_q,\cdot) \ \rightarrow\ v(q,\cdot)
\end{array}
$$

- The optimal value of the KPS instance is $v(q, c)$.

Step 2: computation of an optimal solution

- Two choice tables are stored:
  - $s[h][i][k] = 1$ if the i-th item of class h is selected in the optimal value $u[h][i][k]$;
  - $o[h][k] = 1$ if class h is open in the optimal value $v[h][k]$.
- The setup weight s_h has one index; the choice $s[h][i][k]$ has three.

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**,
which stores the choice tables together with the values.

Algorithm: KPS_DP (1/4)

Input : A KPS instance given by q, the classes $\mathscr{J}_1, \ldots, \mathscr{J}_q$ numbered as above, the setup costs f_h and weights s_h, arrays $p[1..n]$, $w[1..n]$, and capacity c

Output: The optimal value of the KPS instance and the choice tables s and o

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

Algorithm: KPS_DP (2/4)

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

```
6   |   for h := 1 to q do
7   |   |   for k := 0 to c do
8   |   |   |   u[h][0][k] ← −∞
9   |   |   |   if k ≥ s_h then
10  |   |   |   |   u[h][0][k] ← v[h − 1][k − s_h] − f_h
11  |   |   |   end
12  |   |   end
```

Algorithm: KPS_DP (3/4)

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

```
    for h := 1 to q do (continued)
13      for i := 1 to t_h do
14          j := ℓ_{h-1} + i
15          for k := 0 to c do
16              u[h][i][k] ← u[h][i − 1][k]; s[h][i][k] ← 0
17              if w[j] ≤ k and
                   p[j] + u[h][i − 1][k − w[j]] > u[h][i][k]
                   then
18                 u[h][i][k] ← p[j] + u[h][i − 1][k − w[j]];
                      s[h][i][k] ← 1
19              end
20          end
21      end
```

Algorithm: KPS_DP (4/4)

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

```
    for h := 1 to q do (continued)
22      for k := 0 to c do
23          v[h][k] ← v[h − 1][k]; o[h][k] ← 0
24          if u[h][t_h][k] > v[h][k] then
25              v[h][k] ← u[h][t_h][k]; o[h][k] ← 1
26          end
27      end
28  end
29  return v[q][c], s, o
30 end
```

How KPS_DP works

- For each class:
  - the first loop over k pays the setup where it fits;
  - the loops over i and k run the DP algorithm of the KP over the items of the class;
  - the last loop over k compares the closed and the open class.
- In case of ties the item is not selected and the class stays closed, because the values are updated only when they are strictly larger.

Example: execution of the DP algorithm (1/6)

We consider the KPS instance of the example above: $c = 10$, $\mathscr{J}_1 = \{1, 2\}$ with $f_1 = 2$ and $s_1 = 2$, $\mathscr{J}_2 = \{3, 4, 5\}$ with $f_2 = 3$ and $s_2 = 1$, and

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

The rows are computed in the order of the table of the values, each one for $k = 0, 1, \ldots, 10$.

Example: execution of the DP algorithm (2/6)

These are some of the computations:

- $u[1][0][1] = -\infty$, since $k = 1 < s_1 = 2$
- $v[1][7] = \max\{v[0][7], u[1][2][7]\} = \max\{0, 9\} = 9$
- $v[2][10] = \max\{v[1][10], u[2][3][10]\} = \max\{9, 11\} = 11$
- $u[2][3][10] = \max\{u[2][2][10], p_5 + u[2][2][9]\} = \max\{10, 3 + 8\} = 11$

Example: execution of the DP algorithm (3/6)

These are some of the computations (continued):

- $u[2][2][9] = \max\{u[2][1][9],\ p_4 + u[2][1][5]\} = \max\{6,\ 7 + 1\} = 8$
- $u[2][1][5] = \max\{u[2][0][5],\ p_3 + u[2][0][3]\} = \max\{0,\ 4 - 3\} = 1$
- $u[2][0][3] = v[1][2] - f_2 = 0 - 3 = -3$
- $v[1][2] = \max\{v[0][2],\ u[1][2][2]\} = \max\{0,\ -2\} = 0$

