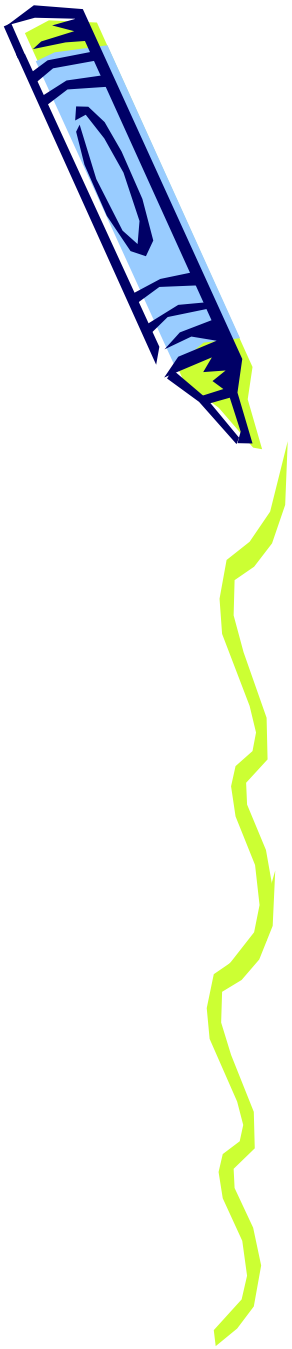


Numero di tardy job

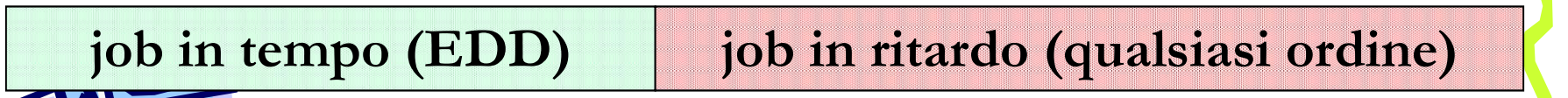
$1//\Sigma U_j$

Struttura delle soluzioni ottime

• ogni schedule ottimo è composto da un insieme di job che arrivano in tempo e da un secondo insieme di job in ritardo (rispetto alle proprie due date)

• il primo insieme rispetta EDD, in modo che L_{max} sia minimizzata (e minore o uguale a zero).

• l'ordine dei job del secondo insieme non ha impatto sulla funzione obiettivo



Algoritmo di *Moore*

- Costruisce lo schedule a partire dall'inizio
- J job già schedulati (schedule parziale, secondo EDD)
- J^d job già esaminati e fissati come tardy job
- J^c job ancora non esaminati

Inizializzazione:

$J = \emptyset$, $J^d = \emptyset$, $J^c = \{1, \ldots, n\}$.

Repeat:

1. Schedula il job j^* in J^c con la minima due date:
   aggiorna $J^c := J^c \setminus \{j^*\}$

2. Se j^* è in ritardo, cioè $\sum_{j \in J} p_j > d_{j^*}$

   elimina dallo schedule parziale il job più lungo, sia k^*.
   Aggiorna $J^d := J^d \cup \{k^*\}$

Until ($J^c \neq \emptyset$)

Correttezza

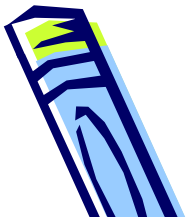
Teorema. L'algoritmo di Moore restituisce una soluzione ottima di $1//\Sigma U_j$

Dimostrazione. Senza perdita di generalità si può assumere $d_1 \leq d_2 \leq \ldots \leq d_n$ Sia J_k un sottoinsieme di job che soddisfa le seguenti proprietà:

1. ha il massimo numero N_k di job in tempo fra quelli di $\{1, \ldots, k\}$
2. fra tutti i sottoinsiemi di $\{1, \ldots, k\}$ con N_k job in tempo, J_k ha il minimo tempo di processamento

