Process and Operations Scheduling

instructor: Stefano Smriglio
stefano.smriglio@univaq.it

Recommended skills:

Basics in complexity theory: classes P and NP
Linear Programming
Integer Linear Programming
Dynamic programming
Branch-and-bound method
Flows and paths in networks
Matching in graphs

Material:

- M. Pinedo, *Scheduling* (*Theory, Algorithms and Systems*) – Prentice Hall
- A. Agnetis, dispense http://www.dii.unisi.it/~agnetis/didattica
- Slides **http://www.di.univaq.it/~smriglio/teach.html**

Outline

1. Introduction: elements of a scheduling problem, notation
2. Mixed integer programming formulations of scheduling problems
3. Single machine problems
   - $1//\Sigma w_j C_j$ — WSPT rule
   - $1/chain/\Sigma w_j C_j$
   - $1/prec/h_{max}$ — Lawler algorithm
   - $1/r_j, prmp/L_{max}$ — preemptive EDD rule
   - $1/r_j/L_{max}$ — complexity, branch-and-bound
   - $1//\Sigma U_j$ — Moore algorithm
   - $1//\Sigma h_j$ — Dynamic programming
   - $1//\Sigma T_j$
   - $1//\Sigma w_j T_j$

Introduction

scheduling problems
examples
classification

Introduction

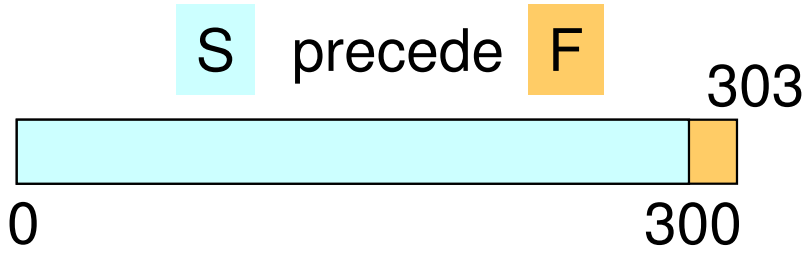
Scheduling deals with the **allocation of (scarce) resources to tasks over given time periods** and its purpose is to **optimize some performance indicator**

Course objectives:
- recognize and model scheduling problems
- apply integer programming to scheduling
- classify problems from a complexity point of view
- derive (heurisric and exact) solution algorithms

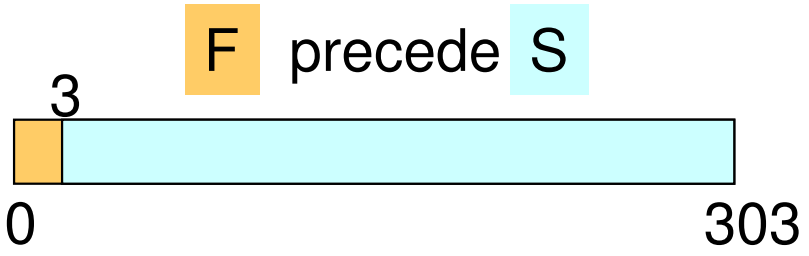
Esempio

Silvia (S) e Francesca (F) giungono nello stesso momento ad una macchina fotocopiatrice. F deve fare 1 fotocopia, S ne deve fare 100.
Il galateo imporrebbe a S di lasciar passare per prima F, che ha un compito molto più breve.
È una buona idea?

Analisi. Supponiamo che l'esecuzione di una fotocopia richieda 3 secondi. Due casi:



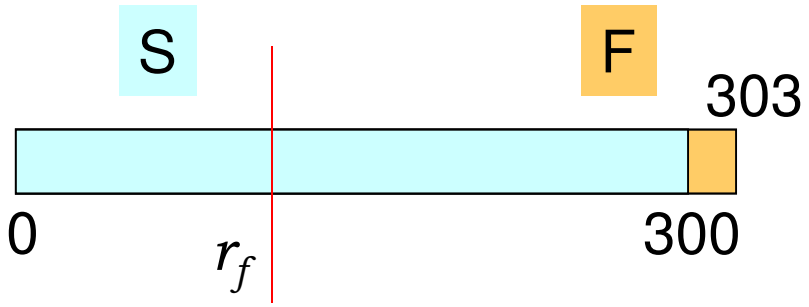
Attesa totale = 300 + 303 = 603



Attesa totale = 3 + 303 = 306

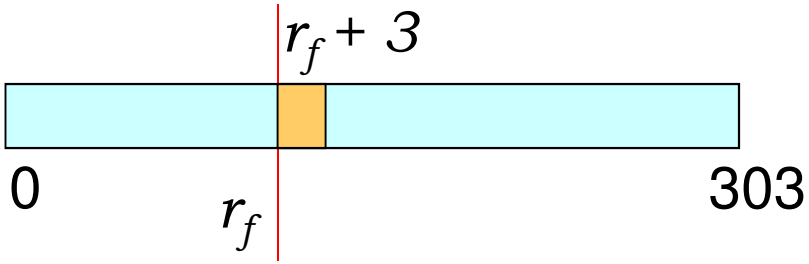
Esempio

E se Francesca F arrivasse all'istante r_f?



Attesa totale = 300 + 303 – r_f =
= 603 – r_f

Di nuovo, il galateo imporrebbe a Silvia di interrompere le proprie copie in favore di Francesca:

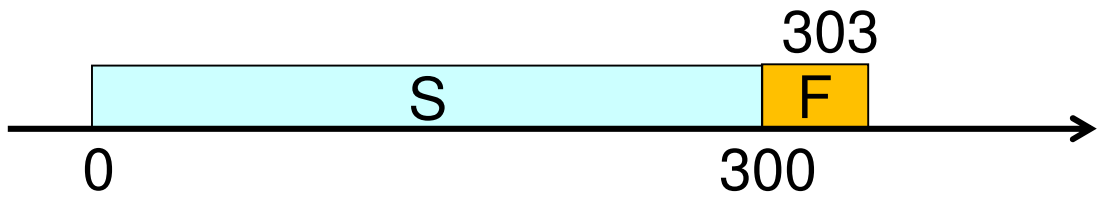


Attesa totale = 3 + 303 = 306

Ricapitolando

- **Attività**: 2 blocchi di fotocopie (di durata nota)
- **Risorse**: una macchina fotocopiatrice
- **Vincoli**:
  - ✓ Capacità: al più una fotocopia alla volta
  - ✓ (caso 2) F non può iniziare prima dell'istante r_f
- **Misura**: attesa complessiva presso la macchina fotocopiatrice

- Rappresentazione di una soluzione: diagramma di Gantt



CPU scheduling

• Il sistema operativo disciplina l'accesso alla CPU dei diversi programmi di calcolo.

• tecnica *time-sharing* (Linux): il tempo di CPU è suddiviso in intervalli (definiti dal *timer interrupt*) e, in ogni intervallo, è possibile eseguire al più un processo.

• diversi obiettivi: rispondere prontamente a ciascun processo, evitare interruzioni prolungate, evitare ritardi nei processi più urgenti.

• Linux attribuisce dinamicamente una priorità ai processi. Ad es., la priorità è proporzionale all'attesa del processo

CPU scheduling

- **Attività**: programmi
- **Risorse**: CPU
- **Vincoli**: Capacità: al più un programma in ciascun intervallo per ciascuna CPU
- **Misure**:
  - tempo massimo di attesa di un programma
  - tempo di completamento di tutti i programmi
  - ritardi nei processi “urgenti”

Scheduling dei velivoli in atterraggio

• La torre di controllo (ATC) ha il compito di assegnare una pista ed un istante di atterraggio a ciascun velivolo nel proprio campo radar (area terminale)

• Ogni velivolo ha una finestra temporale [*r*,*s*] definita da due istanti di atterraggio estremi, a seconda che il velivolo viaggi:

- alla sua massima velocità (istante *r*)
- in modalità di minimo consumo (istante *s*)

• Un velivolo in atterraggio impegna la pista per un tempo noto ma genera turbolenza: quello successivo deve attendere un tempo di "separazione" (funzione delle dimensioni dei velivoli)

Scheduling dei velivoli in atterraggio

• Ad ogni velivolo è associato un orario di atterraggio predeterminato (e pubblicato nelle tabelle)
• "L'importanza" di un velivolo dipende da fattori quali la dimensione, la compagnia, la distanza percorsa…

• **Attività**: atterraggi
• **Risorse**: k piste
• **Vincoli**:
  •Capacità: su ogni pista, al più un atterraggio in un certo istante
  • finestre temporali di atterraggio
  • separazione fra atterraggi consecutivi
• **Misura**:
  • somma (pesata in base all'importanza) dei ritardi rispetto all'orario pubblicato

Ancora nella vita quotidiana…

Aldo (A), Bruno (B), Carlo (C), Duilio (D) condividono un appartamento e, ogni mattino, ricevono 4 giornali: Financial Times (FT), Guardian (G), Daily Express (DE), Sun (S).

Ognuno ha i propri gusti…inizia la lettura ad una certa ora, legge i giornali nella sua sequenza preferita e (ciascuno) per un tempo prefissato:

lettore	inizio	Sequenza giornali (min.)			
Aldo	8.30	FT(60)	G (30)	DE (2)	S(5)
Bruno	8.45	G(75)	DE(3)	FT(25)	S(10)
Carlo	8.45	DE(5)	G(15)	FT(10)	S(30)
Duilio	9.30	S(90)	FT(1)	G(1)	DE(1)

Ancora nella vita quotidiana…

inoltre, ciascun lettore:

- ✓ rilascia un giornale solo dopo averlo letto completamente.
- ✓ termina la lettura di tutti i giornali prima di uscire.

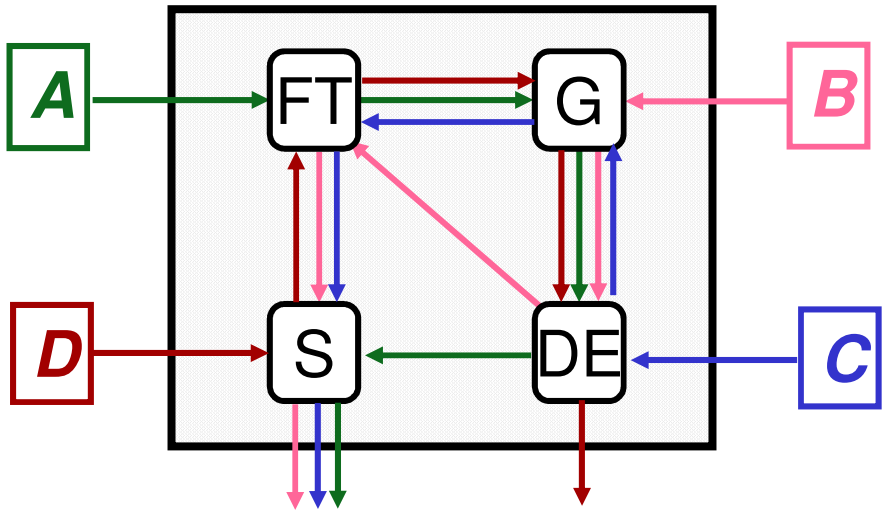
infine, i quattro lettori attendono che tutti abbiano terminato di leggere prima di uscire di casa

Problema:

A che ora A, B, C, D riescono (finalmente!) ad uscire?

Interpretazione

lettore	inizio	Sequenza giornali (min.)			
Aldo	8.30	FT(60)	G (30)	DE (2)	S(5)
Bruno	8.45	G(75)	DE(3)	FT(25)	S(10)
Carlo	8.45	DE(5)	G(15)	FT(10)	S(30)
Duilio	9.30	S(90)	FT(1)	G(1)	DE(1)



determinare in quale sequenza ciascun giornale "accetta" i lettori in modo da minimizzare il tempo di lettura totale.

Soluzioni

lettore	inizio	Sequenza giornali (min.)			
Aldo	8.30	FT(60)	G (30)	DE (2)	S(5)
Bruno	8.45	G(75)	DE(3)	FT(25)	S(10)
Carlo	8.45	DE(5)	G(15)	FT(10)	S(30)
Duilio	9.30	S(90)	FT(1)	G(1)	DE(1)

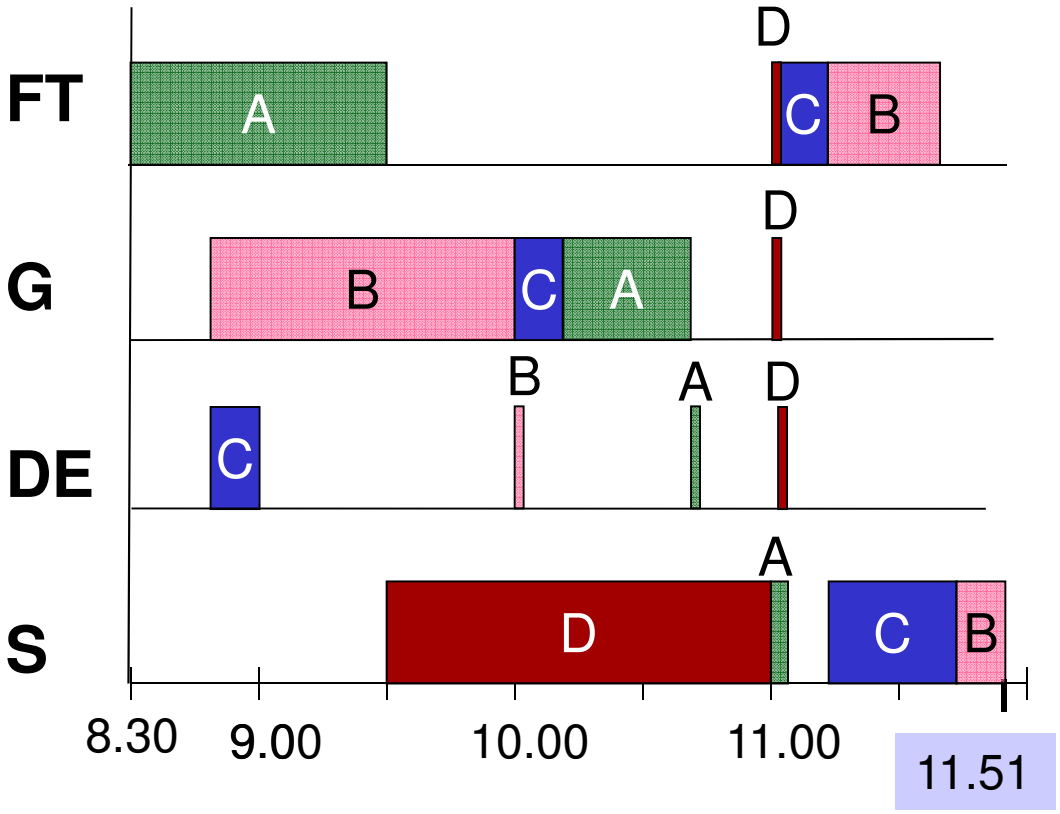
una possibile soluzione:

giornale	Sequenza lettori			
FT	A	D	C	B
G	B	C	A	D
DE	C	B	A	D
S	D	A	C	B

ESERCIZIO:
valutare l'ora in cui A,B,C e D escono di casa

ESERCIZIO:
data la soluzione in tabella valutare l'ora in cui A,B,C e D escono di casa

giornale	Sequenza lettori			
FT	A	D	C	B
G	B	C	A	D
DE	C	B	A	D
S	D	A	C	B



Enumerazione totale

- Il numero di soluzioni è $(4!)^4 = 331.776$
- per un problema con n lettori e m giornali diventa $(n!)^m$.
- esaminando 10.000 soluzioni al secondo (4 giornali e n lettori):

n	tempo
5	354 min
10	$2 \cdot 10^{17}$ giorni

In generale

Con il termine ***scheduling*** si indica l'allocazione temporale di risorse scarse ad attività

Elementi di un problema di scheduling:

job: attività da svolgere – insieme $J = \{1,\ldots,n\}$

Fotocopia, esecuzione di un programma di calcolo

Un job può rappresentare una singola attività o un insieme di attività (*task*) tecnologicamente legate

macchine: risorse che eseguono le attività – insieme $M=\{1,\ldots,m\}$

Fotocopiatrice, CPU, giornali

Attributi dei job

- ***tempo di processamento*** p_{ij} durata del processamento del job j sulla macchina i

