PD per il problema del lot sizing senza capacità

Indice

Perché il lot sizing?

Notazione, modello e soluzioni ammissibili
- Codifica delle soluzioni ammissibili
- La proprietà zero-inventory ordering

Un algoritmo di PD per l'ULS
- Il costo di un lotto di produzione
- Passo 1 e Passo 2
- Numero di passi

Errori tipici

In sintesi

Obiettivi

Al termine di questo capitolo saremo in grado di:

- enunciare la proprietà zero-inventory ordering e dimostrarla con l'argomento di scambio;
- calcolare il costo $c_{j,t}$ di un lotto di produzione, e la matrice dei costi, in $O(m^2)$ passi;
- scrivere la ricorrenza $v(t)$ sull'ultimo lotto di produzione, eseguire a mano l'algoritmo di PD e ricostruire un piano ottimo;
- leggere la ricorrenza come un cammino minimo nella rete dei lotti di produzione;
- contare i passi dell'algoritmo di PD e mostrare che sono $\Theta(m^2)$.

Prerequisiti

- Dal modulo *Problemi di ottimizzazione combinatoria*, i capitoli «Problemi di lot sizing» e «Formulazioni dei problemi di lot sizing»: il problema a un solo prodotto, la sua istanza di riferimento e il suo modello PLI con le variabili x_t, s_t, y_t.
- Da questo modulo, il capitolo «Principi della programmazione dinamica» e gli algoritmi di PD dei capitoli precedenti: ricorrenza, casi base, tabella di lookup e ricostruzione di una soluzione.
- Dal modulo *Algoritmi*: lo pseudocodice e la dimensione dell'input.

Dove eravamo

- **Come problema e come modello**: il lot sizing a un solo prodotto e le sue formulazioni PLI (*Problemi di ottimizzazione combinatoria*, capitoli «Problemi di lot sizing» e «Formulazioni dei problemi di lot sizing»).

- **Ora, come tabella**: l'algoritmo di PD di Wagner e Whitin dà il valore ottimo del problema senza capacità in $\Theta(m^2)$ passi.

Perché il lot sizing?

Introduzione: perché il lot sizing?

- Un'azienda deve decidere **quando** produrre e **quanto**, periodo dopo periodo
- Ogni produzione ha un costo fisso, e ogni unità tenuta a scorta ha un costo
- In questo capitolo risolviamo in modo esatto la versione senza capacità di questo problema, con un algoritmo di PD

Pianificare la produzione di un solo prodotto

- Un'azienda pianifica la produzione di **un solo prodotto** su un orizzonte di m periodi di tempo (settimane, mesi, ...).
- La domanda di ogni periodo deve essere soddisfatta, dalla produzione del periodo oppure dalla scorta mantenuta dai periodi precedenti.

Setup contro scorte

- Produrre in un periodo richiede un **setup**, che ha un costo fisso, indipendente dalla quantità prodotta.
- Ogni unità prodotta ha un costo variabile.
- Ogni unità tenuta a scorta alla fine di un periodo ha un costo di mantenimento a scorta.

Produrre di rado fa risparmiare costi di setup
ma aumenta i costi di mantenimento a scorta:

il problema è trovare il **miglior compromesso**.

Il problema di questo capitolo

Il problema di questo capitolo è il
problema di lot sizing senza capacità (ULS):

la sua definizione e le sue ipotesi sono nella sua pagina.

- Scriviamo l'algoritmo di PD di Wagner e Whitin, che dà il valore ottimo di un'istanza dell'ULS
- Lo stesso algoritmo dà anche un piano di produzione ottimo

Riepilogo

Il problema:

- un solo prodotto, m periodi, una domanda da soddisfare in ogni periodo
- produzione nel periodo oppure scorta dai periodi precedenti

I costi:

- un costo fisso di setup in ogni periodo con produzione
- un costo variabile per unità prodotta e un costo di mantenimento per unità a scorta
- pochi setup significano scorte grandi: il miglior compromesso è il piano ottimo

Notazione, modello e soluzioni ammissibili

Introduzione: notazione, modello e soluzioni ammissibili

- In questa sezione fissiamo la notazione e richiamiamo il modello PLI dell'ULS
- Descriviamo una soluzione ammissibile con il vettore delle quantità prodotte
- Dimostriamo la proprietà zero-inventory ordering: mostra che basta considerare 2^{m-1} soluzioni
- Questa proprietà è la base dell'algoritmo di PD della sezione successiva

La domanda totale di periodi consecutivi

- Per ogni $j, t \in T$, indichiamo con

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

la domanda totale dei periodi $j, j+1, \ldots, t$, con $D_{j,t} = 0$ se $j > t$.

- Per esempio, $D_{1,m}$ è la domanda totale dell'intero orizzonte, e $D_{t,m}$ è la domanda totale dei periodi rimanenti $t, t+1, \ldots, m$.

Il modello PLI dell'ULS

- Il modello PLI è il modello $\mathrm{ILP}_{\mathrm{ULS}}$ del capitolo «Formulazioni dei problemi di lot sizing», sezione «Modelli PLI per le varianti».
- È il modello $\mathrm{ILP}_{\mathrm{CLSP}}$ senza i vincoli di capacità e con il valore di big-M $D_{t,m}$ del periodo t, la domanda totale dei periodi rimanenti.
- Le sue variabili sono, per ogni periodo $t \in T$:
  - la produzione x_t,
  - la scorta s_t alla fine del periodo t,
  - il setup $y_t \in \{0, 1\}$.

Il modello $\mathrm{ILP}_{\mathrm{ULS}}$

Richiamiamo il modello $\mathrm{ILP}_{\mathrm{ULS}}$ del problema di lot sizing senza capacità (ULS):

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

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

Questo modello PLI si chiama modello $\mathrm{ILP}_{\mathrm{ULS}}$.

Descrizione del modello $\mathrm{ILP}_{\mathrm{ULS}}$

Descrizione della funzione obiettivo e dei vincoli del modello $\mathrm{ILP}_{\mathrm{ULS}}$:

- la funzione obiettivo minimizza il costo totale di setup, produzione e mantenimento a scorta;
- i vincoli $s_{t-1} + x_t = d_t + s_t$ sono i vincoli di bilanciamento del flusso;
- i vincoli $x_t \leq D_{t,m}\, y_t$ impongono un setup in ogni periodo con produzione positiva;
- il vincolo $s_0 = s_m = 0$ impone una scorta vuota all'inizio e alla fine dell'orizzonte;
- infine, le ultime tre righe definiscono le variabili del modello PLI.

Esempio: istanza dell'ULS

Consideriamo $m = 5$ periodi con i seguenti dati:

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

- Usiamo questa istanza dell'ULS come esempio di tutto il capitolo.
- È l'istanza di riferimento del capitolo «Problemi di lot sizing» di *Problemi di ottimizzazione combinatoria*, contata in singoli pezzi invece che in decine e senza le capacità (costo di setup 50, costo di mantenimento a scorta 1, nessun costo variabile di produzione).

