DP for the uncapacitated lot sizing problem

Outline

Why lot sizing?

Notation, model and feasible solutions
- Encoding of feasible solutions
- The zero-inventory ordering property

A DP algorithm for the ULS
- The cost of a production run
- Step 1 and Step 2
- Number of steps

Typical mistakes

In summary

Learning objectives

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

- state the zero-inventory ordering property, and prove it with the exchange argument;
- compute the cost $c_{j,t}$ of a production run, and the matrix of the costs, in $O(m^2)$ steps;
- write the recurrence $v(t)$ over the last production run, run the DP algorithm by hand and reconstruct an optimal plan;
- read the recurrence as a shortest path in the network of the production runs;
- count the steps of the DP algorithm, and show that they are $\Theta(m^2)$.

Prerequisites

- From the module *Combinatorial Optimization Problems*, the chapters «Lot sizing problems» and «Formulations of lot sizing problems»: the single-item problem, its reference instance, and its ILP model with the variables x_t, s_t, y_t.
- From this module, the chapter «Principles of dynamic programming» and the DP algorithms of the previous chapters: 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**: single-item lot sizing and its ILP formulations (*Combinatorial Optimization Problems*, chapters «Lot sizing problems» and «Formulations of lot sizing problems»).
- **Now, as a table**: the DP algorithm of Wagner and Whitin gives the optimal value of the uncapacitated problem in $\Theta(m^2)$ steps.

Why lot sizing?

Introduction: why lot sizing?

- A company must decide **when** to produce and **how much**, period after period
- Every production has a fixed cost, and every unit kept in stock has a cost
- In this chapter we solve the uncapacitated version of this problem exactly, with a DP algorithm

Planning the production of one product

- A company plans the production of **one product** over a horizon of m time periods (weeks, months, ...).
- The demand of each period must be satisfied, either by the production of the period or by the inventory kept from the previous periods.

Setups against inventories

- Producing in a period requires a **setup**, which has a fixed cost, independent of the produced quantity.
- Each produced unit has a variable cost.
- Each unit kept in stock at the end of a period has a holding cost.

Producing rarely saves setup costs
but increases the holding costs:

the problem is to find the **best compromise**.

The problem of this chapter

The problem of this chapter is the
uncapacitated lot sizing problem (ULS):

its definition and its assumptions are on its page.

- We write the DP algorithm of Wagner and Whitin, which gives the optimal value of a ULS instance
- The same algorithm also gives an optimal production plan

Summary

The problem:

- one product, m periods, a demand to satisfy in each period
- production now or inventory from the previous periods

The costs:

- a fixed setup cost in every period with production
- a variable cost per unit produced and a holding cost per unit in stock
- few setups mean large inventories: the best compromise is the optimal plan

Notation, model and feasible solutions

Introduction: notation, model and feasible solutions

- In this section we fix the notation and we recall the ILP model of the ULS
- We describe a feasible solution by the vector of the produced quantities
- We prove the zero-inventory ordering property: it shows that only 2^{m-1} solutions need to be considered
- This property is the basis of the DP algorithm of the next section

The total demand of consecutive periods

- For any $j, t \in T$, we denote by

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

the total demand of periods $j, j+1, \ldots, t$, with $D_{j,t} = 0$ if $j > t$.

- For example, $D_{1,m}$ is the total demand of the whole horizon, and $D_{t,m}$ is the total demand of the remaining periods $t, t+1, \ldots, m$.

The ILP model of the ULS

- The ILP model is the $\mathrm{ILP}_{\mathrm{ULS}}$ model of the chapter «Formulations of lot sizing problems», section «ILP models for the variants».
- It is the $\mathrm{ILP}_{\mathrm{CLSP}}$ model without the capacity constraints and with the big-M value $D_{t,m}$ of period t, the total demand of the remaining periods.
- Its variables are, for every period $t \in T$:
  - the production x_t,
  - the inventory s_t at the end of period t,
  - the setup $y_t \in \{0, 1\}$.

The $\mathrm{ILP}_{\mathrm{ULS}}$ model

We recall the $\mathrm{ILP}_{\mathrm{ULS}}$ model of the uncapacitated lot sizing problem (ULS):

$$\min \quad \sum_{t \in T} (sc_t\, y_t + vc_t\, x_t + hc_t\, s_t)$$

$$\begin{aligned}
\text{subject to} \quad & s_{t-1} + x_t = d_t + s_t, && \forall t \in T, \\
& x_t \leq D_{t,m}\, y_t, && \forall t \in T, \\
& s_0 = s_m = 0, && \\
& x_t \in \mathbb{Z}_{\geq 0}, && \forall t \in T, \\
& s_t \in \mathbb{Z}_{\geq 0}, && \forall t \in T, \\
& y_t \in \{0, 1\}, && \forall t \in T.
\end{aligned}$$

This ILP model is referred to as the $\mathrm{ILP}_{\mathrm{ULS}}$ model.

Description of the $\mathrm{ILP}_{\mathrm{ULS}}$ model

Description of the objective function and constraints of the $\mathrm{ILP}_{\mathrm{ULS}}$ model:

- the objective function minimizes the total cost of setups, production and holding;
- the constraints $s_{t-1} + x_t = d_t + s_t$ are the flow balancing constraints;
- the constraints $x_t \leq D_{t,m}\, y_t$ force a setup in every period with positive production;
- the constraint $s_0 = s_m = 0$ imposes an empty inventory at the beginning and at the end of the horizon;
- finally, the last three rows define the variables of the ILP model.

Example: ULS instance

We consider $m = 5$ periods with the following data:

t	1	2	3	4	5
d_t	20	10	30	40	10
sc_t	50	50	50	50	50
vc_t	0	0	0	0	0
hc_t	1	1	1	1	1

- We use this ULS instance as the example of the whole chapter.
- It is the reference instance of the chapter «Lot sizing problems» of *Combinatorial Optimization Problems*, counted in single items instead of units of ten and without the capacities (setup cost 50, holding cost 1, no variable production cost).

Feasible solutions of the ULS

A *feasible solution* of the ULS is a production plan that satisfies the demand of every period, starting and ending with an empty inventory.

- We now describe a feasible solution by a vector, and we compute its inventories and its total cost from this vector.

Encoding of feasible solutions (1/2)

A ULS feasible solution can be described by a solution vector

$$(x_1, x_2, \ldots, x_m)$$

where each component has the following meaning:

$$x_t = \text{number of units produced in period } t, \quad \forall t \in \{1, 2, \ldots, m\}.$$

The inventory at the end of period t is determined by the vector:

$$s_t = \sum_{j=1}^{t} x_j - D_{1,t}, \qquad \forall t \in \{1, 2, \ldots, m\}.$$

Encoding of feasible solutions (2/2)

- Since the solution is feasible, the inventories are non-negative and the last one is empty:

$$s_t \geq 0 \quad \forall t \in \{1, 2, \ldots, m-1\}, \qquad s_m = 0.$$

- A setup is paid in period t if and only if $x_t > 0$.