- ***release date*** r_j tempo in cui j arriva nel sistema, rendendosi disponibile al processamento

- ***due date*** d_j tempo entro il quale si desidera che il job j sia completato (data di consegna)

- ***peso*** w_j indica l'importanza del job j

Classificazione

Il processamento di un job su una macchina è detto ***operazione.***
I problemi di scheduling si classificano in base alle caratteristiche dei task e all'architettura delle macchine

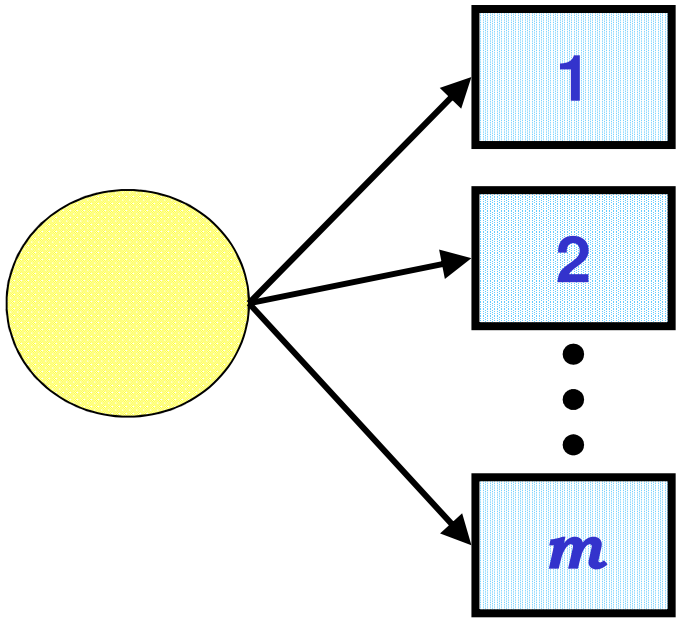
Notazione a tre campi: $\alpha/\beta/\gamma$:

α descrive il sistema di macchine
β rappresenta vincoli e modalità di processamento (0, 1 o più componenti)
γ indica l'obiettivo

Campo α

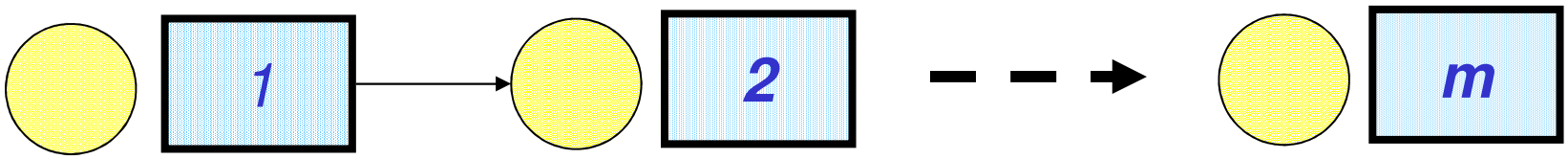
Macchina singola (1) : ciascun job richiede una singola operazione da eseguirsi sull'unica macchina disponibile

Macchine identiche parallele (P_m) : ciascun job richiede una singola operazione da eseguirsi su una qualunque delle m macchine identiche.



Campo α

Flow shop **(F_m)** : Ciascun job è composto di *m* task, ciscuno da eseguirsi su una macchina secondo una sequenza fissata, uguale per tutti i job.



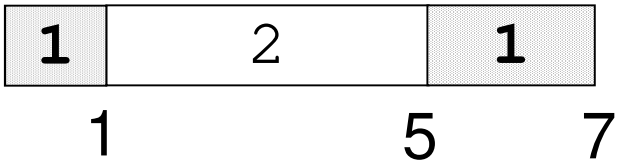
Job shop **(J_m)** : Ciascun job j è composto di $m(j)$ task, ciscuno da eseguirsi su una macchina secondo una sequenza fissata, dipendente da j

Campo β

- ***Release dates*** **(r_j)**: il job j non può iniziare il processamento prima dell'istante r_j

- ***Preemption*** **(*prmp*)**: è ammesso interrompere un'operazione per iniziarne una nuova. Il processamento eseguito prima dell'interruzione non va perso: quando l'operazione viene ripresa, essa richiede solo il tempo *rimanente* di processamento

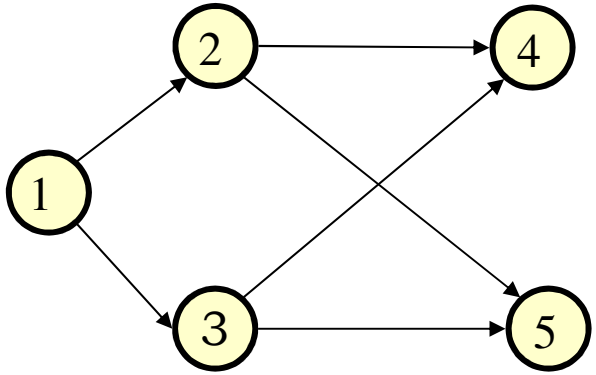
job	1	2
p_j	3	4

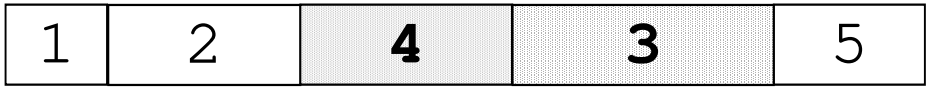


schedule ammissibile

Campo β

prec: è dato un grafo $G = (N, A)$ diretto e aciclico che rappresenta una relazione di precedenza fra job. Un job j può iniziare solo se tutti i suoi predecessori sono stati completati





schedule non ammissibile

chain: caso speciale in cui ogni componente connessa del grafo è un cammino orientato (catena)

Campo β

- ***Tempi di set-up*** **(s_{jk})**: tempo richiesto per il riattrezzaggio delle macchine fra i job j e k. Se dipende dalla macchina i, si esprime con s^i_{jk}

- ***breakdown*** **(*brkdwn*)**: le macchine non sono sempre disponibili, ma hanno periodi fissati di interruzione del servizio

Definizioni

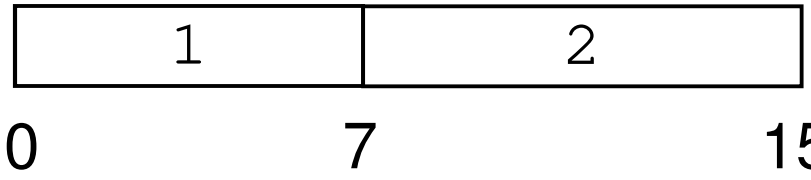
C_j tempo di completamento del job j

$L_j = C_j - d_j$ ***lateness*** del job j

$T_j = max(L_j, 0)$ ***tardiness*** del job j

$U_j = 1$ se $C_j > d_j$ e 0 altrimenti

job	1	2
p_j	7	8
d_j	9	11



$C_1 = 7$ $C_2 = 15$

$L_1 = -2$ $L_2 = 4$

$T_1 = 0$ $T_2 = 4$

$U_1 = 0$ $U_2 = 1$

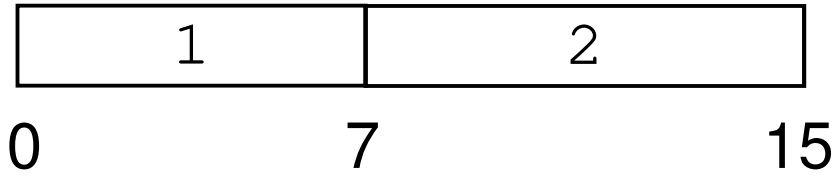
Campo γ: funzioni obiettivo

- ***Makespan*** **(C_{max})** : $max\ (C_1, \ldots, C_n)$ tempo di completamento dell'ultimo job
- **Massima Lateness (L_{max})** : $L_{max} = max\ (L_1, \ldots, L_n)$
- **Tempo totale pesato di completamento ($\Sigma w_j C_j$)**
  o (weighted flow-time)
- **Tardiness totale pesata ($\Sigma w_j T_j$)**
- **Numero di tardy job (ΣU_j)**

tutte funzioni obiettivo **regolari** cioè non-decrescenti in $C_1, \ldots, C_n$

Esempio

job	1	2
w_j	2	3
p_j	7	8
d_j	9	11



$L_{max} = 4$ $\quad\quad$ $\Sigma w_j T_j = 12$

$\Sigma w_j C_j = 59$ $\quad\quad$ $\Sigma U_j = 1$

Sequenze e schedule

• Si definisce **sequenza** una permutazione dei job che definisce l'ordine con cui i job sono processati su una certa macchina

• Si definisce **schedule** un ***assegnamento di un istante di inizio a ciascuno dei task di ogni job*** (se *prmp*, istanti di inizio di ciascuna delle parti in cui viene suddiviso un task)

Schedule nondelay

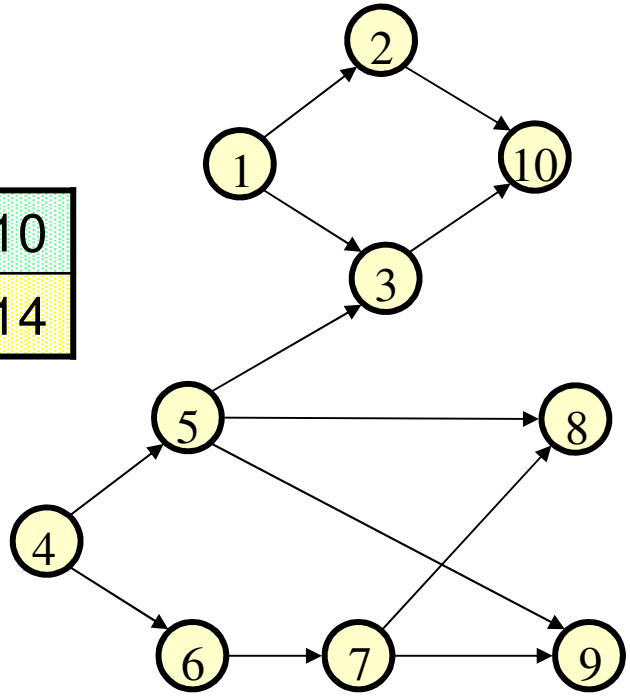
uno schedule ammissibile è detto **nondelay** se nessuna macchina è ferma quando esiste un'operazione disponibile al processamento

In molti casi (sempre se *prmp*) le soluzioni ottime sono nondelay. Ciò non è vero in generale

ESERCIZIO: schedule nondelay

dato il seguente problema $P_2/prec/C_{max}$

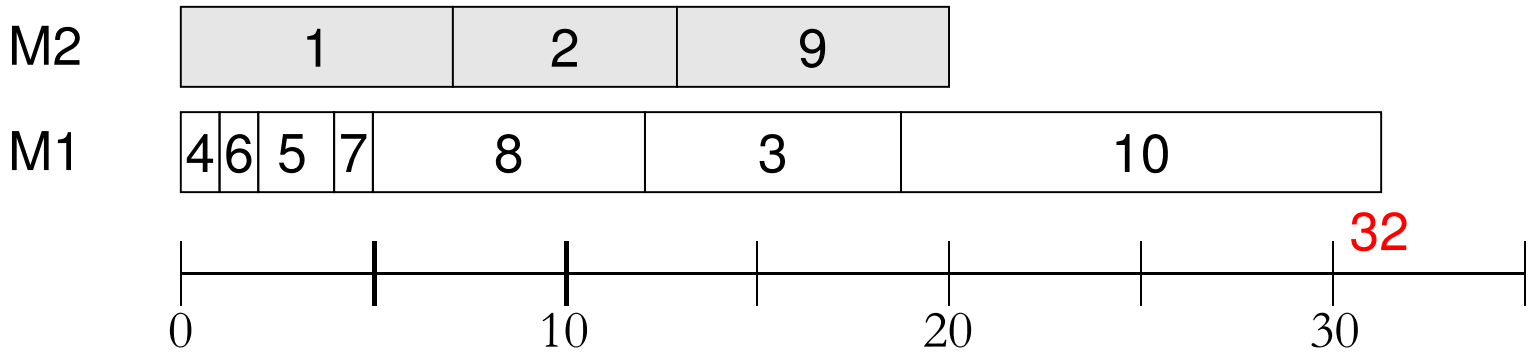
job	1	2	3	4	5	6	7	8	9	10
p_j	7	6	6	1	2	1	1	7	7	14



costruire uno schedule nondelay
e verificarne l'ottimalità

ESERCIZIO: schedule nondelay

Soluzione nondelay:



Soluzione ottima:

1.1 Scheduling a macchina singola

Formulazioni di Programmazione Lineare Booleana e Mista (PL(0-1)-PLM)

Formulazione disgiuntiva

Variabili decisionali “naturali”:

$t_j \in R^n$ istante di inizio del job j, $j = 1, \ldots, n$

Macchina a capacità unitaria:

i precede $j \Rightarrow \quad t_j \geq t_i + p_i$

j precede $i \Rightarrow \quad t_i \geq t_j + p_j$

valgono in alternativa !!

Variabili aggiuntive (binarie):

$$y_{ij} = \begin{cases} 1 & \text{se } i \text{ precede } j \\ 0 & \text{se } j \text{ precede } i \end{cases}$$

Formulazione disgiuntiva (PLM)

i precede *j* $\Rightarrow y_{ij} = 1 \Rightarrow \quad t_j \geq t_i + p_i$

j precede *i* $\Rightarrow y_{ij} = 0 \Rightarrow \quad t_i \geq t_j + p_j$

Formulazione *disgiuntiva*

$$(1 - y_{ij})M + t_j - t_i \geq p_i \quad \forall i, j \in J$$

$$y_{ij}M + t_i - t_j \geq p_j \quad \forall i, j \in J$$

$$t_j \geq 0 \quad \forall j \in J$$

$$y_{ij} \in \{0,1\} \quad \forall i, j \in J$$

M costante grande (quanto?)