Soluzioni ammissibili dell'ULS

Una *soluzione ammissibile* dell'ULS è un piano di produzione che soddisfa la domanda di ogni periodo, partendo e terminando con la scorta vuota.

- Descriviamo ora una soluzione ammissibile con un vettore, e da questo vettore calcoliamo le sue scorte e il suo costo totale.

Codifica delle soluzioni ammissibili (1/2)

Una soluzione ammissibile dell'ULS si può descrivere con un vettore soluzione

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

dove ogni componente ha il seguente significato:

x_t = numero di unità prodotte nel periodo t, $\quad \forall t \in \{1, 2, \ldots, m\}$.

La scorta alla fine del periodo t è determinata dal vettore:

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

Codifica delle soluzioni ammissibili (2/2)

- Poiché la soluzione è ammissibile, le scorte sono non negative e l'ultima è vuota:

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

- Nel periodo t si paga un setup se e solo se $x_t > 0$.

- Il costo totale di una soluzione ammissibile è

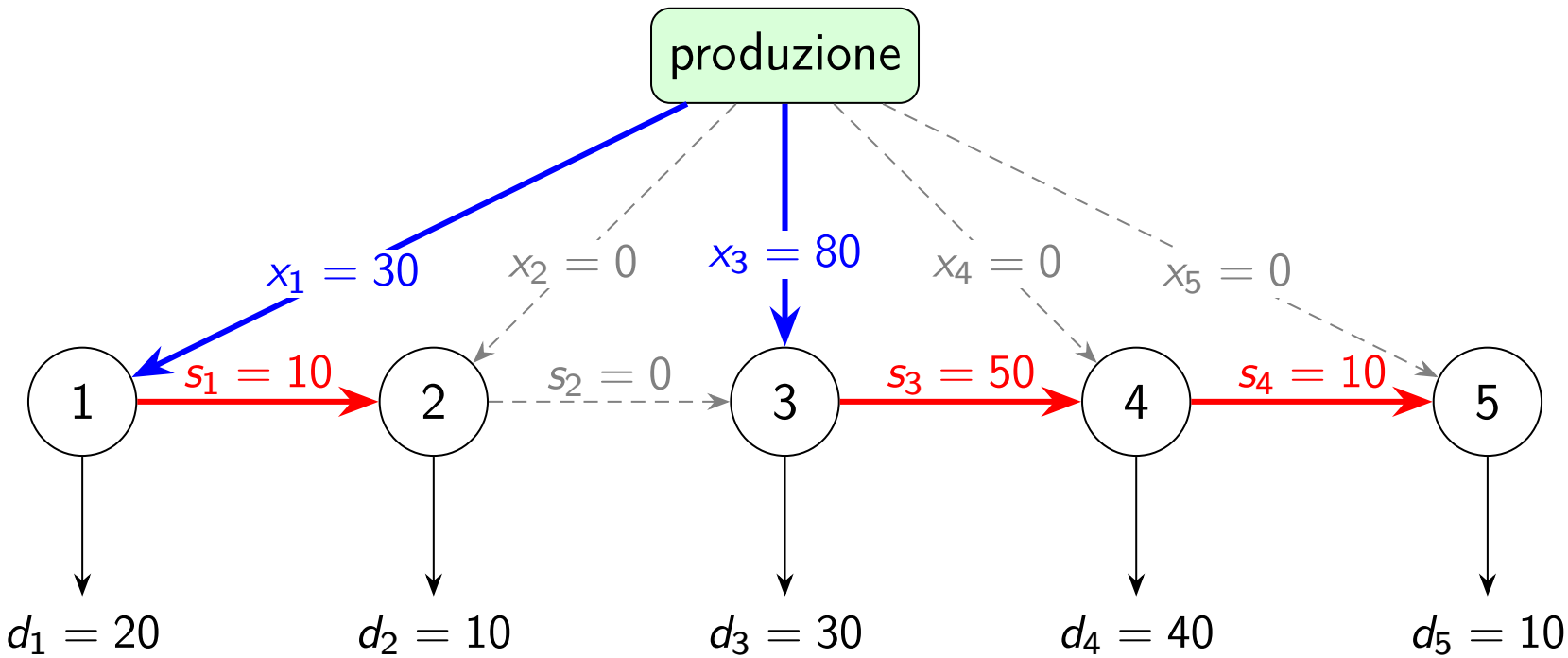
$$\sum_{t=1,\ x_t>0}^{m} sc_t + \sum_{t=1}^{m} vc_t\, x_t + \sum_{t=1}^{m} hc_t\, s_t.$$

Esempio: una soluzione ammissibile (1/3)

- Consideriamo l'istanza dell'ULS dell'esempio precedente.
- La rete seguente rappresenta la soluzione ammissibile $(30, 0, 80, 0, 0)$:
  - ogni periodo è un nodo,
  - la produzione x_t entra nel nodo t, e la domanda d_t ne esce,
  - la scorta s_t scorre dal nodo t al nodo $t + 1$.
- Gli archi spessi portano un flusso positivo.

Esempio: una soluzione ammissibile (2/3)

La rete della soluzione ammissibile $(30, 0, 80, 0, 0)$:



Esempio: una soluzione ammissibile (3/3)

- I setup sono nei periodi 1, 3.
- Le scorte sono $(s_1, \ldots, s_5) = (10, 0, 50, 10, 0)$.
- Il costo totale è (non c'è costo variabile di produzione)

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

Quante soluzioni ammissibili?

- Una quantità prodotta può assumere molti valori, e un'istanza dell'ULS ha un numero enorme di soluzioni ammissibili (infinite, se non si richiede che le quantità siano intere).
- La proprietà seguente mostra che basta considerarne pochissime.

La proprietà zero-inventory ordering

Teorema 1: Zero-inventory ordering (Wagner e Whitin, 1958)

Ogni istanza dell'ULS ha una soluzione ottima in cui nessun periodo con produzione positiva inizia con una scorta positiva, cioè

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

- A parole: un periodo produce solo quando la sua scorta iniziale è vuota.

L'idea della dimostrazione

- La dimostrazione è per assurdo, con un **argomento di scambio**.
- Prendiamo una soluzione ottima con il minimo numero di periodi con produzione positiva.
- Se un periodo produce mentre la scorta è positiva, spostiamo unità di produzione fra questo periodo e il periodo precedente con produzione.
- O il costo diminuisce, oppure sparisce un setup: in entrambi i casi si contraddice la scelta della soluzione.

Dimostrazione (1/4)

- Anzitutto, una soluzione ottima esiste: il vettore $(d_1, d_2, \ldots, d_m)$ è ammissibile e, per ciascuna delle 2^m scelte dell'insieme dei periodi con setup, il problema che resta è un problema LP su un poliedro limitato (poiché $s_m = 0$, ogni x_t è compreso fra 0 e $D_{1,m}$), quindi ha una soluzione ottima ogni volta che è ammissibile.

