PD per il problema dello zaino multidimensionale

Indice

Un algoritmo di PD
- Numero di passi
- La maledizione della dimensionalità

Errori tipici

In sintesi

Obiettivi

Al termine di questo capitolo saremo in grado di:

- scrivere la ricorrenza $v(j, k_1, \ldots, k_m)$ sugli oggetti e sul vettore delle capacità;
- eseguire a mano l'algoritmo di PD su un'istanza con $m = 2$ dimensioni, una matrice per ogni oggetto;
- ricostruire una soluzione ottima dalle tabelle s;
- contare i $\Theta(m\, n \prod_{d=1}^{m}(c_d + 1))$ passi e spiegare perché il numero di stati cresce esponenzialmente con m.

Prerequisiti

- Da questo modulo, il capitolo «PD per il problema dello zaino 0/1»: la ricorrenza $v(j, k)$ del problema dello zaino 0/1 (KP) su oggetti e capacità, la matrice s delle decisioni e la ricostruzione di una soluzione.

- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di zaino»: il problema dello zaino multidimensionale, con una capacità c_d e un peso w_{dj} per ogni dimensione d.

- Dal modulo *Algoritmi*: le convenzioni dello pseudocodice (Capitolo 2) e la dimensione dell'input (Capitolo 11).

Il problema di questo capitolo

Il problema di questo capitolo è il
problema dello zaino multidimensionale (MDKP):

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

- Estendiamo all'MDKP l'algoritmo di PD del KP: la capacità k diventa un vettore di capacità, una per ogni dimensione
- Vediamo perché il numero di stati cresce esponenzialmente con il numero di dimensioni

Un algoritmo di PD

Introduzione: Un algoritmo di PD

- In questa sezione scriviamo un algoritmo di PD per l'MDKP, in due passi: prima il valore ottimo, poi una soluzione ottima
- L'algoritmo segue l'algoritmo di PD del KP, ma la capacità diventa il vettore $\boldsymbol{k} = (k_1, k_2, \ldots, k_m)$: con $m = 2$ dimensioni, ogni riga della tabella è una matrice
- Lo eseguiamo a mano su un'istanza con quattro oggetti, contiamo i suoi passi e vediamo dove l'algoritmo di PD smette di essere utile: la maledizione della dimensionalità

Vettori dei pesi e vettori delle capacità

- Indichiamo con m il numero di dimensioni, quindi

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

perché nei capitoli di PD la lettera k indica una capacità.

- L'oggetto j ha il **vettore dei pesi**

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

e lo zaino ha il vettore delle capacità

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

Confronto fra vettori e soluzioni ammissibili

- Per due vettori, $\boldsymbol{w}_j \leq \boldsymbol{k}$ significa

$$w_{dj} \leq k_d \quad \text{per ogni dimensione } d \in \mathscr{D}.$$

- Una soluzione si descrive con il vettore binario $(x_1, x_2, \ldots, x_n)$, dove $x_j = 1$ se l'oggetto j è selezionato; è ammissibile se

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

Esempio: istanza dell'MDKP

Consideriamo uno zaino con $m = 2$ dimensioni e capacità $c_1 = 5$ e $c_2 = 4$, e un insieme $\mathscr{I} = \{1, 2, 3, 4\}$ di 4 oggetti ($n = 4$), con i seguenti profitti e pesi:

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

Usiamo questa istanza dell'MDKP come esempio di tutto il capitolo.

