DP for the capacitated lot sizing problem

Sapienza Università di Roma

Outline

Notation, model and feasible solutions

A DP algorithm for the CLS
- Step 1 and Step 2
- Number of steps
- The case of constant capacity

Typical mistakes

In summary

Learning objectives

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

- explain why the zero-inventory ordering property fails with capacities, on an example;
- write the recurrence $v(t, k)$ on the pairs (period, final inventory), with its base cases;
- run the DP algorithm by hand, fill the matrices v and r, and reconstruct an optimal plan;
- count the steps of the DP algorithm, and say why it is pseudo-polynomial;
- state the weaker property that holds when the capacity is the same in all periods.

Prerequisites (1/2)

- From this module, the chapter «DP for the uncapacitated lot sizing problem»: the total demands $D_{j,t}$, the encoding of a production plan and its network, the zero-inventory ordering property, and the instance of its first example.

- From the module *Combinatorial Optimization Problems*, the chapters «Lot sizing problems» and «Formulations of lot sizing problems»: the capacitated problem, its reference instance, and its ILP model with the variables x_t, s_t, y_t.

Prerequisites (2/2)

- From this module, the chapter «Principles of dynamic programming»: recurrence, base cases, lookup table and reconstruction of a solution.
- From the module *Algorithms*: pseudocode, and the size of the input.

Where we were

- **As a problem and as a model**: the capacitated lot sizing problem and its ILP formulations (*Combinatorial Optimization Problems*, chapters «Lot sizing problems» and «Formulations of lot sizing problems»).
- **As a table, without capacities**: the DP algorithm of Wagner and Whitin of the chapter «DP for the uncapacitated lot sizing problem», over the production runs, in $\Theta(m^2)$ steps.
- **Now, with capacities**: a DP algorithm on the pairs (period, inventory), which is pseudo-polynomial.

The problem of this chapter

The problem of this chapter is the

capacitated lot sizing problem (CLS):

its definition and its assumptions are on its page.

Notation, model and feasible solutions

Introduction: notation, model and feasible solutions

- We recall the notation of the DP algorithm without capacities: the total demands $D_{j,t}$
- We recall the ILP model of the CLS and how a feasible solution is encoded
- We see why the zero-inventory ordering property, on which the DP algorithm without capacities rests, fails with capacities
- We fix the CLS instance that we use in all the examples of the chapter

Total demands

- As in the chapter «DP for the uncapacitated lot sizing problem», for any $j, t \in T$ we denote by

$$D_{j,t} = \sum_{k=j}^{t} d_k$$

  the total demand of periods $j, j+1, \ldots, t$

- By convention, $D_{j,t} = 0$ if $j > t$

The ILP model

- The ILP model is the $\mathrm{ILP}_{\mathrm{CLSP}}$ model of the chapter «Formulations of lot sizing problems», with the big-M value

$$\min\{cap_t, D_{t,m}\}$$

of period t

- It is the $\mathrm{ILP}_{\mathrm{ULS}}$ model with the additional constraints

$$x_t \leq cap_t \qquad \forall\, t \in T$$

Feasible solutions

- A feasible solution is encoded by the same vector $(x_1, x_2, \ldots, x_m)$ as in the ULS, with the inventories

$$s_t = \sum_{j=1}^{t} x_j - D_{1,t}$$

- With the additional requirement

$$x_t \leq cap_t \qquad \forall\, t \in T$$

No zero-inventory ordering with capacities

- The **zero-inventory ordering property does not hold** for the CLS
- A period may have to produce while units are still in stock, because the capacity does not allow to produce everything in the period where it would be cheapest
- The property, and the DP algorithm over the production runs that rests on it, are those of the chapter «DP for the uncapacitated lot sizing problem»

Example: capacitated lot sizing (1/2)

We consider the uncapacitated instance of the first example of the chapter «DP for the uncapacitated lot sizing problem», measured in units of 10 products (hence the holding cost is 10 per unit), with a constant capacity of 4 units per period:

t	1	2	3	4	5
d_t	2	1	3	4	1
sc_t	50	50	50	50	50
vc_t	0	0	0	0	0
hc_t	10	10	10	10	10
cap_t	4	4	4	4	4

Example: capacitated lot sizing (2/2)

- We refer to this CLS instance as **the example instance**: it is the reference instance of the chapter «Lot sizing problems» of *Combinatorial Optimization Problems*

- The optimal solution of the uncapacitated instance, in units of 10, is the vector

$$(3, 0, 8, 0, 0),$$

  of cost 170

- It is not feasible for the example instance, since period 3 produces more than 4 units

Summary

Notation and model:

- $D_{j,t}$ is the total demand of periods $j, \ldots, t$
- The model $\mathrm{ILP}_{\mathrm{CLSP}}$ is $\mathrm{ILP}_{\mathrm{ULS}}$ with the constraints $x_t \leq cap_t$
- A feasible solution is a vector $(x_1, \ldots, x_m)$ with $x_t \leq cap_t$ for all $t \in T$

Zero-inventory ordering:

- It does not hold with capacities: a period may produce while units are in stock
- On the example instance ($cap_t = 4$), the uncapacitated optimum $(3, 0, 8, 0, 0)$ is not feasible

A DP algorithm for the CLS

Introduction: a DP algorithm for the CLS

- Without capacities the DP algorithm works over the production runs; with capacities we need a different state: the pair (period, final inventory)
- We write the recurrence $v(t, k)$, verify optimal substructure and overlapping subproblems, and write the DP algorithm with the bottom-up approach
- We run it by hand on the example instance and reconstruct an optimal production plan
- We count its steps, which makes it pseudo-polynomial, and we look at the case of constant capacity

Step 1: computation of the optimal value

- For any period $t \in \{1, \ldots, m\}$, the **production cost function** of period t is

$$f_t(x) = \begin{cases} 0 & \text{if } x = 0, \\ sc_t + vc_t\, x & \text{if } x \in \{1, 2, \ldots, cap_t\}. \end{cases}$$

- The inventory at the end of period t never exceeds the demand $D_{t+1,m}$ of the following periods, since the inventory must be empty at the end of period m

The subinstances

For any $t \in \{1, 2, \ldots, m\}$ and $k \in \{0, 1, \ldots, D_{t+1,m}\}$, let $v(t, k)$ be the optimal value of the CLS subinstance given by:

1. a set $\{1, 2, \ldots, t\}$ of t time periods,
2. for each period $j \in \{1, 2, \ldots, t\}$, a demand d_j,
3. for each period $j \in \{1, 2, \ldots, t\}$, a setup cost sc_j, a variable production cost vc_j, and a holding cost hc_j,
4. for each period $j \in \{1, 2, \ldots, t\}$, a capacity cap_j,
5. an inventory equal to k at the end of period t.