- Fra tutte le soluzioni ottime, consideriamone una con il **minimo numero di periodi con produzione positiva**, e supponiamo, per assurdo, che esista un periodo t con $x_t > 0$ e $s_{t-1} > 0$.

Dimostrazione (2/4)

- Poiché $s_0 = 0$ e $s_{t-1} > 0$, qualcosa viene prodotto prima del periodo t: sia $j < t$ l'ultimo periodo prima di t con $x_j > 0$.
- Nei periodi $j+1, \ldots, t-1$ non si produce alcuna unità, quindi la scorta può solo diminuire dal periodo j al periodo $t-1$:

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

- Per un numero ε, spostiamo ε unità di produzione dal periodo j al periodo t: sostituiamo x_j con $x_j - \varepsilon$, x_t con $x_t + \varepsilon$ e s_k con $s_k - \varepsilon$ per ogni $k \in \{j, \ldots, t-1\}$.
- Tutti i vincoli di bilanciamento restano soddisfatti, e il nuovo piano è ammissibile per ogni ε con

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

Dimostrazione (3/4)

- Finché x_j e x_t restano positivi, i setup non cambiano e il costo varia di $\varepsilon\,\Delta$, dove

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

  è il costo di produrre un'unità nel periodo t meno il costo di produrla nel periodo j e tenerla a scorta fino al periodo t.

- Se $\Delta < 0$, un $\varepsilon > 0$ piccolo dà una soluzione ammissibile di costo minore, il che contraddice l'ottimalità.

Dimostrazione (4/4)

- Se $\Delta \geq 0$, scegliamo $\varepsilon = -x_t$: tutta la produzione del periodo t viene spostata nel periodo j, il costo varia di $-x_t \Delta \leq 0$, e si risparmia il costo di setup $sc_t \geq 0$ del periodo t.
- La nuova soluzione è ottima e ha un periodo con produzione positiva in meno, il che contraddice la scelta della soluzione.
- Quindi un tale periodo t non esiste.

□

Ogni produzione copre domande consecutive

In una soluzione ottima con la proprietà zero-inventory ordering, ogni periodo j con produzione positiva produce esattamente le domande dei periodi $j, j+1, \ldots, t$, dove $t+1$ è il periodo successivo con produzione positiva (oppure $t = m$ se non ce n'è uno):

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

- Infatti $s_{j-1} = 0$ (zero-inventory ordering nel periodo j), $s_t = 0$ (zero-inventory ordering nel periodo $t+1$, oppure $s_m = 0$), e nei periodi $j+1, \ldots, t$ non si produce nulla.
- Sommando i vincoli di bilanciamento dei periodi $j, \ldots, t$ si ottiene $x_j = D_{j,t}$.

Soluzioni con la proprietà zero-inventory ordering

- Il periodo 1 produce sempre, perché $s_0 = 0$ e $d_1 \geq 1$.
- Una soluzione con la proprietà zero-inventory ordering è quindi determinata dall'insieme dei suoi periodi di produzione, che contiene il periodo 1 e un qualsiasi sottoinsieme di $\{2, 3, \ldots, m\}$.

Il numero di soluzioni dell'ULS con la proprietà zero-inventory ordering è 2^{m-1},
perché ogni periodo $t \in \{2, 3, \ldots, m\}$ può, indipendentemente dagli altri, produrre o no.

Un numero esponenziale di soluzioni

- Il numero di queste soluzioni cresce esponenzialmente con il numero m di periodi:

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

- Quindi enumerarle diventa rapidamente impraticabile al crescere di m.

Vediamo ora l'enumerazione sull'esempio,
che ha solo $m = 5$ periodi.

Esempio: enumerazione delle soluzioni (1/6)

Consideriamo l'istanza dell'ULS dell'esempio precedente:

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

- Ci sono $2^{5-1} = 16$ soluzioni con la proprietà zero-inventory ordering.
- Per ciascuna di esse, le tabelle seguenti riportano i periodi di produzione, il vettore soluzione, le scorte e il costo totale, scritto come costo di setup più costo di mantenimento a scorta.
- I costi ottimi sono evidenziati.

Esempio: enumerazione delle soluzioni (2/6)

Soluzioni (1)–(8): periodi di produzione e vettori soluzione.

Etichetta	Periodi di produzione	Vettore soluzione
(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)	$\{1, 3, 4, 5\}$	$(30, 0, 30, 40, 10)$

Esempio: enumerazione delle soluzioni (3/6)

Soluzioni (1)–(8): scorte e costi totali.

Etichetta	Periodi di produzione	Scorte $(s_1, \ldots, s_5)$	Costo totale
(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)	$\{1, 3, 4, 5\}$	$(10, 0, 0, 0, 0)$	$200 + 10 = 210$

Esempio: enumerazione delle soluzioni (4/6)

Soluzioni (9)–(16): periodi di produzione e vettori soluzione.

Etichetta	Periodi di produzione	Vettore soluzione
(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)	$\{1, 2, 3, 4, 5\}$	$(20, 10, 30, 40, 10)$

Esempio: enumerazione delle soluzioni (5/6)

Soluzioni (9)–(16): scorte e costi totali.

Etichetta	Periodi di produzione	Scorte $(s_1, \ldots, s_5)$	Costo totale
(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)	$\{1, 2, 3, 4, 5\}$	$(0, 0, 0, 0, 0)$	$250 + 0 = 250$

Esempio: enumerazione delle soluzioni (6/6)

Il valore ottimo dell'istanza dell'ULS è 170,

raggiunto da 2 soluzioni: (5) e (7).

- La soluzione (5) è $(30, 0, 80, 0, 0)$, con periodi di produzione $\{1, 3\}$.
- La soluzione (7) è $(30, 0, 30, 50, 0)$, con periodi di produzione $\{1, 3, 4\}$.
- Con m periodi l'enumerazione elenca 2^{m-1} soluzioni: la sezione successiva calcola il valore ottimo senza elencarle.