Per definizione, J_n corrisponde ad uno schedule ottimo.
Dimostriamo **per induzione** che l'algoritmo di Moore restituisce J_n

Induzione

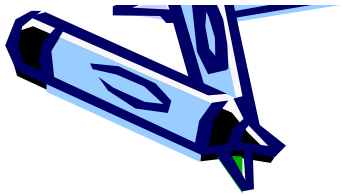


- l'algoritmo costruisce J_1 in modo da rispettare (1) e (2).
- Assumiamo che (1) e (2) valgano per k, cioè che Moore costruisca J_k e mostriamo che esse valgono per $k+1$

Sussistono due casi:

Caso a. Il job k+1 è aggiunto all'insieme J_k ed è completato in tempo. Ovviamente, è impossibile avere un maggior numero di job in tempo fra quelli in $\{1,\ldots,k+1\}$, quindi vale (1).

Inoltre, il job $k+1$ deve appartenere all'insieme. Quindi, il tempo di processamento totale è minimo fra tutti gli insiemi che soddisfano (1). Quindi vale (2).



Induzione

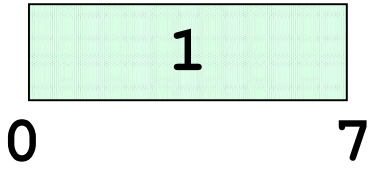
Caso b. Il job k+1 è aggiunto all'insieme J_k ed è completato in ritardo. Dato che J_k soddisfa (1) e (2) deve essere $N_k = N_{k+1}$.

Quindi, aggiungere il job k+1 a J_k non aumenta il numero di job in tempo. Ma agguingere k+1 e cancellare il job più lungo fra quelli di $J_k \cup \{k+1\}$ mantiene uguale il numero di job in tempo e riduce il tempo di processamento complessivo. Questo completa la prova.

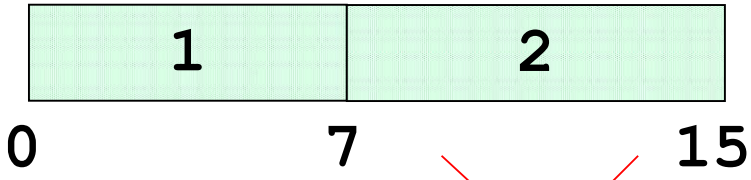
Esempio

job	1	2	3	4	5
p_j	7	8	4	6	6
d_j	9	17	18	19	21

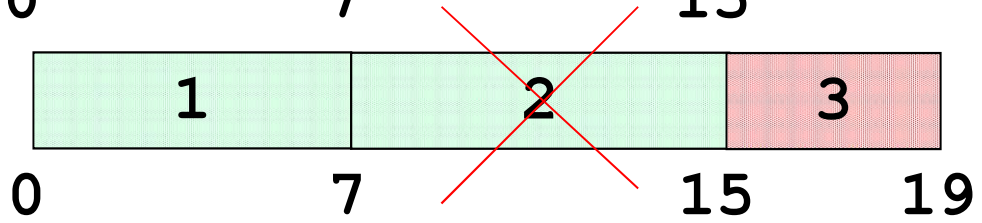
iter 1.

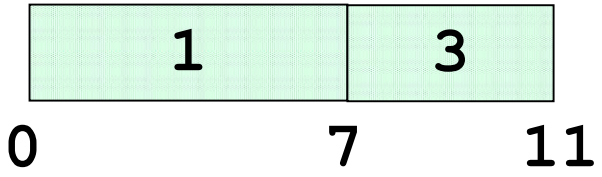


iter 2.



iter 3.