Problemi minsum: $1//\sum_j w_j C_j$

$$\min \sum_{j \in J} w_j (t_i + p_i) = \sum_{j \in J} w_j t_j + \sum_{j \in J} w_j p_j$$

$$(1 - y_{ij}) M + t_j - t_i \geq p_i \quad \forall i, j \in J$$

$$y_{ij} M + t_i - t_j \geq p_j \quad \forall i, j \in J$$

$$t_j \geq 0 \quad \forall j \in J$$

$$y_{ij} \in \{0,1\} \quad \forall i, j \in J$$

$(n^2–n)/2$ + n variabili
n^2 vincoli

Esercizio

Formulare come problema di PLM la seguente istanza del problema $1/r_j/\sum_j w_j C_j$

job	1	2	3
p_j	3	1	2
r_j	0	2	0
w_j	3	2	4

Problemi minsum: $1//\sum_j w_j T_j$

$$\min \quad \sum_{j \in J} w_j \max\{t_j + p_j - d_j, 0\}$$

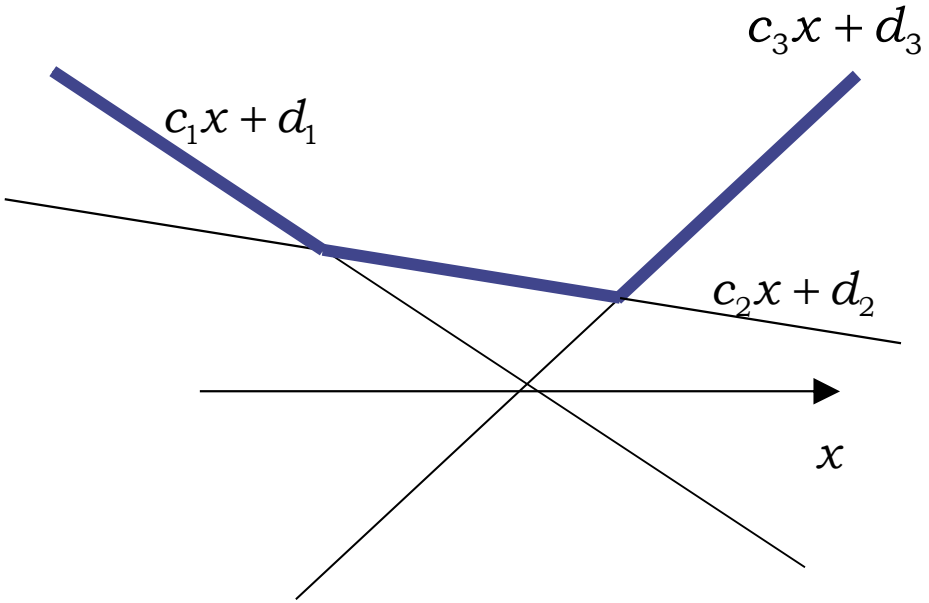
$$(1 - y_{ij})M + t_j - t_i \geq p_i \quad \forall i, j \in J$$

$$y_{ij} M + t_i - t_j \geq p_j \quad \forall i, j \in J$$

$$t_j \geq 0 \quad \forall j \in J$$

$$y_{ij} \in \{0,1\} \quad \forall i, j \in J$$

RICHIAMO: Funzioni convesse lineari a tratti



$$\max_{i=1,\ldots,m}(c_i^T x + d_i)$$

è il più piccolo valore z per cui

$$z \geq c_i^T x + d_i \quad i = 1,\ldots,m$$

$$\min \quad \max_{i=1,\ldots,m}(c_i^T x + d_i)$$

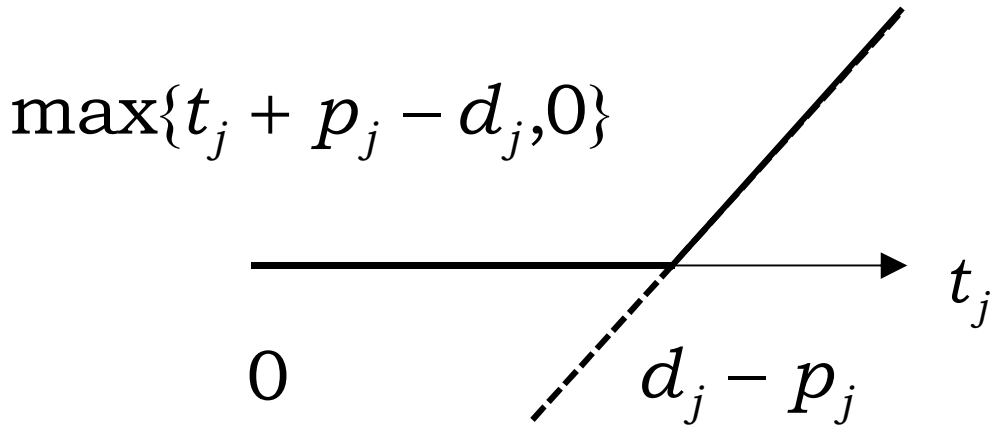
$$Ax \geq b$$

$$\min \quad z$$

$$Ax \geq b$$

$$c_i^T x + d_i \leq z \qquad i = 1,\ldots,m$$

Formulazione PLM



$$\min \sum_{j \in J} w_j \max\{t_j + p_j - d_j, 0\}$$

$$\min \quad \sum w_j f_j$$

$$f_j \geq t_j + p_j - d_j, \; j \in J$$

$$f_j \geq 0$$

Formulazione PLM

$$\min \sum w_j f_j$$

$$f_j - t_j \geq p_j - d_j, \quad j \in J$$

$$(1 - y_{ij})M + t_j - t_i \geq p_i \quad \forall i, j \in J$$

$$y_{ij} M + t_i - t_j \geq p_j \quad \forall i, j \in J$$

$$t_j \geq 0 \quad \forall j \in J$$

$$y_{ij} \in \{0,1\} \quad \forall i, j \in J$$

$$f_j \geq 0$$

Problemi min-max: L_{max}

$$\min \quad f$$

$$f - t_j \geq p_j - d_j, \quad j \in J$$

$$(1 - y_{ij})M + t_j - t_i \geq p_i \quad \forall i, j \in J$$

$$y_{ij}M + t_i - t_j \geq p_j \quad \forall i, j \in J$$

$$t_j \geq 0 \quad \forall j \in J$$

$$y_{ij} \in \{0,1\} \quad \forall i, j \in J$$

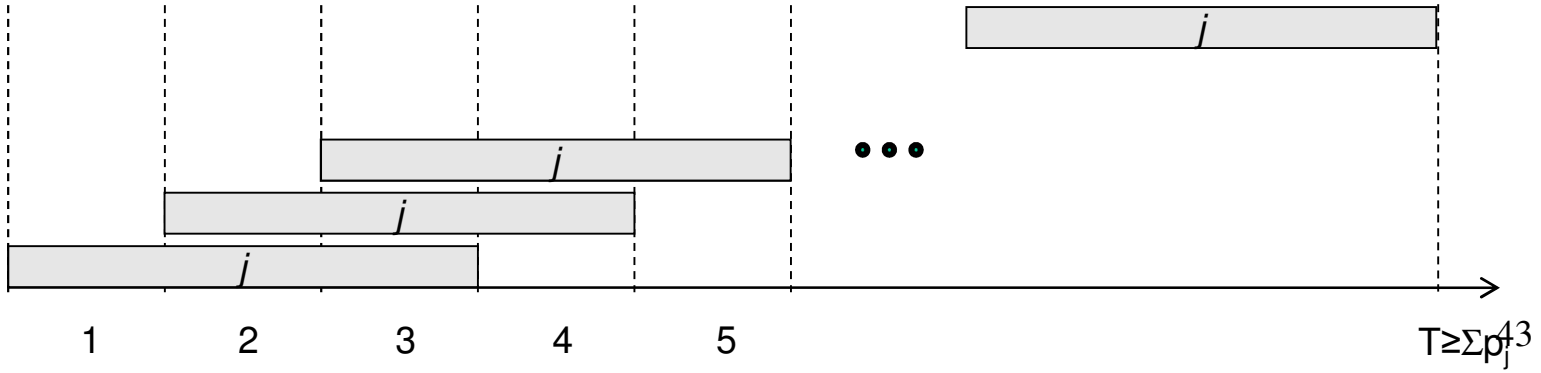
Formulazioni Time-indexed

Orizzonte temporale $[0, T]$ discretizzato in periodi di durata unitaria. Il periodo t concide con l'intervallo di $[t-1,\ t]$

Variabili binarie: $x_{jt} = 1$ se j inizia nel periodo t; 0 altrimenti

vincolo 1 (Completezza): ogni job inizia in un qualche t

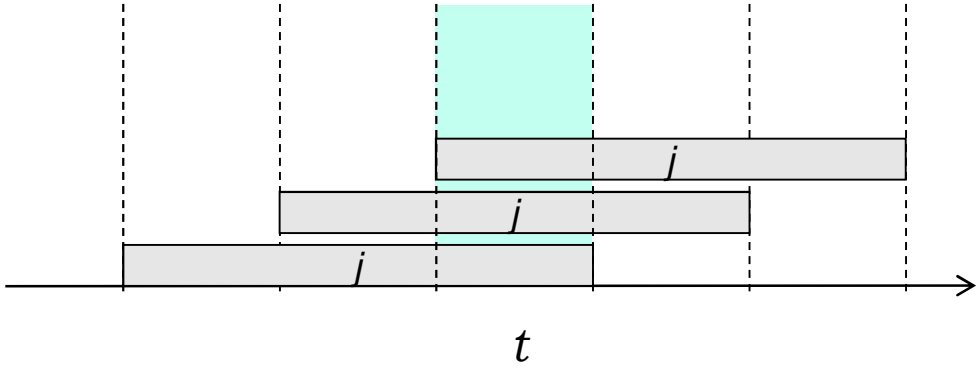
$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j = 1,\ldots,n$$



Formulazioni Time-indexed

vincolo 2 (Capacità): al più un job processato nel periodo t

un job occupa il periodo t *se* inizia nell'intervallo $[t - p_j+1,\ t]$



quindi:

$$\sum_{j=1}^{n} \sum_{s=\max(1,\,t-p_j+1)}^{\min(t,\,T-p_j+1)} x_{js} \le 1, \qquad t = 1,\ldots,T$$

Formulazione PL(0-1)

$$\min \quad \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}$$

c_{jt} costo di assegnazione del job j all'istante t

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j = 1,\ldots,n$$

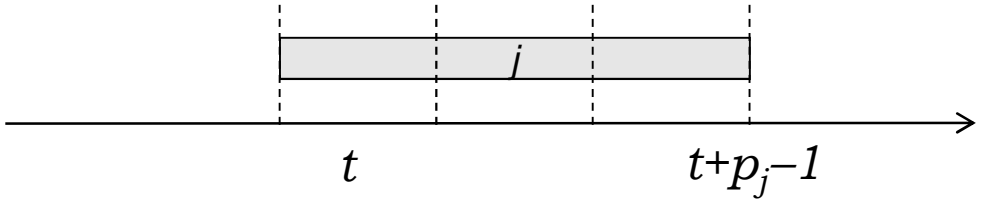
$$\sum_{j=1}^{n} \sum_{s=\max(1,\,t-p_j+1)}^{\min(t,\,T-p_j+1)} x_{js} \le 1, \qquad t = 1,\ldots,T$$

$$x_{jt} \in \{0,1\},\ j = 1,\ldots,n; \quad t = 1,\ldots,T-p_j+1$$

nT variabili; $n + T$ vincoli

Modellazione funzioni obiettivo

se j inizia nel periodo t si ha $C_j = t+p_j-1$



cioè:

$$C_j = \sum_{t=1}^{T-p_j+1} (t + p_j - 1) x_{jt}$$

Quindi, basta porre $c_{jt} = w_j(t+p_j-1)$ per ottenere

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} w_j (t + p_j - 1) x_{jt} = \sum_{j=1}^{n} w_j C_j$$

Modellazione funzioni obiettivo

Analogamente, definendo

$$c_{jt} = w_j \max\{0, t + p_j - d_j - 1\}$$

si ottiene la tardiness totale:

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} w_j \max(0, t + p_j - d_j - 1) x_{jt} = \sum_{j=1}^{n} w_j T_j$$

Problemi min-max

$$\min \quad \max_{j=1,\ldots,n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j = 1,\ldots,n$$

$$\sum_{j=1}^{n} \sum_{s=\max(1,\, t-p_j+1)}^{\min(t,\, T-p_j+1)} x_{js} \le 1, \qquad t = 1,\ldots,T$$

$$x_{jt} \in \{0,1\},\ j = 1,\ldots,n; \quad t = 1,\ldots,T-p_j+1$$

Linearizzazione

$$\min \quad f$$

$$f - \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt} \geq 0, \quad j = 1, \ldots, n$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j = 1, \ldots, n$$

$$\sum_{j=1}^{n} \sum_{s=\max(1, t-p_j+1)}^{\min(t, T-p_j+1)} x_{js} \leq 1, \quad t = 1, \ldots, T$$

$$x_{jt} \in \{0,1\},\ j = 1, \ldots, n; \quad t = 1, \ldots, T - p_j + 1$$

Esempio: L_{max}

Analogamente, definendo

$$c_{jt} = t + p_j - d_j - 1$$

I vincoli $f - \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt} \geq 0, \quad j = 1, \ldots, n$

diventano

$$f - \underbrace{\sum_{t=1}^{T-p_j+1} (t + p - d_j - 1) x_{jt}}_{= L_j} \geq 0, \quad j = 1, \ldots, n$$

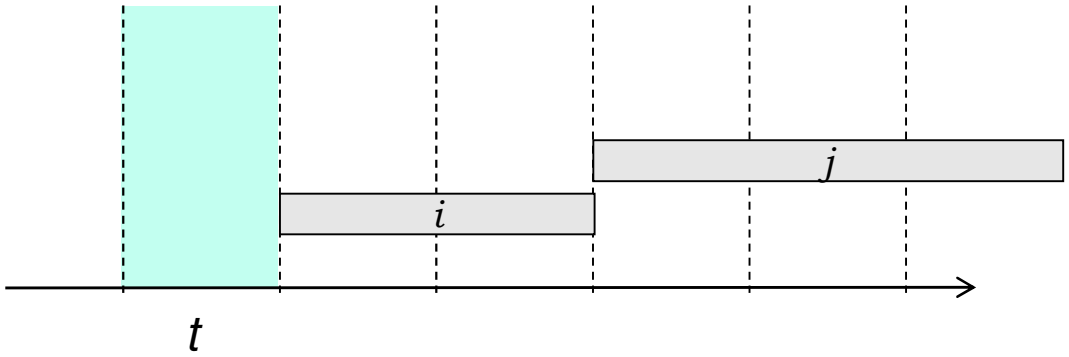
Esercizio

Formulare come problema di PL(0-1) la seguente istanza del problema $1/r_j/\sum_j w_j C_j$

job	1	2	3
p_j	3	1	2
r_j	0	2	0
w_j	3	2	4

Vincoli di precedenza

vincolo 3 (Precedenza $i \rightarrow j$): se il job i non è iniziato nei periodi $1,2,\ldots,t$ allora il job j non può iniziare nei periodi 1, $2,\ldots,\ t + p_i$



quindi:

$$\sum_{s=1}^{t+p_i} x_{js} \leq \sum_{r=1}^{t} x_{ir}, \qquad t = 1,\ldots,T - p_i - p_j + 1$$

1.2 Tempo totale pesato di completamento

$\Sigma w_j C_j$

nondelay schedule

Proprietà 0.1 Dato un problema del tipo $1//\gamma$ con γ funzione obiettivo regolare, esiste uno schedule ottimo in cui la macchina processa ininterrottamente dall'istante 0 all'istante $C_{max} = \Sigma p_j$

Tempo totale pesato di completamento
$1//\Sigma w_j C_j$

Definiamo *Weighted Shortest Processing Time* (WSPT) una regola che ordina i job per valori non crescenti del rapporto w_j/p_j

Sussiste il seguente

Teorema 1.1 La regola WSPT calcola una soluzione ottima del problema $1//\Sigma w_j C_j$

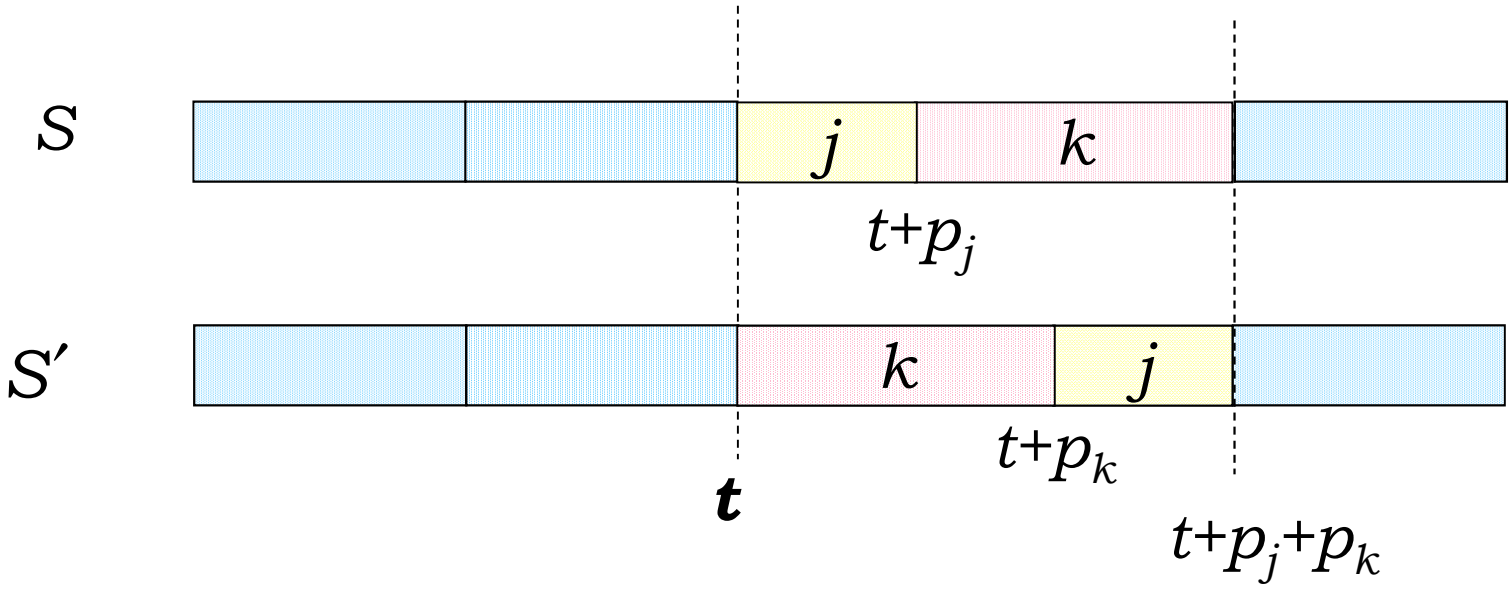
Dimostrazione. (Per contraddizione)
Assumiamo che S sia uno schedule ottimo e che non rispetti WSPT