- The total cost of a feasible solution is

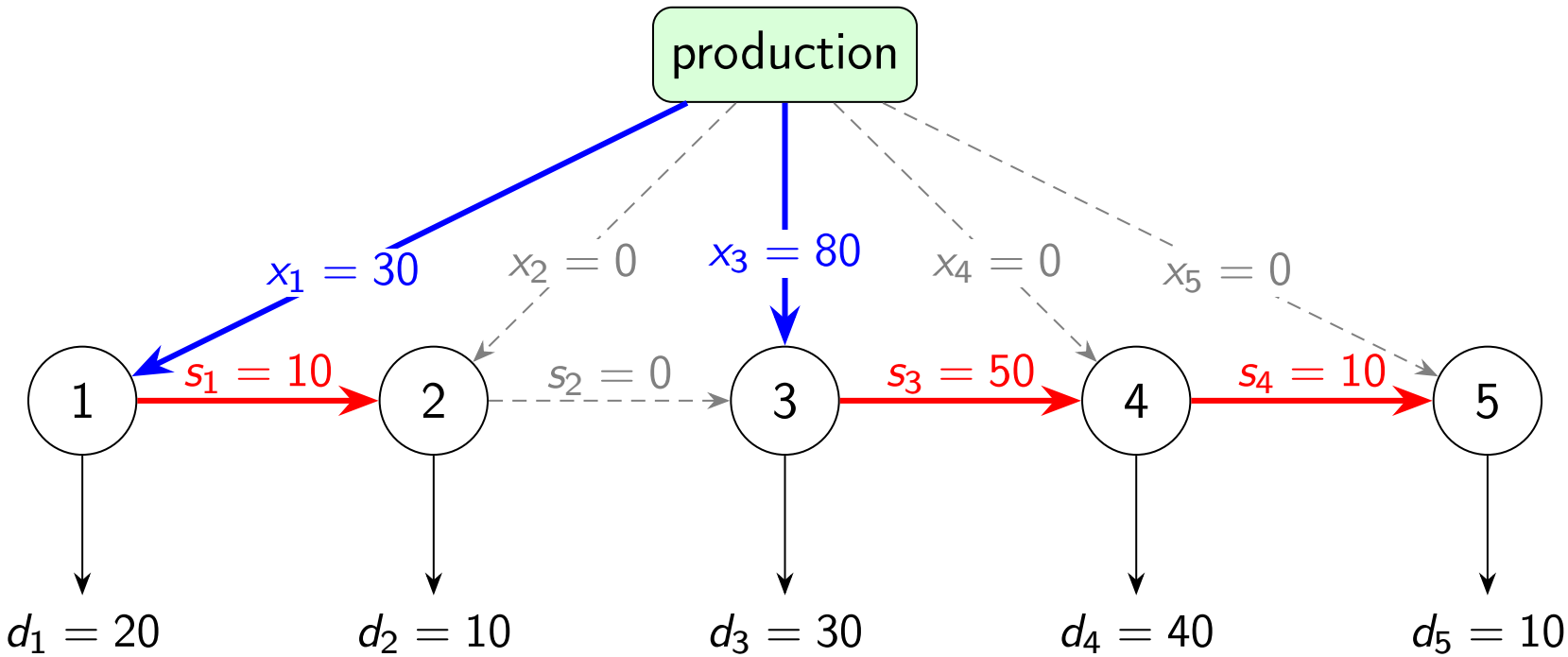
$$\sum_{t=1,\ x_t>0}^{m} sc_t + \sum_{t=1}^{m} vc_t\, x_t + \sum_{t=1}^{m} hc_t\, s_t.$$

Example: a feasible solution (1/3)

- We consider the ULS instance of the example above.
- The next network represents the feasible solution $(30, 0, 80, 0, 0)$:
  - each period is a node,
  - the production x_t enters node t, and the demand d_t leaves it,
  - the inventory s_t flows from node t to node $t + 1$.
- Thick arcs carry a positive flow.

Example: a feasible solution (2/3)

The network of the feasible solution $(30, 0, 80, 0, 0)$:



Example: a feasible solution (3/3)

- The setups are in periods 1, 3.
- The inventories are $(s_1, \ldots, s_5) = (10, 0, 50, 10, 0)$.
- The total cost is (there is no variable production cost)

$$\underbrace{50 + 50}_{\text{setup}} + \underbrace{1 \cdot 10 + 1 \cdot 0 + 1 \cdot 50 + 1 \cdot 10 + 1 \cdot 0}_{\text{holding}} = 100 + 70 = 170.$$

How many feasible solutions?

- A production quantity can take many values, and a ULS instance has a huge number of feasible solutions (infinitely many, if the quantities are not required to be integer).
- The next property shows that only very few of them need to be considered.

The zero-inventory ordering property

Theorem 1: Zero-inventory ordering (Wagner and Whitin, 1958)

Every ULS instance has an optimal solution in which no period with positive production starts with a positive inventory, that is,

$$s_{t-1}\, x_t = 0 \qquad \forall t \in \{1, 2, \ldots, m\}.$$

- In words: a period produces only when its initial inventory is empty.

The idea of the proof

- The proof is by contradiction, with an **exchange argument**.
- We take an optimal solution with the smallest number of periods with positive production.
- If a period produces while the inventory is positive, we move units of production between this period and the previous period with production.
- Either the cost decreases, or one setup disappears: both contradict the choice of the solution.

Proof (1/4)

- First, an optimal solution exists: the vector $(d_1, d_2, \ldots, d_m)$ is feasible, and for each of the 2^m choices of the set of periods with a setup, the remaining problem is a linear program over a bounded polyhedron (since $s_m = 0$, every x_t is between 0 and $D_{1,m}$), so it has an optimal solution whenever it is feasible.

- Among all optimal solutions, consider one with the **smallest number of periods with positive production**, and suppose, by contradiction, that there exists a period t with $x_t > 0$ and $s_{t-1} > 0$.

Proof (2/4)

- Since $s_0 = 0$ and $s_{t-1} > 0$, something is produced before period t: let $j < t$ be the last period before t with $x_j > 0$.
- No unit is produced in periods $j+1, \ldots, t-1$, so the inventory can only decrease from period j to period $t-1$:

$$s_k \geq s_{t-1} > 0 \qquad \forall k \in \{j, j+1, \ldots, t-1\}.$$

- For a number ε, move ε units of production from period j to period t: replace x_j by $x_j - \varepsilon$, x_t by $x_t + \varepsilon$, and s_k by $s_k - \varepsilon$ for all $k \in \{j, \ldots, t-1\}$.
- All balancing constraints remain satisfied, and the new plan is feasible for every ε with

$$-x_t \leq \varepsilon \leq \min\{x_j,\ s_{t-1}\}.$$

Proof (3/4)

- As long as x_j and x_t stay positive, the setups do not change and the cost changes by $\varepsilon\,\Delta$, where

$$\Delta = vc_t - vc_j - \sum_{k=j}^{t-1} hc_k$$

  is the cost of producing one unit in period t minus the cost of producing it in period j and holding it until period t.

- If $\Delta < 0$, a small $\varepsilon > 0$ gives a feasible solution with a smaller cost, which contradicts optimality.

Proof (4/4)

- If $\Delta \geq 0$, choose $\varepsilon = -x_t$: the whole production of period t is moved to period j, the cost changes by $-x_t \, \Delta \leq 0$, and the setup cost $sc_t \geq 0$ of period t is saved.
- The new solution is optimal and has one period with positive production less, which contradicts the choice of the solution.
- Hence no such period t exists.

□

Each production covers consecutive demands

In an optimal solution with the zero-inventory ordering property, each period j with positive production produces exactly the demands of periods $j, j+1, \ldots, t$, where $t+1$ is the next period with positive production (or $t = m$ if there is none):

$$x_j = D_{j,t}.$$

- Indeed, $s_{j-1} = 0$ (zero-inventory ordering in period j), $s_t = 0$ (zero-inventory ordering in period $t+1$, or $s_m = 0$), and nothing is produced in periods $j+1, \ldots, t$.
- Summing the balancing constraints of periods $j, \ldots, t$ gives $x_j = D_{j,t}$.