The recurrence

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

$$v(t,k) = \begin{cases} \min\Big\{v(t-1, k+d_t-x) \quad \text{if } 1 \le t \le m, \\ \qquad + f_t(x) : x \in X_t(k)\Big\} \\ + hc_t\, k \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{if } t = 0 \text{ and } k = 0, \\ +\infty \qquad\qquad\qquad\qquad\qquad\qquad \text{if } t = 0 \text{ and } k > 0, \end{cases}$$

where

$$X_t(k) = \{0, 1, \ldots, \min\{cap_t,\ k + d_t\}\}$$

is the set of the possible productions of period t.

States and subproblems

- The pair (t, k) is the **state**: the last period of the subinstance and the inventory at its end
- For $t = m$ and $k = 0$ we obtain the CLS instance itself, so its optimal value is $v(m, 0)$
- The recursive decomposition process of the recurrence generates at most $cap_t + 1$ **independent subproblems**, each corresponding to a possible choice of the production x of the last period t

The residual subproblem

- If x units are produced in period t and k units remain at its end, the inventory at the end of period $t-1$ is $k + d_t - x$
- The residual subproblem is the CLS subinstance with periods $\{1, \ldots, t-1\}$ and final inventory $k + d_t - x$
- The production x is at most cap_t, and at most $k + d_t$ because the inventory at the end of period $t-1$ cannot be negative
- Moreover $k + d_t - x \leq D_{t+1,m} + d_t = D_{t,m}$, so the residual subproblem is a state

Base cases

- The base case is $v(0, 0) = 0$: no period, no inventory, no cost
- For $k > 0$ we set $v(0, k) = +\infty$: the initial inventory is empty, so no feasible solution has a positive inventory before period 1
- The value $+\infty$ rules out this case, and it propagates to every state that cannot be reached within the capacities, with the convention $+\infty + a = +\infty$

Optimal substructure

- The optimal value $v(t, k)$ is obtained by trying all possible productions x of period t and selecting the one that minimizes the sum

$$v(t-1, k+d_t-x)+f_t(x)$$

  of the optimal value of the residual subproblem and the production cost of period t
- The holding cost $hc_t\, k$ of the final inventory does not depend on x
- Hence, an optimal solution of the subinstance of state (t, k) is built by combining the chosen production of period t with an optimal solution of the residual subproblem of state $(t-1, k+d_t-x)$

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

Overlapping subproblems

- Different sequences of production decisions can lead to the same state
- For instance, in the example instance, starting from the state $(5, 0)$:
  - the decisions $x_5 = 0$ and $x_4 = 4$ lead to the state $(3, 1)$
  - the decisions $x_5 = 1$ and $x_4 = 3$ lead to the same state $(3, 1)$
- Therefore the same value $v(t, k)$ may need to be computed several times during the recursive evaluation of the recurrence

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

First method: the top-down approach

- A **first method** to compute the optimal value $v(m, 0)$ of the CLS instance is the **top-down approach**
- The method starts from the call with $(t, k) = (m, 0)$ and evaluates the recurrence recursively
- Each call for a state (t, k) triggers further calls for the states $(t - 1, k + d_t - x)$, for all $x \in \{0, \ldots, \min\{cap_t, k + d_t\}\}$, until one of the base cases $v(0, k)$ is reached

The lookup table

- The value $v(t, k)$ can be computed once the values $v(t-1, k')$ are known for all $k' \in \{0, \ldots, D_{t,m}\}$
- We can avoid solving the same subproblems repeatedly by storing the value of each subproblem as soon as it is computed, in a **lookup table**: the matrix

$$v[0..m][0..D_{1,m}]$$

- The cell $v[t][k]$ contains $v(t, k)$ for $k \leq D_{t+1,m}$ (the other cells are not used)

Second method: the bottom-up approach

- A **second method** to compute the optimal value $v(m, 0)$ of the CLS instance is the **bottom-up approach**
- The method starts from the base cases $v(0, 0) = 0$ and $v(0, k) = +\infty$ for all $k \in \{1, \ldots, D_{1,m}\}$
- It evaluates the recurrence for increasing values of $t \in \{1, \ldots, m\}$ and, for each fixed t, for $k \in \{0, 1, \ldots, D_{t+1,m}\}$: each value $v(t, k)$ uses already-computed values of row $t - 1$, until $v(m, 0)$ is obtained

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**.