Example: execution of the DP algorithm (4/6)

The table of the values after the execution of the DP algorithm, row by row in the order of computation ($k = 0, 1, \ldots, 5$):

Row	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$v[0][k]$	0	0	0	0	0	0
$u[1][0][k]$ (setup)	$-\infty$	$-\infty$	-2	-2	-2	-2
$u[1][1][k]$ ($j=1$)	$-\infty$	$-\infty$	-2	-2	-2	4
$u[1][2][k]$ ($j=2$)	$-\infty$	$-\infty$	-2	-2	3	4
$v[1][k]$	0	0	0	0	3	4
$u[2][0][k]$ (setup)	$-\infty$	-3	-3	-3	-3	0
$u[2][1][k]$ ($j=3$)	$-\infty$	-3	-3	1	1	1
$u[2][2][k]$ ($j=4$)	$-\infty$	-3	-3	1	1	4
$u[2][3][k]$ ($j=5$)	$-\infty$	-3	0	1	4	4
$v[2][k]$	0	0	0	1	4	4

Example: execution of the DP algorithm (5/6)

Row	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$v[0][k]$	0	0	0	0	0
$u[1][0][k]$ (setup)	-2	-2	-2	-2	-2
$u[1][1][k]$ $(j=1)$	4	4	4	4	4
$u[1][2][k]$ $(j=2)$	4	9	9	9	9
$v[1][k]$	4	9	9	9	9
$u[2][0][k]$ (setup)	1	1	6	6	6
$u[2][1][k]$ $(j=3)$	1	4	6	6	10
$u[2][2][k]$ $(j=4)$	4	8	8	8	10
$u[2][3][k]$ $(j=5)$	7	8	11	11	11
$v[2][k]$	7	9	11	11	11

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

- The optimal value of the instance is

$$v[2][10] = 11.$$

- The values -2 and -3 in the setup rows are the setup costs paid before any item is selected.

Example: the choice tables (1/2)

For the instance of the example, these are the choice tables computed by the DP algorithm: the rows $s[h][i][\cdot]$ of the items, and the rows $o[h][\cdot]$ of the classes ($k = 0, 1, \ldots, 5$).

Row	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$s[1][1][k]$ $(j=1)$	0	0	0	0	0	1
$s[1][2][k]$ $(j=2)$	0	0	0	0	1	0
$o[1][k]$	0	0	0	0	1	1
$s[2][1][k]$ $(j=3)$	0	0	0	1	1	1
$s[2][2][k]$ $(j=4)$	0	0	0	0	0	1
$s[2][3][k]$ $(j=5)$	0	0	1	0	1	0
$o[2][k]$	0	0	0	1	1	0

Example: the choice tables (2/2)

The choice tables ($k = 6, 7, \ldots, 10$):

Row	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
$s[1][1][k]$ $(j=1)$	1	1	1	1	1
$s[1][2][k]$ $(j=2)$	0	1	1	1	1
$o[1][k]$	1	1	1	1	1
$s[2][1][k]$ $(j=3)$	0	1	0	0	1
$s[2][2][k]$ $(j=4)$	1	1	1	1	0
$s[2][3][k]$ $(j=5)$	1	0	1	1	1
$o[2][k]$	1	0	1	1	1

Reconstructing an optimal solution

- An optimal solution is reconstructed by scanning the classes from $h = q$ down to $h = 1$, starting from $k = c$ and $x_j = 0$ for every item j.
- If $o[h][k] = 0$, class h is closed and k does not change.
- If $o[h][k] = 1$, the items of class h are scanned from $i = t_h$ down to $i = 1$ as in the KP, and at the end the setup weight s_h is subtracted from k.