Riepilogo

Notazione e modello:

- $D_{j,t}$ è la domanda totale dei periodi $j, \ldots, t$
- il modello $\mathrm{ILP}_{\mathrm{ULS}}$: vincoli di bilanciamento, vincoli di setup con big-M $D_{t,m}$, scorta vuota all'inizio e alla fine

Soluzioni ammissibili:

- una soluzione è il vettore $(x_1, \ldots, x_m)$; le scorte s_t e il costo ne derivano
- zero-inventory ordering: qualche soluzione ottima ha $s_{t-1}\, x_t = 0$ per ogni periodo t
- allora ogni produzione è $x_j = D_{j,t}$, e restano solo 2^{m-1} soluzioni
- sull'esempio: valore ottimo 170, raggiunto da 2 soluzioni

Un algoritmo di PD per l'ULS

Introduzione: un algoritmo di PD per l'ULS

- In questa sezione scriviamo l'algoritmo di PD di Wagner e Whitin per l'ULS
- Prima calcoliamo il costo di ogni lotto di produzione, poi la ricorrenza sceglie l'ultimo lotto
- Eseguiamo l'algoritmo a mano sull'esempio e ricostruiamo un piano ottimo
- Infine contiamo i suoi passi: sono $\Theta(m^2)$, invece delle 2^{m-1} soluzioni dell'enumerazione

Lotti di produzione

- Per la proprietà zero-inventory ordering, una soluzione ottima è una sequenza di **lotti di produzione**:
  - un lotto produce in un periodo j le domande dei periodi $j, \ldots, t$,
  - il lotto successivo inizia nel periodo $t+1$ con la scorta vuota.
- Per ogni $j, t \in T$ con $j \leq t$, sia $c_{j,t}$ il costo del lotto di produzione che produce nel periodo j le domande dei periodi $j, \ldots, t$.

Il costo di un lotto di produzione

Il costo del lotto di produzione dei periodi $j, \ldots, t$ è

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

- Il primo termine è il costo di setup del periodo j.
- Il secondo è il costo variabile delle $D_{j,t}$ unità prodotte.
- L'ultimo è il costo di mantenimento a scorta: alla fine di ogni periodo $k \in \{j, \ldots, t-1\}$ la scorta contiene le domande $D_{k+1,t}$ dei periodi successivi del lotto.

Calcolare i costi uno dopo l'altro

- Per un periodo j fissato, i costi $c_{j,j}, c_{j,j+1}, \ldots, c_{j,m}$ si possono calcolare uno dopo l'altro.
- Un'unità usata nel periodo t e prodotta nel periodo j costa

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

quindi $u_{j,j} = vc_j$ e $u_{j,t} = u_{j,t-1} + hc_{t-1}$.

- Aggiungere il periodo t al lotto aggiunge d_t unità di costo $u_{j,t}$:

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

Vediamo ora lo pseudocodice che calcola la matrice dei costi dei lotti di produzione.

Algorithm: CostMatrix

Input : Un'istanza dell'ULS data dal numero m di periodi e dai vettori $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$

Output : La matrice dei costi dei lotti di produzione

```
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
```

Come funziona CostMatrix

- La variabile u è il costo unitario $u_{j,t}$.
- La variabile g è il costo $c_{j,t}$ del lotto corrente.
- Si usano solo le celle $c[j][t]$ con $j \leq t$.

Esempio: calcolo dei costi dei lotti di produzione (1/6)

Consideriamo l'istanza dell'ULS dell'esempio precedente:

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

- Per ogni passo, le tabelle seguenti riportano i valori di j e t, il valore corrente di u, la cella $c[j][t]$ e l'operazione eseguita.
- La cella $c[j][t]$ è evidenziata in blu quando viene assegnata.

Esempio: calcolo dei costi dei lotti di produzione (2/6)

j	t	u	$c[j][t]$	Azione
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	$u \leftarrow u+hc[4] = 3+1 = 4$, $g \leftarrow g+u \cdot d[5] = 240+4 \cdot 10 = 280$, $c[1][5] \leftarrow 280$

Esempio: calcolo dei costi dei lotti di produzione (3/6)

j	t	u	$c[j][t]$	Azione
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	$u \leftarrow u+hc[4] = 2+1 = 3$, $g \leftarrow g+u \cdot d[5] = 160+3 \cdot 10 = 190$, $c[2][5] \leftarrow 190$

Esempio: calcolo dei costi dei lotti di produzione (4/6)

j	t	u	$c[j][t]$	Azione
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	$u \leftarrow u+hc[4] = 1+1 = 2$, $g \leftarrow g+u \cdot d[5] = 90+2 \cdot 10 = 110$, $c[3][5] \leftarrow 110$

Esempio: calcolo dei costi dei lotti di produzione (5/6)

j	t	u	$c[j][t]$	Azione
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	$u \leftarrow vc[5] = 0$, $g \leftarrow sc[5]+vc[5] \cdot d[5] = 50+0 \cdot 10 = 50$, $c[5][5] \leftarrow 50$

Esempio: calcolo dei costi dei lotti di produzione (6/6)

Questa è la matrice c dopo l'esecuzione dell'algoritmo:

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

Per esempio, per la formula del costo di un lotto di produzione:

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

I due passi dell'algoritmo di PD

- Passo 1: calcolo del valore ottimo
  - una formula ricorsiva dà il valore ottimo $v(t)$ dei primi t periodi, scegliendo l'ultimo lotto di produzione
- Passo 2: calcolo di una soluzione ottima
  - si memorizza il periodo dell'ultimo lotto scelto, e un piano ottimo si legge all'indietro

Passo 1: le sottoistanze

Per ogni $t \in \{1, 2, \ldots, m\}$, sia $v(t)$ il valore ottimo della sottoistanza dell'ULS data da:

1. un insieme $\{1, 2, \ldots, t\}$ di t periodi di tempo,
2. per ogni periodo $j \in \{1, 2, \ldots, t\}$, una domanda d_j,
3. per ogni periodo $j \in \{1, 2, \ldots, t\}$, un costo di setup sc_j, un costo variabile di produzione vc_j e un costo di mantenimento a scorta hc_j.

Passo 1: la formula ricorsiva

La formula ricorsiva per calcolare $v(t)$ è la seguente:

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

Leggere la formula ricorsiva

- La sottoistanza con i periodi $\{1, \ldots, t\}$ è un'istanza dell'ULS dello stesso tipo: la sua scorta deve essere vuota alla fine del periodo t.
- Qui $v(0) = 0$, perché un insieme vuoto di periodi non ha domanda né costo.
- Il processo di decomposizione ricorsiva della formula genera t **sottoproblemi indipendenti**, ciascuno corrispondente a una possibile scelta del periodo j dell'**ultimo lotto di produzione**, che produce le domande dei periodi $j, \ldots, t$.

Sottostruttura ottima

- Per la proprietà zero-inventory ordering applicata alla sottoistanza, la scorta è vuota alla fine del periodo $j-1$, e il sottoproblema residuo è la sottoistanza dell'ULS con i periodi $\{1,\ldots,j-1\}$.
- Il valore ottimo $v(t)$ si ottiene provando tutti i possibili periodi dell'ultimo lotto di produzione e scegliendo quello che minimizza la somma $v(j-1)+c_{j,t}$ del valore ottimo del sottoproblema residuo e del costo dell'ultimo lotto.
- Quindi una soluzione ottima per i periodi $\{1,\ldots,t\}$ si costruisce combinando l'ultimo lotto di produzione scelto nel periodo j con una soluzione ottima del sottoproblema residuo con i periodi $\{1,\ldots,j-1\}$.