Dimostrazione

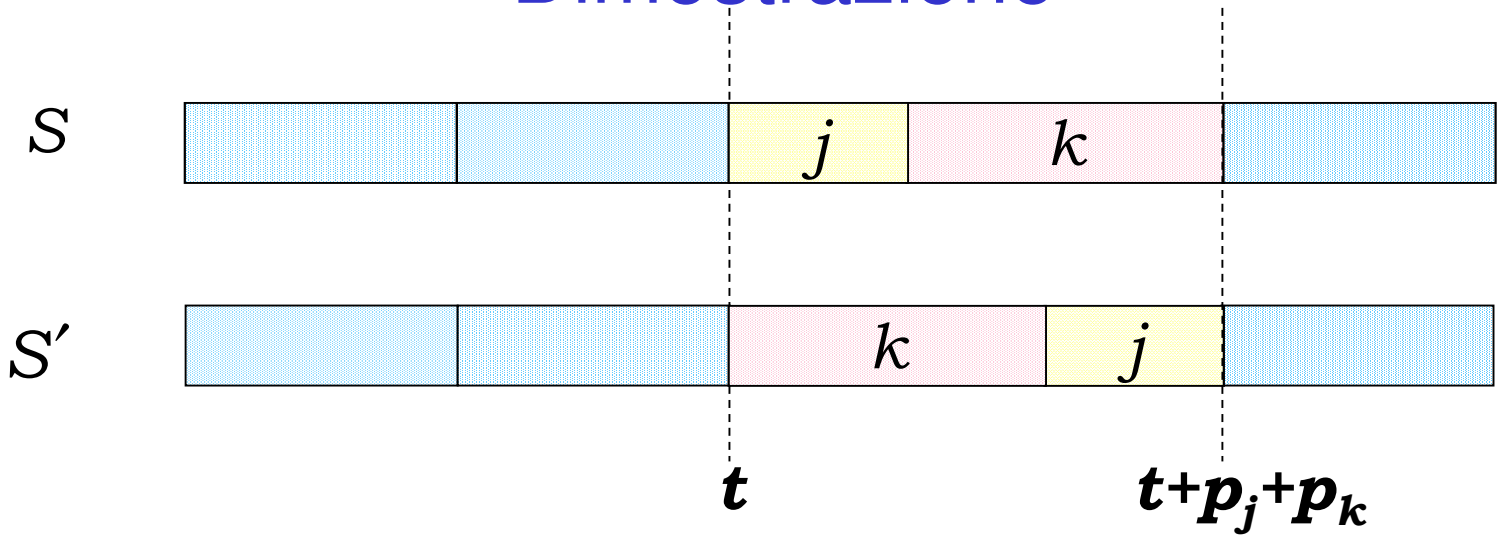
Allora devono esserci in S due job adiacenti, diciamo j seguito da k tali che

$$w_j / p_j < w_k / p_k$$

Assumiamo che il job j inizi all'istante t.
Eseguiamo uno scambio dei job j e k ottenendo un nuovo schedule S'



Dimostrazione



Il tempo totale (pesato) di completamento dei job che precedono e seguono la coppia (j,k) non è modificato dallo scambio. Il contributo dei job j e k in S è:

$$(t+p_j)w_j + (t+p_j+p_k)w_k$$

mentre in S' è:

$$(t+p_k)w_k + (t+p_k+p_j)w_j$$

Dimostrazione

in S] $(t+p_j)w_j + (t+p_j+p_k)w_k$

in S'] $(t+p_k)w_k + (t+p_k+p_j)w_j$

Eliminando i termini uguali:

in S] $p_j w_k$

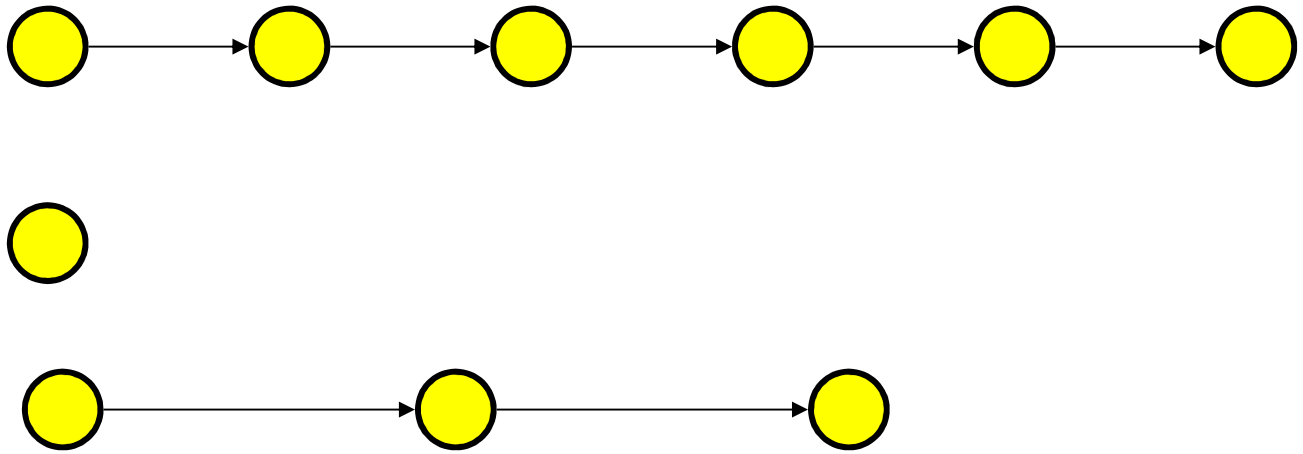
in S'] $p_k w_j$

Quindi, se $w_j/p_j < w_k/p_k$, risulta $p_k\ w_j < p_j\ w_k$, cioè il tempo totale pesato in S' è strettamente inferiore a quello in S, contraddizione

□

Vincoli di precedenza
$1/prec/\ \Sigma w_j C_j$

Nel caso generale il problema è NP-Hard. Consideriamo invece il caso in cui le precedenze sono rappresentate da un insieme di catene in parallelo: ***1/chain/*$\Sigma w_j C_j$**



Per questo caso esiste un algoritmo di soluzione polinomiale.

Catene non interrompibili

Date due catene:

$$1 \rightarrow 2 \rightarrow \ldots \rightarrow k$$
$$k+1 \rightarrow k+2 \rightarrow \ldots \rightarrow n$$

Minimizziamo $\Sigma w_j C_j$ con il vincolo che i job di ciascuna catena debbano essere processati consecutivamente.

Lemma 1.2 Se

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

allora la catena 1, …,k precede la catena k+1, …, n

Dimostrazione

- Per la sequenza $1,2,\ldots,k,k+1,k+2,\ldots,n$ (**S1**) il tempo totale di completamento è

$$w_1 p_1 + \ldots + w_k \sum_{j=1}^{k} p_j + w_{k+1}\left(\sum_{j=1}^{k} p_j + p_{k+1}\right) + \ldots + w_n \sum_{j=1}^{n} p_j$$

- Per la sequenza $k+1,k+2,\ldots,n,1,2,\ldots,k$ (**S2**) il tempo totale di completamento è

$$w_{k+1} p_{k+1} + \ldots + w_n \sum_{j=k+1}^{n} p_j + w_1\left(\sum_{j=k+1}^{n} p_j + p_1\right) + \ldots + w_k \sum_{j=1}^{n} p_j$$

Semplificando

- **S1**

$$w_{k+1}\sum_{j=1}^{k} p_j + \ldots + w_n \sum_{j=1}^{k} p_j = (\sum_{j=k+1}^{n} w_j)(\sum_{j=1}^{k} p_j)$$

- **S2**

$$w_1\sum_{j=k+1}^{n} p_j + \ldots + w_k \sum_{j=k+1}^{n} p_j = (\sum_{j=1}^{k} w_j)(\sum_{j=k+1}^{n} p_j)$$

- Quindi, ricordando l'ipotesi:

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j} \Rightarrow (\sum_{j=k+1}^{n} w_j)(\sum_{j=1}^{k} p_j) < (\sum_{j=1}^{k} w_j)(\sum_{j=k+1}^{n} p_j)$$

cioè, S1migliore di S2

□

Coefficiente ρ di una catena

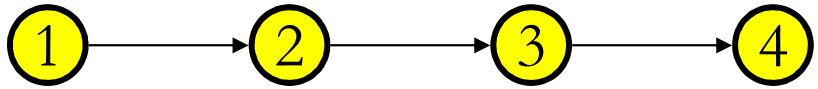
L'idea chiave dell'algoritmo è ricondurre $1/chain/\Sigma w_j C_j$ al problema di sequenziare catene non-interrompibili, in modo da utilizzare il Lemma 1.2.

Definizione 1.3 Data una catena $1 \rightarrow 2 \rightarrow \ldots \rightarrow k$ definiamo coefficiente ρ della catena il seguente rapporto:

$$\rho(1,\ldots,k) = \frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1 \le l \le k} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right)$$

Esempio

job	1	2	3	4
p_j	3	4	2	6
w_j	10	4	17	8



$$\rho(1,\ldots,4) = \max_{1 \le l \le 4} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right) = \left\{ \frac{10}{3}, \frac{14}{7}, \frac{31}{9}, \frac{39}{15} \right\}$$

$l^* = 3$

Proprietà del coefficiente ρ

Proprietà 1.4 Sia *l** il primo job della catena che determina il coefficiente ρ(*1*,...,*k*) allora per qualunque $1 \leq u < l^*$ risulta:

$$\frac{\sum_{j=u+1}^{l^*} w_j}{\sum_{j=u+1}^{l^*} p_j} > \frac{\sum_{j=1}^{u} w_j}{\sum_{j=1}^{u} p_j}$$

Dimostrazione

Per definizione:

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} > \frac{\sum_{j=1}^{u} w_j}{\sum_{j=1}^{u} p_j}$$

da cui:

$$p_1 \sum_{j=1}^{l^*} w_j + \ldots + p_u \sum_{j=1}^{l^*} w_j > p_1 \sum_{j=1}^{u} w_j + \ldots + p_{l^*} \sum_{j=1}^{u} w_j$$

$$p_1 \sum_{j=u+1}^{l^*} w_j + \ldots + p_u \sum_{j=u+1}^{l^*} w_j > p_{u+1} \sum_{j=1}^{u} w_j + \ldots + p_{l^*} \sum_{j=1}^{u} w_j$$

$$(\sum_{j=1}^{u} p_j)(\sum_{j=u+1}^{l^*} w_j) > (\sum_{j=u+1}^{l^*} p_j)(\sum_{j=1}^{u} w_j)$$

□

Proprietà dello schedule ottimo

Lemma 1.5 Si consideri una generica catena (1, ..., k) e sia l^* il primo job che determina il suo coefficiente $\rho(1,\ldots,k)$. Allora esiste uno schedule ottimo che processa i job $1, \ldots, l^*$ consecutivamente.

Dimostrazione. (per contraddizione) Assumiamo che in una sequenza ottima la sequenza $1,\ldots,l^*$ sia interrotta da un job $v \notin (1,\ldots,k)$, cioè, contenga la sottosequenza

(S) $\quad 1,\ldots,u,v,u+1,\ldots l^*$

Proprietà dello schedule ottimo

(**S**) ***1,...,u,v,u+1,...l****.

Consideriamo le sottosequenze:

(**S'**) ***v,1,...,u,u+1,...,l****

(**S''**) ***1,...,u,u+1,...l*,v***

Mostriamo che per almeno una fra **S'** e **S''** il tempo di completamento è non superiore a quello di **S**.

Dimostrazione

• Applicando il Lemma 1.2 alle catene (v) e ($1,\ldots,u$) si ha che se il valore ($\sum w_j C_j$) di **S** è inferiore a quello di **S'**, allora:

$$\frac{w_v}{p_v} < \frac{w_1 + w_2 + \ldots + w_u}{p_1 + p_2 + \ldots + p_u}$$

• Applicando il Lemma 1.2 alle catene (v) e ($u+1,\ldots,l^*$) si ha che se il valore ($\sum w_j C_j$) di **S** è inferiore a quello di **S''**, allora:

$$\frac{w_v}{p_v} > \frac{w_{u+1} + w_{u+2} + \ldots + w_{l^*}}{p_{u+1} + p_{u+2} + \ldots + p_{l^*}}$$

Dimostrazione

- Per la Proprietà 1.4 risulta inoltre:

$$\frac{w_{u+1} + w_{u+2} + \ldots + w_{l^*}}{p_{u+1} + p_{u+2} + \ldots + p_{l^*}} > \frac{w_1 + w_2 + \ldots + w_u}{p_1 + p_2 + \ldots + p_u}$$

- Quindi, se **S** è meglio di **S''**, allora:

$$\frac{w_v}{p_v} > \frac{w_{u+1} + w_{u+2} + \ldots + w_{l^*}}{p_{u+1} + p_{u+2} + \ldots + p_{l^*}} > \frac{w_1 + w_2 + \ldots + w_u}{p_1 + p_2 + \ldots + p_u}$$

Contraddizione (avendo assunto **S** meglio di **S'**). Lo stesso argomento si applica quando la catena è interrotta da molteplici job.

□

Algoritmo

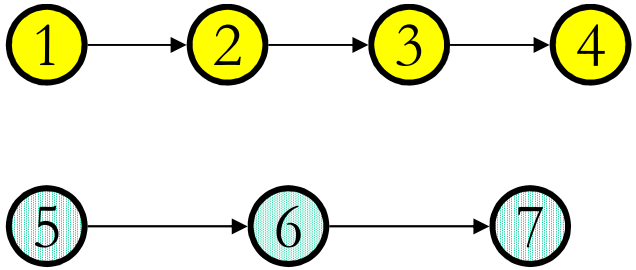
I due lemmi precedenti sono il fondamento di un algoritmo polinomiale:

Algoritmo 1.6
In ogni istante in cui la macchina è libera, seleziona fra le rimanenti catene quella con il massimo coefficiente ρ e la processa senza interruzione fino al primo job che definisce ρ (incluso)

L'algoritmo ha complessità $O(n^2)$

Esempio

job	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10



step1.

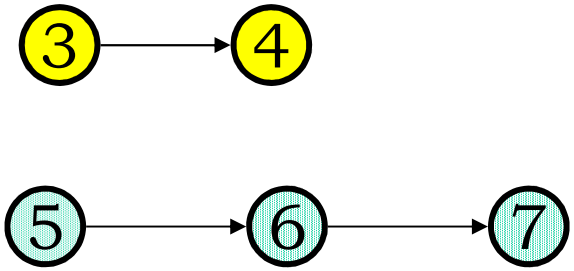
$\rho(1,2,3,4) = (6+18)/(3+6) = 24/9,\ l^*=2$

$\rho(5,6,7) = (8+17)/(4+8) = 25/12,\ l^*=6$

1	2

Esempio

job	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10



step2.