Algorithm: DP_CLS (1/2)

Input : A CLS instance given by the number m of periods and the arrays $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$, $cap[1..m]$

Output : The optimal value

```
1  function DP_CLS(m, d, sc, vc, hc, cap)
2      R := new array[0..m]
3      R[m] ← 0
4      for t := m − 1 downto 0 do
5          R[t] ← R[t + 1] + d[t + 1]
6      end
7      v := new array[0..m][0..R[0]]
8      v[0][0] ← 0
9      for k := 1 to R[0] do
10         v[0][k] ← +∞
11     end
```

Algorithm: DP_CLS (2/2)

function *DP_CLS*(m, d, sc, vc, hc, cap) (continued)

```
12      for t := 1 to m do
13          for k := 0 to R[t] do
14              b := +∞
15              for x := 0 to min{cap[t], k + d[t]} do
16                  if v[t − 1][k + d[t] − x] + f_t(x) + hc[t] · k < b
                    then
17                      b ← v[t − 1][k + d[t] − x] + f_t(x) + hc[t] · k
18                  end
19              end
20              v[t][k] ← b
21          end
22      end
23      return v[m][0]
24  end
```

How DP_CLS works (1/2)

- The vector $R[0..m]$ contains the remaining demands $R[t] = D_{t+1,m}$, computed backwards from $R[m] = 0$: the states of period t are the pairs (t, k) with $k \leq R[t]$
- The matrix v is the **lookup table**. The initialization sets $v[0][0] = 0$ and $v[0][k] = +\infty$ for all $k \in \{1, \ldots, R[0]\}$, since the initial inventory is empty
- The outer loop, over $t \in \{1, 2, \ldots, m\}$, considers the periods in increasing order, and the second loop, over $k \in \{0, \ldots, R[t]\}$, considers all final inventories of period t

How DP_CLS works (2/2)

- The inner loop, over the production x of period t, keeps in b the smallest value

$$v[t-1][k+d[t]-x]+f_t(x)+hc[t]\cdot k$$

- The value $f_t(x)$ is 0 for $x=0$ and $sc[t]+vc[t]\cdot x$ otherwise
- If all candidates are $+\infty$, the comparison is never true and $v[t][k]=+\infty$: the state (t,k) cannot be reached within the capacities
- Finally, the optimal value $v[m][0]$ of the CLS instance is returned

Example: execution of the DP algorithm (1/11)

We consider the CLS instance of the example:

t	1	2	3	4	5
d_t	2	1	3	4	1
sc_t	50	50	50	50	50
vc_t	0	0	0	0	0
hc_t	10	10	10	10	10
cap_t	4	4	4	4	4

The remaining demands are $(R[0], \ldots, R[5]) = (11, 9, 8, 5, 1, 0)$.

For each step of the DP algorithm, the table on the next slides shows the values of t and k, the cell $v[t][k]$, and the operation performed. The candidates of the minimum are listed for $x = 0, 1, \ldots, \min\{cap_t, k + d_t\}$.

Example: execution of the DP algorithm (2/11)

t	k	$v[t][k]$	Action
0	0	0	$v[0][0] \leftarrow 0$
0	1	$+\infty$	$v[0][1] \leftarrow +\infty$
0	2	$+\infty$	$v[0][2] \leftarrow +\infty$
0	3	$+\infty$	$v[0][3] \leftarrow +\infty$
0	4	$+\infty$	$v[0][4] \leftarrow +\infty$
0	5	$+\infty$	$v[0][5] \leftarrow +\infty$
0	6	$+\infty$	$v[0][6] \leftarrow +\infty$
0	7	$+\infty$	$v[0][7] \leftarrow +\infty$
0	8	$+\infty$	$v[0][8] \leftarrow +\infty$
0	9	$+\infty$	$v[0][9] \leftarrow +\infty$
0	10	$+\infty$	$v[0][10] \leftarrow +\infty$
0	11	$+\infty$	$v[0][11] \leftarrow +\infty$

Example: execution of the DP algorithm (3/11)

t	k	$v[t][k]$	Action
1	0	50	$\min\{v[0][2]+f_1(0), v[0][1]+f_1(1),$ $v[0][0]+f_1(2)\}+hc_1 \cdot 0$ $= \min\{+\infty, +\infty, 0+50\}+10 \cdot 0 = 50$
1	1	60	$\min\{v[0][3]+f_1(0), v[0][2]+f_1(1), v[0][1]+f_1(2),$ $v[0][0]+f_1(3)\}+hc_1 \cdot 1$ $= \min\{+\infty, +\infty, +\infty, 0+50\}+10 \cdot 1 = 60$
1	2	70	$\min\{v[0][4]+f_1(0), v[0][3]+f_1(1), v[0][2]+f_1(2),$ $v[0][1]+f_1(3), v[0][0]+f_1(4)\}+hc_1 \cdot 2$ $= \min\{+\infty, +\infty, +\infty, +\infty, 0+50\}+10 \cdot 2 = 70$
1	3	$+\infty$	the candidates use the cells $v[0][1], \ldots, v[0][5]$, which are all $+\infty$: $v[1][3] \leftarrow +\infty$
1	4	$+\infty$	the candidates use the cells $v[0][2], \ldots, v[0][6]$, which are all $+\infty$: $v[1][4] \leftarrow +\infty$

Example: execution of the DP algorithm (4/11)

t	k	$v[t][k]$	Action
1	5	$+\infty$	the candidates use the cells $v[0][3], \ldots, v[0][7]$, which are all $+\infty$: $v[1][5] \leftarrow +\infty$
1	6	$+\infty$	the candidates use the cells $v[0][4], \ldots, v[0][8]$, which are all $+\infty$: $v[1][6] \leftarrow +\infty$
1	7	$+\infty$	the candidates use the cells $v[0][5], \ldots, v[0][9]$, which are all $+\infty$: $v[1][7] \leftarrow +\infty$
1	8	$+\infty$	the candidates use the cells $v[0][6], \ldots, v[0][10]$, which are all $+\infty$: $v[1][8] \leftarrow +\infty$
1	9	$+\infty$	the candidates use the cells $v[0][7], \ldots, v[0][11]$, which are all $+\infty$: $v[1][9] \leftarrow +\infty$