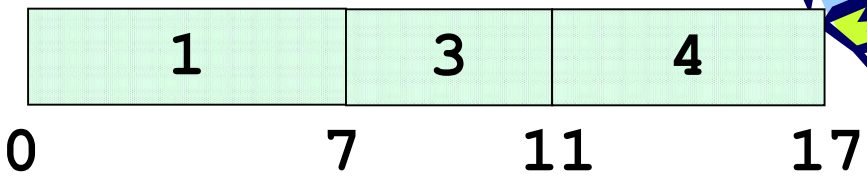




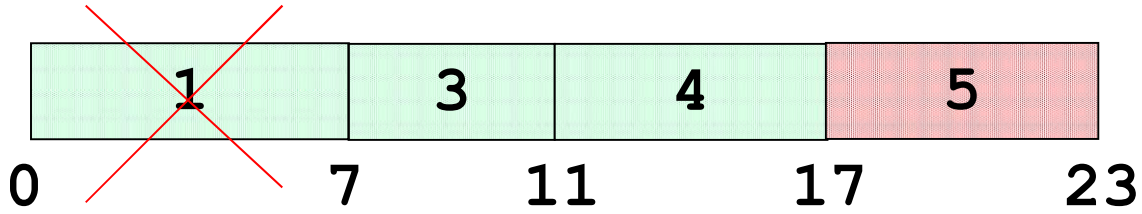
Esempio

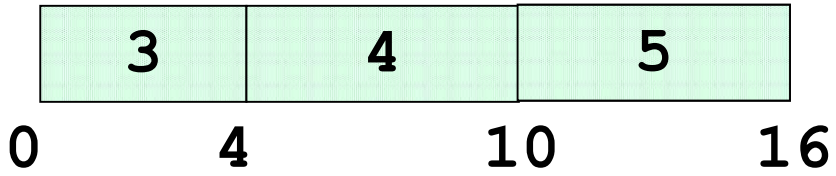
job	1	2	3	4	5
p_j	7	8	4	6	6
d_j	9	17	18	19	21

iter 4.

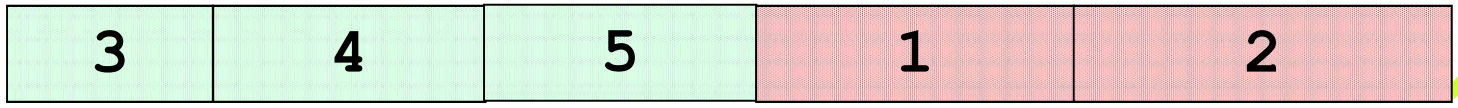


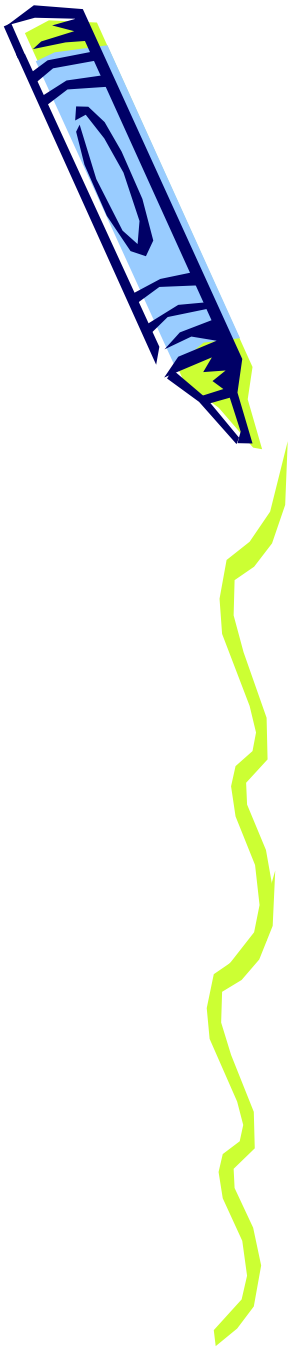
iter 5.





Soluzione





Massima Lateness

L_{max}

$1/prec/h_{max}$

Forma della funzione obiettivo:

$$h_{max} = max\ (h_1(C_1), \ldots, h_n(C_n))$$

In cui $h_j(C_j)$ è una arbitraria funzione non decrescente di C_j

Esempi:

$h_j(C_j) = C_j - d_j = L_j \Rightarrow h_{max} = L_{max}$

$h_j(C_j) = max\ (0, L_j) = T_j \Rightarrow h_{max} = T_{max}$

Precedenza: G_P generico grafo aciclico.