Algorithm: KPS_Reconstruct (1/2)

Input : The number q of classes, the setup weights s_h, the array $w[1..n]$, the capacity c, and the choice tables s and o

Output: An optimal solution of the KPS instance

```
1 function KPS_Reconstruct(q, w, c, s, o)
2 |   x := new array[1..n], with x[j] = 0 for every j
3 |   k := c
```

Algorithm: KPS_Reconstruct (2/2)

function *KPS_Reconstruct*(q, w, c, s, o) (continued)

```
4   |  for h := q downto 1 do
5   |  |  if o[h][k] = 1 then
6   |  |  |  for i := t_h downto 1 do
7   |  |  |  |  j := ℓ_{h-1} + i; x[j] ← s[h][i][k]
8   |  |  |  |  k ← k − x[j] w[j]
9   |  |  |  end
10  |  |  |  k ← k − s_h
11  |  |  end
12  |  end
13  |  return x
14  end
```

Example: computing an optimal solution (1/4)

After the execution of the DP algorithm on the instance of the example, we start from $h = 2$ and $k = 10$. The choices read in the tables:

Iteration	Stage	k	Choice	Action
–	–	–	–	$k = 10$
1	$h = 2$	10	$o[2][10] = 1$	class 2 open
2	$j = 5$	10	$s[2][3][10] = 1$	$k \leftarrow 10 - 1 = 9$
3	$j = 4$	9	$s[2][2][9] = 1$	$k \leftarrow 9 - 4 = 5$
4	$j = 3$	5	$s[2][1][5] = 1$	$k \leftarrow 5 - 2 = 3$
–	–	3	–	setup: $k \leftarrow 3 - s_2 = 2$
5	$h = 1$	2	$o[1][2] = 0$	class 1 closed

Example: computing an optimal solution (2/4)

The same iterations, with the vector x:

Iteration	Stage	k	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$
–	–	–	0	0	0	0	0
1	$h = 2$	10	0	0	0	0	0
2	$j = 5$	10	0	0	0	0	1
3	$j = 4$	9	0	0	0	1	1
4	$j = 3$	5	0	0	1	1	1
–	–	3	0	0	1	1	1
5	$h = 1$	2	0	0	1	1	1

Example: computing an optimal solution (3/4)

- The vector
$$x = (0, 0, 1, 1, 1)$$
is an optimal solution of the KPS instance of the example.
- Only class 2 is open, and the items $\{3, 4, 5\}$ are selected.
- Its total weight is $2 + 4 + 1 + s_2 = 8 \leq c$.
- Its optimal value is $v(2, 10) = 4 + 7 + 3 - f_2 = 11$.

Example: computing an optimal solution (4/4)

- The items of class 1 have the larger ratio between total revenue and total weight (11/5 against 14/7), but class 1 stays closed.
- With class 1 alone the best value is $v[1][10] = 9$, while the open class 2 gives $u[2][3][10] = 11$.
- After its items and its setup weight only $k = 2$ is left for class 1, and $o[1][2] = 0$.

Number of steps

- Let $S(n, q, c)$ denote the number of steps executed by `KPS_DP` in the worst case.
- The loops over i run over the items of each class, so over all the classes they run $t_1 + t_2 + \cdots + t_q = n$ times.
- We assume that the conditions at lines 9, 17 and 24 are always true.
- The line numbers are those of the four parts of the pseudo-code of `KPS_DP`.

Number of executions of the lines

- Lines 2 and 29 are executed 1 time each.
- Lines 3 and 4 are executed $c + 1$ times each.
- Line 6 is executed q times.
- Lines 7, 8, 9, 10, 22, 23, 24 and 25 are executed $q(c + 1)$ times each.
- Lines 13 and 14 are executed n times each.
- Lines 15, 16, 17 and 18 are executed $n(c + 1)$ times each.