Example: execution of the DP algorithm (5/11)

t	k	$v[t][k]$	Action
2	0	60	$\min\{v[1][1]+f_2(0), v[1][0]+f_2(1)\}+hc_2 \cdot 0$ $= \min\{60+0, 50+50\}+10 \cdot 0 = 60$
2	1	80	$\min\{v[1][2]+f_2(0), v[1][1]+f_2(1),$ $v[1][0]+f_2(2)\}+hc_2 \cdot 1$ $= \min\{70+0, 60+50, 50+50\}+10 \cdot 1 = 80$
2	2	120	$\min\{v[1][3]+f_2(0), v[1][2]+f_2(1), v[1][1]+f_2(2),$ $v[1][0]+f_2(3)\}+hc_2 \cdot 2$ $= \min\{+\infty, 70+50, 60+50, 50+50\}+10 \cdot 2 = 120$
2	3	130	$\min\{v[1][4]+f_2(0), v[1][3]+f_2(1), v[1][2]+f_2(2),$ $v[1][1]+f_2(3), v[1][0]+f_2(4)\}+hc_2 \cdot 3$ $= \min\{+\infty, +\infty, 70+50, 60+50,$ $50+50\}+10 \cdot 3 = 130$

Example: execution of the DP algorithm (6/11)

t	k	$v[t][k]$	Action
2	4	150	$\min\{v[1][5]+f_2(0), v[1][4]+f_2(1), v[1][3]+f_2(2),$ $v[1][2]+f_2(3), v[1][1]+f_2(4)\}+hc_2 \cdot 4$ $= \min\{+\infty, +\infty, +\infty, 70+50, 60+50\}+10 \cdot 4 = 150$
2	5	170	$\min\{v[1][6]+f_2(0), v[1][5]+f_2(1), v[1][4]+f_2(2),$ $v[1][3]+f_2(3), v[1][2]+f_2(4)\}+hc_2 \cdot 5$ $= \min\{+\infty, +\infty, +\infty, +\infty, 70+50\}+10 \cdot 5 = 170$
2	6	$+\infty$	the candidates use the cells $v[1][3], \ldots, v[1][7]$, which are all $+\infty$: $v[2][6] \leftarrow +\infty$
2	7	$+\infty$	the candidates use the cells $v[1][4], \ldots, v[1][8]$, which are all $+\infty$: $v[2][7] \leftarrow +\infty$
2	8	$+\infty$	the candidates use the cells $v[1][5], \ldots, v[1][9]$, which are all $+\infty$: $v[2][8] \leftarrow +\infty$

Example: execution of the DP algorithm (7/11)

t	k	$v[t][k]$	Action
3	0	110	$\min\{v[2][3]+f_3(0), v[2][2]+f_3(1), v[2][1]+f_3(2), v[2][0]+f_3(3)\}+hc_3 \cdot 0$ $= \min\{130+0, 120+50, 80+50, 60+50\}+10 \cdot 0 = 110$
3	1	120	$\min\{v[2][4]+f_3(0), v[2][3]+f_3(1), v[2][2]+f_3(2), v[2][1]+f_3(3), v[2][0]+f_3(4)\}+hc_3 \cdot 1$ $= \min\{150+0, 130+50, 120+50, 80+50, 60+50\}+10 \cdot 1 = 120$
3	2	150	$\min\{v[2][5]+f_3(0), v[2][4]+f_3(1), v[2][3]+f_3(2), v[2][2]+f_3(3), v[2][1]+f_3(4)\}+hc_3 \cdot 2$ $= \min\{170+0, 150+50, 130+50, 120+50, 80+50\}+10 \cdot 2 = 150$

Example: execution of the DP algorithm (8/11)

t	k	$v[t][k]$	Action
3	3	200	$\min\{v[2][6]+f_3(0), v[2][5]+f_3(1), v[2][4]+f_3(2),$ $v[2][3]+f_3(3), v[2][2]+f_3(4)\}+hc_3 \cdot 3$ $= \min\{+\infty, 170+50, 150+50, 130+50,$ $120+50\}+10 \cdot 3 = 200$
3	4	220	$\min\{v[2][7]+f_3(0), v[2][6]+f_3(1), v[2][5]+f_3(2),$ $v[2][4]+f_3(3), v[2][3]+f_3(4)\}+hc_3 \cdot 4$ $= \min\{+\infty, +\infty, 170+50, 150+50,$ $130+50\}+10 \cdot 4 = 220$
3	5	250	$\min\{v[2][8]+f_3(0), v[2][7]+f_3(1), v[2][6]+f_3(2),$ $v[2][5]+f_3(3), v[2][4]+f_3(4)\}+hc_3 \cdot 5$ $= \min\{+\infty, +\infty, +\infty, 170+50,$ $150+50\}+10 \cdot 5 = 250$

Example: execution of the DP algorithm (9/11)

t	k	$v[t][k]$	Action
4	0	160	$\min\{v[3][4]+f_4(0), v[3][3]+f_4(1), v[3][2]+f_4(2),$ $v[3][1]+f_4(3), v[3][0]+f_4(4)\}+hc_4 \cdot 0$ $= \min\{220+0, 200+50, 150+50, 120+50,$ $110+50\}+10 \cdot 0 = 160$
4	1	180	$\min\{v[3][5]+f_4(0), v[3][4]+f_4(1), v[3][3]+f_4(2),$ $v[3][2]+f_4(3), v[3][1]+f_4(4)\}+hc_4 \cdot 1$ $= \min\{250+0, 220+50, 200+50, 150+50,$ $120+50\}+10 \cdot 1 = 180$
5	0	180	$\min\{v[4][1]+f_5(0), v[4][0]+f_5(1)\}+hc_5 \cdot 0$ $= \min\{180+0, 160+50\}+10 \cdot 0 = 180$

Example: execution of the DP algorithm (10/11)

This is the matrix v after the execution of the DP algorithm (a dot marks a pair (t, k) with $k > D_{t+1,m}$, which is not a state), columns $k = 0, \ldots, 5$:

$v[t][k]$	0	1	2	3	4	5
$t = 0$	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$t = 1$	50	60	70	$+\infty$	$+\infty$	$+\infty$
$t = 2$	60	80	120	130	150	170
$t = 3$	110	120	150	200	220	250
$t = 4$	160	180	.	.	.	.
$t = 5$	180	.	.	.	.	.

