DP for the multidimensional knapsack problem

Outline

A DP algorithm
- Number of steps
- The curse of dimensionality

Typical mistakes

In summary

Sapienza Università di Roma

Learning objectives

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

- write the recurrence $v(j, k_1, \ldots, k_m)$ over the items and the vector of the capacities;
- run the DP algorithm by hand on an instance with $m = 2$ dimensions, one matrix for each item;
- reconstruct an optimal solution from the tables s;
- count the $\Theta(m\, n \prod_{d=1}^{m}(c_d + 1))$ steps and explain why the number of states grows exponentially with m.

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 multidimensional knapsack problem, with one capacity c_d and one weight w_{dj} for every dimension d.
- From the module *Algorithms*: pseudocode conventions (Chapter 2) and the size of the input (Chapter 11).

The problem of this chapter

The problem of this chapter is the
multidimensional knapsack problem (MDKP):

its definition and its assumptions are on its page.

- We extend the DP algorithm of the KP to the MDKP: the capacity k becomes a vector of capacities, one for each dimension
- We see why the number of states grows exponentially with the number of dimensions

A DP algorithm

Introduction: A DP algorithm

- In this section we write a DP algorithm for the MDKP, in two steps: first the optimal value, then an optimal solution
- The algorithm follows the DP algorithm of the KP, but the capacity becomes the vector $\boldsymbol{k} = (k_1, k_2, \ldots, k_m)$: with $m = 2$ dimensions, each row of the table is a matrix
- We run it by hand on an instance with four items, we count its steps, and we see where the DP algorithm stops being useful: the curse of dimensionality

Weight vectors and capacity vectors

- We write m for the number of dimensions, so

$$\mathscr{D} = \{1, 2, \ldots, m\},$$

because in the DP chapters the letter k denotes a capacity.

- Item j has the **weight vector**

$$\boldsymbol{w}_j = (w_{1j}, w_{2j}, \ldots, w_{mj}),$$

and the knapsack has the capacity vector

$$\boldsymbol{c} = (c_1, c_2, \ldots, c_m).$$

Comparing vectors and feasible solutions

- For two vectors, $\boldsymbol{w}_j \leq \boldsymbol{k}$ means

$$w_{dj} \leq k_d \quad \text{for every dimension } d \in \mathscr{D}.$$

- A solution is described by the binary vector $(x_1, x_2, \ldots, x_n)$, where $x_j = 1$ if item j is selected; it is feasible if

$$\sum_{j=1}^{n} w_{dj} x_j \leq c_d \quad \text{for every } d \in \mathscr{D}.$$

Example: MDKP instance

We consider a knapsack with $m = 2$ dimensions and capacities $c_1 = 5$ and $c_2 = 4$, and a set $\mathscr{I} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following profits and weights:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$
p_j	9	8	3	9
w_{1j}	1	4	2	3
w_{2j}	2	1	1	4

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

The two steps of the DP algorithm

- Step 1: computation of the optimal value
  - a recursive formula gives the optimal value of each subinstance, with a vector of capacities
- Step 2: computation of an optimal solution
  - the tables s of the decisions give an optimal solution, read backwards

Step 1: the subinstances

For any $j \in \{0, 1, \ldots, n\}$ and any vector $\boldsymbol{k} = (k_1, k_2, \ldots, k_m)$ with $k_d \in \{0, 1, \ldots, c_d\}$ for every $d \in \mathscr{D}$, let

$$v(j, \boldsymbol{k}) = v(j, k_1, \ldots, k_m)$$

be the optimal value of the MDKP subinstance given by:

1. a knapsack with capacity k_d in each dimension $d \in \mathscr{D}$,
2. a set $\{1, 2, \ldots, j\}$ of available items,
3. for each item $h \in \{1, 2, \ldots, j\}$, a profit p_h and a weight vector $\boldsymbol{w}_h$.

Step 1: the recursive formula

The recursive formula to compute $v(j, \boldsymbol{k})$ reads as follows:

$$v(j, \boldsymbol{k}) = \begin{cases} 0 & \text{if } j = 0, \\ v(j-1, \boldsymbol{k}) & \text{if } j \geq 1 \text{ and} \\ & w_{dj} > k_d \text{ for some } d, \\ \max \left\{ v(j-1, \boldsymbol{k}), \right. & \text{if } j \geq 1 \text{ and } \boldsymbol{w}_j \leq \boldsymbol{k}. \\ \qquad \left. p_j + v(j-1, \boldsymbol{k} - \boldsymbol{w}_j) \right\} & \end{cases}$$

Reading the recursive formula

- With $m = 1$ the recursive formula is the recurrence of the KP: the only change is that the capacity k becomes the vector $\boldsymbol{k}$, and «item j fits» means $\boldsymbol{w}_j \leq \boldsymbol{k}$.