Solutions with the zero-inventory ordering property

- Period 1 always produces, since $s_0 = 0$ and $d_1 \geq 1$.
- A solution with the zero-inventory ordering property is therefore determined by the set of its production periods, which contains period 1 and any subset of $\{2, 3, \ldots, m\}$.

The number of ULS solutions with the zero-inventory ordering property is 2^{m-1},
since each period $t \in \{2, 3, \ldots, m\}$ can independently produce or not.

An exponential number of solutions

- The number of these solutions grows exponentially with the number m of periods:

$$2^{m-1} = O(2^m).$$

- Therefore, enumerating them quickly becomes impractical as m increases.

Next: the enumeration on the example,
which has only $m = 5$ periods.

Example: enumeration of the solutions (1/6)

We consider the ULS instance of the example above:

t	1	2	3	4	5
d_t	20	10	30	40	10
sc_t	50	50	50	50	50
vc_t	0	0	0	0	0
hc_t	1	1	1	1	1

- There are $2^{5-1} = 16$ solutions with the zero-inventory ordering property.
- For each of them, the next tables report the production periods, the solution vector, the inventories, and the total cost, written as setup cost plus holding cost.
- The optimal costs are highlighted.

Example: enumeration of the solutions (2/6)

Solutions (1)–(8): production periods and solution vectors.

Label	Production periods	Solution vector
(1)	$\{1\}$	$(110, 0, 0, 0, 0)$
(2)	$\{1, 5\}$	$(100, 0, 0, 0, 10)$
(3)	$\{1, 4\}$	$(60, 0, 0, 50, 0)$
(4)	$\{1, 4, 5\}$	$(60, 0, 0, 40, 10)$
(5)	$\{1, 3\}$	$(30, 0, 80, 0, 0)$
(6)	$\{1, 3, 5\}$	$(30, 0, 70, 0, 10)$
(7)	$\{1, 3, 4\}$	$(30, 0, 30, 50, 0)$
(8)	$\{1, 3, 4, 5\}$	$(30, 0, 30, 40, 10)$

Example: enumeration of the solutions (3/6)

Solutions (1)–(8): inventories and total costs.

Label	Production periods	Inventories $(s_1, \ldots, s_5)$	Total cost
(1)	$\{1\}$	$(90, 80, 50, 10, 0)$	$50 + 230 = 280$
(2)	$\{1, 5\}$	$(80, 70, 40, 0, 0)$	$100 + 190 = 290$
(3)	$\{1, 4\}$	$(40, 30, 0, 10, 0)$	$100 + 80 = 180$
(4)	$\{1, 4, 5\}$	$(40, 30, 0, 0, 0)$	$150 + 70 = 220$
(5)	$\{1, 3\}$	$(10, 0, 50, 10, 0)$	$100 + 70 = 170$
(6)	$\{1, 3, 5\}$	$(10, 0, 40, 0, 0)$	$150 + 50 = 200$
(7)	$\{1, 3, 4\}$	$(10, 0, 0, 10, 0)$	$150 + 20 = 170$
(8)	$\{1, 3, 4, 5\}$	$(10, 0, 0, 0, 0)$	$200 + 10 = 210$

Example: enumeration of the solutions (4/6)

Solutions (9)–(16): production periods and solution vectors.

Label	Production periods	Solution vector
(9)	$\{1, 2\}$	$(20, 90, 0, 0, 0)$
(10)	$\{1, 2, 5\}$	$(20, 80, 0, 0, 10)$
(11)	$\{1, 2, 4\}$	$(20, 40, 0, 50, 0)$
(12)	$\{1, 2, 4, 5\}$	$(20, 40, 0, 40, 10)$
(13)	$\{1, 2, 3\}$	$(20, 10, 80, 0, 0)$
(14)	$\{1, 2, 3, 5\}$	$(20, 10, 70, 0, 10)$
(15)	$\{1, 2, 3, 4\}$	$(20, 10, 30, 50, 0)$
(16)	$\{1, 2, 3, 4, 5\}$	$(20, 10, 30, 40, 10)$

Example: enumeration of the solutions (5/6)

Solutions (9)–(16): inventories and total costs.

Label	Production periods	Inventories $(s_1, \ldots, s_5)$	Total cost
(9)	$\{1, 2\}$	$(0, 80, 50, 10, 0)$	$100 + 140 = 240$
(10)	$\{1, 2, 5\}$	$(0, 70, 40, 0, 0)$	$150 + 110 = 260$
(11)	$\{1, 2, 4\}$	$(0, 30, 0, 10, 0)$	$150 + 40 = 190$
(12)	$\{1, 2, 4, 5\}$	$(0, 30, 0, 0, 0)$	$200 + 30 = 230$
(13)	$\{1, 2, 3\}$	$(0, 0, 50, 10, 0)$	$150 + 60 = 210$
(14)	$\{1, 2, 3, 5\}$	$(0, 0, 40, 0, 0)$	$200 + 40 = 240$
(15)	$\{1, 2, 3, 4\}$	$(0, 0, 0, 10, 0)$	$200 + 10 = 210$
(16)	$\{1, 2, 3, 4, 5\}$	$(0, 0, 0, 0, 0)$	$250 + 0 = 250$

Example: enumeration of the solutions (6/6)

The optimal value of the ULS instance is 170,

attained by 2 solutions: (5) and (7).

- Solution (5) is $(30, 0, 80, 0, 0)$, with production periods $\{1, 3\}$.
- Solution (7) is $(30, 0, 30, 50, 0)$, with production periods $\{1, 3, 4\}$.
- With m periods the enumeration lists 2^{m-1} solutions: the next section computes the optimal value without listing them.

Summary

Notation and model:

- $D_{j,t}$ is the total demand of periods $j, \ldots, t$
- the $\mathrm{ILP}_{\mathrm{ULS}}$ model: balancing constraints, setup constraints with big-M $D_{t,m}$, empty inventory at the beginning and at the end

Feasible solutions:

- a solution is the vector $(x_1, \ldots, x_m)$; the inventories s_t and the cost follow from it
- zero-inventory ordering: some optimal solution has $s_{t-1}\, x_t = 0$ for every period t
- then each production is $x_j = D_{j,t}$, and only 2^{m-1} solutions remain
- on the example: optimal value 170, attained by 2 solutions

A DP algorithm for the ULS

Introduction: a DP algorithm for the ULS

- In this section we write the DP algorithm of Wagner and Whitin for the ULS
- First we compute the cost of every production run, then the recurrence chooses the last run
- We run the algorithm by hand on the example, and we reconstruct an optimal plan
- Finally, we count its steps: they are $\Theta(m^2)$, instead of the 2^{m-1} solutions of the enumeration

Production runs

- By the zero-inventory ordering property, an optimal solution is a sequence of **production runs**:
  - a run produces in a period j the demands of periods $j, \ldots, t$,
  - the next run starts in period $t+1$ with an empty inventory.
- For any $j, t \in T$ with $j \leq t$, let $c_{j,t}$ be the cost of the production run that produces in period j the demands of periods $j, \ldots, t$.

The cost of a production run

The cost of the production run of periods $j, \ldots, t$ is

$$c_{j,t} = sc_j + vc_j\, D_{j,t} + \sum_{k=j}^{t-1} hc_k\, D_{k+1,t}.$$

- The first term is the setup cost of period j.
- The second term is the variable cost of the $D_{j,t}$ produced units.
- The last term is the holding cost: at the end of each period $k \in \{j, \ldots, t-1\}$ the inventory contains the demands $D_{k+1,t}$ of the following periods of the run.