Example: execution of the DP algorithm (11/11)

The matrix v after the execution of the DP algorithm, columns $k = 6, \ldots, 11$:

$v[t][k]$	6	7	8	9	10	11
$t = 0$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$t = 1$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	.	.
$t = 2$	$+\infty$	$+\infty$	$+\infty$	.	.	.
$t = 3$	.	.	.	.	.	.
$t = 4$	.	.	.	.	.	.
$t = 5$	.	.	.	.	.	.

The optimal value of the CLS instance is $v[5][0] = 180$.

Step 2: computation of an optimal solution

- After the execution of the DP algorithm, the matrix v contains the optimal value $v[t][k]$ for every state (t, k)
- To compute an optimal solution, it is necessary to store some additional information
- For each state (t, k), in addition to the optimal value $v[t][k]$, we store $r[t][k]$, the production of period t in an optimal solution of the subinstance of state (t, k); for the states with value $+\infty$ we store 0

Reading the matrix r

- Starting from $(t, k) = (m, 0)$, the value $r[t][k]$ gives the production of period t
- The inventory at the end of period $t-1$ is then $k + d_t - r[t][k]$
- The same rule can be applied again to period $t-1$, and so on, until $t = 0$

Next: the pseudo-code of the **DP algorithm using the bottom-up approach**,
with the storing of the information necessary to compute an optimal solution.

Algorithm: DP_CLS_InfoSol (1/4)

Input : A CLS instance given by the number m of periods and the arrays $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$, $cap[1..m]$

Output : The optimal value and the information matrix $r[0..m][0..D_{1,m}]$

```
1 function DP_CLS_InfoSol(m, d, sc, vc, hc, cap)
2 |   R := new array[0..m]
3 |   R[m] ← 0
4 |   for t := m − 1 downto 0 do
5 |   |   R[t] ← R[t + 1] + d[t + 1]
6 |   end
7 |   v := new array[0..m][0..R[0]]
8 |   r := new array[0..m][0..R[0]]
```

Algorithm: DP_CLS_InfoSol (2/4)

function *DP_CLS_InfoSol*(m, d, sc, vc, hc, cap) (continued)

```
9   |  v[0][0] ← 0
10  |  r[0][0] ← 0
11  |  for k := 1 to R[0] do
12  |  |  v[0][k] ← +∞
13  |  |  r[0][k] ← 0
14  |  end
```

Algorithm: DP_CLS_InfoSol (3/4)

function *DP_CLS_InfoSol*(m, d, sc, vc, hc, cap) (continued)

```
15      for t := 1 to m do
16          for k := 0 to R[t] do
17              b := +∞
18              a := 0
19              for x := 0 to min{cap[t], k + d[t]} do
20                  if v[t − 1][k + d[t] − x] + f_t(x) + hc[t] · k < b
                    then
21                      b ← v[t − 1][k + d[t] − x] + f_t(x) + hc[t] · k
22                      a ← x
23                  end
24              end
```

Algorithm: DP_CLS_InfoSol (4/4)

```
function DP_CLS_InfoSol(m, d, sc, vc, hc, cap) (continued)
    for t := 1 to m (continued) do
        for k := 0 to R[t] (continued) do
25          v[t][k] ← b
26          r[t][k] ← a
27      end
28  end
29  return v[m][0], r
30 end
```

Storing the decisions

- The DP algorithm uses the additional temporary variable a to keep track of the decision made when b is updated
- If the production x is selected, $a \leftarrow x$ stores this decision
- At the end of the inner loop $r[t][k] \leftarrow a$ stores it, together with the optimal value $v[t][k] \leftarrow b$
- Because of the strict comparison, in case of ties the smallest production is stored

Example: computation of the additional information (1/8)

We consider the CLS instance of the example:

t	1	2	3	4	5
d_t	2	1	3	4	1
sc_t	50	50	50	50	50
vc_t	0	0	0	0	0
hc_t	10	10	10	10	10
cap_t	4	4	4	4	4

For each step of the DP algorithm, the table on the next slides shows the values of t and k, the cell $r[t][k]$, and the production x for which the minimum of the corresponding row of the previous example is attained first.

Example: computation of the additional information (2/8)

t	k	$r[t][k]$	Action
0	0	0	$r[0][0] \leftarrow 0$
0	1	0	$r[0][1] \leftarrow 0$
0	2	0	$r[0][2] \leftarrow 0$
0	3	0	$r[0][3] \leftarrow 0$
0	4	0	$r[0][4] \leftarrow 0$
0	5	0	$r[0][5] \leftarrow 0$
0	6	0	$r[0][6] \leftarrow 0$
0	7	0	$r[0][7] \leftarrow 0$
0	8	0	$r[0][8] \leftarrow 0$
0	9	0	$r[0][9] \leftarrow 0$
0	10	0	$r[0][10] \leftarrow 0$
0	11	0	$r[0][11] \leftarrow 0$

Example: computation of the additional information (3/8)

t	k	$r[t][k]$	Action
1	0	2	first minimum at $x = 2$, i.e., $v[0][0]+f_1(2)+hc_1 \cdot 0 = 0+50+0 = 50$: $r[1][0] \leftarrow 2$
1	1	3	first minimum at $x = 3$, i.e., $v[0][0]+f_1(3)+hc_1 \cdot 1 = 0+50+10 = 60$: $r[1][1] \leftarrow 3$
1	2	4	first minimum at $x = 4$, i.e., $v[0][0]+f_1(4)+hc_1 \cdot 2 = 0+50+20 = 70$: $r[1][2] \leftarrow 4$
1	3	0	all candidates equal $+\infty$: $r[1][3] \leftarrow 0$
1	4	0	all candidates equal $+\infty$: $r[1][4] \leftarrow 0$
1	5	0	all candidates equal $+\infty$: $r[1][5] \leftarrow 0$
1	6	0	all candidates equal $+\infty$: $r[1][6] \leftarrow 0$
1	7	0	all candidates equal $+\infty$: $r[1][7] \leftarrow 0$
1	8	0	all candidates equal $+\infty$: $r[1][8] \leftarrow 0$
1	9	0	all candidates equal $+\infty$: $r[1][9] \leftarrow 0$