- The base cases are

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

since no item is available.

Optimal substructure

- The recursive formula generates at most 2 **independent subproblems**:
  - item j is not selected, and the residual subproblem has the same vector $\boldsymbol{k}$, or
  - item j is selected, and the residual subproblem has the vector $\boldsymbol{k} - \boldsymbol{w}_j$.

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

Overlapping subproblems and the bottom-up approach

- Different decisions on the items $j+1, \ldots, n$ can leave the same vector of residual capacities, and therefore the same value $v(j, \boldsymbol{k})$ may be needed several times.

$$\rightsquigarrow \textbf{Overlapping Subproblems verified!}$$

- The value $v(j, \boldsymbol{k})$ depends only on values of row $j-1$.
- Therefore, the **bottom-up approach** computes the rows $j = 0, 1, \ldots, n$ in this order, and in each row all the vectors $\boldsymbol{k}$ in any order.

The state of the DP algorithm

- The **state** of the DP algorithm is the pair $(j, \boldsymbol{k})$.
- For every j the table has one cell for each vector $\boldsymbol{k}$, that is,

$$\prod_{d=1}^{m}(c_d + 1) \text{ cells.}$$

- With $m = 2$ the row j is a matrix, with one row for each k_1 and one column for each k_2.

Step 2: computation of an optimal solution

- For each item j and vector $\boldsymbol{k}$ we store:
  - $s[j][\boldsymbol{k}] = 1$ if item j is selected in the optimal solution found for $(j, \boldsymbol{k})$,
  - $s[j][\boldsymbol{k}] = 0$ otherwise.

- The set of all the vectors $\boldsymbol{k}$ is

$$\mathscr{C} = \{0, 1, \ldots, c_1\} \times \{0, 1, \ldots, c_2\} \times \cdots \times \{0, 1, \ldots, c_m\}.$$

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

Algorithm: MDKP_DP (1/2)

Input : An MDKP instance given by n, m, the array $p[1..n]$, the weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$, and the capacity vector $\boldsymbol{c}$

Output: The optimal value of the MDKP instance and the tables $s[j][\boldsymbol{k}]$

```
1 function MDKP_DP(n, m, p, w, c)
2 |   v := new array[0..n][0..c_1] ··· [0..c_m];
  |    s := new array[0..n][0..c_1] ··· [0..c_m]
3 |   foreach k ∈ 𝒞 do
4 |   |   v[0][k] ← 0; s[0][k] ← 0
5 |   end
```

Algorithm: MDKP_DP (2/2)

function *MDKP_DP*$(n, m, p, \boldsymbol{w}, \boldsymbol{c})$ (continued)

```
6   |   for j := 1 to n do
7   |   |   foreach k ∈ 𝒞 do
8   |   |   |   v[j][k] ← v[j − 1][k]; s[j][k] ← 0
9   |   |   |   if w_j ≤ k and
    |   |   |     p[j] + v[j − 1][k − w_j] > v[j][k] then
10  |   |   |   |   v[j][k] ← p[j] + v[j − 1][k − w_j];
    |   |   |   |     s[j][k] ← 1
11  |   |   |   end
12  |   |   end
13  |   end
14  |   return v[n][c], s
15  end
```

How MDKP_DP works

- The structure is the one of the DP algorithm of the KP; the loop over the capacities becomes a loop over all the vectors $\boldsymbol{k} \in \mathscr{C}$.
- The test $\boldsymbol{w}_j \leq \boldsymbol{k}$ compares m pairs of numbers, and the vector $\boldsymbol{k} - \boldsymbol{w}_j$ needs m subtractions.
- In case of ties item j is not selected, because the value is updated only when it is strictly larger.

Example: execution of the DP algorithm (1/7)

- We consider the MDKP instance of the example above, with capacities $c_1 = 5$ and $c_2 = 4$.
- The row $j = 0$ is $v[0][\boldsymbol{k}] = 0$ for every $\boldsymbol{k}$.
- For each item j we compute the matrix $v[j][k_1][k_2]$, with:
  - one row for each $k_1 \in \{0, 1, \ldots, 5\}$,
  - one column for each $k_2 \in \{0, 1, \ldots, 4\}$.