$\rightsquigarrow$ **Sottostruttura ottima verificata!**

Sottoproblemi sovrapposti

- Scelte diverse dell'ultimo lotto di produzione, fatte eventualmente per valori diversi di t, possono portare allo stesso insieme residuo di periodi.
- Quindi lo stesso valore $v(t)$ può dover essere calcolato più volte durante la valutazione ricorsiva della formula.

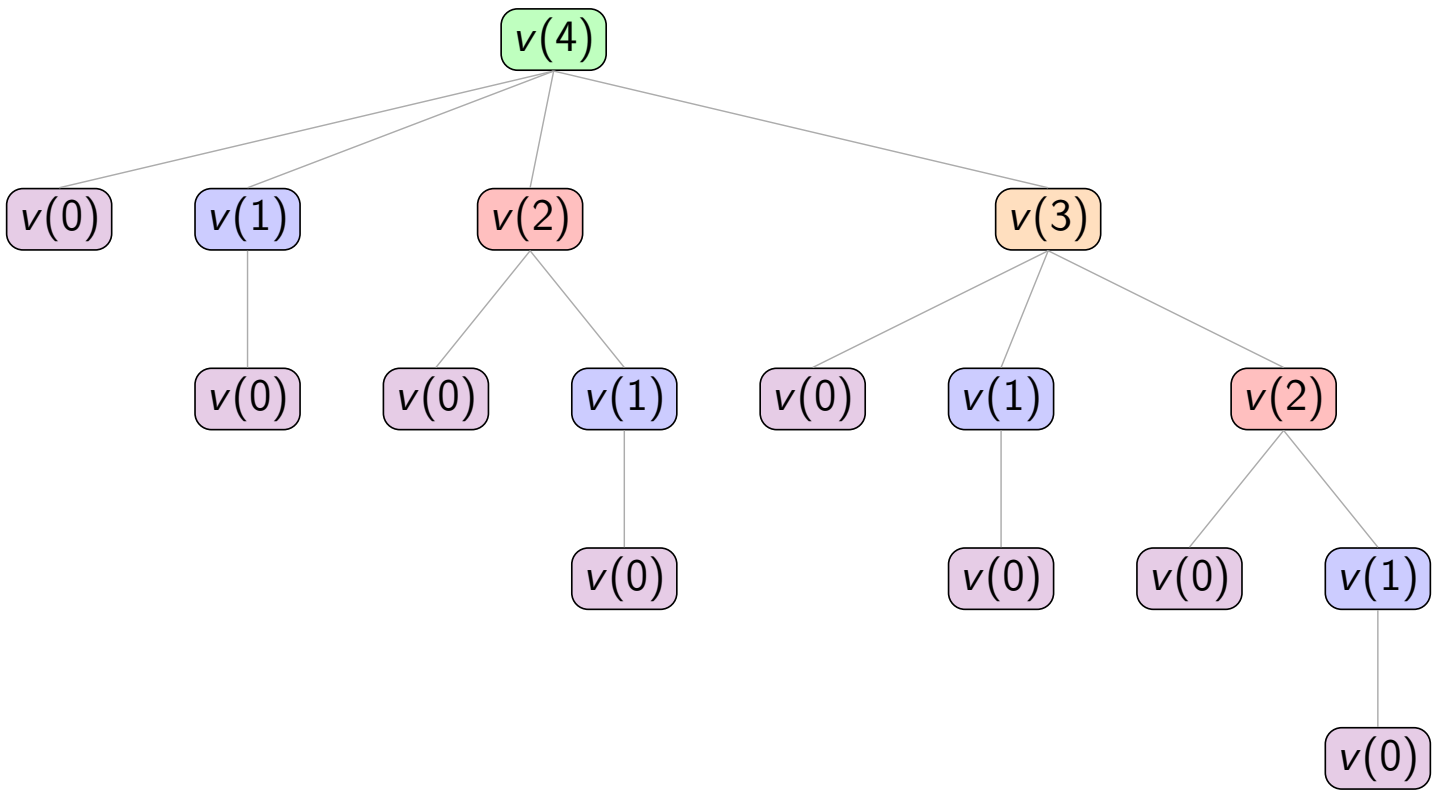
$\rightsquigarrow$ **Sottoproblemi sovrapposti verificati!**

L'approccio top-down

- Un **primo metodo** per calcolare il valore ottimo $v(m)$ dell'istanza dell'ULS è l'**approccio top-down**.
- Il metodo parte dalla chiamata con $t = m$ e valuta ricorsivamente la formula: ogni chiamata per i periodi $\{1, \ldots, t\}$ genera altre chiamate per i periodi $\{1, \ldots, j-1\}$, per ogni $j \in \{1, 2, \ldots, t\}$, finché non si raggiunge il caso base $v(0)$.
- L'**albero di ricorsione** della funzione v ha un nodo per ogni chiamata ricorsiva, etichettato con la chiamata che rappresenta:
  - la radice rappresenta la chiamata iniziale;
  - i figli di un nodo rappresentano le chiamate ricorsive effettuate da quella chiamata;
  - le foglie rappresentano i casi base.

Esempio: albero di ricorsione (1/2)

Consideriamo i primi quattro periodi dell'istanza dell'ULS dell'esempio (con cinque periodi l'albero ha 32 nodi). Questo è l'albero di ricorsione dell'approccio top-down per calcolare $v(4)$:



Esempio: albero di ricorsione (2/2)

- I figli di un nodo $v(t)$ sono ordinati secondo $j \in \{1, \ldots, t\}$: il figlio $v(j-1)$ corrisponde a un ultimo lotto di produzione nel periodo j.
- La funzione v viene chiamata 16 volte, e gli stessi sottoproblemi vengono risolti più volte:
  - $v(2)$ viene chiamata 2 volte,
  - $v(1)$ viene chiamata 4 volte,
  - $v(0)$ viene chiamata 8 volte.
- Tutte le altre chiamate vengono effettuate una sola volta.

Calcolare i valori in ordine crescente

Il valore $v(t)$, per un periodo generico t,
si può calcolare una volta noti i valori $v(j)$
per tutti i periodi minori $j < t$.

- Vediamo ora il calcolo di $v(0), v(1), \ldots, v(5)$ in quest'ordine sull'esempio.

Esempio: calcolo dei valori della funzione v (1/3)

Consideriamo l'istanza dell'ULS dell'esempio, con la matrice c dei costi dei lotti di produzione:

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

Il valore della funzione v si può calcolare partendo dal caso base $v(0)=0$ e calcolando poi i valori di $v(1)$, $v(2)$, $v(3)$, $v(4)$ e $v(5)$ nell'ordine.