Example: computation of the additional information (4/8)

t	k	$r[t][k]$	Action
2	0	0	first minimum at $x = 0$, i.e., $v[1][1]+f_2(0)+hc_2 \cdot 0 = 60+0+0 = 60$: $r[2][0] \leftarrow 0$
2	1	0	first minimum at $x = 0$, i.e., $v[1][2]+f_2(0)+hc_2 \cdot 1 = 70+0+10 = 80$: $r[2][1] \leftarrow 0$
2	2	3	first minimum at $x = 3$, i.e., $v[1][0]+f_2(3)+hc_2 \cdot 2 = 50+50+20 = 120$: $r[2][2] \leftarrow 3$
2	3	4	first minimum at $x = 4$, i.e., $v[1][0]+f_2(4)+hc_2 \cdot 3 = 50+50+30 = 130$: $r[2][3] \leftarrow 4$

Example: computation of the additional information (5/8)

t	k	$r[t][k]$	Action
2	4	4	first minimum at $x = 4$, i.e., $v[1][1]+f_2(4)+hc_2 \cdot 4 = 60+50+40 = 150$: $r[2][4] \leftarrow 4$
2	5	4	first minimum at $x = 4$, i.e., $v[1][2]+f_2(4)+hc_2 \cdot 5 = 70+50+50 = 170$: $r[2][5] \leftarrow 4$
2	6	0	all candidates equal $+\infty$: $r[2][6] \leftarrow 0$
2	7	0	all candidates equal $+\infty$: $r[2][7] \leftarrow 0$
2	8	0	all candidates equal $+\infty$: $r[2][8] \leftarrow 0$

Example: computation of the additional information (6/8)

t	k	$r[t][k]$	Action
3	0	3	first minimum at $x = 3$, i.e., $v[2][0]+f_3(3)+hc_3 \cdot 0 = 60+50+0 = 110$: $r[3][0] \leftarrow 3$
3	1	4	first minimum at $x = 4$, i.e., $v[2][0]+f_3(4)+hc_3 \cdot 1 = 60+50+10 = 120$: $r[3][1] \leftarrow 4$
3	2	4	first minimum at $x = 4$, i.e., $v[2][1]+f_3(4)+hc_3 \cdot 2 = 80+50+20 = 150$: $r[3][2] \leftarrow 4$
3	3	4	first minimum at $x = 4$, i.e., $v[2][2]+f_3(4)+hc_3 \cdot 3 = 120+50+30 = 200$: $r[3][3] \leftarrow 4$

Example: computation of the additional information (7/8)

t	k	$r[t][k]$	Action
3	4	4	first minimum at $x = 4$, i.e., $v[2][3]+f_3(4)+hc_3 \cdot 4 = 130+50+40 = 220$: $r[3][4] \leftarrow 4$
3	5	4	first minimum at $x = 4$, i.e., $v[2][4]+f_3(4)+hc_3 \cdot 5 = 150+50+50 = 250$: $r[3][5] \leftarrow 4$
4	0	4	first minimum at $x = 4$, i.e., $v[3][0]+f_4(4)+hc_4 \cdot 0 = 110+50+0 = 160$: $r[4][0] \leftarrow 4$
4	1	4	first minimum at $x = 4$, i.e., $v[3][1]+f_4(4)+hc_4 \cdot 1 = 120+50+10 = 180$: $r[4][1] \leftarrow 4$
5	0	0	first minimum at $x = 0$, i.e., $v[4][1]+f_5(0)+hc_5 \cdot 0 = 180+0+0 = 180$: $r[5][0] \leftarrow 0$

Example: computation of the additional information (8/8)

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

$r[t][k]$	0	1	2	3	4	5	6	7	8	9	10	11
$t = 0$	0	0	0	0	0	0	0	0	0	0	0	0
$t = 1$	2	3	4	0	0	0	0	0	0	0	·	·
$t = 2$	0	0	3	4	4	4	0	0	0	·	·	·
$t = 3$	3	4	4	4	4	4	·	·	·	·	·	·
$t = 4$	4	4	·	·	·	·	·	·	·	·	·	·
$t = 5$	0	·	·	·	·	·	·	·	·	·	·	·

Reconstructing an optimal solution

- Once the matrix r has been computed, an optimal solution of the CLS instance can be computed by applying the following rule for $t = m, m-1, \ldots, 1$, starting from $k = 0$:

$$x_t \leftarrow r[t][k], \qquad k \leftarrow k + d_t - x_t.$$

- This yields the vector $(x_1, x_2, \ldots, x_m)$, where each entry x_t is the number of units produced in period t in an optimal solution

- The following pseudo-code implements the computation of an optimal solution, given the matrix r

Algorithm: DP_CLS_SolComputation

Input : The number m of periods and the demands $d[1..m]$ of the CLS instance, and the matrix $r[0..m][0..D_{1,m}]$

Output: An optimal solution of the CLS instance

```
1 function DP_CLS_SolComputation(m, d, r)
2     x := new array[1..m]
3     k := 0
4     for t := m downto 1 do
5         x[t] ← r[t][k]
6         k ← k + d[t] − x[t]
7     end
8     return x
9 end
```

Example: computing an optimal solution (1/3)

After the execution of the DP algorithm on the example instance, we have the matrix r of the previous example. We can compute an optimal solution as follows:

Iteration	t	k	$r[t][k]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$k \leftarrow k + d[t] - x[t]$
–	–	–	–	0	0	0	0	0	$t = 5,\ k = 0$
1	5	0	0	0	0	0	0	0	$k = 0 + 1 - 0 = 1$
2	4	1	4	0	0	0	4	0	$k = 1 + 4 - 4 = 1$
3	3	1	4	0	0	4	4	0	$k = 1 + 3 - 4 = 0$
4	2	0	0	0	0	4	4	0	$k = 0 + 1 - 0 = 1$
5	1	1	3	3	0	4	4	0	$k = 1 + 2 - 3 = 0$