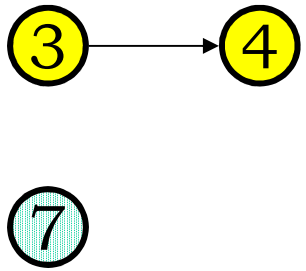
$\rho(3,4) = 12/6,\ l^*=3$

$\rho(5,6,7) = (8+17)/(4+8) = 25/12,\ l^*=6$

1	2	**5**	**6**

Esempio

job	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10



step3.

$\rho(3,4) = 12/6$, $l^*=3$

$\rho(7) = 18/10$, $l^*=7$

1	2	5	6	**3**

Esempio

job	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10

④

⑦

step4.

$\rho(4) = 8/5,\ l^*=4$

$\rho(7) = 18/10,\ l^*=7$

1	2	5	6	3	**7**	4

Release date e preemption

L'introduzione delle release date complica il problema. Nel caso in esame, il problema $1/r_j/\Sigma w_j C_j$ è NP-hard. [Ex1]

Richiamo. Dato un problema di ottimizzazione $P=(z,S)$ (di minimo), si definisce ***rilassamento*** di P un nuovo problema RP=(w,Φ) tale che:

(i) $S \subseteq \Phi$
(ii) $\forall\ x \in S$ risulta $w(x) \leq z(x)$

Rilassamento preemptivo

Il problema $1/r_j, prmp/\Sigma w_j C_j$ si dice rilassamento (perché lo è?) preemptivo.

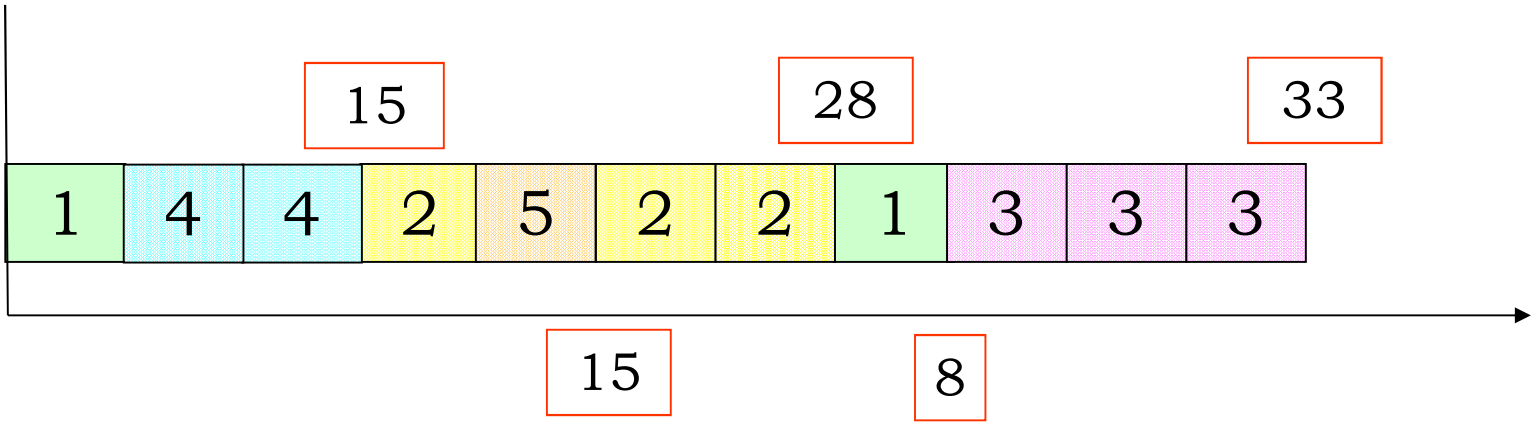
Per esso è interessante valutare il comportamento della naturale estensione della regola WSPT:

Preemptive WSPT: in ogni istante (intero) di tempo, si processa il job disponibile con il massimo rapporto

peso / tempo residuo di processamento

Esempio

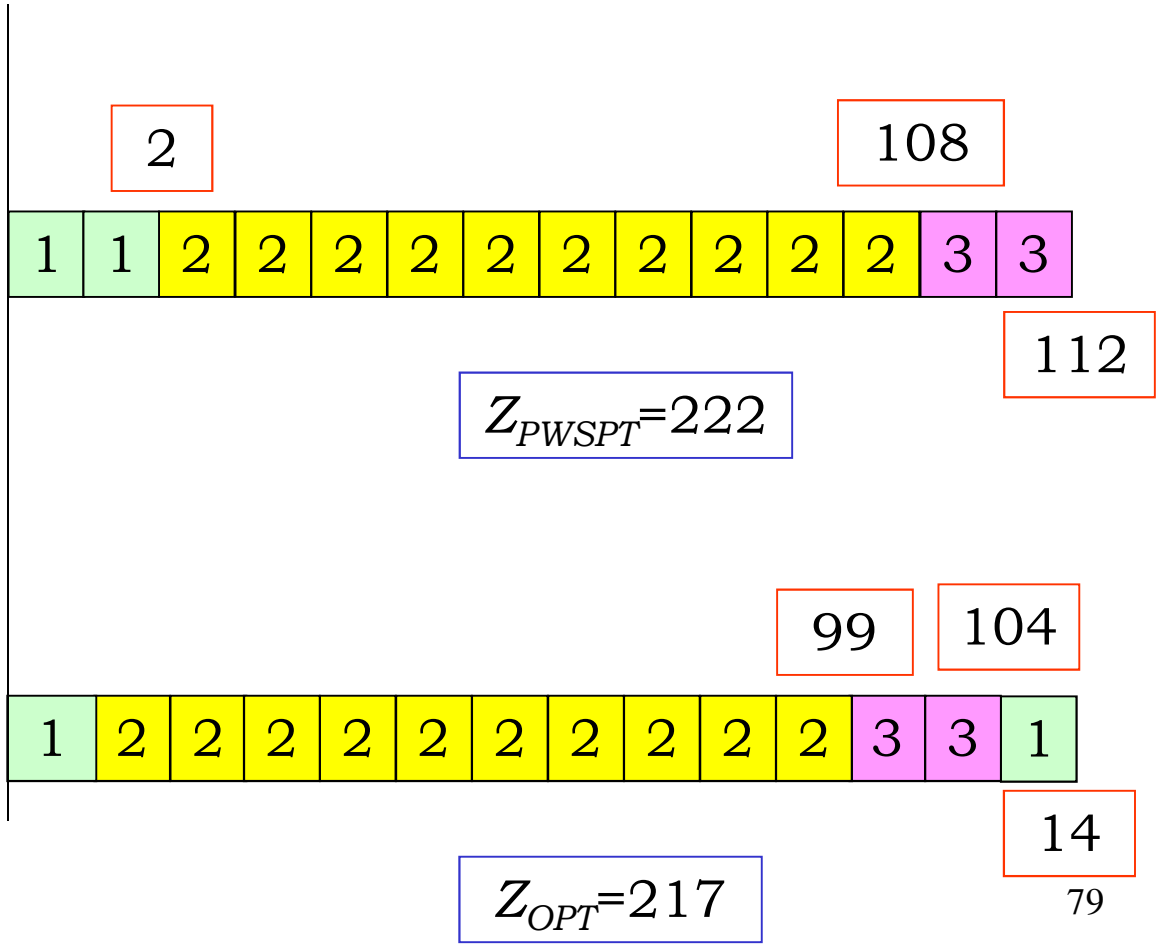
job	1	2	3	4	5
w_j	1	4	3	5	3
p_j	2	3	3	2	1
r_j	0	2	3	1	4



$Z_{PWSPT}=99$

PWSPT non è ottima per $1/r_j, prmp/\sum w_j C_j$

job	1	2	3
w_j	1	9	8
p_j	2	10	2
r_j	0	1	11



Implementazione

Gli eventi significativi sono di due tipi: (*i*) completamento di un job e (*ii*) rilascio di un nuovo job. Quindi, ci sono $O(n)$ eventi.

Inizializzazione. Ordina i job per valori w_j/p_j e per r_j crescenti; etichetta job rilasciati

Iterazione:
- Sceglie il primo job *j* rilasciato
- calcola l'evento successivo $t_{next} = \min(t + p_j(t), r_{next})$, che richiede tempo costante
- aggiorna la lista dei job rilasciati a t_{next}, [O(log n) per ogni nuovo job inserito]

PWSPT si esegue in tempo $O(n \log n)$

Pesi unitari

Nel caso di pesi tutti unitari PWSPT schedula, in ogni istante (intero) di tempo, il job disponibile con il minimo tempo residuo di processamento (**Shortest Remaining Processing Time, SRPT**)

Si ha il seguente:

Lemma 1.6
La regola SRPT è ottima per $1/r_j, prmp/\Sigma C_j$

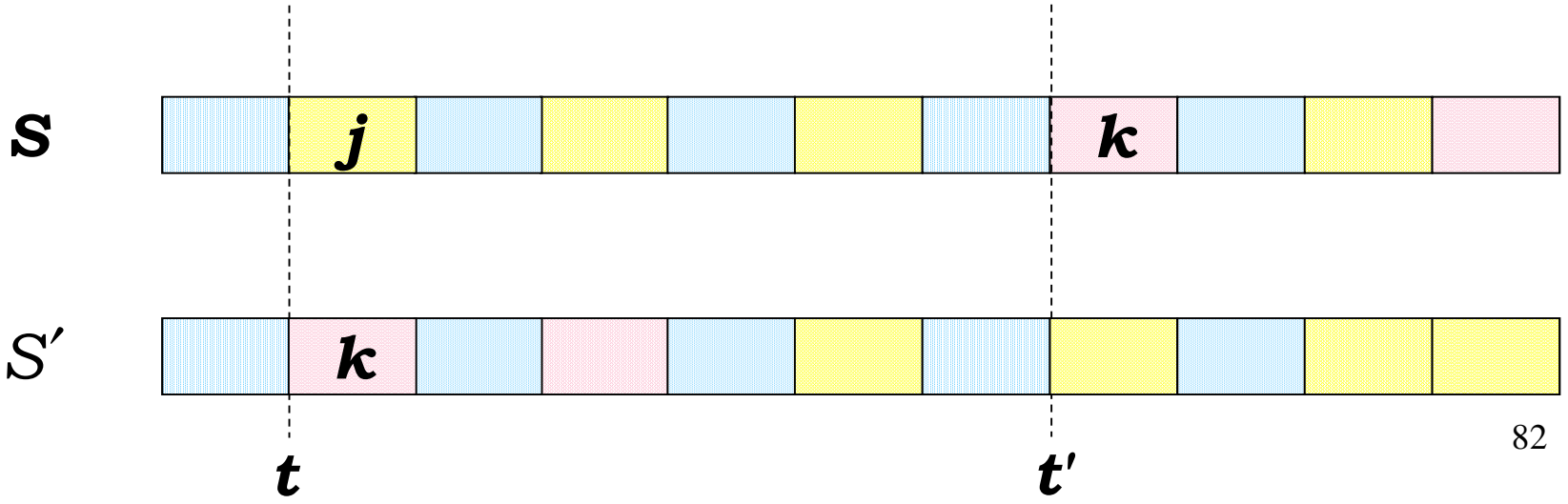
Dimostrazione. Mostriamo che, comunque preso uno schedule *S* che non rispetta SRPT, ne esiste uno *S'* che la rispetta con un tempo di completamento non peggiore.

Dimostrazione

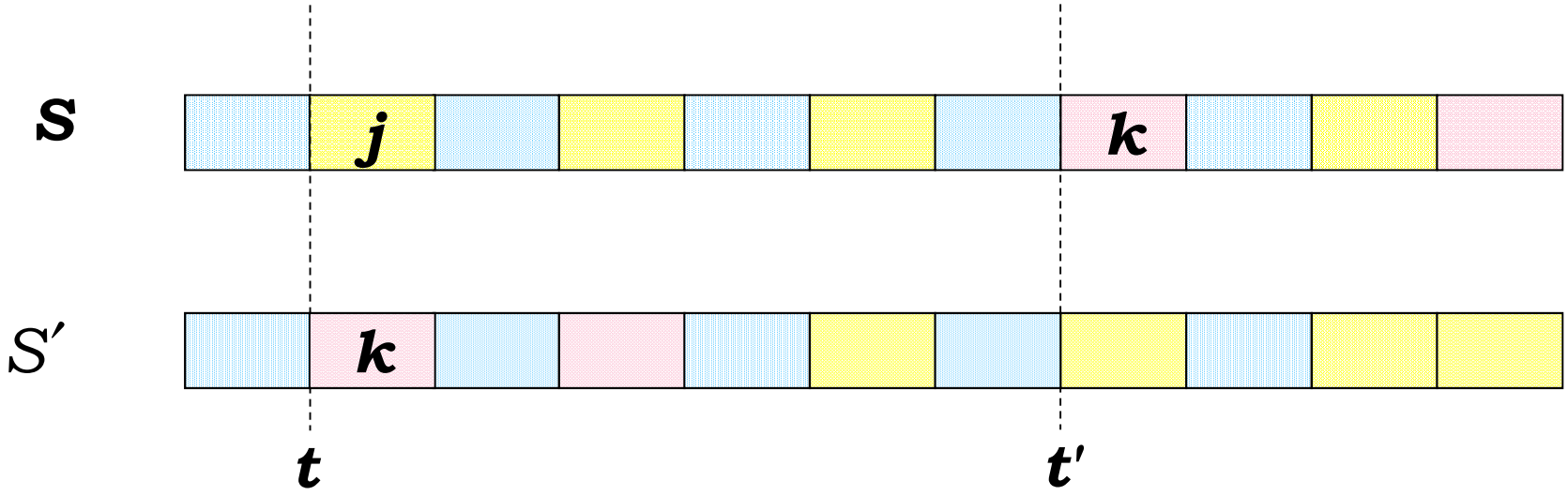
Se S non rispetta SRPT, deve esserci un istante t in cui è processata una unità del job j pur esistendo k tale che

$$p_j(t) > p_k(t)$$

Costruiamo un nuovo schedule S' in cui le rimanenti $p_k(t)$ unità del job k sono scambiate con le prime $p_k(t)$ del job j.