Example: execution of the DP algorithm (2/7)

These are the computations of the cell $\boldsymbol{k} = \boldsymbol{c} = (5, 4)$ for each item, and of the cells used by the reconstruction (items 1 and 2):

$$\begin{aligned} v[1][(5,4)] &= \max\{v[0][(5,4)],\ p_1 + v[0][(4,2)]\} \\ &= \max\{0,\ 9+0\} = 9 \end{aligned}$$

$$\begin{aligned} v[1][(1,3)] &= \max\{v[0][(1,3)],\ p_1 + v[0][(0,1)]\} \\ &= \max\{0,\ 9+0\} = 9 \end{aligned}$$

$$\begin{aligned} v[2][(5,4)] &= \max\{v[1][(5,4)],\ p_2 + v[1][(1,3)]\} \\ &= \max\{9,\ 8+9\} = 17 \end{aligned}$$

Example: execution of the DP algorithm (3/7)

These are the computations of the cell $\boldsymbol{k} = \boldsymbol{c} = (5, 4)$ for each item, and of the cells used by the reconstruction (items 3 and 4):

$$\begin{aligned} v[3][(5,4)] &= \max\{v[2][(5,4)],\ p_3 + v[2][(3,3)]\} \\ &= \max\{17,\ 3+9\} = 17 \end{aligned}$$

$$\begin{aligned} v[4][(5,4)] &= \max\{v[3][(5,4)],\ p_4 + v[3][(2,0)]\} \\ &= \max\{17,\ 9+0\} = 17 \end{aligned}$$

Example: execution of the DP algorithm (4/7)

Item $j = 1$, with $p_1 = 9$ and $\boldsymbol{w}_1 = (1, 2)$:

$v[1][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	9	9	9
$k_1 = 2$	0	0	9	9	9
$k_1 = 3$	0	0	9	9	9
$k_1 = 4$	0	0	9	9	9
$k_1 = 5$	0	0	9	9	9

Example: execution of the DP algorithm (5/7)

Item $j = 2$, with $p_2 = 8$ and $\boldsymbol{w}_2 = (4, 1)$:

$v[2][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	9	9	9
$k_1 = 2$	0	0	9	9	9
$k_1 = 3$	0	0	9	9	9
$k_1 = 4$	0	8	9	9	9
$k_1 = 5$	0	8	9	17	17

Example: execution of the DP algorithm (6/7)

Item $j = 3$, with $p_3 = 3$ and $\boldsymbol{w}_3 = (2, 1)$:

$v[3][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	9	9	9
$k_1 = 2$	0	3	9	9	9
$k_1 = 3$	0	3	9	12	12
$k_1 = 4$	0	8	9	12	12
$k_1 = 5$	0	8	9	17	17

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

Item $j = 4$, with $p_4 = 9$ and $\boldsymbol{w}_4 = (3, 4)$:

$v[4][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	9	9	9
$k_1 = 2$	0	3	9	9	9
$k_1 = 3$	0	3	9	12	12
$k_1 = 4$	0	8	9	12	12
$k_1 = 5$	0	8	9	17	17

Example: the tables s (1/4)

For the instance of the example above, these are the matrices $s[j][k_1][k_2]$ computed by the DP algorithm; a 1 means that item j is selected in the optimal solution found for (j, k_1, k_2).

Item $j = 1$, with $p_1 = 9$ and $\boldsymbol{w}_1 = (1, 2)$:

$s[1][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	1	1	1
$k_1 = 2$	0	0	1	1	1
$k_1 = 3$	0	0	1	1	1
$k_1 = 4$	0	0	1	1	1
$k_1 = 5$	0	0	1	1	1

Example: the tables s (2/4)

Item $j = 2$, with $p_2 = 8$ and $\boldsymbol{w}_2 = (4, 1)$:

$s[2][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	0	0	0
$k_1 = 2$	0	0	0	0	0
$k_1 = 3$	0	0	0	0	0
$k_1 = 4$	0	1	0	0	0
$k_1 = 5$	0	1	0	1	1

Example: the tables s (3/4)

Item $j = 3$, with $p_3 = 3$ and $\boldsymbol{w}_3 = (2, 1)$:

$s[3][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	0	0	0
$k_1 = 2$	0	1	0	0	0
$k_1 = 3$	0	1	0	1	1
$k_1 = 4$	0	0	0	1	1
$k_1 = 5$	0	0	0	0	0

Example: the tables s (4/4)

Item $j = 4$, with $p_4 = 9$ and $\boldsymbol{w}_4 = (3, 4)$:

$s[4][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	0	0	0	0	0
$k_1 = 1$	0	0	0	0	0
$k_1 = 2$	0	0	0	0	0
$k_1 = 3$	0	0	0	0	0
$k_1 = 4$	0	0	0	0	0
$k_1 = 5$	0	0	0	0	0