Example: computing an optimal solution (2/3)

An optimal solution is the vector:

$$(3, 0, 4, 4, 0)$$

The setups are in periods 1, 3, 4, the inventories are $(s_1, \ldots, s_5) = (1, 0, 1, 1, 0)$, and the total cost is (there is no variable production cost)

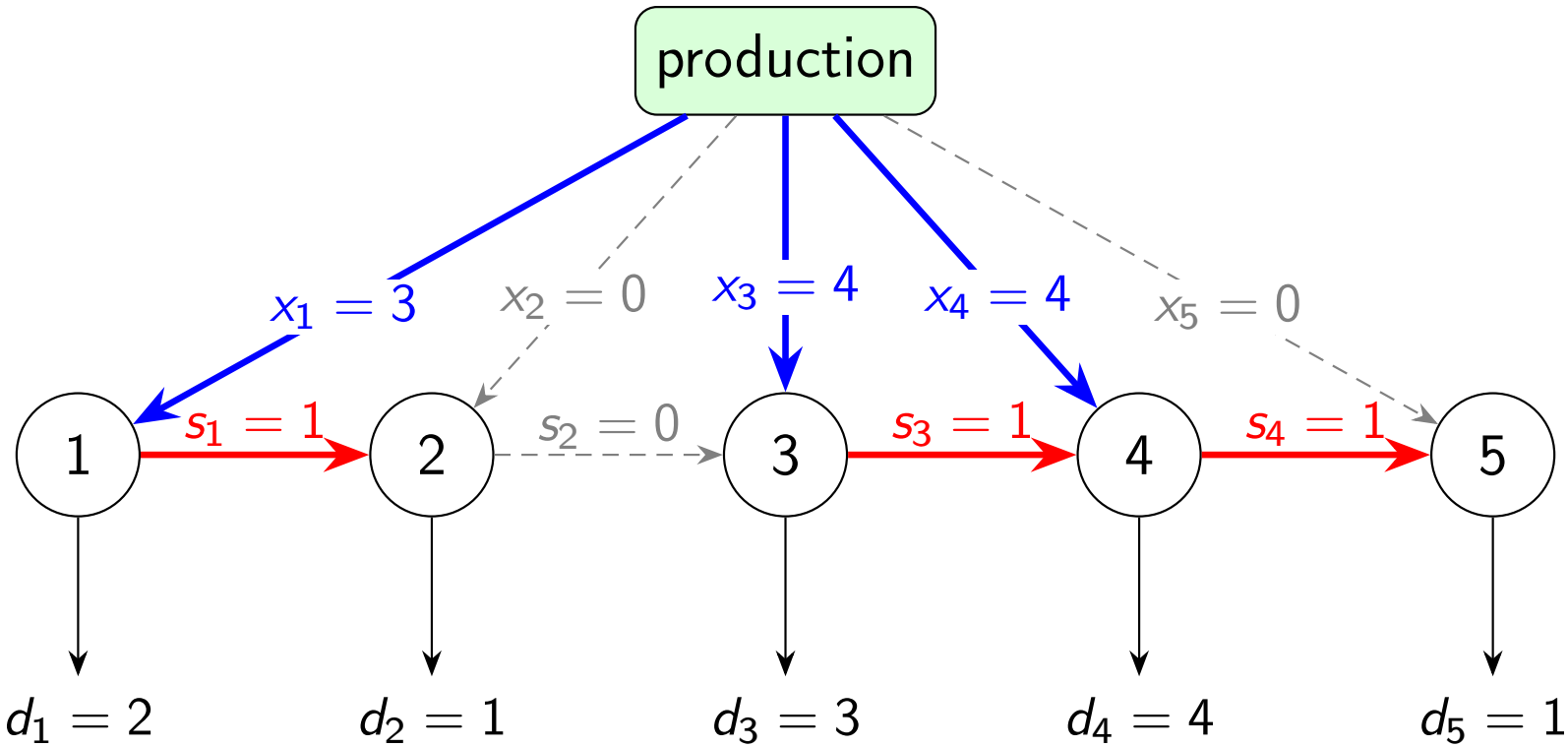
$$\underbrace{50 + 50 + 50}_{\text{setup}} + \underbrace{10 \cdot 1 + 10 \cdot 0 + 10 \cdot 1 + 10 \cdot 1 + 10 \cdot 0}_{\text{holding}}$$

$$= 150 + 30 = 180.$$

The corresponding optimal value is $v[5][0] = 180$, the same value given by Gurobi on the $\mathrm{ILP}_{\mathrm{CLSP}}$ model; the capacity costs $180 - 170$ with respect to the uncapacitated instance.

Example: computing an optimal solution (3/3)

The network of the solution is:



Period 4 produces although one unit is in stock at the end of period 3: the solution does not have the zero-inventory ordering property (and it is the unique optimal solution with integer quantities).

Number of steps

- Let $S(m, D, C)$ denote the number of steps executed by DP_CLS in the worst case, where:
  - m is the number of periods,
  - $D = D_{1,m}$ is the total demand,
  - $C = \max_{t \in T} cap_t$ is the largest capacity

- For every $t \geq 1$ we have $R[t] + 1 \leq D$ (since $d_1 \geq 1$), and the inner loop has at most $C + 1$ iterations

- In the worst case, the condition at line 16 is always true

Counting the executions (1/2)

The number of executions of the lines is:

- Line 2, Line 3, Line 7, and Line 8 are executed 1 time each.
- Line 4 and Line 5 are executed m times each.
- Line 9 and Line 10 are executed D times each.
- Line 12 is executed m times.

Counting the executions (2/2)

The number of executions of the lines is:

- Line 13, Line 14, and Line 20 are executed $\sum_{t=1}^{m}(R[t]+1) \leq mD$ times each.
- Line 15, Line 16, and Line 17 are executed at most $mD(C+1)$ times each.
- Line 23 is executed 1 time.