Computing the costs one after the other

- For a fixed period j, the costs $c_{j,j}, c_{j,j+1}, \ldots, c_{j,m}$ can be computed one after the other.
- A unit used in period t and produced in period j costs

$$u_{j,t} = vc_j + \sum_{k=j}^{t-1} hc_k,$$

  so that $u_{j,j} = vc_j$ and $u_{j,t} = u_{j,t-1} + hc_{t-1}$.
- Adding period t to the run adds d_t units of cost $u_{j,t}$:

$$c_{j,j} = sc_j + vc_j\, d_j, \qquad c_{j,t} = c_{j,t-1} + u_{j,t}\, d_t \quad \text{if } t > j.$$

Next: the pseudo-code that computes the matrix of the production-run costs.

Algorithm: CostMatrix

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

Output : The matrix of the production-run costs

```
1  function CostMatrix(m, d, sc, vc, hc)
2      c := new array[1..m][1..m]
3      for j := 1 to m do
4          u := vc[j]
5          g := sc[j] + vc[j] · d[j]
6          c[j][j] ← g
7          for t := j + 1 to m do
8              u ← u + hc[t − 1]
9              g ← g + u · d[t]
10             c[j][t] ← g
11         end
12     end
13     return c
14 end
```


Sapienza Università di Roma

How CostMatrix works

- The variable u is the unit cost $u_{j,t}$.
- The variable g is the cost $c_{j,t}$ of the current run.
- Only the cells $c[j][t]$ with $j \leq t$ are used.

Example: computation of the production-run costs (1/6)

We consider the ULS instance of the example above:

t	1	2	3	4	5
d_t	20	10	30	40	10
sc_t	50	50	50	50	50
vc_t	0	0	0	0	0
hc_t	1	1	1	1	1

- For each step, the next tables show the values of j and t, the current value of u, the cell $c[j][t]$, and the operation performed.
- The cell $c[j][t]$ is highlighted in blue when assigned.

Example: computation of the production-run costs (2/6)

j	t	u	$c[j][t]$	Action
1	1	0	50	$u \leftarrow vc[1] = 0$, $g \leftarrow sc[1]+vc[1] \cdot d[1] = 50+0 \cdot 20 = 50$, $c[1][1] \leftarrow 50$
1	2	1	60	$u \leftarrow u+hc[1] = 0+1 = 1$, $g \leftarrow g+u \cdot d[2] = 50+1 \cdot 10 = 60$, $c[1][2] \leftarrow 60$
1	3	2	120	$u \leftarrow u+hc[2] = 1+1 = 2$, $g \leftarrow g+u \cdot d[3] = 60+2 \cdot 30 = 120$, $c[1][3] \leftarrow 120$
1	4	3	240	$u \leftarrow u+hc[3] = 2+1 = 3$, $g \leftarrow g+u \cdot d[4] = 120+3 \cdot 40 = 240$, $c[1][4] \leftarrow 240$
1	5	4	280	$u \leftarrow u+hc[4] = 3+1 = 4$, $g \leftarrow g+u \cdot d[5] = 240+4 \cdot 10 = 280$, $c[1][5] \leftarrow 280$

Example: computation of the production-run costs (3/6)

j	t	u	$c[j][t]$	Action
2	2	0	50	$u \leftarrow vc[2] = 0$, $g \leftarrow sc[2]+vc[2] \cdot d[2] = 50+0 \cdot 10 = 50$, $c[2][2] \leftarrow 50$
2	3	1	80	$u \leftarrow u+hc[2] = 0+1 = 1$, $g \leftarrow g+u \cdot d[3] = 50+1 \cdot 30 = 80$, $c[2][3] \leftarrow 80$
2	4	2	160	$u \leftarrow u+hc[3] = 1+1 = 2$, $g \leftarrow g+u \cdot d[4] = 80+2 \cdot 40 = 160$, $c[2][4] \leftarrow 160$
2	5	3	190	$u \leftarrow u+hc[4] = 2+1 = 3$, $g \leftarrow g+u \cdot d[5] = 160+3 \cdot 10 = 190$, $c[2][5] \leftarrow 190$

Example: computation of the production-run costs (4/6)

j	t	u	$c[j][t]$	Action
3	3	0	50	$u \leftarrow vc[3] = 0$, $g \leftarrow sc[3]+vc[3] \cdot d[3] = 50+0 \cdot 30 = 50$, $c[3][3] \leftarrow 50$
3	4	1	90	$u \leftarrow u+hc[3] = 0+1 = 1$, $g \leftarrow g+u \cdot d[4] = 50+1 \cdot 40 = 90$, $c[3][4] \leftarrow 90$
3	5	2	110	$u \leftarrow u+hc[4] = 1+1 = 2$, $g \leftarrow g+u \cdot d[5] = 90+2 \cdot 10 = 110$, $c[3][5] \leftarrow 110$

Example: computation of the production-run costs (5/6)

j	t	u	$c[j][t]$	Action
4	4	0	50	$u \leftarrow vc[4] = 0$, $g \leftarrow sc[4]+vc[4] \cdot d[4] = 50+0 \cdot 40 = 50$, $c[4][4] \leftarrow 50$
4	5	1	60	$u \leftarrow u+hc[4] = 0+1 = 1$, $g \leftarrow g+u \cdot d[5] = 50+1 \cdot 10 = 60$, $c[4][5] \leftarrow 60$
5	5	0	50	$u \leftarrow vc[5] = 0$, $g \leftarrow sc[5]+vc[5] \cdot d[5] = 50+0 \cdot 10 = 50$, $c[5][5] \leftarrow 50$

Example: computation of the production-run costs (6/6)

This is the matrix c after the execution of the algorithm:

$c[j][t]$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$j=1$	50	60	120	240	280
$j=2$	–	50	80	160	190
$j=3$	–	–	50	90	110
$j=4$	–	–	–	50	60
$j=5$	–	–	–	–	50

For instance, by the formula of the cost of a production run:

$$\begin{aligned} c_{1,3} &= sc_1 + vc_1\, D_{1,3} + hc_1\, D_{2,3} + hc_2\, D_{3,3} \\ &= 50 + 0 \cdot 60 + 1 \cdot 40 + 1 \cdot 30 = 120. \end{aligned}$$

The two steps of the DP algorithm

- Step 1: computation of the optimal value
  - a recursive formula gives the optimal value $v(t)$ of the first t periods, by choosing the last production run
- Step 2: computation of an optimal solution
  - the period of the chosen last run is stored, and an optimal plan is read backwards

Step 1: the subinstances

For any $t \in \{1, 2, \ldots, m\}$, let $v(t)$ be the optimal value of the ULS 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.

Step 1: the recursive formula

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

$$v(t) = \begin{cases} \min\left\{ v(j-1) + c_{j,t} : j \in \{1, 2, \ldots, t\} \right\} & \text{if } 1 \leq t \leq m, \\ 0 & \text{if } t = 0. \end{cases}$$

Reading the recursive formula

- The subinstance with periods $\{1, \ldots, t\}$ is a ULS instance of the same type: its inventory must be empty at the end of period t.
- Here, $v(0) = 0$, since an empty set of periods has no demand and no cost.
- The recursive decomposition process of the formula generates t **independent subproblems**, each corresponding to a possible choice of the period j of the **last production run**, which produces the demands of periods $j, \ldots, t$.