- Dato che le soluzioni ottime sono nondelay, l'ultimo job termina all'istante $C_{max} = \sum_{j=1}^{n} p_j$

Algoritmo di *Lawler*

- Costruisce lo schedule a partire dal fondo
- J insieme dei job già schedulati nell'intervallo $[C_{max} - \sum_{j \in J} p_j, C_{max}]$
- $J^c = \{1, \ldots, n\} - J$
- J' insieme dei job <u>schedulabili</u> immediatamente prima di J (= tutti i successori sono in J)

Inizializzazione:
$J = \emptyset$, $J^c = \{1, \ldots, n\}$, J' insieme dei job privi di successori

Loop: while ($J^c \neq \emptyset$)

Schedula in ultima posizione il job j^* tale che

$$h_{j^*}\left(\sum_{j \in J^c} p_j\right) = \min_{j \in J'} \left(h_j\left(\sum_{j \in J^c} p_j\right)\right)$$

$J := J \cup \{j^*\}$, $J^c := J^c \setminus \{j^*\}$, aggiorna J'

Endloop.

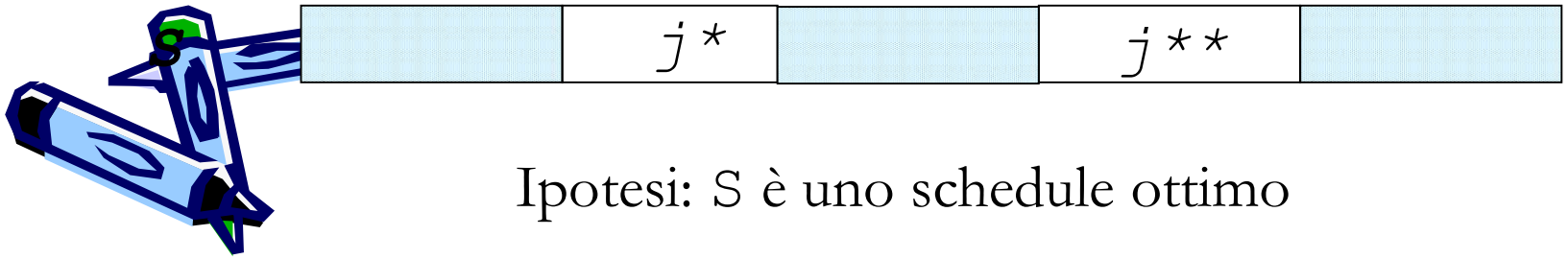
Corretezza

Teorema. L'algoritmo di Lawler restituisce uno schedule ottimo per $1/prec/h_{max}$

Dimostrazione. Per contraddizione: ad una generica iterazione è selezionato il job $j^{**} \in J'$ che non ha il minimo costo di completamento

$$h_{j^*}\left(\sum_{j \in J^c} p_j\right)$$

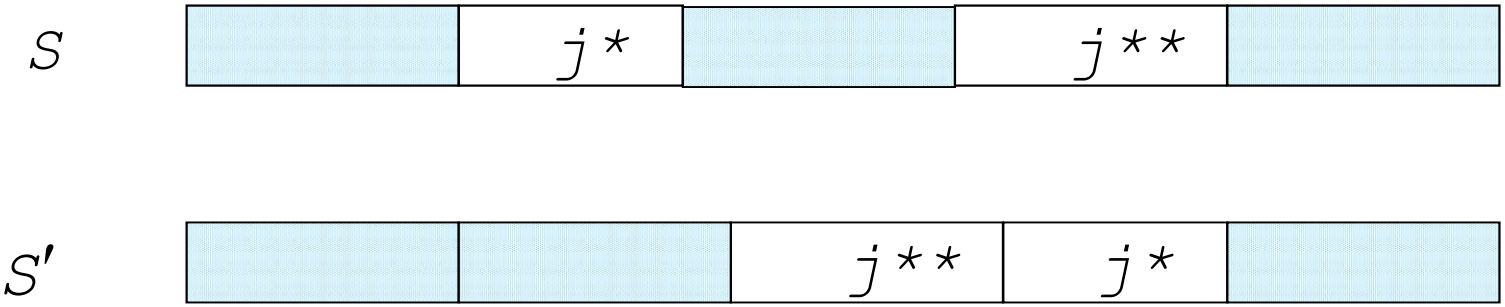
Quindi, j^* è schedulato prima di j^{**}