Esempio: calcolo dei valori della funzione v (2/3)

- Poiché conosciamo il valore di $v(0)$, possiamo calcolare $v(1)$ come segue:

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

- Calcolati $v(1)$ e $v(0)$, possiamo calcolare $v(2)$ come segue:

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

- Calcolati $v(2)$, $v(1)$ e $v(0)$, possiamo calcolare $v(3)$ come segue:

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

Esempio: calcolo dei valori della funzione v (3/3)

- Calcolati $v(3)$, $v(2)$, $v(1)$ e $v(0)$, possiamo calcolare $v(4)$ come segue:

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

- Calcolati $v(4)$, $v(3)$, $v(2)$, $v(1)$ e $v(0)$, possiamo calcolare $v(5)$ come segue:

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

La tabella di lookup

- Possiamo evitare di risolvere ripetutamente gli stessi sottoproblemi memorizzando il valore di ciascun sottoproblema appena viene calcolato.

- I valori sono memorizzati in una **tabella di lookup**, il vettore $v[0..m]$, la cui cella $v[t]$ contiene $v(t)$.

- Poiché la soluzione di un sottoproblema con valore $v(t)$ dipende solo dai sottoproblemi con valori $v(j-1)$, con $j-1 < t$, possiamo risolvere i sottoproblemi per valori crescenti di t, partendo dal caso base $v(0) = 0$ fino ad arrivare a $v(m)$.

- Questa osservazione porta in modo naturale a un **metodo molto più efficiente** per calcolare il valore ottimo $v(m)$ dell'istanza dell'ULS.

L'approccio bottom-up

- Un **secondo metodo** per calcolare il valore ottimo $v(m)$ dell'istanza dell'ULS è l'**approccio bottom-up**.
- Il metodo parte dal caso base $v(0) = 0$ e valuta la formula per periodi crescenti $t \in \{1, 2, \ldots, m\}$.
- Ogni valore $v(t)$ si calcola usando i valori già calcolati $v(0), v(1), \ldots, v(t-1)$, fino a ottenere il valore $v(m)$ dell'istanza dell'ULS.

Vediamo ora lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**.

Algorithm: DP_WW (1/2)

Input : Un'istanza dell'ULS data dal numero m di periodi e dai vettori $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$

Output : Il valore ottimo

```
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
```

Come funziona l'algoritmo di PD (1/2)

- La matrice c contiene i costi dei lotti di produzione, calcolati dall'algoritmo CostMatrix.
- Il vettore $v[0..m]$ è la **tabella di lookup**: memorizza i valori ottimi di tutti i sottoproblemi, con il caso base $v[0] = 0$.
- Il ciclo esterno, su $t \in \{1, 2, \ldots, m\}$, calcola $v[t]$ in ordine crescente dell'ultimo periodo.

Come funziona l'algoritmo di PD (2/2)

- Il ciclo interno, su $j \in \{1, 2, \ldots, t\}$, valuta tutti i possibili periodi dell'ultimo lotto di produzione e conserva in b il valore minimo $v[j-1] + c[j][t]$.
- La variabile b parte da $+\infty$, quindi il primo confronto la aggiorna sempre.
- Al termine del ciclo interno, $v[t] \leftarrow b$ memorizza il valore ottimo $v[t]$.
- Infine, viene restituito il valore ottimo $v[m]$ dell'istanza dell'ULS.

Esempio: esecuzione dell'algoritmo di PD (1/6)

Consideriamo l'istanza dell'ULS dell'esempio, con la matrice c dei costi dei lotti di produzione:

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

- Per ogni passo, le tabelle seguenti riportano i valori di t e j, il valore corrente di b e la cella $v[t]$.
- La cella di b è evidenziata in rosso quando viene aggiornata; la cella di $v[t]$ è evidenziata in blu quando viene assegnata.

Esempio: esecuzione dell'algoritmo di PD (2/6)

t	j	$v[t]$	b	Azione
–	–	0	–	$v[0] \leftarrow 0$
1	–	–	$+\infty$	$b \leftarrow +\infty$
	1	–	50	$v[0]+c[1][1] = 0+50 = 50 < +\infty$: $b \leftarrow 50$
	–	50	50	$v[1] \leftarrow b = 50$
2	–	–	$+\infty$	$b \leftarrow +\infty$
	1	–	60	$v[0]+c[1][2] = 0+60 = 60 < +\infty$: $b \leftarrow 60$
	2	–	60	$v[1]+c[2][2] = 50+50 = 100 > 60$: nessun aggiornamento
	–	60	60	$v[2] \leftarrow b = 60$

Esempio: esecuzione dell'algoritmo di PD (3/6)

t	j	$v[t]$	b	Azione
	–	–	$+\infty$	$b \leftarrow +\infty$
3	1	–	120	$v[0]+c[1][3] = 0+120 = 120 < +\infty$: $b \leftarrow 120$
	2	–	120	$v[1]+c[2][3] = 50+80 = 130 > 120$: nessun aggiornamento
	3	–	110	$v[2]+c[3][3] = 60+50 = 110 < 120$: $b \leftarrow 110$
	–	110	110	$v[3] \leftarrow b = 110$

Esempio: esecuzione dell'algoritmo di PD (4/6)

t	j	$v[t]$	b	Azione
	–	–	$+\infty$	$b \leftarrow +\infty$
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	$v[3]+c[4][4] = 110+50 = 160 > 150$: nessun aggiornamento
	–	150	150	$v[4] \leftarrow b = 150$

Esempio: esecuzione dell'algoritmo di PD (5/6)

t	j	$v[t]$	b	Azione
	–	–	$+\infty$	$b \leftarrow +\infty$
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	$v[3]+c[4][5] = 110+60 = 170 = b$: nessun aggiornamento
	5	–	170	$v[4]+c[5][5] = 150+50 = 200 > 170$: nessun aggiornamento
	–	170	170	$v[5] \leftarrow b = 170$

Esempio: esecuzione dell'algoritmo di PD (6/6)

Questo è il vettore v dopo l'esecuzione dell'algoritmo di PD:

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

- Dopo l'esecuzione dell'algoritmo di PD, il vettore v contiene il valore ottimo $v[t]$ per ogni insieme di periodi $\{1, \ldots, t\}$.
- Il valore ottimo dell'istanza dell'ULS è $v[5] = 170$.

Osservazione: un cammino minimo

- La formula ricorsiva di $v(t)$ calcola un cammino minimo in una rete con:
  - nodi $0, 1, \ldots, m$,
  - un arco dal nodo $j-1$ al nodo t, di costo $c_{j,t}$, per ogni $j \leq t$.
- Il valore $v(t)$ è la lunghezza di un cammino minimo dal nodo 0 al nodo t.
- È la formulazione a cammino minimo del capitolo «Formulazioni dei problemi di lot sizing», senza le capacità: un arco è un lotto di produzione.

Passo 2: calcolo di una soluzione ottima

- Per calcolare una soluzione ottima è necessario memorizzare alcune informazioni aggiuntive.