I due passi dell'algoritmo di PD

- Passo 1: calcolo del valore ottimo
  - una formula ricorsiva dà il valore ottimo di ogni sottoistanza, con un vettore di capacità
- Passo 2: calcolo di una soluzione ottima
  - le tabelle s delle decisioni danno una soluzione ottima, letta all'indietro

Passo 1: le sottoistanze

Per ogni $j \in \{0, 1, \ldots, n\}$ e ogni vettore $\boldsymbol{k} = (k_1, k_2, \ldots, k_m)$ con $k_d \in \{0, 1, \ldots, c_d\}$ per ogni $d \in \mathscr{D}$, sia

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

il valore ottimo della sottoistanza dell'MDKP data da:

1. uno zaino con capacità k_d in ogni dimensione $d \in \mathscr{D}$,
2. un insieme $\{1, 2, \ldots, j\}$ di oggetti disponibili,
3. per ogni oggetto $h \in \{1, 2, \ldots, j\}$, un profitto p_h e un vettore dei pesi $\boldsymbol{w}_h$.

Passo 1: la formula ricorsiva

La formula ricorsiva per calcolare $v(j, \boldsymbol{k})$ è la seguente:

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

Leggere la formula ricorsiva

- Con $m = 1$ la formula ricorsiva è la ricorrenza del KP: l'unica differenza è che la capacità k diventa il vettore $\boldsymbol{k}$, e «l'oggetto j entra nello zaino» significa $\boldsymbol{w}_j \leq \boldsymbol{k}$.

- I casi base sono

$$v(0, \boldsymbol{k}) = 0 \quad \text{per ogni vettore } \boldsymbol{k},$$

perché nessun oggetto è disponibile.

Sottostruttura ottima

- La formula ricorsiva genera al più 2 **sottoproblemi indipendenti**:
  - l'oggetto j non è selezionato, e il sottoproblema residuo ha lo stesso vettore $\boldsymbol{k}$, oppure
  - l'oggetto j è selezionato, e il sottoproblema residuo ha il vettore $\boldsymbol{k} - \boldsymbol{w}_j$.

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

Sottoproblemi sovrapposti e approccio bottom-up

- Decisioni diverse sugli oggetti $j+1, \ldots, n$ possono lasciare lo stesso vettore di capacità residue, e quindi lo stesso valore $v(j, \boldsymbol{k})$ può servire più volte.

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

- Il valore $v(j, \boldsymbol{k})$ dipende solo da valori della riga $j-1$.
- Quindi l'**approccio bottom-up** calcola le righe $j = 0, 1, \ldots, n$ in quest'ordine, e in ogni riga tutti i vettori $\boldsymbol{k}$ in un ordine qualsiasi.

Lo stato dell'algoritmo di PD

- Lo **stato** dell'algoritmo di PD è la coppia $(j, \boldsymbol{k})$.
- Per ogni j la tabella ha una cella per ogni vettore $\boldsymbol{k}$, cioè

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

- Con $m = 2$ la riga j è una matrice, con una riga per ogni k_1 e una colonna per ogni k_2.

Passo 2: calcolo di una soluzione ottima

- Per ogni oggetto j e vettore $\boldsymbol{k}$ memorizziamo:
  - $s[j][\boldsymbol{k}] = 1$ se l'oggetto j è selezionato nella soluzione ottima trovata per $(j, \boldsymbol{k})$,
  - $s[j][\boldsymbol{k}] = 0$ altrimenti.

- L'insieme di tutti i vettori $\boldsymbol{k}$ è

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

Segue lo pseudocodice dell'**algoritmo di PD con l'approccio bottom-up**,
che memorizza le tabelle s insieme alle tabelle v.

Algorithm: MDKP_DP (1/2)

Input : Un'istanza dell'MDKP data da n, m, dal vettore $p[1..n]$, dai vettori dei pesi $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$ e dal vettore delle capacità $\boldsymbol{c}$

Output: Il valore ottimo dell'istanza dell'MDKP e le tabelle $s[j][\boldsymbol{k}]$

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

Algorithm: MDKP_DP (2/2)

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

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

Come funziona MDKP_DP

- La struttura è quella dell'algoritmo di PD del KP; il ciclo sulle capacità diventa un ciclo su tutti i vettori $\boldsymbol{k} \in \mathscr{C}$.
- Il test $\boldsymbol{w}_j \leq \boldsymbol{k}$ confronta m coppie di numeri, e il vettore $\boldsymbol{k} - \boldsymbol{w}_j$ richiede m sottrazioni.
- In caso di pareggio l'oggetto j non viene selezionato, perché il valore viene aggiornato solo quando è strettamente maggiore.