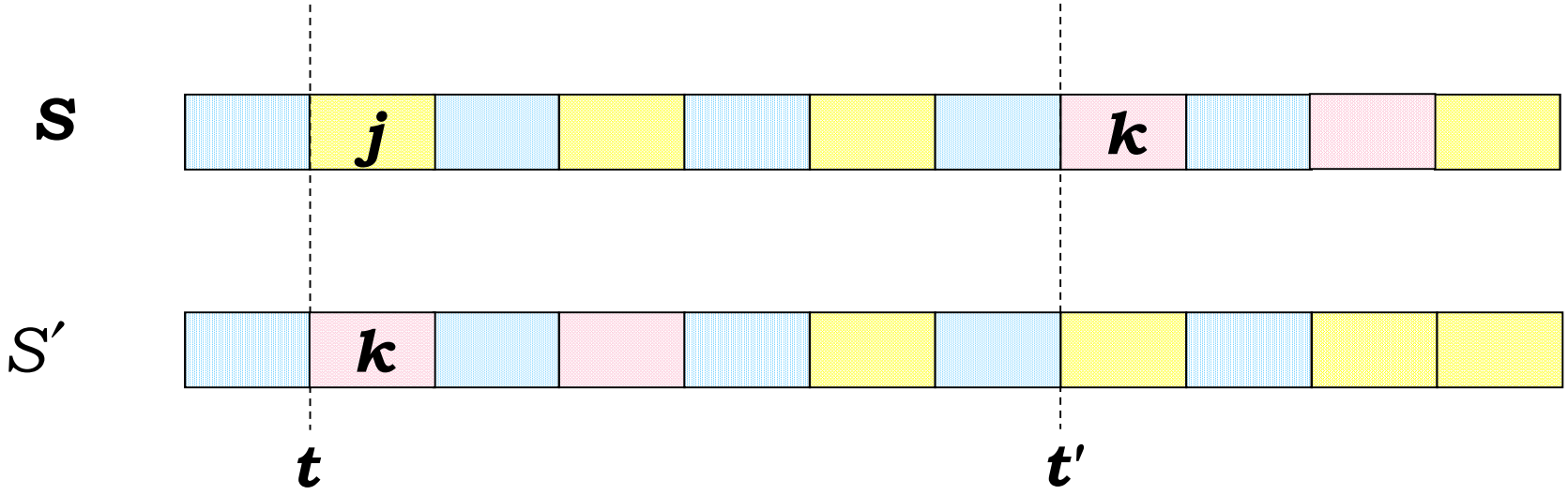
Dimostrazione



- i job diversi da j e k non subiscono alcuna variazione;
- primo caso: $C_k(S) > C_j(S)$
  - Essendo $p_j(t) > p_k(t)$ si ha che $C_k(S') \leq C_j(S)$;
  - inoltre, per costruzione, $C_j(S') = C_k(S)$;

Quindi: $C_k(S') + C_j(S') \leq C_k(S) + C_j(S)$

Dimostrazione



- secondo caso: $C_k(S) < C_j(S)$

  - $C_k(S') \leq C_k(S)$;
  - $C_j(S') = C_j(S)$;

  Quindi: $C_k(S') + C_j(S') \leq C_k(S) + C_j(S)$

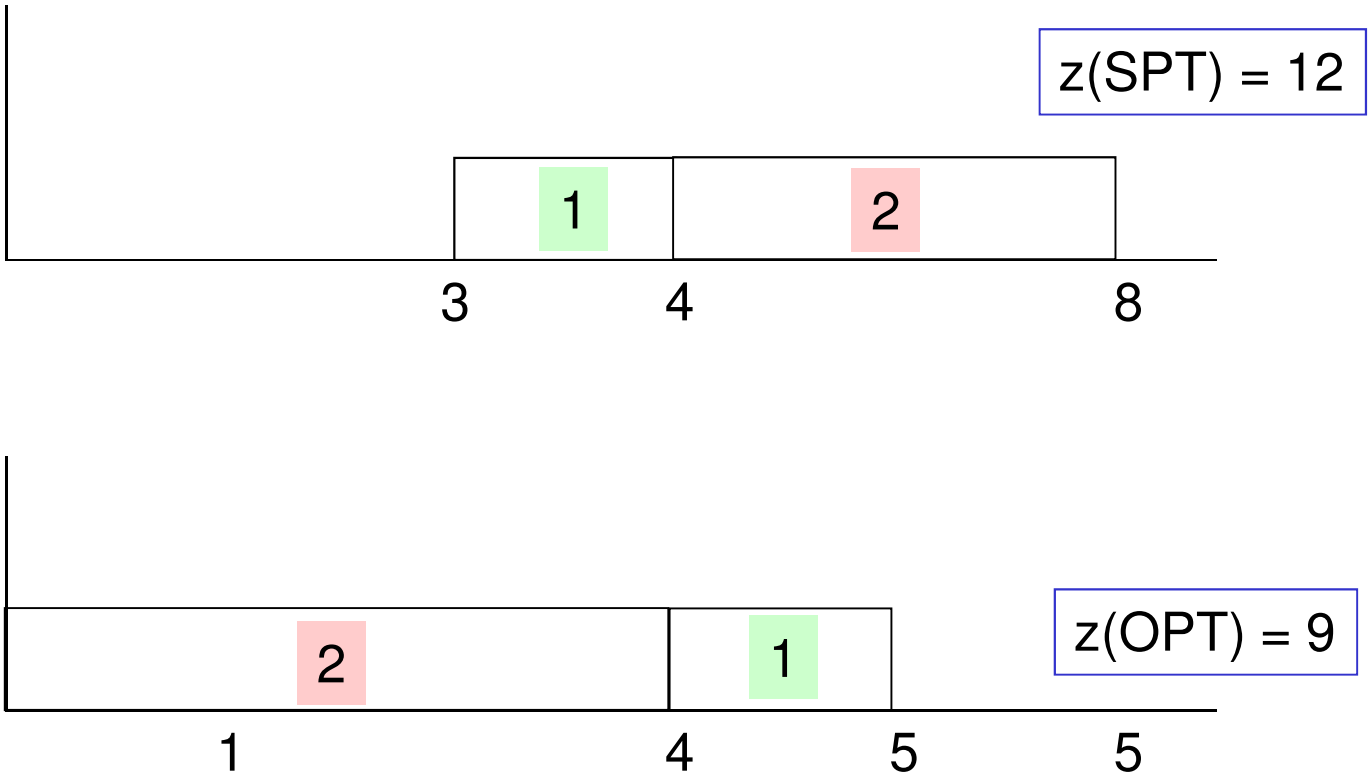
□

Esercizi

Esercizio 1

Dimostrare che la regola SPT non è ottima per il problema $1/r_j/\sum C_j$

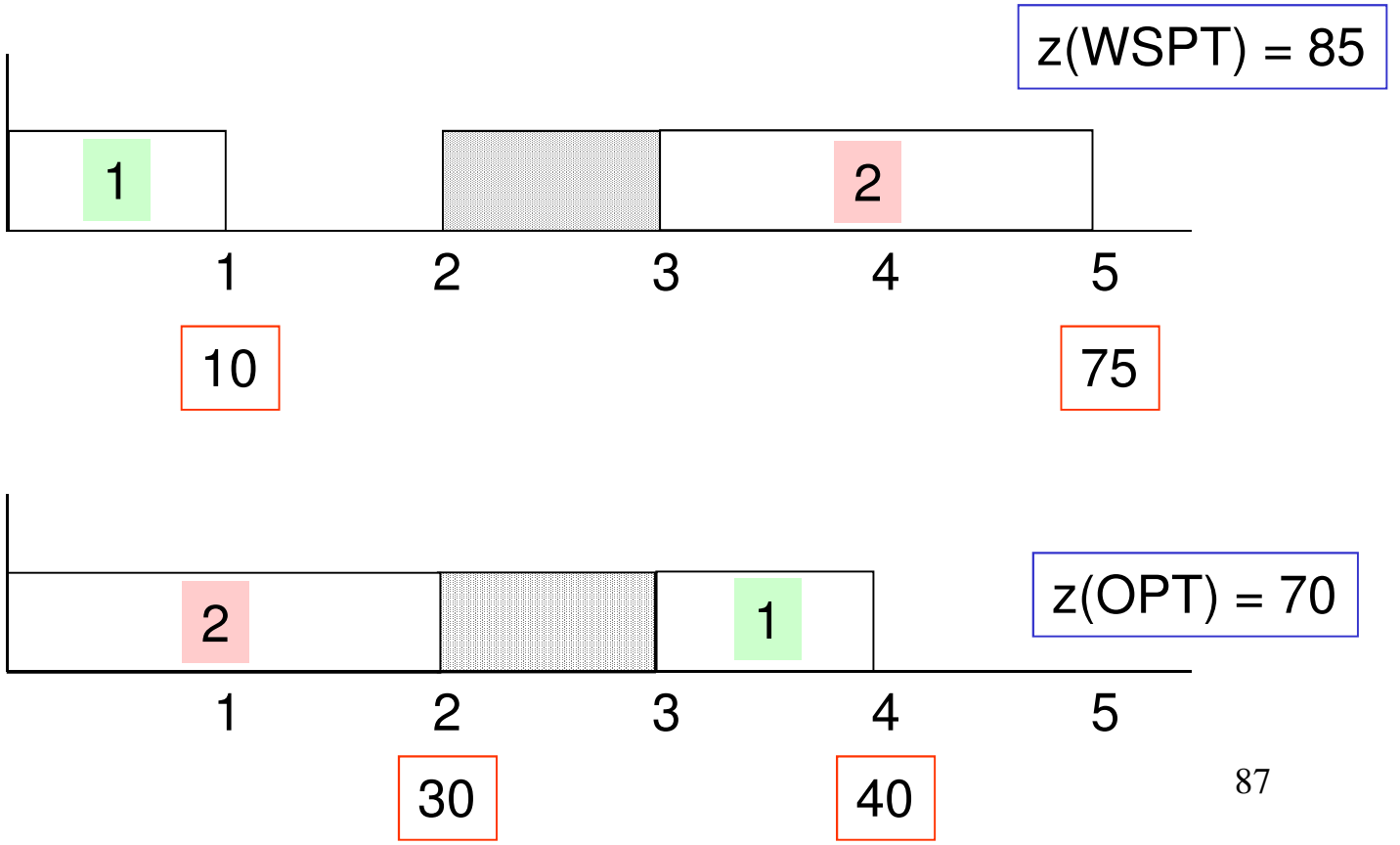
job	1	2
r_j	3	0
p_j	1	4



Esercizio 2

Dimostrare che la regola WSPT non è ottima per il problema $1/brkdwn/\sum w_j C_j$

job	1	2
w_j	10	15
p_j	1	2

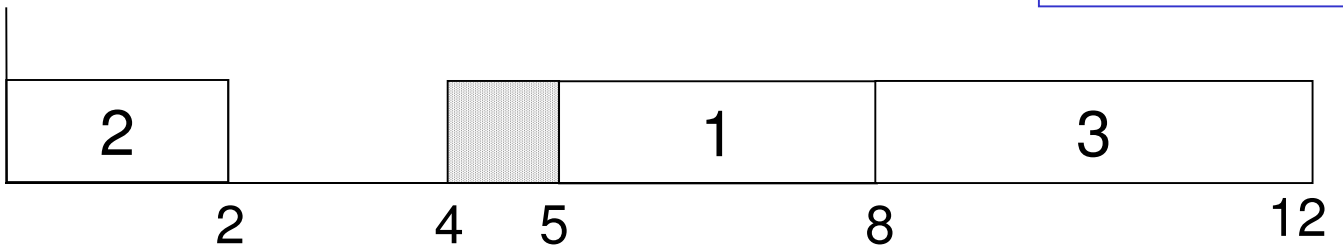


Esercizio 3

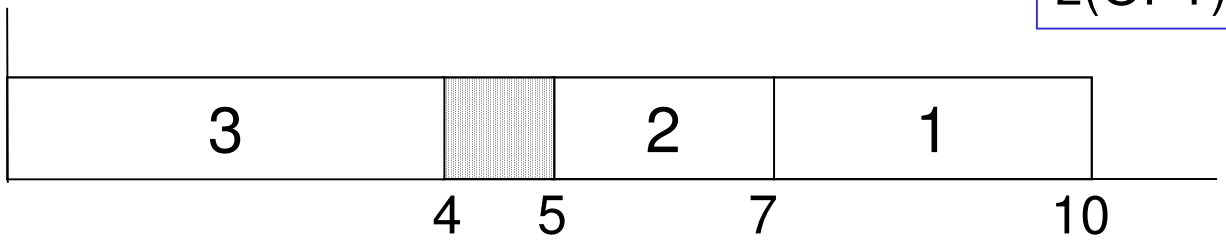
Dimostrare o confutare che la regola SPT è ottima per il problema $1/brkdwn/\sum C_j$

job	1	2	3
p_j	3	2	4

z(SPT) = 22



z(OPT) = 21



1.3 Problemi min-max (Massima Lateness L_{max})

$1/prec/h_{max}$

Forma della funzione obiettivo:

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

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

Esempi:

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

Precedenze: G_P generico grafo aciclico.

Algoritmo di *Lawler*

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

Costruisce lo schedule a partire dal fondo (*backward dynamic programming*)

- J job già schedulati, processati nell'intervallo

$$[C_{\max} - \sum_{j \in J} p_j, \quad C_{\max}]$$

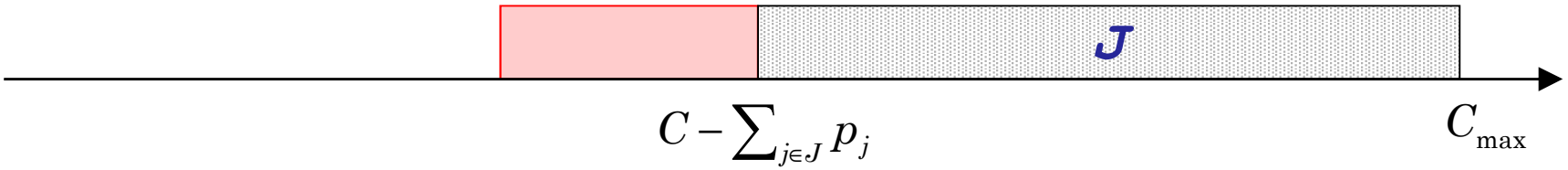
- $J^c = \{1, \ldots, n\} \setminus J$

- J' job schedulabili immediatamente prima di J (= tutti i successori sono in J)

  si osservi che $\sum_{j \in J^c} p_j = C_{\max} - \sum_{j \in J} p_j$

Algoritmo di *Lawler*

posizione corrente



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

Loop: while ($J^c \neq \emptyset$)
Schedula nella posizione corrente il job j^* tale che

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

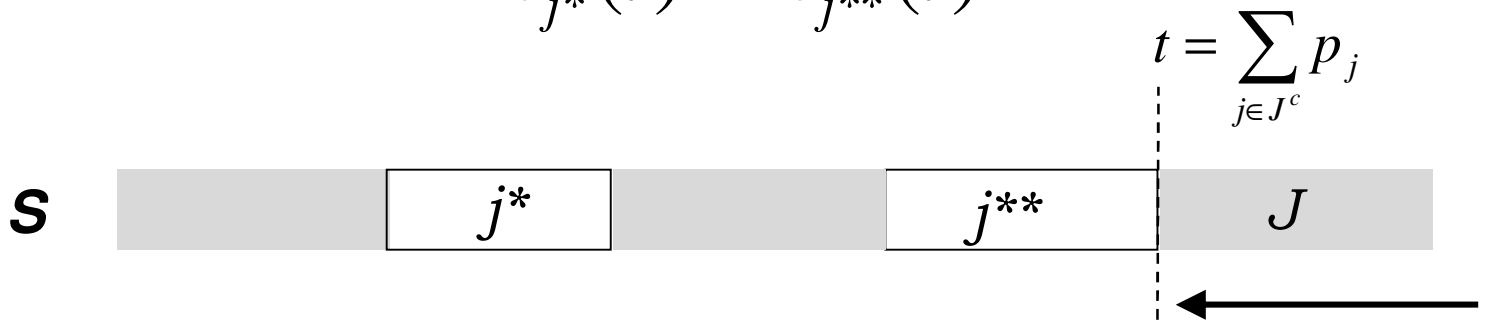
$J := J \cup \{j^*\}$, $J^c := J^c \setminus \{j^*\}$, aggiorna J'
Endloop.