Optimal substructure

- By the zero-inventory ordering property applied to the subinstance, the inventory is empty at the end of period $j-1$, and the residual subproblem is the ULS subinstance with periods $\{1,\ldots,j-1\}$.
- The optimal value $v(t)$ is obtained by trying all possible periods of the last production run and selecting the one that minimizes the sum $v(j-1)+c_{j,t}$ of the optimal value of the residual subproblem and the cost of the last run.
- Hence, an optimal solution for periods $\{1,\ldots,t\}$ is built by combining the chosen last production run in period j with an optimal solution of the residual subproblem with periods $\{1,\ldots,j-1\}$.

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

Overlapping subproblems

- Different choices of the last production run, possibly made for different values of t, can lead to the same residual set of periods.
- Therefore the same value $v(t)$ may need to be computed several times during the recursive evaluation of the formula.

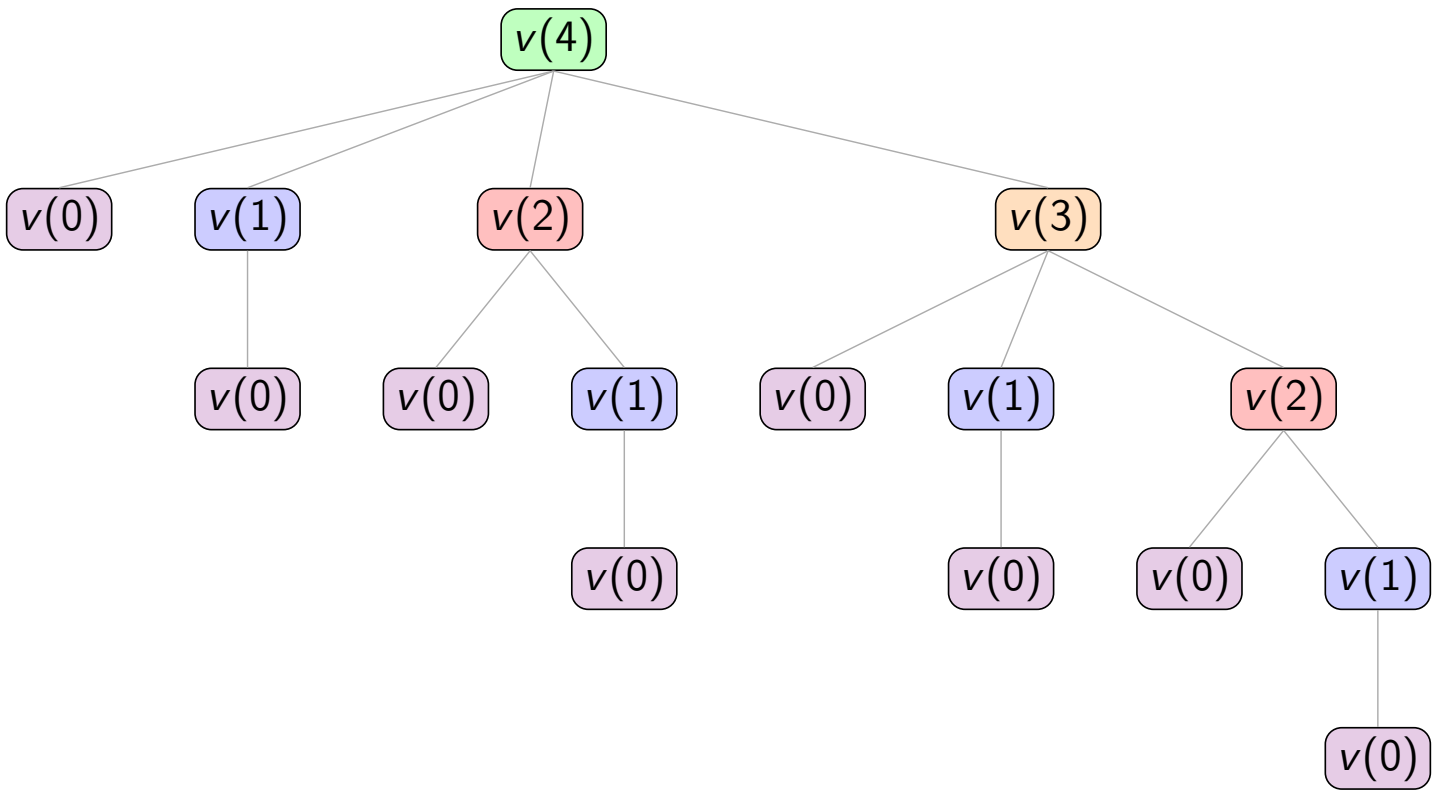
$\rightsquigarrow$ **Overlapping Subproblems verified!**

The top-down approach

- A **first method** to compute the optimal value $v(m)$ of the ULS instance is the **top-down approach**.
- The method starts from the call with $t = m$ and evaluates the formula recursively: each call for periods $\{1, \ldots, t\}$ triggers further calls for periods $\{1, \ldots, j-1\}$, for all $j \in \{1, 2, \ldots, t\}$, until the base case $v(0)$ is reached.
- The **recursion tree** of the function v has one node for each recursive call, labeled with the function call it represents:
  - the root represents the initial call;
  - the children of a node represent the recursive calls made by that call;
  - the leaf nodes represent the base cases.

Example: recursion tree (1/2)

We consider the first four periods of the ULS instance of the example (with five periods the tree has 32 nodes). This is the recursion tree of the top-down approach to compute $v(4)$:



Example: recursion tree (2/2)

- The children of a node $v(t)$ are ordered by $j \in \{1, \ldots, t\}$: the child $v(j-1)$ corresponds to a last production run in period j.
- The function v is called 16 times, and the same subproblems are solved multiple times:
  - $v(2)$ is called 2 times,
  - $v(1)$ is called 4 times,
  - $v(0)$ is called 8 times.
- All the other calls are made only once.

Computing the values in increasing order

The value $v(t)$, for a generic period t,
can be computed once the values $v(j)$ are known
for all smaller periods $j < t$.

- Next: we compute $v(0), v(1), \ldots, v(5)$ in this order on the example.

Example: computing the values of the v function (1/3)

We consider the ULS instance of the example, with the matrix c of the production-run costs:

$c[j][t]$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$j=1$	50	60	120	240	280
$j=2$	–	50	80	160	190
$j=3$	–	–	50	90	110
$j=4$	–	–	–	50	60
$j=5$	–	–	–	–	50

The value of the function v can be computed by starting from the base case $v(0)=0$ and then computing the values of $v(1)$, $v(2)$, $v(3)$, $v(4)$, and $v(5)$ in order.