Esempio: esecuzione dell'algoritmo di PD (1/7)

- Consideriamo l'istanza dell'MDKP dell'esempio precedente, con capacità $c_1 = 5$ e $c_2 = 4$.
- La riga $j = 0$ è $v[0][\boldsymbol{k}] = 0$ per ogni $\boldsymbol{k}$.
- Per ogni oggetto j calcoliamo la matrice $v[j][k_1][k_2]$, con:
  - una riga per ogni $k_1 \in \{0, 1, \ldots, 5\}$,
  - una colonna per ogni $k_2 \in \{0, 1, \ldots, 4\}$.

Esempio: esecuzione dell'algoritmo di PD (2/7)

Questi sono i calcoli della cella $\boldsymbol{k} = \boldsymbol{c} = (5, 4)$ per ogni oggetto, e delle celle usate dalla ricostruzione (oggetti 1 e 2):

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

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

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

Esempio: esecuzione dell'algoritmo di PD (3/7)

Questi sono i calcoli della cella $\boldsymbol{k} = \boldsymbol{c} = (5, 4)$ per ogni oggetto, e delle celle usate dalla ricostruzione (oggetti 3 e 4):

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

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

Esempio: esecuzione dell'algoritmo di PD (4/7)

Oggetto $j = 1$, con $p_1 = 9$ e $\boldsymbol{w}_1 = (1, 2)$:

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

Esempio: esecuzione dell'algoritmo di PD (5/7)

Oggetto $j = 2$, con $p_2 = 8$ e $\boldsymbol{w}_2 = (4, 1)$:

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

Esempio: esecuzione dell'algoritmo di PD (6/7)

Oggetto $j = 3$, con $p_3 = 3$ e $\boldsymbol{w}_3 = (2, 1)$:

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

Esempio: esecuzione dell'algoritmo di PD (7/7)

Oggetto $j = 4$, con $p_4 = 9$ e $\boldsymbol{w}_4 = (3, 4)$:

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

Esempio: le tabelle s (1/4)

Per l'istanza dell'esempio precedente, queste sono le matrici $s[j][k_1][k_2]$ calcolate dall'algoritmo di PD; un 1 indica che l'oggetto j è selezionato nella soluzione ottima trovata per (j, k_1, k_2).

Oggetto $j = 1$, con $p_1 = 9$ e $\boldsymbol{w}_1 = (1, 2)$:

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

Esempio: le tabelle s (2/4)

Oggetto $j = 2$, con $p_2 = 8$ e $\boldsymbol{w}_2 = (4, 1)$:

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

Esempio: le tabelle s (3/4)

Oggetto $j = 3$, con $p_3 = 3$ e $\boldsymbol{w}_3 = (2, 1)$:

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

Esempio: le tabelle s (4/4)

Oggetto $j = 4$, con $p_4 = 9$ e $\boldsymbol{w}_4 = (3, 4)$:

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

Ricostruzione di una soluzione ottima

- Calcolate le tabelle s, si ricostruisce una soluzione ottima:
  - scorrendo gli oggetti da $j = n$ fino a $j = 1$,
  - partendo da $\boldsymbol{k} = \boldsymbol{c}$ e da $x_j = 0$ per ogni oggetto j:

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

Algorithm: MDKP_Reconstruct

Input : Il numero n di oggetti, il vettore delle capacità $\boldsymbol{c}$, i vettori dei pesi $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$ e le tabelle s

Output: Una soluzione ottima dell'istanza dell'MDKP

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

Esempio: calcolo di una soluzione ottima (1/3)

Dopo l'esecuzione dell'algoritmo di PD sull'istanza dell'esempio precedente, partiamo da $j = 4$ e $\boldsymbol{k} = (5, 4)$. I valori del vettore x a ogni iterazione:

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

Esempio: calcolo di una soluzione ottima (2/3)

L'azione di ogni iterazione, che aggiorna x e il vettore $\boldsymbol{k}$:

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

Esempio: calcolo di una soluzione ottima (3/3)

Il vettore $x = (1, 1, 0, 0)$ è una soluzione ottima dell'istanza dell'MDKP dell'esempio.