Corretezza

Teorema 1.7 L'algoritmo di Lawler restituisce uno schedule ottimo per $1/prec/h_{max}$

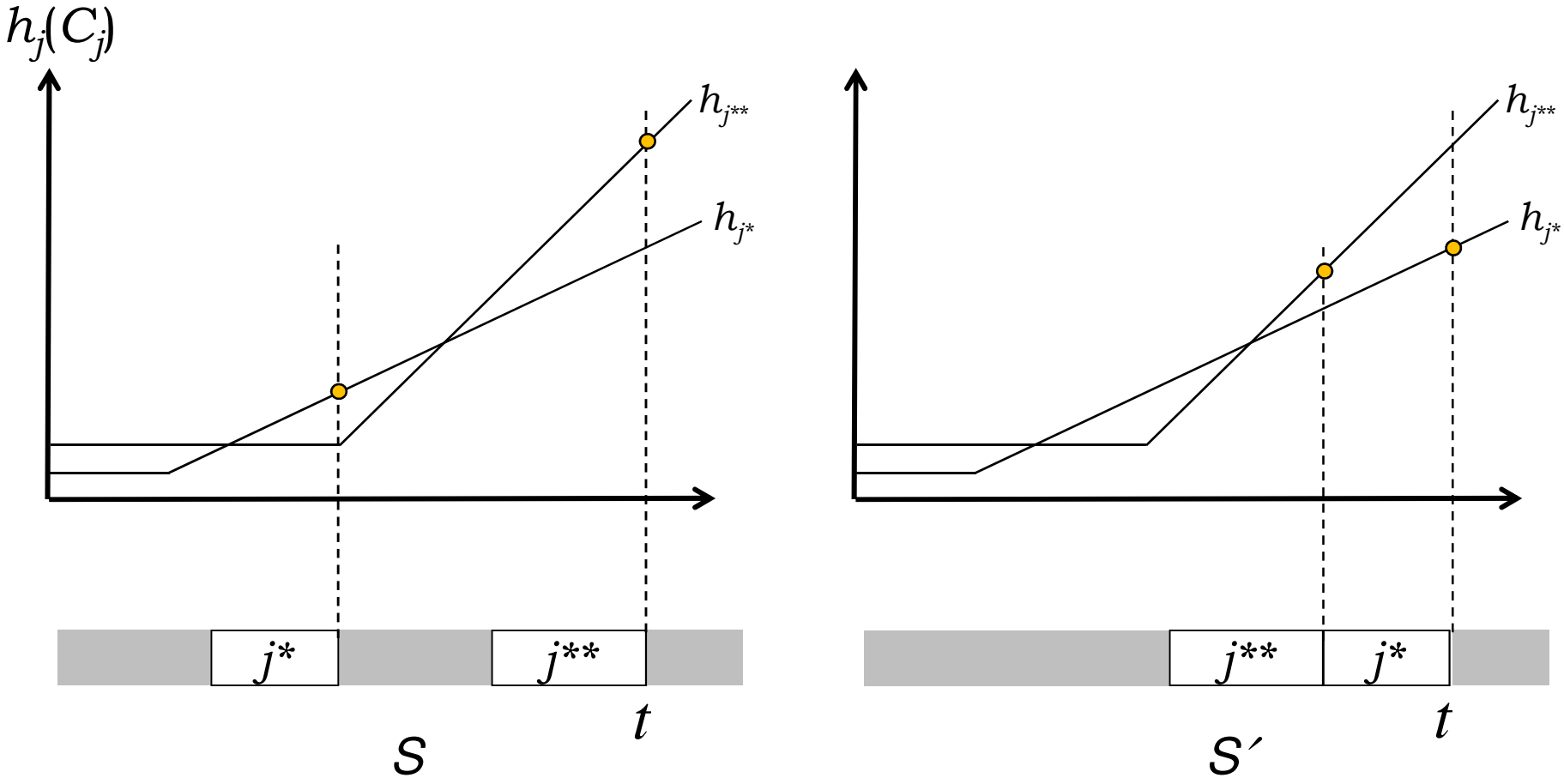
Dimostrazione. (Per contr.) Sia S una sequenza ottima per cui, ad una generica iterazione, è selezionato il job $j^{**} \in J'$ ma esiste $\exists\, j^*$ schedulabile tale che

$$h_{j^*}(t) < h_{j^{**}}(t)$$



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

Dimostrazione (continua)



Consideriamo un nuovo schedule S′ ottenuto da *S* spostando il job j^{**} subito dopo j^*. [L'unico job che peggiora è j^*]

Corretezza

$$h'_{j^*}(t) = h_{j^*}(t) < h_{j^{**}}(t)$$

$$h'_{\max} = \max\left\{h'_{j^*}(t), \max_{j \neq j^*}\left\{h'_j(C'_j)\right\}\right\} \leq \max\left\{h_{j^{**}}(t), \max_{j \neq j^*}\left\{h'_j(C'_j)\right\}\right\} \leq$$

$$\leq \max\left\{h_{j^{**}}(t), \max_{j \neq j^*}\left\{h_j(C_j)\right\}\right\} = h_{\max}$$

$$C'_j \leq C_j$$

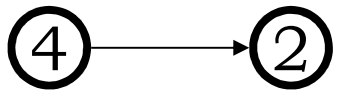
Quindi *S'* meglio di S, contraddizione.

□

Esempio

job	1	2	3	4
p_j	3	4	2	6
h_j	10	$1+C_2$	$1.5\ C_3$	$2\ (C_4/5)$

G_p

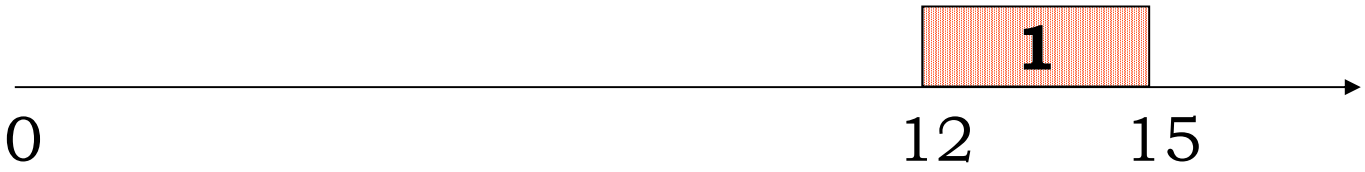


Iter 1.

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

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

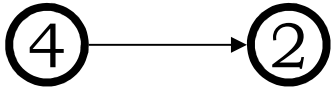
job	1	2	3
costo	10	16	22.5



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)

G_p

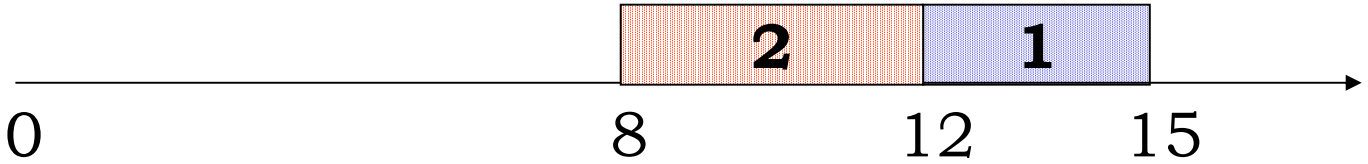


Iter 2.

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

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

job	2	3
costo	13	18



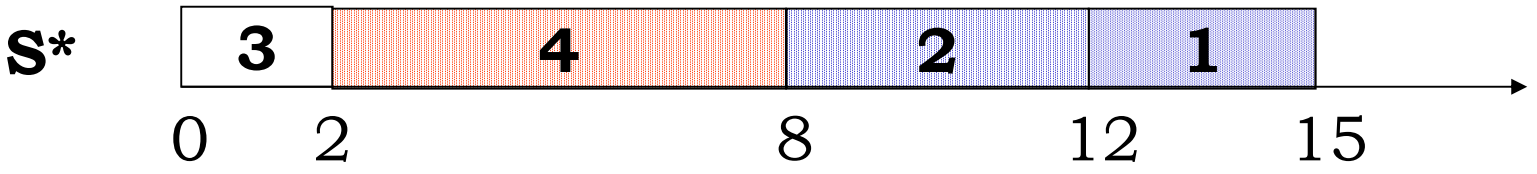
Esempio

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



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

Algoritmo di Lawler: caso speciale

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 $1 \mid\mid L_{max}$ l'algoritmo di Lawler si specializza:

$$h_j(C_j) = C_j - d_j$$

$$\min_{j \in J'}(h_j(\sum_{j \in J^c} p_j)) = \max_{j \in J'}(d_j)$$

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

$1|r_j|C_{max}$

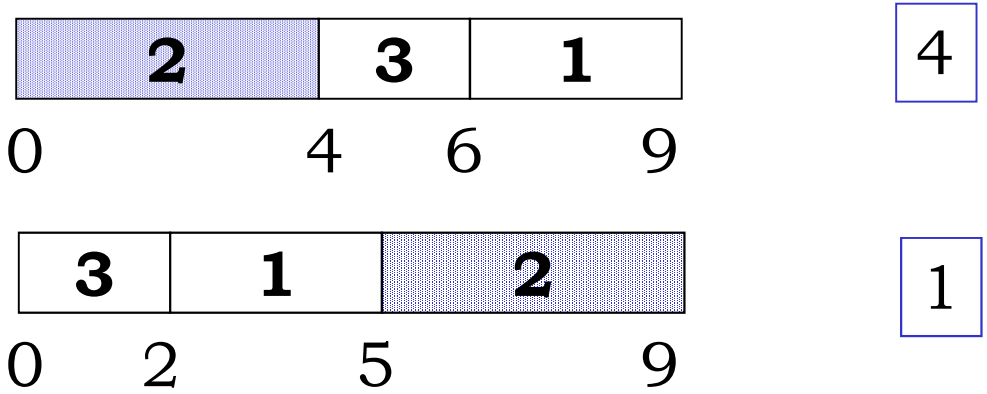
- anche il problema $1|r_j|C_{max}$ si riduce a $1||L_{max}$:
  - definiamo $d_j := K - r_j$, con $K > \max r_j$
  - calcoliamo uno schedule ottimo S del corrispondente problema $1||L_{max}$
  - sia $\overline{S}$ lo schedule ottenuto invertendo S: $\overline{S}$ è ottimo per $1|r_j|C_{max}$

Esercizio

Dimostrare che la regola EDD non è ottima per il problema $1 \mid \mid \sum L_j$

Controesempio:

job	1	2	3
p_j	3	4	2
d_j	6	4	5



$1/r_j/L_{max}$: complessità

Teorema 1.8 $1/r_j/L_{max}$ è NP-hard in senso forte

Dimostrazione. Riduzione polinomiale da 3-PARTITION

Istanza: interi $a_1, \ldots, a_{3t}, b$ tali che $b/4 < a_j < b/2, \sum_{j=1}^{3t} a_j = tb$

Problema: $\exists$ partizione in t terne tutte di peso b?

Costruiamo la seguente istanza di $1/r_j/L_{max}$ con $n = 4t-1$

$r_j = jb + (j-1), \quad p_j = 1, \quad d_j = jb + j, \qquad j = 1, \ldots, t-1$

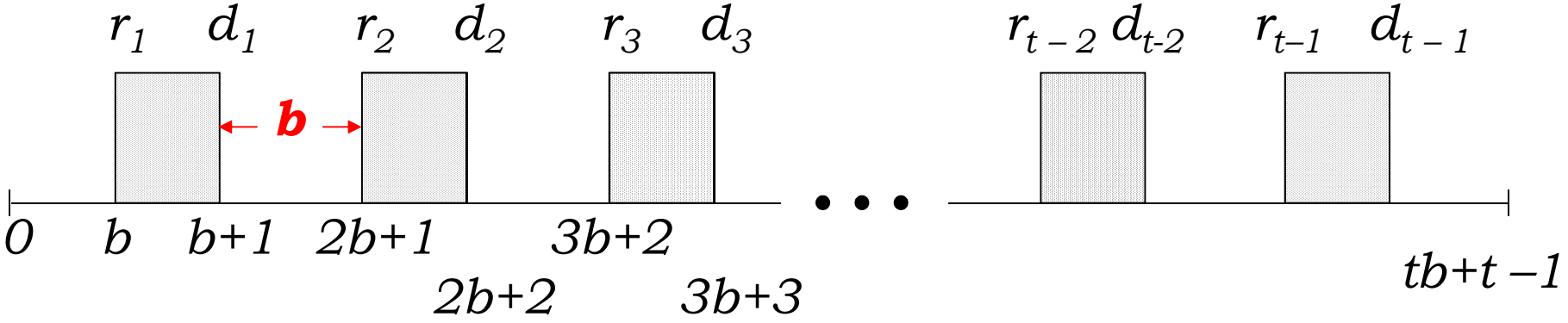
$r_j = 0, \quad p_j = a_{j-t+1}, \quad d_j = tb + (t-1), \qquad j = t, \ldots, 4t-1$

Problema (target): $\exists$ uno schedule di valore $z \leq 0$?

$1/r_j/L_{max}$: complessità

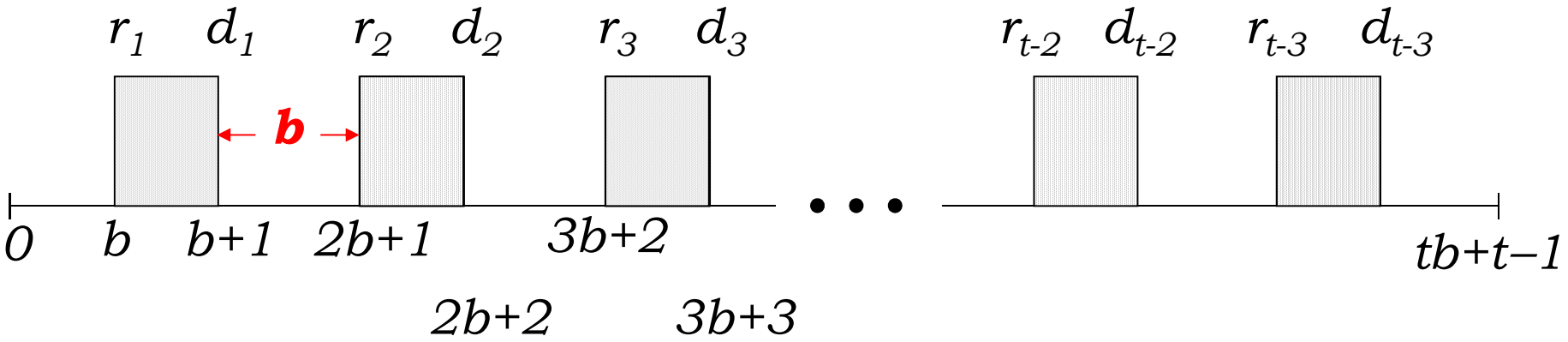
$r_j=jb+(j-1), \quad p_j=1, \quad d_j=jb+j, \quad j=1,\ldots,t-1$

$\exists$ schedule con $L_{max} \leq z = 0$ se e solo se ogni job j, $j = 1,\ldots,t-1$, è processato fra r_j e $d_j = r_j+p_j$



Fissando i primi $t-1$ job "bloccanti", rimangono **t intervalli di lunghezza b**

$1/r_j/L_{max}$: complessità



$\exists$ schedule con $L_{max} \leq 0$ se e solo se i restanti $3t$ job:

$r_j = 0$, $p_j = a_{j-t+1}$, $d_j = tb + (t-1)$, $j = t,\ldots,4t-1$

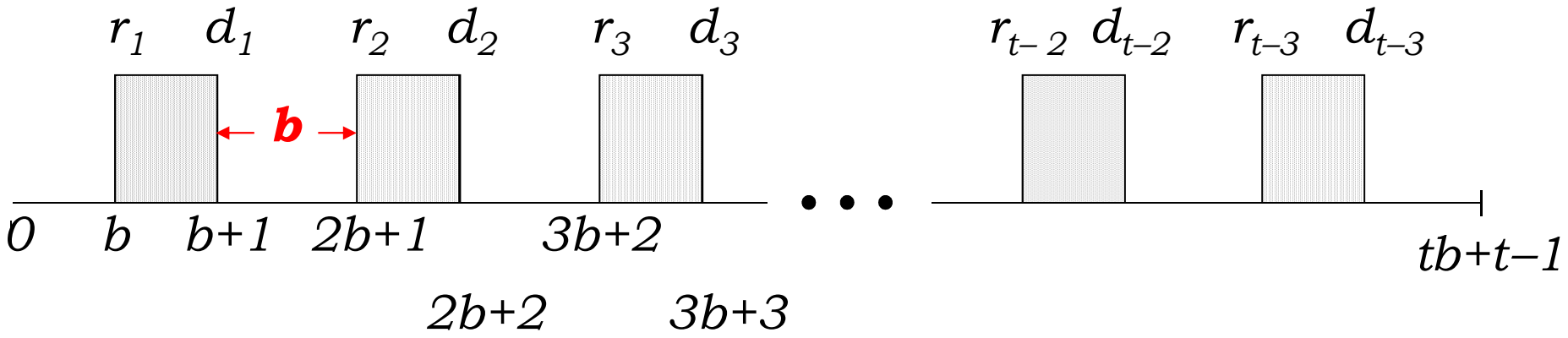
per cui vale:

$$\sum_{j=t}^{4t-1} p_j = \sum_{j=t}^{4t-1} a_{j-t+1} = tb$$

possono essere schedulati negli t intervalli di lunghezza b.