Example: computing the values of the v function (2/3)

- Since we know the value of $v(0)$, we can compute $v(1)$, as follows:

$$v(1) = v(0) + c_{1,1} = 0 + 50 = 50$$

- Once we have computed $v(1)$ and $v(0)$, we can compute $v(2)$, as follows:

$$\begin{aligned} v(2) &= \min\{v(0) + c_{1,2}, v(1) + c_{2,2}\} \\ &= \min\{0 + 60, 50 + 50\} = 60 \end{aligned}$$

- Once we have computed $v(2)$, $v(1)$ and $v(0)$, we can compute $v(3)$, as follows:

$$\begin{aligned} v(3) &= \min\{v(0) + c_{1,3}, v(1) + c_{2,3}, v(2) + c_{3,3}\} \\ &= \min\{0 + 120, 50 + 80, 60 + 50\} = 110 \end{aligned}$$

Example: computing the values of the v function (3/3)

- Once we have computed $v(3)$, $v(2)$, $v(1)$ and $v(0)$, we can compute $v(4)$, as follows:

$$\begin{aligned} v(4) &= \min\{v(0) + c_{1,4}, v(1) + c_{2,4}, v(2) + c_{3,4}, v(3) + c_{4,4}\} \\ &= \min\{0 + 240, 50 + 160, 60 + 90, 110 + 50\} = 150 \end{aligned}$$

- Once we have computed $v(4)$, $v(3)$, $v(2)$, $v(1)$ and $v(0)$, we can compute $v(5)$, as follows:

$$\begin{aligned} v(5) = \min\{&v(0) + c_{1,5}, v(1) + c_{2,5}, v(2) + c_{3,5}, \\ &v(3) + c_{4,5}, v(4) + c_{5,5}\} \\ = \min\{&0 + 280, 50 + 190, 60 + 110, \\ &110 + 60, 150 + 50\} = 170 \end{aligned}$$

The lookup table

- We can avoid solving the same subproblems repeatedly by storing the value of each subproblem as soon as it is computed.
- The values are stored in a **lookup table**, the vector $v[0..m]$, whose cell $v[t]$ contains $v(t)$.
- Since solving a subproblem with value $v(t)$ depends only on subproblems with values $v(j-1)$, with $j-1 < t$, we can solve the subproblems for increasing values of t, starting from the base case $v(0) = 0$ and building up to $v(m)$.
- This observation naturally leads to a **much more efficient method** to compute the optimal value $v(m)$ of the ULS instance.

The bottom-up approach

- A **second method** to compute the optimal value $v(m)$ of the ULS instance is the **bottom-up approach**.
- The method starts from the base case $v(0) = 0$ and evaluates the formula for increasing periods $t \in \{1, 2, \ldots, m\}$.
- Each value $v(t)$ is computed using the already-computed values $v(0), v(1), \ldots, v(t-1)$, until the value $v(m)$ of the ULS instance is obtained.

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

Algorithm: DP_WW (1/2)

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

Output : The optimal value

```
1 function DP_WW(m, d, sc, vc, hc)
2 |   c := CostMatrix(m, d, sc, vc, hc)
3 |   v := new array[0..m]
4 |   v[0] ← 0
```

Algorithm: DP_WW (2/2)

function *DP_WW*(m, d, sc, vc, hc) (continued)

```
5     for t := 1 to m do
6         b := +∞
7         for j := 1 to t do
8             if v[j − 1] + c[j][t] < b then
9                 b ← v[j − 1] + c[j][t]
10            end
11        end
12        v[t] ← b
13    end
14    return v[m]
15 end
```

How the DP algorithm works (1/2)

- The matrix c contains the production-run costs, computed by the algorithm CostMatrix.
- The vector $v[0..m]$ is the **lookup table**: it stores the optimal values of all subproblems, with the base case $v[0] = 0$.
- The outer loop, over $t \in \{1, 2, \ldots, m\}$, computes $v[t]$ in increasing order of the last period.

How the DP algorithm works (2/2)

- The inner loop, over $j \in \{1, 2, \ldots, t\}$, evaluates all possible periods of the last production run and keeps in b the smallest value $v[j-1] + c[j][t]$.
- The variable b starts from $+\infty$, so the first comparison always updates it.
- At the end of the inner loop, $v[t] \leftarrow b$ stores the optimal value $v[t]$.
- Finally, the optimal value $v[m]$ of the ULS instance is returned.

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

We consider the ULS instance of the example, with the matrix c of the production-run costs:

$c[j][t]$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$j=1$	50	60	120	240	280
$j=2$	–	50	80	160	190
$j=3$	–	–	50	90	110
$j=4$	–	–	–	50	60
$j=5$	–	–	–	–	50

- For each step, the next tables show the values of t and j, the current value of b, and the cell $v[t]$.
- The cell of b is highlighted in red when updated; the cell of $v[t]$ is highlighted in blue when assigned.

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

t	j	$v[t]$	b	Action
–	–	0	–	$v[0] \leftarrow 0$
	–	–	$+\infty$	$b \leftarrow +\infty$
1	1	–	50	$v[0]+c[1][1] = 0+50 = 50 < +\infty$: $b \leftarrow 50$
	–	50	50	$v[1] \leftarrow b = 50$
	–	–	$+\infty$	$b \leftarrow +\infty$
2	1	–	60	$v[0]+c[1][2] = 0+60 = 60 < +\infty$: $b \leftarrow 60$
	2	–	60	$v[1]+c[2][2] = 50+50 = 100 > 60$: no update
	–	60	60	$v[2] \leftarrow b = 60$

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

t	j	$v[t]$	b	Action
	–	–	$+\infty$	$b \leftarrow +\infty$
3	1	–	120	$v[0]+c[1][3] = 0+120 = 120 < +\infty$: $b \leftarrow 120$
	2	–	120	$v[1]+c[2][3] = 50+80 = 130 > 120$: no update
	3	–	110	$v[2]+c[3][3] = 60+50 = 110 < 120$: $b \leftarrow 110$
	–	110	110	$v[3] \leftarrow b = 110$

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

t	j	$v[t]$	b	Action
	–	–	$+\infty$	$b \leftarrow +\infty$
4	1	–	240	$v[0]+c[1][4] = 0+240 = 240 < +\infty$: $b \leftarrow 240$
	2	–	210	$v[1]+c[2][4] = 50+160 = 210 < 240$: $b \leftarrow 210$
	3	–	150	$v[2]+c[3][4] = 60+90 = 150 < 210$: $b \leftarrow 150$
	4	–	150	$v[3]+c[4][4] = 110+50 = 160 > 150$: no update
	–	150	150	$v[4] \leftarrow b = 150$

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

t	j	$v[t]$	b	Action
	–	–	$+\infty$	$b \leftarrow +\infty$
5	1	–	280	$v[0]+c[1][5] = 0+280 = 280 < +\infty$: $b \leftarrow 280$
	2	–	240	$v[1]+c[2][5] = 50+190 = 240 < 280$: $b \leftarrow 240$
	3	–	170	$v[2]+c[3][5] = 60+110 = 170 < 240$: $b \leftarrow 170$
	4	–	170	$v[3]+c[4][5] = 110+60 = 170 = b$: no update
	5	–	170	$v[4]+c[5][5] = 150+50 = 200 > 170$: no update
	–	170	170	$v[5] \leftarrow b = 170$

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

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

	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$v[t]$	0	50	60	110	150	170

- After the execution of the DP algorithm, the vector v contains the optimal value $v[t]$ for every set of periods $\{1, \ldots, t\}$.
- The optimal value of the ULS instance is $v[5] = 170$.

Remark: a shortest path

- The recursive formula of $v(t)$ computes a shortest path in a network with:
  - nodes $0, 1, \ldots, m$,
  - one arc from node $j-1$ to node t, of cost $c_{j,t}$, for every $j \leq t$.
- The value $v(t)$ is the length of a shortest path from node 0 to node t.
- This is the shortest-path formulation of the chapter «Formulations of lot sizing problems», without the capacities: an arc is a production run.

Step 2: computation of an optimal solution

- To compute an optimal solution, it is necessary to store some additional information.
- For each period t, in addition to the optimal value $v[t]$, we store $r[t]$, the period of the last production run in an optimal solution for periods $\{1, \ldots, t\}$.
- Starting from $t = m$, the value $r[t]$ gives the period of the last production run, which produces $D_{r[t],t}$ units; the remaining periods are $\{1, \ldots, r[t] - 1\}$, and the same rule can be applied again, and so on, until no period remains.