I suoi pesi sono $1 + 4 = 5 \leq c_1$ e $2 + 1 = 3 \leq c_2$, e il suo valore ottimo è $v(4, 5, 4) = 17$.

- Con la sola prima dimensione, il KP con capacità $c_1 = 5$ ha valore ottimo 18, raggiunto dagli oggetti $\{1, 4\}$; il loro peso totale nella seconda dimensione è $2 + 4 = 6 > c_2$.
- Le due dimensioni vanno controllate contemporaneamente, come fa la formula ricorsiva.

Quanti passi esegue l'algoritmo di PD?

Contiamo quante volte viene eseguita
ogni riga dello pseudocodice di MDKP_DP.

Numero di passi (1/3)

- Sia

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

  il numero di vettori $\boldsymbol{k}$.

- Indichiamo con $S(n, m, \boldsymbol{c})$ il numero di passi eseguiti da `MDKP_DP` nel caso peggiore.

- Le righe 9 e 10 lavorano su vettori di m componenti, quindi contiamo m passi per ciascuna delle loro esecuzioni.

Numero di passi (2/3)

- Le righe 2 e 14 vengono eseguite 1 volta ciascuna.
- Le righe 3 e 4 vengono eseguite N volte ciascuna.
- La riga 6 viene eseguita n volte.
- Le righe 7 e 8 vengono eseguite nN volte ciascuna.
- Le righe 9 e 10 vengono eseguite al più nN volte ciascuna, con m passi ciascuna.

Numero di passi (3/3)

Quindi il numero totale di passi nel caso peggiore è

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

e, poiché

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

abbiamo

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

Il numero di passi dell'algoritmo di PD

L'algoritmo di PD per l'MDKP esegue

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

passi nel caso peggiore.

Per un numero m fissato di dimensioni questo è

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

un numero pseudo-polinomiale di passi.

- La ricostruzione esegue $\Theta(m\, n)$ passi.

Il numero di passi contiene il prodotto delle capacità.

Che cosa succede quando il numero m di dimensioni cresce?

La maledizione della dimensionalità

- Ogni nuova dimensione moltiplica il numero di stati per $c_d + 1$.
- Se tutte le capacità sono uguali a c, la tabella ha

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

  celle: il numero di stati cresce **esponenzialmente** con m.
- La dimensione dell'input cresce solo linearmente con m: una dimensione in più aggiunge $n+1$ numeri, i pesi w_{dj} e la capacità c_d.
- Questa esplosione del numero di stati quando lo stato diventa un vettore si chiama **maledizione della dimensionalità**.

Quanto cresce la tabella

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

- Con $m = 1$ la tabella ha qualche migliaio di celle; con $m = 5$ ne ha più di $5 \cdot 10^{11}$, troppe per essere memorizzate.
- L'algoritmo di PD è quindi utile per l'MDKP solo quando m e le capacità sono piccoli.

Riepilogo (1/2)

Passo 1: il valore ottimo:

- la capacità k del KP diventa il vettore $\boldsymbol{k} = (k_1, \ldots, k_m)$
- $v(j, \boldsymbol{k}) = \max\{v(j-1, \boldsymbol{k}),\ p_j + v(j-1, \boldsymbol{k} - \boldsymbol{w}_j)\}$ se $\boldsymbol{w}_j \leq \boldsymbol{k}$, e $v(j, \boldsymbol{k}) = v(j-1, \boldsymbol{k})$ altrimenti
- casi base $v(0, \boldsymbol{k}) = 0$; bottom-up, riga per riga, tutti i vettori $\boldsymbol{k}$ in un ordine qualsiasi