The total number of steps

Therefore,

$$\begin{aligned} S(m, D, C) &\leq 5 + 3m + 2D + 3mD + 3mD(C + 1) \\ &= 5 + 3m + 2D + 6mD + 3mDC, \end{aligned}$$

and hence

$$S(m, D, C) = O(mDC).$$

Number of steps of the DP algorithm

The DP algorithm executes

$$S(m, D, C) = O(m\,D\,C)$$

steps in the worst case.

- The number of steps depends on the values D and C, not only on the number m of periods: the DP algorithm is **pseudo-polynomial**
- DP_CLS_InfoSol only adds a constant number of operations inside the same loops, and therefore its number of steps has the same order of growth

Example: an instance with 6 periods

- Consider the capacitated instance with 6 periods, demands

$$(50, 30, 10, 10, 40, 10),$$

  setup cost 40, holding cost 1, and capacity 80
- Measured in units of 10, it has 35 states with $t \geq 1$

- The DP algorithm gives the optimal value 160, with the production plan

$$(50, 50, 0, 0, 50, 0)$$

  (in the original units): the same value given by Gurobi on the $\mathrm{ILP}_{\mathrm{CLSP}}$ model

General and constant capacities

- With general capacities cap_t the CLS is NP-hard, so a polynomial algorithm is not expected
- The pseudo-polynomial DP algorithm is a natural exact method for small values of D and C
- When the capacity is the same in all periods, $cap_t = C$ for all $t \in T$, the CLS can be solved in polynomial time, because the zero-inventory ordering property is replaced by a weaker one

A weaker property

If $cap_t = C$ for all $t \in T$, there is an optimal solution in which, between two consecutive periods with empty inventory, at most one period produces a quantity strictly between 0 and C; all the other periods produce 0 or C.

- The CLS with constant capacity has its own page in the catalogue of problems

Example: the property on the example instance

- The example instance has constant capacity $C = 4$
- Its optimal solution $(3, 0, 4, 4, 0)$ has inventories $(1, 0, 1, 1, 0)$, empty at the end of periods 0, 2, and 5
- In periods 1–2 only period 1 produces a quantity strictly between 0 and 4
- In periods 3–5 every period produces 0 or 4, as stated by the property

Summary (1/2)

Step 1: the optimal value:

- The state is (t, k), with $k \leq D_{t+1,m}$; the recurrence tries every production $x \in X_t(k)$ and adds $hc_t\, k$ outside the minimum
- Base cases $v(0, 0) = 0$ and $v(0, k) = +\infty$ for $k > 0$
- DP_CLS fills the matrix v row by row: on the example, $v[5][0] = 180$

Step 2: an optimal solution:

- $r[t][k]$ stores the chosen production (the smallest one in case of ties)
- Backwards from $(m, 0)$: $x_t \leftarrow r[t][k]$, $k \leftarrow k + d_t - x_t$; on the example, $(3, 0, 4, 4, 0)$

Summary (2/2)

Number of steps:

- $S(m, D, C) \leq 5 + 3m + 2D + 6mD + 3mDC$, hence $S(m, D, C) = O(mDC)$
- The DP algorithm is pseudo-polynomial

Constant capacity:

- With general capacities the CLS is NP-hard
- With $cap_t = C$, between two consecutive periods with empty inventory at most one period produces a quantity strictly between 0 and C, and the CLS is polynomial

Typical mistakes

Introduction: typical mistakes

- We list the mistakes that are made most often with the DP algorithm for the CLS
- They concern the zero-inventory ordering property, the base cases, the range of the inventories, the holding cost and the number of steps
- Knowing them helps to run the algorithm by hand correctly in the exercises

Observation 1: Typical mistakes (1/2)

- ***Applying the zero-inventory ordering property with capacities.*** *With capacities a period can produce while units are in stock: in the example instance the unique optimal plan violates the property.*
- ***Forgetting the base cases with value*** $+\infty$***.*** *The initial inventory is empty, so* $v(0, k) = +\infty$ *for* $k > 0$*; this value marks every state that cannot be reached within the capacities.*
- ***Allowing too large an inventory.*** *At the end of period* t *the inventory is at most* $D_{t+1,m}$*, since it must be empty at the end of period* m*: the other pairs* (t, k) *are not states.*

Observation: Typical mistakes (2/2)

- ***Putting the holding cost inside the choice.*** *The cost $hc_t\, k$ of the final inventory is paid in the state (t, k) whatever the production x: it is added outside the minimum.*
- ***Calling $O(mDC)$ polynomial.*** *The running time contains the total demand D and the largest capacity C, which are numbers of the input: the DP algorithm is pseudo-polynomial.*

Summary

Typical mistakes:

- With capacities the zero-inventory ordering property does not hold
- $v(0, k) = +\infty$ for $k > 0$ marks the states that cannot be reached
- At the end of period t the inventory is at most $D_{t+1,m}$
- The holding cost $hc_t\, k$ is added outside the minimum
- $O(mDC)$ is pseudo-polynomial, not polynomial

In summary

Introduction: in summary

- We collect the key ideas of the chapter on the DP algorithm for the CLS
- We close with the dashboard of the example instance: the bounds found in this chapter

Summary (1/2)

- With capacities the zero-inventory ordering property fails: a period may produce while units are in stock, and the DP algorithm over the production runs does not apply.
- The state is the pair (t, k) of the period and the final inventory, with $k \leq D_{t+1,m}$; the recurrence $v(t, k)$ chooses the production x of period t, and the optimal value is $v(m, 0)$.
- The matrix r of the chosen productions, read backwards from the state $(m, 0)$, gives an optimal plan.
- The DP algorithm takes $O(mDC)$ steps, which is pseudo-polynomial.
- With a constant capacity a weaker property holds, and the problem is polynomial; with general capacities it is NP-hard.

Summary (2/2)

Dashboard of the example.

Instance	Lower bound	Upper bound	Origin
The example instance (capacitated, $C = 4$)	180	180	DP algorithm on (period, inventory)