Ipotesi: S è uno schedule ottimo

Dimostrazione (continua)

Consideriamo un nuovo schedule S' ottenuto da S spostando il job $j**$ subito dopo $j*$.



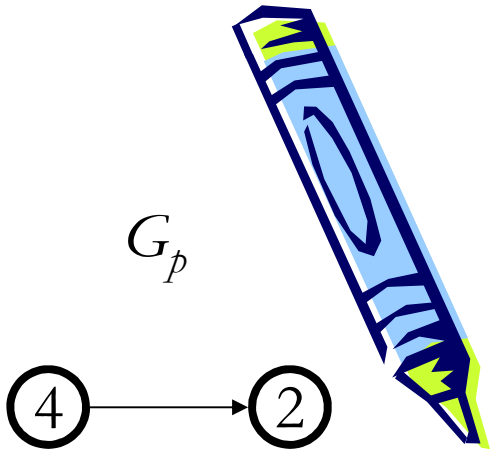
- L'unico job che peggiora il suo costo di completamento è $j*$.

- Tuttavia, il suo costo di completamento in S' è, per definizione, non superiore al costo di $j**$ in S. Quindi S non è ottimo.

□

Esempio

job	1	2	3	4
p_j	3	4	2	6
h_j	10	$1+C_2$	$1.5\ C_3$	$2^{(C4/5)}$



Iter 1.

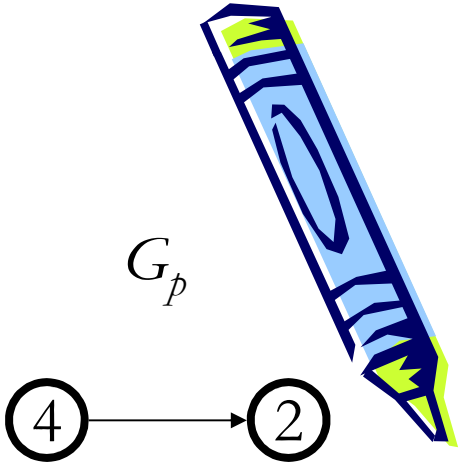
$$\sum_{j \in J^c} p_j = 15$$

J	$\varnothing$
J^c	{1,2,3,4}
J'	{1,2,3}

job	1	2	3
costo	10	16	22.5

Esempio

job	1	2	3	4
p_j	3	4	2	6
h_j	10	$1+C_2$	$1.5\ C_3$	2 (C4/5)

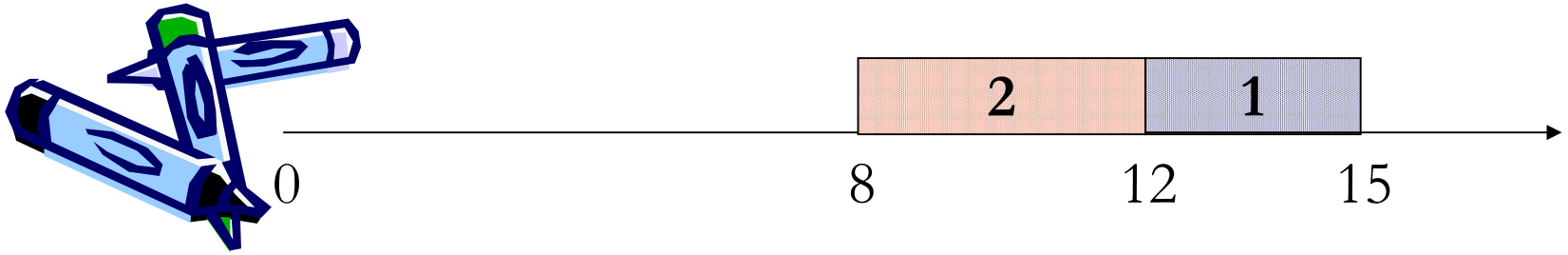


Iter 2.

