

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 $v(j, k_1, \dots, k_m)$ 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 $\Theta(mn \prod_{d=1}^m (c_d + 1))$ 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.
- ▶ Da questo modulo, 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 $\mathbf{k} = (k_1, k_2, \dots, k_m)$: 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

$$\mathcal{D} = \{1, 2, \dots, m\},$$

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

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

$$\mathbf{w}_j = (w_{1j}, w_{2j}, \dots, w_{mj}),$$

e lo zaino ha il vettore delle capacità

$$\mathbf{c} = (c_1, c_2, \dots, c_m).$$

Confronto fra vettori e soluzioni ammissibili

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

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

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

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

Esempio: istanza dell'MDKP

Consideriamo uno zaino con $m = 2$ dimensioni e capacità $c_1 = 5$ e $c_2 = 4$, e un insieme $\mathcal{J} = \{1, 2, 3, 4\}$ 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 $j \in \{0, 1, \dots, n\}$ e ogni vettore $\mathbf{k} = (k_1, k_2, \dots, k_m)$ con $k_d \in \{0, 1, \dots, c_d\}$ per ogni $d \in \mathcal{D}$, sia

$$v(j, \mathbf{k}) = v(j, k_1, \dots, k_m)$$

il valore ottimo della sottoistanza dell'MDKP data da:

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

Passo 1: la formula ricorsiva

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

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

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 $\mathbf{k}$, e «l'oggetto j entra nello zaino» significa $\mathbf{w}_j \leq \mathbf{k}$.
- ▶ I casi base sono

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

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 $\mathbf{k}$, oppure
 - ▶ l'oggetto j è selezionato, e il sottoproblema residuo ha il vettore $\mathbf{k} - \mathbf{w}_j$.

~> **Sottostruttura ottima verificata!**

Sottoproblemi sovrapposti e approccio bottom-up

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

⇒ **Sottoproblemi sovrapposti verificati!**

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

Lo stato dell'algoritmo di PD

- ▶ Lo **stato** dell'algoritmo di PD è la coppia $(j, \mathbf{k})$.
- ▶ Per ogni j la tabella ha una cella per ogni vettore $\mathbf{k}$, 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 $\mathbf{k}$ memorizziamo:
 - ▶ $s[j][\mathbf{k}] = 1$ se l'oggetto j è selezionato nella soluzione ottima trovata per $(j, \mathbf{k})$,
 - ▶ $s[j][\mathbf{k}] = 0$ altrimenti.
- ▶ L'insieme di tutti i vettori $\mathbf{k}$ è

$$\mathcal{C} = \{0, 1, \dots, c_1\} \times \{0, 1, \dots, c_2\} \times \dots \times \{0, 1, \dots, c_m\}.$$

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 $\mathbf{w}_1, \dots, \mathbf{w}_n$ e dal vettore delle capacità $\mathbf{c}$

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

```
1 function MDKP_DP( $n, m, p, \mathbf{w}, \mathbf{c}$ )
2   |  $v := \text{new array}[0..n][0..c_1] \cdots [0..c_m];$ 
   |  $s := \text{new array}[0..n][0..c_1] \cdots [0..c_m]$ 
3   | foreach  $\mathbf{k} \in \mathcal{C}$  do
4   |   |  $v[0][\mathbf{k}] \leftarrow 0; s[0][\mathbf{k}] \leftarrow 0$ 
5   | end
```

Algorithm: MDKP_DP (2/2)

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

```
6   for  $j := 1$  to  $n$  do
7       foreach  $k \in \mathcal{C}$  do
8            $v[j][k] \leftarrow v[j-1][k]; s[j][k] \leftarrow 0$ 
9           if  $w_j \leq k$  e  $p[j] + v[j-1][k - w_j] > v[j][k]$ 
              then
10               $v[j][k] \leftarrow p[j] + v[j-1][k - w_j];$ 
11               $s[j][k] \leftarrow 1$ 
12          end
13      end
14  end
15  return  $v[n][c], s$ 
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 $\mathbf{k} \in \mathcal{C}$.
- ▶ Il test $\mathbf{w}_j \leq \mathbf{k}$ confronta m coppie di numeri, e il vettore $\mathbf{k} - \mathbf{w}_j$ 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 $v[0][k] = 0$ per ogni k .
- ▶ Per ogni oggetto j calcoliamo la matrice $v[j][k_1][k_2]$, con:
 - ▶ una riga per ogni $k_1 \in \{0, 1, \dots, 5\}$,
 - ▶ una colonna per ogni $k_2 \in \{0, 1, \dots, 4\}$.

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

Questi sono i calcoli della cella $\mathbf{k} = \mathbf{c} = (5, 4)$ 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 $\mathbf{k} = \mathbf{c} = (5, 4)$ 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 $\mathbf{w}_1 = (1, 2)$:

$v[1][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_2 = (4, 1)$:

$v[2][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_3 = (2, 1)$:

$v[3][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_4 = (3, 4)$:

$v[4][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_1 = (1, 2)$:

$s[1][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_2 = (4, 1)$:

$s[2][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_3 = (2, 1)$:

$s[3][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{w}_4 = (3, 4)$:

$s[4][k_1][k_2]$	$k_2 = 0$	$k_2 = 1$	$k_2 = 2$	$k_2 = 3$	$k_2 = 4$
$k_1 = 0$	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 $\mathbf{k} = \mathbf{c}$ e da $x_j = 0$ per ogni oggetto j :

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

Algorithm: MDKP_Reconstruct

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

Output: Una soluzione ottima dell'istanza dell'MDKP

```
1 function MDKP_Reconstruct( $n, \mathbf{c}, \mathbf{w}, s$ )
2    $x := \text{new array}[1..n]$ 
3    $\mathbf{k} := \mathbf{c}$ 
4   for  $j := n$  downto 1 do
5      $x[j] \leftarrow s[j][\mathbf{k}]$ 
6      $\mathbf{k} \leftarrow \mathbf{k} - x[j] \mathbf{w}_j$ 
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 $\mathbf{k} = (5, 4)$. I valori del vettore x a ogni iterazione:

Iterazione	j	$\mathbf{k}$	$s[j][\mathbf{k}]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$
—	—	—	—	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 k :

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

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

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

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

- Indichiamo con $S(n, m, \mathbf{c})$ 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

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

e, poiché

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

abbiamo

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

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 $\mathbf{k} = (k_1, \dots, k_m)$
- ▶ $v(j, \mathbf{k}) = \max\{v(j-1, \mathbf{k}), p_j + v(j-1, \mathbf{k} - \mathbf{w}_j)\}$ se $\mathbf{w}_j \leq \mathbf{k}$,
e $v(j, \mathbf{k}) = v(j-1, \mathbf{k})$ altrimenti
- ▶ casi base $v(0, \mathbf{k}) = 0$; bottom-up, riga per riga, tutti i vettori $\mathbf{k}$ in un ordine qualsiasi

Passo 2: una soluzione ottima:

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

Il vettore delle capacità:

- ▶ l'oggetto j entra solo se $w_j \leq k$, in ogni dimensione
- ▶ selezionare l'oggetto j fa diminuire tutte le componenti di 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 $\mathbf{k} = (k_1, \dots, k_m)$:

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

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

- ▶ La tabella ha $(n+1) \prod_{d=1}^m (c_d + 1)$ celle e l'algoritmo di DP 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 $(n, \mathbf{c})$.

Riepilogo (2/2)

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

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