- Per ogni periodo t, oltre al valore ottimo $v[t]$, memorizziamo $r[t]$, il periodo dell'ultimo lotto di produzione in una soluzione ottima per i periodi $\{1, \ldots, t\}$.

- Partendo da $t = m$, il valore $r[t]$ dà il periodo dell'ultimo lotto di produzione, che produce $D_{r[t],t}$ unità; i periodi rimanenti sono $\{1, \ldots, r[t] - 1\}$, e la stessa regola si può applicare di nuovo, e così via, finché non resta alcun periodo.

Vediamo ora lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**,
con la memorizzazione delle informazioni necessarie a calcolare una soluzione ottima.

Algorithm: DP_WW_InfoSol (1/2)

Input : Un'istanza dell'ULS data dal numero m di periodi e dai vettori $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$

Output : Il valore ottimo e il vettore delle informazioni $r[1..m]$

```
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
```

Come si memorizzano le informazioni

- L'algoritmo di PD usa la variabile temporanea aggiuntiva a per tenere traccia della decisione presa quando b viene aggiornato.
- Se viene scelto l'ultimo lotto di produzione nel periodo j, $a \leftarrow j$ memorizza questa decisione.
- Al termine del ciclo interno, $r[t] \leftarrow a$ memorizza il periodo dell'ultimo lotto di produzione scelto, insieme al valore ottimo $v[t] \leftarrow b$.
- A causa del confronto stretto, in caso di pareggio si memorizza il periodo j minimo.

Esempio: calcolo delle informazioni aggiuntive (1/6)

Consideriamo l'istanza dell'ULS dell'esempio, con la matrice c dei costi dei lotti di produzione:

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

- Per ogni passo, le tabelle seguenti riportano i valori di t e j, la cella $r[t]$, il valore corrente di b e il valore corrente di a.
- La cella di b è evidenziata in rosso quando viene aggiornata; le celle di a e di $r[t]$ sono evidenziate in blu quando vengono aggiornate.

Esempio: calcolo delle informazioni aggiuntive (2/6)

t	j	$r[t]$	b	a	Azione
–	–	–	–	–	Stato iniziale
	–	–	$+\infty$	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	$v[1]+c[2][2] = 50+50 = 100 > 60$: nessun aggiornamento
	–	1	60	1	$v[2] \leftarrow b = 60$, $r[2] \leftarrow a = 1$

Esempio: calcolo delle informazioni aggiuntive (3/6)

t	j	$r[t]$	b	a	Azione
	–	–	$+\infty$	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	$v[1]+c[2][3] = 50+80 = 130 > 120$: nessun aggiornamento
	3	–	110	3	$v[2]+c[3][3] = 60+50 = 110 < 120$: $b \leftarrow 110$, $a \leftarrow 3$
	–	3	110	3	$v[3] \leftarrow b = 110$, $r[3] \leftarrow a = 3$

Esempio: calcolo delle informazioni aggiuntive (4/6)

t	j	$r[t]$	b	a	Azione
	–	–	$+\infty$	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	$v[3]+c[4][4] = 110+50 = 160 > 150$: nessun aggiornamento
	–	3	150	3	$v[4] \leftarrow b = 150$, $r[4] \leftarrow a = 3$

Esempio: calcolo delle informazioni aggiuntive (5/6)

t	j	$r[t]$	b	a	Azione
	–	–	$+\infty$	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	$v[3]+c[4][5] = 110+60 = 170 = b$: nessun aggiornamento
	5	–	170	3	$v[4]+c[5][5] = 150+50 = 200 >$ 170: nessun aggiornamento
	–	3	170	3	$v[5] \leftarrow b = 170$, $r[5] \leftarrow a = 3$

Esempio: calcolo delle informazioni aggiuntive (6/6)

Questo è il vettore r dopo l'esecuzione dell'algoritmo di PD:

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

- Per esempio, $r[5] = 3$: in una soluzione ottima per i periodi $\{1, \ldots, 5\}$ l'ultimo lotto di produzione è nel periodo 3.

Calcolare una soluzione ottima dal vettore r

- Calcolato il vettore r, si può calcolare una soluzione ottima dell'istanza dell'ULS applicando ripetutamente la regola seguente, partendo da $t = m$ e da $x_j = 0$ per ogni $j \in \{1, 2, \ldots, m\}$:

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

  finché $t = 0$.

- Si ottiene così il vettore $(x_1, x_2, \ldots, x_m)$, in cui ogni componente x_j è il numero di unità prodotte nel periodo j in una soluzione ottima.

Lo pseudocodice seguente implementa il calcolo di una soluzione ottima, dato il vettore r.

Algorithm: DP_WW_SolComputation

Input : Il numero m di periodi e le domande $d[1..m]$ dell'istanza dell'ULS, e il vettore $r[1..m]$

Output : Una soluzione ottima dell'istanza dell'ULS

```
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
```

Esempio: calcolo di una soluzione ottima (1/2)

Dopo l'esecuzione dell'algoritmo di PD sull'istanza dell'esempio, abbiamo il seguente vettore r:

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

Possiamo calcolare una soluzione ottima per $m=5$ periodi come segue:

Iterazione	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	$t=1-1=0$

Esempio: calcolo di una soluzione ottima (2/2)

- $t = 5$, $r[5] = 3$: l'ultimo lotto di produzione dei periodi $1, \ldots, 5$ è nel periodo 3 e copre le domande dei periodi $3, \ldots, 5$, quindi $x[3] \leftarrow d_3 + d_4 + d_5 = 30 + 40 + 10 = 80$
- $t = 2$, $r[2] = 1$: l'ultimo lotto di produzione dei periodi $1, \ldots, 2$ è nel periodo 1 e copre le domande dei periodi $1, \ldots, 2$, quindi $x[1] \leftarrow d_1 + d_2 = 20 + 10 = 30$

Una soluzione ottima è il vettore:

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

Il valore ottimo corrispondente è $v[5] = 170$, lo stesso valore ottenuto dall'enumerazione delle soluzioni con la proprietà zero-inventory ordering e da Gurobi sul modello $\mathrm{ILP}_{\mathrm{ULS}}$.

Osservazione: pareggi

- Nel calcolo di $v[5]$, il lotto nel periodo 4 dà lo stesso valore del lotto nel periodo 3, ma il confronto stretto conserva $j = 3$.
- Scegliendo invece $j = 4$ si ottiene l'altra soluzione ottima $(30, 0, 30, 50, 0)$ dell'enumerazione.
- L'algoritmo di PD restituisce una soluzione ottima, non tutte.

Numero di passi

Quanti passi esegue l'algoritmo di PD?

Contiamo prima i passi di CostMatrix,
poi quelli di DP_WW, che lo chiama.

Richiamiamo i due pseudocodici
prima di contare le esecuzioni delle loro righe.