$$\sum_{j \in J^c} p_j = 12$$

J	{1}
J^c	{2, 3,4}
J'	{2,3}

job	2	3
costo	13	18



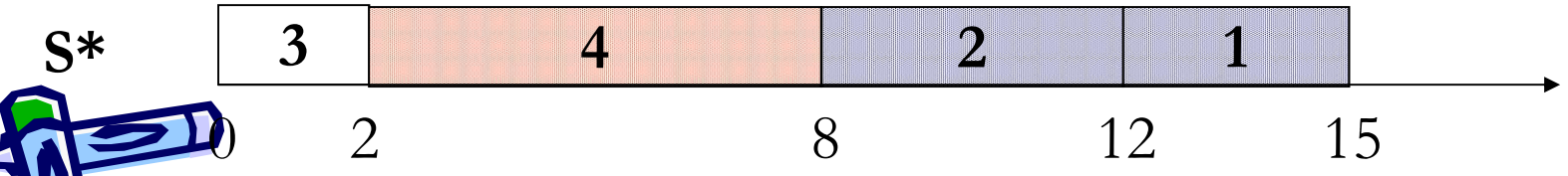
Esempio

job	1	2	3	4
p_j	3	4	2	6
h_j	10	$1+C_2$	$1.5\ C_3$	$2^{(C4/5)}$

Iter 3. $\sum_{j \in J^c} p_j = 8$

J	{1,2}
J^c	{3,4}
J'	{3,4}

job	3	4
costo	12	3.03



$z(S^*) = \max(10, 13, 3.03, 3)$

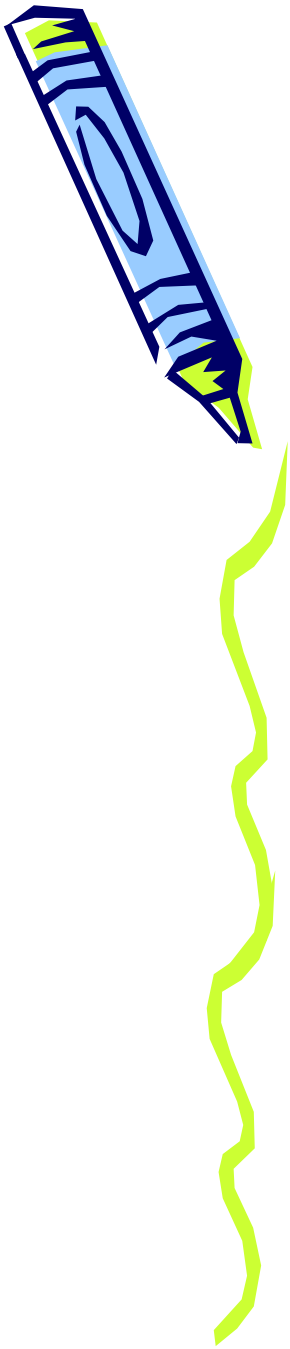
Complessità

L'algoritmo di Lawler si esegue in tempo $O(n^2)$

Infatti, assumendo che il calcolo del valore delle funzioni h_j richieda tempo costante, l'algoritmo richiede n passi, ciascuno basato su n confronti

- Nel caso $1//T_{max}$ l'algoritmo di Lawler si specializza:

Equivale alla regola **EDD** (Earliest Due Date)



$1/r_j/L_{max}$

Algoritmo branch-and-bound