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_WW_InfoSol (1/2)

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

Output : The optimal value and the information vector $r[1..m]$

```
1 function DP_WW_InfoSol(m, d, sc, vc, hc)
2 |   c := CostMatrix(m, d, sc, vc, hc)
3 |   v := new array[0..m]
4 |   r := new array[1..m]
5 |   v[0] ← 0
```

Algorithm: DP_WW_InfoSol (2/2)

function *DP_WW_InfoSol*(m, d, sc, vc, hc) (continued)

```
6     for t := 1 to m do
7         b := +∞
8         a := 0
9         for j := 1 to t do
10            if v[j − 1] + c[j][t] < b then
11                b ← v[j − 1] + c[j][t]
12                a ← j
13            end
14        end
15        v[t] ← b
16        r[t] ← a
17    end
18    return v[m], r
19 end
```

How the information is stored

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

Example: computation of the additional information (1/6)

We consider the ULS instance of the example, with the matrix c of the production-run costs:

$c[j][t]$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$j=1$	50	60	120	240	280
$j=2$	–	50	80	160	190
$j=3$	–	–	50	90	110
$j=4$	–	–	–	50	60
$j=5$	–	–	–	–	50

- For each step, the next tables show the values of t and j, the cell $r[t]$, the current value of b, and the current value of a.
- The cell of b is highlighted in red when updated; the cells of a and of $r[t]$ are highlighted in blue when updated.

Example: computation of the additional information (2/6)

t	j	$r[t]$	b	a	Action
–	–	–	–	–	Initial state
	–	–	$+\infty$	0	$b \leftarrow +\infty$, $a \leftarrow 0$
1	1	–	50	1	$v[0]+c[1][1] = 0+50 = 50 < +\infty$: $b \leftarrow 50$, $a \leftarrow 1$
	–	1	50	1	$v[1] \leftarrow b = 50$, $r[1] \leftarrow a = 1$
	–	–	$+\infty$	0	$b \leftarrow +\infty$, $a \leftarrow 0$
2	1	–	60	1	$v[0]+c[1][2] = 0+60 = 60 < +\infty$: $b \leftarrow 60$, $a \leftarrow 1$
	2	–	60	1	$v[1]+c[2][2] = 50+50 = 100 > 60$: no update
	–	1	60	1	$v[2] \leftarrow b = 60$, $r[2] \leftarrow a = 1$

Example: computation of the additional information (3/6)

t	j	$r[t]$	b	a	Action
	–	–	$+\infty$	0	$b \leftarrow +\infty$, $a \leftarrow 0$
3	1	–	120	1	$v[0]+c[1][3] = 0+120 = 120 < +\infty$: $b \leftarrow 120$, $a \leftarrow 1$
	2	–	120	1	$v[1]+c[2][3] = 50+80 = 130 > 120$: no update
	3	–	110	3	$v[2]+c[3][3] = 60+50 = 110 < 120$: $b \leftarrow 110$, $a \leftarrow 3$
	–	3	110	3	$v[3] \leftarrow b = 110$, $r[3] \leftarrow a = 3$

Example: computation of the additional information (4/6)

t	j	$r[t]$	b	a	Action
	–	–	$+\infty$	0	$b \leftarrow +\infty$, $a \leftarrow 0$
4	1	–	240	1	$v[0]+c[1][4] = 0+240 = 240 < +\infty$: $b \leftarrow 240$, $a \leftarrow 1$
	2	–	210	2	$v[1]+c[2][4] = 50+160 = 210 < 240$: $b \leftarrow 210$, $a \leftarrow 2$
	3	–	150	3	$v[2]+c[3][4] = 60+90 = 150 < 210$: $b \leftarrow 150$, $a \leftarrow 3$
	4	–	150	3	$v[3]+c[4][4] = 110+50 = 160 > 150$: no update
	–	3	150	3	$v[4] \leftarrow b = 150$, $r[4] \leftarrow a = 3$

Example: computation of the additional information (5/6)

t	j	$r[t]$	b	a	Action
	–	–	$+\infty$	0	$b \leftarrow +\infty$, $a \leftarrow 0$
5	1	–	280	1	$v[0]+c[1][5] = 0+280 = 280 < +\infty$: $b \leftarrow 280$, $a \leftarrow 1$
	2	–	240	2	$v[1]+c[2][5] = 50+190 = 240 < 280$: $b \leftarrow 240$, $a \leftarrow 2$
	3	–	170	3	$v[2]+c[3][5] = 60+110 = 170 < 240$: $b \leftarrow 170$, $a \leftarrow 3$
	4	–	170	3	$v[3]+c[4][5] = 110+60 = 170 = b$: no update
	5	–	170	3	$v[4]+c[5][5] = 150+50 = 200 > 170$: no update
	–	3	170	3	$v[5] \leftarrow b = 170$, $r[5] \leftarrow a = 3$

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

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

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$r[t]$	1	1	3	3	3

- For example, $r[5] = 3$: in an optimal solution for periods $\{1, \ldots, 5\}$ the last production run is in period 3.

Computing an optimal solution from the vector r

- Once the vector r has been computed, an optimal solution of the ULS instance can be computed by repeatedly applying the following rule, starting from $t = m$ and $x_j = 0$ for all $j \in \{1, 2, \ldots, m\}$:

$$x_{r[t]} \leftarrow D_{r[t],t}, \qquad t \leftarrow r[t] - 1,$$

  until $t = 0$.

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

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

Algorithm: DP_WW_SolComputation

Input : The number m of periods and the demands $d[1..m]$ of the ULS instance, and the vector $r[1..m]$

Output : An optimal solution of the ULS instance

```
1  function DP_WW_SolComputation(m, d, r)
2      x := new array[1..m]
3      for j := 1 to m do
4          x[j] ← 0
5      end
6      t := m
7      while t > 0 do
8          x[r[t]] ← Σ_{k=r[t]}^{t} d[k]
9          t ← r[t] − 1
10     end
11     return x
12 end
```

Example: computing an optimal solution (1/2)

After the execution of the DP algorithm on the instance of the example, we have the following vector r:

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
$r[t]$	1	1	3	3	3

We can compute an optimal solution for $m = 5$ periods as follows:

Iteration	t	$r[t]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$t \leftarrow r[t]-1$
–	–	–	0	0	0	0	0	$t=5$
1	5	3	0	0	80	0	0	$t=3-1=2$
2	2	1	30	0	80	0	0	$t=1-1=0$

Example: computing an optimal solution (2/2)

- $t = 5$, $r[5] = 3$: the last production run of periods $1, \ldots, 5$ is in period 3 and covers the demands of periods $3, \ldots, 5$, so $x[3] \leftarrow d_3 + d_4 + d_5 = 30 + 40 + 10 = 80$
- $t = 2$, $r[2] = 1$: the last production run of periods $1, \ldots, 2$ is in period 1 and covers the demands of periods $1, \ldots, 2$, so $x[1] \leftarrow d_1 + d_2 = 20 + 10 = 30$

An optimal solution is the vector:

$$(30, 0, 80, 0, 0)$$

The corresponding optimal value is $v[5] = 170$, the same value obtained by the enumeration of the solutions with the zero-inventory ordering property and by Gurobi on the $\mathrm{ILP}_{\mathrm{ULS}}$ model.