Passo 2: una soluzione ottima:

- $s[j][\boldsymbol{k}] = 1$ se l'oggetto j è selezionato nella soluzione ottima trovata per $(j, \boldsymbol{k})$
- letta all'indietro a partire da $(n, \boldsymbol{c})$: $x_j \leftarrow s[j][\boldsymbol{k}]$, $\boldsymbol{k} \leftarrow \boldsymbol{k} - s[j][\boldsymbol{k}]\, \boldsymbol{w}_j$
- nell'esempio: $x = (1, 1, 0, 0)$, con valore ottimo 17

Riepilogo (2/2)

Numero di passi:

- $\Theta(m\,n\prod_{d=1}^{m}(c_d+1))$ per MDKP_DP, $\Theta(m\,n)$ per la ricostruzione
- per m fissato: $\Theta(n\prod_{d=1}^{m}c_d)$, pseudo-polinomiale

La maledizione della dimensionalità:

- con capacità tutte uguali a c, la tabella ha $(n+1)(c+1)^m$ celle: esponenziale in m
- la dimensione dell'input cresce solo linearmente con m
- l'algoritmo di PD è utile solo quando m e le capacità sono piccoli

Errori tipici

Introduzione: Errori tipici

- Raccogliamo gli errori più frequenti con l'algoritmo di PD per l'MDKP
- Due riguardano il vettore delle capacità, due il legame con il KP e il numero di passi
- Controlliamoli prima di eseguire a mano l'algoritmo di PD

Errori tipici (1/2)

Osservazione 1: Controllare una sola dimensione

L'oggetto j entra in $\boldsymbol{k}$ solo se $w_{dj} \leq k_d$ per ogni dimensione d; una dimensione non basta.

Osservazione 2: Aggiornare una sola capacità

Quando l'oggetto j è selezionato, tutte le componenti di $\boldsymbol{k}$ diminuiscono, di $\boldsymbol{w}_j$.

Errori tipici (2/2)

Osservazione 3: Risolvere un KP per dimensione

Il KP con la sola capacità c_d dà un upper bound, ma la sua soluzione ottima può violare un'altra dimensione.

Osservazione 4: Chiamare polinomiale l'algoritmo di PD

Il suo numero di passi contiene il prodotto delle capacità, che è esponenziale in m e nella dimensione delle capacità.

Riepilogo

Il vettore delle capacità:

- l'oggetto j entra solo se $\boldsymbol{w}_j \leq \boldsymbol{k}$, in ogni dimensione
- selezionare l'oggetto j fa diminuire tutte le componenti di $\boldsymbol{k}$

Il KP e il numero di passi:

- un KP per dimensione dà un upper bound, non una soluzione ammissibile
- l'algoritmo di PD è pseudo-polinomiale, non polinomiale

In sintesi

Introduzione: In sintesi

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

Riepilogo (1/2)

- La ricorrenza del KP si estende all'MDKP sostituendo la capacità k con il vettore $\boldsymbol{k} = (k_1, \ldots, k_m)$:

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

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

- La tabella ha $(n+1)\prod_{d=1}^{m}(c_d+1)$ celle e l'algoritmo di PD esegue $\Theta(m\,n\prod_{d=1}^{m}(c_d+1))$ passi, cioè $\Theta(n\prod_{d=1}^{m} c_d)$ per m fissato.

- Il numero di stati cresce esponenzialmente con m: la maledizione della dimensionalità.

- Le tabelle s danno una soluzione ottima, letta all'indietro a partire da $(n, \boldsymbol{c})$.

Riepilogo (2/2)

Quadro riassuntivo dell'esempio dell'MDKP ($\boldsymbol{c} = (5, 4)$, quattro oggetti):

Metodo	Lower bound	Upper bound	Origine
Algoritmo di PD	17	17	valore ottimo, soluzione $x = (1, 1, 0, 0)$