The total number of steps

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

$$S(n, q, c) = 4n(c + 1) + 8q(c + 1) + 2n + q + 2c + 4.$$

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

$$4n(c + 1) \ \leq \ S(n, q, c) \ \leq \ 12n(c + 1) + 3n + 2c + 4.$$

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 KPS executes

$$S(n, q, c) = \Theta(nc)$$

steps in the worst case: the setups add $2q$ rows to the n rows of the KP recurrence.

- The reconstruction executes $\Theta(n)$ steps.

Summary (1/2)

The recurrences:

- $v(h, k)$: optimal value with capacity k and the classes $\mathscr{J}_1, \ldots, \mathscr{J}_h$; $u(h, i, k)$: the same with class h open and its first i items available
- $u(h, 0, k) = v(h-1, k-s_h) - f_h$ pays the setup (or $-\infty$ if $k < s_h$); the rows $u(h, 1, \cdot), \ldots, u(h, t_h, \cdot)$ are the KP recurrence over the items of class h
- $v(h, k) = \max\{v(h-1, k),\ u(h, t_h, k)\}$: closed or open class

The DP algorithm:

- Optimal substructure and overlapping subproblems verified
- Bottom-up, class by class; on the example $v[2][10] = 11$

Summary (2/2)

An optimal solution:

- The choice tables $s[h][i][k]$ (items) and $o[h][k]$ (classes)
- From $h = q$ and $k = c$: if $o[h][k] = 1$, the items of class h are scanned as in the KP and then s_h is subtracted from k
- On the example: $x = (0, 0, 1, 1, 1)$, only class 2 open

Number of steps:

- $S(n, q, c) = 4n(c + 1) + 8q(c + 1) + 2n + q + 2c + 4 = \Theta(nc)$; the reconstruction executes $\Theta(n)$ steps

Typical mistakes

Introduction: typical mistakes

- We list the mistakes that are made most often with the DP algorithm for the KPS
- They concern where the setup is paid, the start of the rows of an open class, the reconstruction of a solution and the notation
- Knowing them helps to run the DP algorithm correctly by hand

Observation 1: Typical mistakes (1/2)

- ***Paying the setup for every item.*** *The setup of class h is paid once, in the row $u(h, 0, \cdot)$; the rows of the items add only p_j and w_j.*
- ***Starting the items of a class from*** $v(h-1, k)$***.*** *The rows of an open class start from $v(h-1, k-s_h) - f_h$: starting from $v(h-1, k)$ would select items without paying the setup.*

Observation: Typical mistakes (2/2)

- ***Forgetting the setup weight in the reconstruction.*** *After the items of an open class, the capacity decreases also by s_h.*
- ***Confusing s_h with $s[h][i][k]$.*** *The setup weight of class h has one index; the choice of the i-th item of class h has three.*

Summary

Typical mistakes:

- The setup of a class is paid once, in the row $u(h, 0, \cdot)$, not for every item
- The rows of an open class start from $v(h-1, k-s_h) - f_h$, not from $v(h-1, k)$
- In the reconstruction, the capacity of an open class decreases also by s_h
- The setup weight s_h has one index, the choice $s[h][i][k]$ has three

In summary

Introduction: in summary

- We collect the key ideas of the chapter on the DP algorithm for the KPS
- We close with the dashboard of the example of the chapter

Summary (1/2)

- For each class h the DP algorithm compares the closed class, $v(h-1,k)$, with the open class, $u(h,t_h,k)$.
- The open class starts from $u(h,0,k) = v(h-1,k-s_h) - f_h$, which pays the setup once, and continues with the KP recurrence over its items.
- The choice tables s (items) and o (classes) give an optimal solution, read backwards from (q,c).
- The DP algorithm executes $\Theta(nc)$ steps, as the DP algorithm of the KP.

Summary (2/2)

Dashboard of the KPS example ($c = 10$, two classes):

Method	Lower bound	Upper bound	Origin
DP algorithm	11	11	optimal value, solution $x = (0, 0, 1, 1, 1)$