Reconstruction of an optimal solution

- Once the tables s have been computed, an optimal solution is reconstructed:
  - scanning the items from $j = n$ down to $j = 1$,
  - starting from $\boldsymbol{k} = \boldsymbol{c}$ and $x_j = 0$ for every item j:

$$x_j \leftarrow s[j][\boldsymbol{k}], \qquad \boldsymbol{k} \leftarrow \boldsymbol{k} - s[j][\boldsymbol{k}]\, \boldsymbol{w}_j.$$

Algorithm: MDKP_Reconstruct

Input : The number n of items, the capacity vector $\boldsymbol{c}$, the weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$, and the tables s

Output: An optimal solution of the MDKP instance

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

Example: computing an optimal solution (1/3)

After the execution of the DP algorithm on the instance of the example above, we start from $j = 4$ and $\boldsymbol{k} = (5, 4)$. The values of the vector x at each iteration:

Iteration	j	$\boldsymbol{k}$	$s[j][\boldsymbol{k}]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$
–	–	–	–	0	0	0	0
1	4	$(5, 4)$	0	0	0	0	0
2	3	$(5, 4)$	0	0	0	0	0
3	2	$(5, 4)$	1	0	1	0	0
4	1	$(1, 3)$	1	1	1	0	0

Example: computing an optimal solution (2/3)

The action of each iteration, which updates x and the vector $\boldsymbol{k}$:

Iteration	j	$\boldsymbol{k}$	Action
–	–	–	$\boldsymbol{k} = (5, 4)$
1	4	$(5, 4)$	$x[4] \leftarrow 0$, $\boldsymbol{k} = (5, 4)$
2	3	$(5, 4)$	$x[3] \leftarrow 0$, $\boldsymbol{k} = (5, 4)$
3	2	$(5, 4)$	$x[2] \leftarrow 1$, $\boldsymbol{k} \leftarrow (5, 4) - (4, 1) = (1, 3)$
4	1	$(1, 3)$	$x[1] \leftarrow 1$, $\boldsymbol{k} \leftarrow (1, 3) - (1, 2) = (0, 1)$

Example: computing an optimal solution (3/3)

The vector $x = (1, 1, 0, 0)$ is an optimal solution of the MDKP instance of the example.

Its weights are $1 + 4 = 5 \leq c_1$ and $2 + 1 = 3 \leq c_2$, and its optimal value is $v(4, 5, 4) = 17$.

- With the first dimension alone, the KP with capacity $c_1 = 5$ has optimal value 18, attained by the items $\{1, 4\}$; their total weight in the second dimension is $2 + 4 = 6 > c_2$.
- Both dimensions must be checked at the same time, as the recursive formula does.

How many steps does the DP algorithm execute?

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

Number of steps (1/3)

- Let

$$N = |\mathscr{C}| = \prod_{d=1}^{m} (c_d + 1)$$

  be the number of vectors $\boldsymbol{k}$.

- Let $S(n, m, \boldsymbol{c})$ denote the number of steps executed by `MDKP_DP` in the worst case.

- Lines 9 and 10 work on vectors of m components, so we count m steps for each of their executions.

Number of steps (2/3)

- Lines 2 and 14 are executed 1 time each.
- Lines 3 and 4 are executed N times each.
- Line 6 is executed n times.
- Lines 7 and 8 are executed nN times each.
- Lines 9 and 10 are executed at most nN times each, with m steps each.

Number of steps (3/3)

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

$$S(n, m, \boldsymbol{c}) = 2m\,nN + 2nN + 2N + n + 2,$$

and, since

$$2m\,nN \leq S(n, m, \boldsymbol{c}) \leq 9m\,nN,$$

we have

$$S(n, m, \boldsymbol{c}) = \Theta\left(m\,n \prod_{d=1}^{m} (c_d + 1)\right).$$

The number of steps of the DP algorithm

The DP algorithm for the MDKP executes

$$\Theta(m\, n \prod_{d=1}^{m}(c_d + 1))$$

steps in the worst case.

For a fixed number m of dimensions this is

$$\Theta(n \prod_{d=1}^{m} c_d),$$

a pseudo-polynomial number of steps.

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

The number of steps contains the product of the capacities.

What happens when the number m of dimensions grows?

The curse of dimensionality

- Each new dimension multiplies the number of states by $c_d + 1$.
- If all the capacities are equal to c, the table has

$$(n+1)(c+1)^m$$

  cells: the number of states grows **exponentially** with m.
- The size of the input grows only linearly with m: one more dimension adds $n+1$ numbers, the weights w_{dj} and the capacity c_d.
- This explosion of the number of states when the state becomes a vector is called the **curse of dimensionality**.