Algorithm: CostMatrix

Input : Un'istanza dell'ULS data dal numero m di periodi e dai vettori $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$

Output : La matrice dei costi dei lotti di produzione

```
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
```

Numero di passi di CostMatrix

Indichiamo con $S_c(m)$ il numero di passi eseguiti da CostMatrix nel caso peggiore, dove m è il numero di periodi. Il numero di esecuzioni delle sue righe è:

- La riga 2 viene eseguita 1 volta.
- Le righe 3, 4, 5 e 6 vengono eseguite m volte ciascuna.
- Le righe 7, 8, 9 e 10 vengono eseguite $\sum_{j=1}^{m}(m-j) = \frac{m(m-1)}{2}$ volte ciascuna.
- La riga 13 viene eseguita 1 volta.

Quindi

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

Algorithm: DP_WW (1/2)

Input : Un'istanza dell'ULS data dal numero m di periodi e dai vettori $d[1..m]$, $sc[1..m]$, $vc[1..m]$, $hc[1..m]$

Output : Il valore ottimo

```
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
```

Numero di passi di DP_WW (1/3)

Indichiamo con $S(m)$ il numero di passi eseguiti da DP_WW nel caso peggiore.

- La riga 2 esegue CostMatrix, cioè $S_c(m)$ passi.
- Nel caso peggiore, la condizione alla riga 8 è vera a ogni iterazione del ciclo interno, e quindi la riga 9 viene eseguita tante volte quanto la riga 8.

Numero di passi di DP_WW (2/3)

Il numero di esecuzioni delle altre righe è:

- La riga 3 viene eseguita 1 volta.
- La riga 4 viene eseguita 1 volta.
- La riga 5 viene eseguita m volte.
- La riga 6 viene eseguita m volte.
- La riga 7 viene eseguita $\sum_{t=1}^{m} t$ volte.
- La riga 8 viene eseguita $\sum_{t=1}^{m} t$ volte.
- La riga 9 viene eseguita $\sum_{t=1}^{m} t$ volte.
- La riga 12 viene eseguita m volte.
- La riga 14 viene eseguita 1 volta.

Numero di passi di DP_WW (3/3)

Quindi il numero totale di passi nel caso peggiore è

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

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

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

Di conseguenza,

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

Infatti,

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

Il numero di passi dell'algoritmo di PD

L'algoritmo di PD esegue

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

passi nel caso peggiore.

- DP_WW_InfoSol aggiunge solo un numero costante di operazioni negli stessi cicli, e quindi il suo numero di passi ha lo stesso ordine di crescita.
- L'enumerazione delle soluzioni con la proprietà zero-inventory ordering elenca invece 2^{m-1} soluzioni.

Riepilogo (1/2)

Lotti di produzione:

- un lotto produce nel periodo j le domande dei periodi $j, \ldots, t$, al costo $c_{j,t} = sc_j + vc_j\, D_{j,t} + \sum_{k=j}^{t-1} hc_k\, D_{k+1,t}$
- CostMatrix calcola tutti i costi in modo incrementale, in $2m^2 + 2m + 2$ passi

Passo 1: il valore ottimo:

- $v(t) = \min\{v(j-1) + c_{j,t} : j \in \{1, 2, \ldots, t\}\}$, con $v(0) = 0$: scelta dell'ultimo lotto
- sottostruttura ottima e sottoproblemi sovrapposti: bottom-up nella tabella di lookup $v[0..m]$
- la ricorrenza è un cammino minimo dal nodo 0 al nodo m

Riepilogo (2/2)

Passo 2: una soluzione ottima:

- $r[t]$ memorizza il periodo dell'ultimo lotto; in caso di pareggio, il periodo minimo
- il piano si legge all'indietro: $x_{r[t]} \leftarrow D_{r[t],t}$, $t \leftarrow r[t] - 1$
- sull'esempio: $(30, 0, 80, 0, 0)$, con valore ottimo $v[5] = 170$

Numero di passi:

- $S(m) = \frac{7m^2+13m+10}{2} = \Theta(m^2)$ per DP_WW
- lo stesso ordine di crescita per DP_WW_InfoSol

Errori tipici

Introduzione: errori tipici

- Raccogliamo gli errori più frequenti con l'algoritmo di PD per l'ULS
- Due riguardano la proprietà zero-inventory ordering e il costo di un lotto
- Due riguardano le sottoistanze e i piani ottimi
- Conviene controllarli prima di eseguire a mano l'algoritmo di PD

Errori tipici (1/2)

Osservazione 1: Applicare la proprietà zero-inventory ordering con le capacità

La dimostrazione sposta la produzione fra due periodi senza limiti: con le capacità un periodo può produrre mentre ci sono unità a scorta.

Osservazione 2: Calcolare da zero il costo di un lotto

Calcolare ogni $c_{j,t}$ dalla sua definizione costa $O(m^3)$ passi dentro l'algoritmo di PD; il calcolo incrementale di CostMatrix mantiene $\Theta(m^2)$.

Errori tipici (2/2)

Osservazione 3: Dimenticare la scorta finale vuota

La sottoistanza con i periodi $\{1, \ldots, t\}$ termina con la scorta vuota: per questo il lotto nel periodo j produce esattamente $D_{j,t}$ unità.

Osservazione 4: Prendere un piano ottimo per l'unico

L'esempio del capitolo ha due piani ottimi: l'algoritmo di PD restituisce quello con il periodo j minimo, perché aggiorna solo con un valore strettamente minore.

Riepilogo

Proprietà e costi:

- la proprietà zero-inventory ordering vale senza capacità
- i costi $c_{j,t}$ si calcolano in modo incrementale, in $\Theta(m^2)$ passi

Sottoistanze e piani:

- ogni sottoistanza termina con la scorta vuota, quindi un lotto produce esattamente $D_{j,t}$
- l'algoritmo di PD restituisce un piano ottimo, non tutti

In sintesi

Introduzione: in sintesi

- Raccogliamo le idee chiave del capitolo sull'algoritmo di PD per l'ULS
- Il quadro riassuntivo raccoglie il risultato sull'esempio del capitolo

Riepilogo (1/2)

- Se tutti i costi sono non negativi, il problema senza capacità ha un piano ottimo in cui un periodo produce solo quando la scorta è nulla (Wagner e Whitin), quindi $x_j = D_{j,t}$ per ogni lotto di produzione.
- Il costo $c_{j,t}$ di ogni lotto si calcola in $\Theta(m^2)$ passi, e la ricorrenza $v(t) = \min_j\{v(j-1) + c_{j,t}\}$ dà il valore ottimo in $\Theta(m^2)$ passi; è un cammino minimo.
- La proprietà zero-inventory ordering si basa su una produzione senza limiti: con le capacità non vale.

Riepilogo (2/2)

Quadro riassuntivo dell'esempio.

Istanza	Lower bound	Upper bound	Origine
Esempio del capitolo (senza capacità, $m = 5$)	170	170	algoritmo di PD di Wagner e Whitin