Remark: ties

- In the computation of $v[5]$, the run in period 4 gives the same value as the run in period 3, but the strict comparison keeps $j = 3$.
- Choosing $j = 4$ instead leads to the other optimal solution $(30, 0, 30, 50, 0)$ of the enumeration.
- The DP algorithm returns one optimal solution, not all of them.

Number of steps

How many steps does the DP algorithm execute?

We first count the steps of CostMatrix,
then those of DP_WW, which calls it.

We recall the two pseudo-codes
before counting the executions of their lines.

Algorithm: CostMatrix

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

Output : The matrix of the production-run costs

```
1  function CostMatrix(m, d, sc, vc, hc)
2      c := new array[1..m][1..m]
3      for j := 1 to m do
4          u := vc[j]
5          g := sc[j] + vc[j] · d[j]
6          c[j][j] ← g
7          for t := j + 1 to m do
8              u ← u + hc[t − 1]
9              g ← g + u · d[t]
10             c[j][t] ← g
11         end
12     end
13     return c
14 end
```


Sapienza Università di Roma

Number of steps of CostMatrix

Let $S_c(m)$ denote the number of steps executed by CostMatrix in the worst case, where m is the number of periods. The number of executions of its lines is:

- Line 2 is executed 1 time.
- Line 3, Line 4, Line 5, and Line 6 are executed m times each.
- Line 7, Line 8, Line 9, and Line 10 are executed $\sum_{j=1}^{m}(m-j) = \frac{m(m-1)}{2}$ times each.
- Line 13 is executed 1 time.

Therefore,

$$S_c(m) = 2 + 4m + 4 \cdot \frac{m(m-1)}{2} = 2m^2 + 2m + 2.$$

Algorithm: DP_WW (1/2)

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

Output : The optimal value

```
1 function DP_WW(m, d, sc, vc, hc)
2 |   c := CostMatrix(m, d, sc, vc, hc)
3 |   v := new array[0..m]
4 |   v[0] ← 0
```

Algorithm: DP_WW (2/2)

function *DP_WW*(m, d, sc, vc, hc) (continued)

```
5     for t := 1 to m do
6         b := +∞
7         for j := 1 to t do
8             if v[j − 1] + c[j][t] < b then
9                 b ← v[j − 1] + c[j][t]
10            end
11        end
12        v[t] ← b
13    end
14    return v[m]
15 end
```

Number of steps of DP_WW (1/3)

Let $S(m)$ denote the number of steps executed by DP_WW in the worst case.

- Line 2 executes CostMatrix, that is, $S_c(m)$ steps.
- In the worst case, the condition at line 8 is true at every iteration of the inner loop, and therefore line 9 is executed as often as line 8.

Number of steps of DP_WW (2/3)

The number of executions of the other lines is:

- Line 3 is executed 1 time.
- Line 4 is executed 1 time.
- Line 5 is executed m times.
- Line 6 is executed m times.
- Line 7 is executed $\sum_{t=1}^{m} t$ times.
- Line 8 is executed $\sum_{t=1}^{m} t$ times.
- Line 9 is executed $\sum_{t=1}^{m} t$ times.
- Line 12 is executed m times.
- Line 14 is executed 1 time.

Number of steps of DP_WW (3/3)

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

$$S(m) = S_c(m) + 3 + 3m + 3\sum_{t=1}^{m} t$$

$$= 2m^2 + 2m + 2 + 3 + 3m + 3 \cdot \frac{m(m+1)}{2}$$

$$= \frac{7m^2 + 13m + 10}{2}.$$

Hence,

$$S(m) = \Theta(m^2).$$

Indeed,

$$\lim_{m\to\infty} \frac{\frac{7m^2+13m+10}{2}}{m^2} = \lim_{m\to\infty} \frac{1}{2}\left(7 + \frac{13}{m} + \frac{10}{m^2}\right) = \frac{7}{2} \in \mathbb{R}_{>0}.$$

The number of steps of the DP algorithm

The DP algorithm executes

$$S(m) = \Theta(m^2)$$

steps in the worst case.

- DP_WW_InfoSol only adds a constant number of operations inside the same loops, and therefore its number of steps has the same order of growth.
- The enumeration of the solutions with the zero-inventory ordering property lists 2^{m-1} solutions instead.

Summary (1/2)

Production runs:

- a run produces in period j the demands of periods $j, \ldots, t$, at the cost $c_{j,t} = sc_j + vc_j\, D_{j,t} + \sum_{k=j}^{t-1} hc_k\, D_{k+1,t}$
- CostMatrix computes all the costs incrementally, in $2m^2 + 2m + 2$ steps

Step 1: the optimal value:

- $v(t) = \min\{v(j-1) + c_{j,t} : j \in \{1, 2, \ldots, t\}\}$, with $v(0) = 0$: choice of the last run
- optimal substructure and overlapping subproblems: bottom-up in the lookup table $v[0..m]$
- the recurrence is a shortest path from node 0 to node m

Summary (2/2)

Step 2: an optimal solution:

- $r[t]$ stores the period of the last run; with ties, the smallest period
- the plan is read backwards: $x_{r[t]} \leftarrow D_{r[t],t}$, $t \leftarrow r[t] - 1$
- on the example: $(30, 0, 80, 0, 0)$, with optimal value $v[5] = 170$

Number of steps:

- $S(m) = \frac{7m^2+13m+10}{2} = \Theta(m^2)$ for DP_WW
- the same order of growth for DP_WW_InfoSol

Typical mistakes

Introduction: typical mistakes

- We collect the mistakes that are made most often with the DP algorithm for the ULS
- Two of them concern the zero-inventory ordering property and the cost of a run
- Two concern the subinstances and the optimal plans
- Check them before running the DP algorithm by hand

Typical mistakes (1/2)

Observation 1: Applying the zero-inventory ordering property with capacities

The proof moves production between two periods without limits: with capacities a period can produce while units are in stock.

Observation 2: Computing the cost of a run from scratch

Computing every $c_{j,t}$ from its definition costs $O(m^3)$ steps inside the DP algorithm; the incremental computation of Cost-Matrix keeps $\Theta(m^2)$.

Typical mistakes (2/2)

Observation 3: Forgetting the empty final inventory

The subinstance with periods $\{1, \ldots, t\}$ ends with an empty inventory: this is why the run in period j produces exactly $D_{j,t}$ units.

Observation 4: Reading one optimal plan as the only one

The example of the chapter has two optimal plans: the DP algorithm returns the one with the smallest period j, because it updates only on a strictly smaller value.

Summary

Property and costs:

- the zero-inventory ordering property holds without capacities
- the costs $c_{j,t}$ are computed incrementally, in $\Theta(m^2)$ steps

Subinstances and plans:

- every subinstance ends with an empty inventory, so a run produces exactly $D_{j,t}$
- the DP algorithm returns one optimal plan, not all of them

In summary

Introduction: in summary

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

Summary (1/2)

- If all costs are non-negative, the uncapacitated problem has an optimal plan in which a period produces only when the inventory is zero (Wagner and Whitin), so $x_j = D_{j,t}$ for every production run.
- The cost $c_{j,t}$ of every run is computed in $\Theta(m^2)$ steps, and the recurrence $v(t) = \min_j \{v(j-1) + c_{j,t}\}$ gives the optimal value in $\Theta(m^2)$ steps; it is a shortest path.
- The zero-inventory ordering property rests on production without limits: with capacities it fails.

Summary (2/2)

Dashboard of the example.

Instance	Lower bound	Upper bound	Origin
Example of the chapter (uncapacitated, $m = 5$)	170	170	DP algorithm of Wagner and Whitin