How fast the table grows

m	cells $(n+1)(c+1)^m$, with $n = 50$ and $c = 100$
1	5 151
2	520 251
3	52 545 351
4	5 307 080 451
5	536 015 125 551

- With $m = 1$ the table has a few thousand cells; with $m = 5$ it has more than $5 \cdot 10^{11}$, too many to be stored.
- The DP algorithm is therefore useful for the MDKP only when m and the capacities are small.

Summary (1/2)

Step 1: the optimal value:

- the capacity k of the KP becomes the vector $\boldsymbol{k} = (k_1, \ldots, k_m)$
- $v(j, \boldsymbol{k}) = \max\{v(j-1, \boldsymbol{k}),\ p_j + v(j-1, \boldsymbol{k} - \boldsymbol{w}_j)\}$ if $\boldsymbol{w}_j \leq \boldsymbol{k}$, and $v(j, \boldsymbol{k}) = v(j-1, \boldsymbol{k})$ otherwise
- base cases $v(0, \boldsymbol{k}) = 0$; bottom-up, row by row, all the vectors $\boldsymbol{k}$ in any order

Step 2: an optimal solution:

- $s[j][\boldsymbol{k}] = 1$ if item j is selected in the optimal solution found for $(j, \boldsymbol{k})$
- read backwards from $(n, \boldsymbol{c})$: $x_j \leftarrow s[j][\boldsymbol{k}]$, $\boldsymbol{k} \leftarrow \boldsymbol{k} - s[j][\boldsymbol{k}]\, \boldsymbol{w}_j$
- on the example: $x = (1, 1, 0, 0)$, with optimal value 17

Summary (2/2)

Number of steps:

- $\Theta(m\,n\prod_{d=1}^{m}(c_d+1))$ for MDKP_DP, $\Theta(m\,n)$ for the reconstruction
- for a fixed m: $\Theta(n\prod_{d=1}^{m}c_d)$, pseudo-polynomial

The curse of dimensionality:

- with equal capacities c, the table has $(n+1)(c+1)^m$ cells: exponential in m
- the size of the input grows only linearly with m
- the DP algorithm is useful only when m and the capacities are small

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the DP algorithm for the MDKP
- Two of them concern the vector of the capacities, two the relation with the KP and the number of steps
- Check them before running the DP algorithm by hand

Typical mistakes (1/2)

Observation 1: Checking only one dimension

Item j fits in $\boldsymbol{k}$ only if $w_{dj} \leq k_d$ for every dimension d; one dimension is not enough.

Observation 2: Updating only one capacity

When item j is selected, all the components of $\boldsymbol{k}$ decrease, by $\boldsymbol{w}_j$.

Typical mistakes (2/2)

Observation 3: Solving one KP per dimension

The KP with the capacity c_d alone gives an upper bound, but its optimal solution may violate another dimension.

Observation 4: Calling the DP algorithm polynomial

Its number of steps contains the product of the capacities, which is exponential in m and in the size of the capacities.

Summary

The vector of the capacities:

- item j fits only if $\boldsymbol{w}_j \leq \boldsymbol{k}$, in every dimension
- selecting item j decreases all the components of $\boldsymbol{k}$

The KP and the number of steps:

- one KP per dimension gives an upper bound, not a feasible solution
- the DP algorithm is pseudo-polynomial, not polynomial

In summary

Introduction: In summary

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

Summary (1/2)

- The recurrence of the KP extends to the MDKP by replacing the capacity k with the vector $\boldsymbol{k} = (k_1, \ldots, k_m)$:

$$v(j, \boldsymbol{k}) = \max\{v(j-1, \boldsymbol{k}),\ p_j + v(j-1, \boldsymbol{k} - \boldsymbol{w}_j)\}$$

  when $\boldsymbol{w}_j \leq \boldsymbol{k}$.

- The table has $(n+1)\prod_{d=1}^{m}(c_d+1)$ cells and the DP algorithm executes $\Theta(m\,n\prod_{d=1}^{m}(c_d+1))$ steps, that is, $\Theta(n\prod_{d=1}^{m} c_d)$ for a fixed m.

- The number of states grows exponentially with m: the curse of dimensionality.

- The tables s give an optimal solution, read backwards from $(n, \boldsymbol{c})$.

Summary (2/2)

Dashboard of the MDKP example ($\boldsymbol{c} = (5, 4)$, four items):

Method	Lower bound	Upper bound	Origin
DP algorithm	17	17	optimal value, solution $x = (1, 1, 0, 0)$