$1/r_j/L_{max}$: complessità

$r_j = 0,\ p_j = a_{j-t+1},\qquad d_j = tb + (t-1),\quad j = t,\ldots,4t-1$

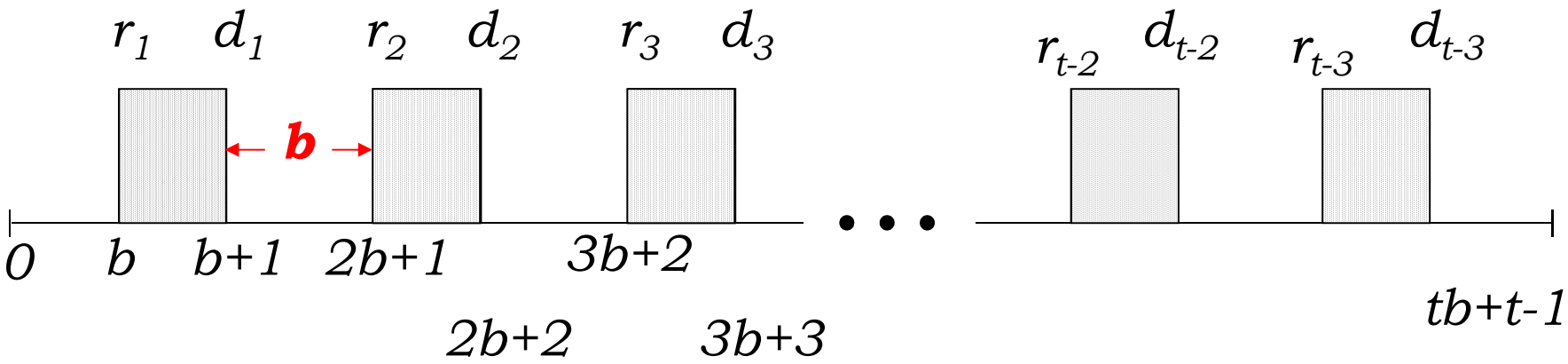


- Essendo $b/4 < a_j$, in ciascun intervallo di lunghezza b non entrano più di tre job
- essendo $a_j < b/2$ e $\sum_{j=1}^{4t-1} p_j = \sum_{j=1}^{4t-1} a_{j-t+1} = tb$

mettendo meno di tre job in un intervallo, almeno un job è schedulato in ritardo

$1/r_j/L_{max}$: complessità

$r_j = 0,\ p_j = a_{j-t+1},\qquad d_j = tb + (t-1),\quad j = t,\ldots,4t-1$



Quindi, i job $t,\ldots, 4t-1$ possono essere schedulati negli t intervalli di lunghezza b sse possono essere suddivisi in t terne di lunghezza b, cioè, sse 3-PARTITION ha soluzione

Esercizio

A={27,27,29,33,33,33,35,35,35,37,37,39}
b=100

esiste una partizione in 4 terne di peso 100 sse la seguente istanza di 1|rj|Lmax ammette soluzione con Lmax ≤ 0

job	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
r_j	100	201	302	0	0	0	0	0	0	0	0	0	0	0	0
d_j	101	202	303	403	403	403	403	403	403	403	403	403	403	403	403
p_j	1	1	1	27	27	29	33	33	33	35	35	35	37	37	39

$1/r_j,prmp/L_{max}$

Preemptive **EDD**: ad ogni istante t, si processa il job disponibile con la minima due date

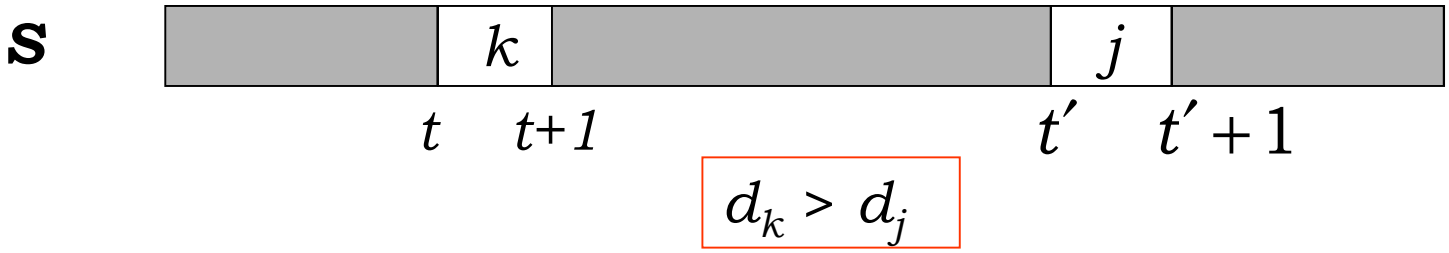
$R(t) = \{j: r_j \leq t\}$ insieme dei job disponibili al tempo t

Teorema 1.9 La regola PEDD calcola una soluzione ottima del problema $1/r_j, prmp/L_{max}$.

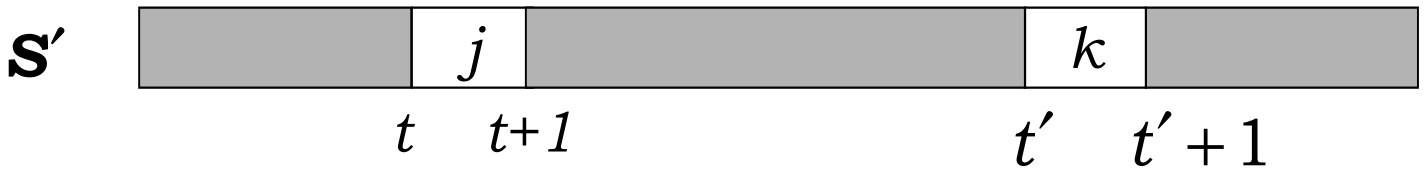
Dimostrazione. Consideriamo uno schedule ***S*** che viola PEDD:
Al tempo t è schedulato un job $k \in R(t)$ per cui esiste un job $j \in R(t)$ disponibile e non ancora terminato
tale che $d_k > d_j$

PEDD - correttezza

Poiché j non è terminato, $\exists$ istante $t' > t$ in cui j è in esecuzione

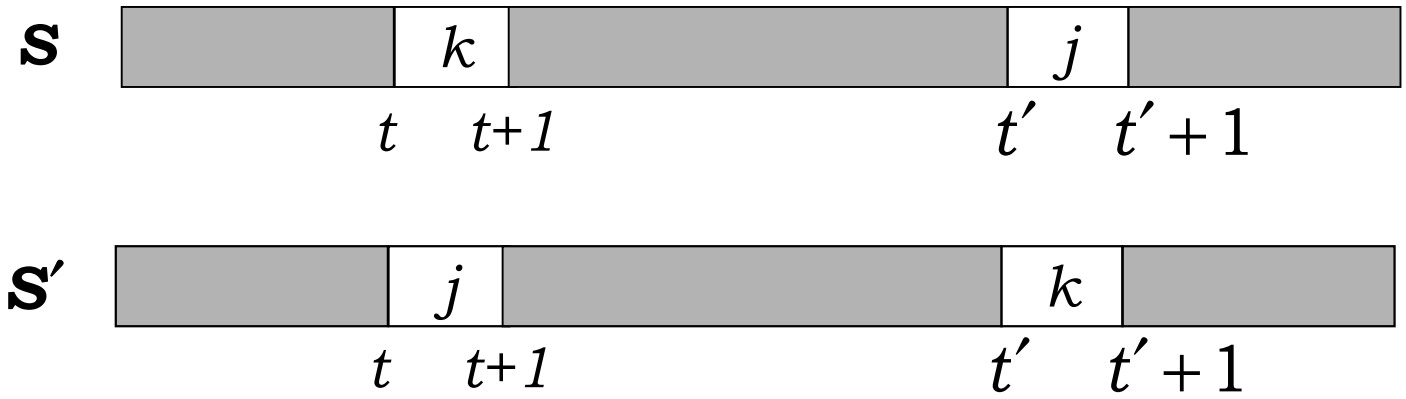


Consideriamo il nuovo schedule **S′** ottenuto da **S** scambiando j e k nei due intervalli di figura:



Il contributo dei job diversi da j e k non cambia

PEDD - correttezza



$d_k > d_j$

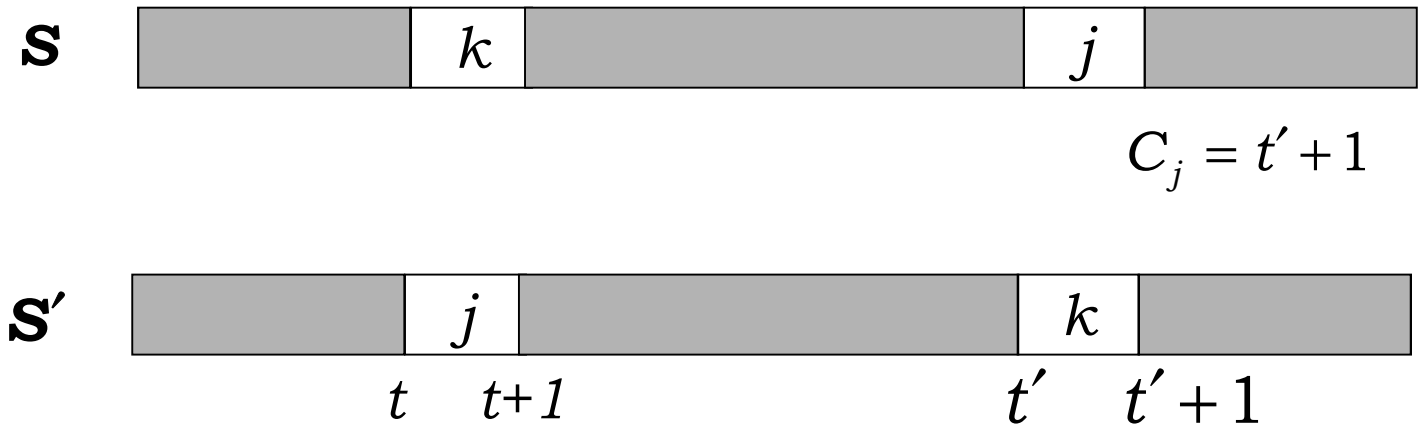
Definiamo C_j e C_k i tempi di completamento di j e k **nello schedule *S***.

Dimostriamo che $L_{max}(S') \leq L_{max}(S)$. Due casi:

1. $C_j \geq t'+1, C_k > t'+1$
2. $C_j \geq t'+1, C_k < t'+1$

PEDD - correttezza

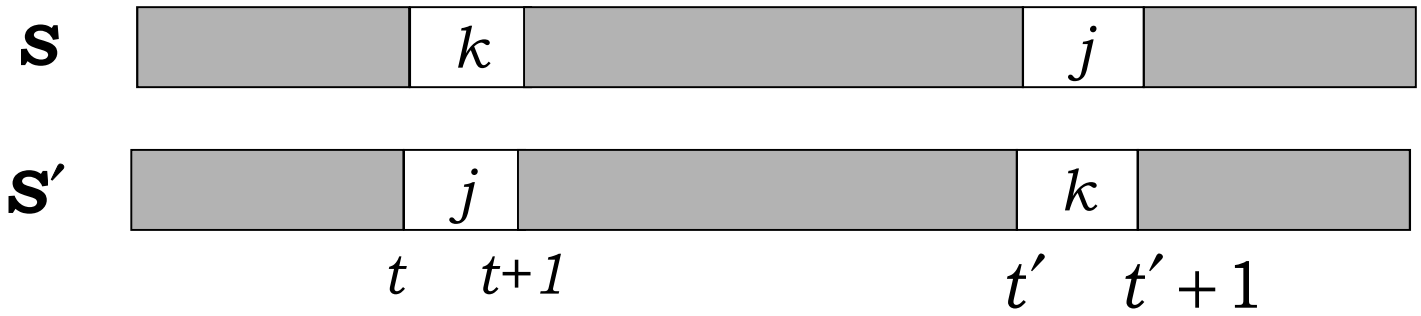
caso 1. $C_j \geq t' + 1, \quad C_k > t' + 1$



Essendo $C_k > t'+1$ il tempo di completamento di k in S' rimane inalterato; al contrario, quello di j non aumenta.
Quindi, $L_{max}(S') \leq L_{max}(S)$

PEDD - correttezza

caso 2. $C_j \geq t' + 1, \quad C_k < t' + 1$



Il tempo di completamento di k aumenta fino a $t'+1$.
Per l'ipotesi $d_k > d_j$, detta L'_k la lateness di k in S', risulta:

$$L'_k = t' + 1 - d_k < t' + 1 - d_j \leq C_j - d_j = L_j$$

PEDD - correttezza

Quindi

$$L_{\max}(S') = \max\left\{L'_k, \max_{i\neq k}\{L'_i\}\right\} \leq \max\left\{L'_k, \max_{i\neq k}\{L_i\}\right\} \leq$$

$$\leq \max\left\{L_j, \max_{i\neq k}\{L_i\}\right\} = L_{\max}(S)$$

$1/r_j/L_{max}$

Algoritmo branch-and-bound

Branch-and-bound

Metodo enumerativo che applichiamo nella soluzione di problemi difficili. Un algoritmo immediato sarebbe:

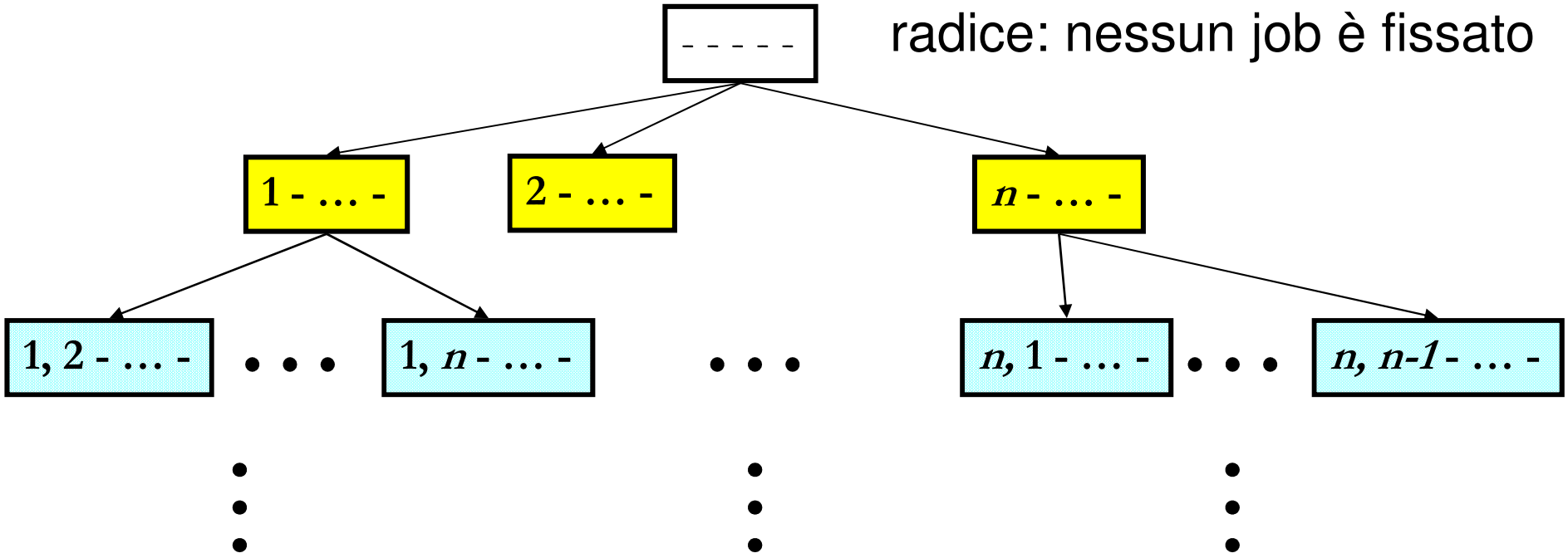
1. Enumera tutte le soluzioni ammissibili (sono in numero finito) e calcolane il costo
2. Scegli la soluzione di costo minimo

Non utilizzabile in pratica a causa del numero elevatissimo di soluzioni (dell'ordine di $n!$ per una singola macchina)

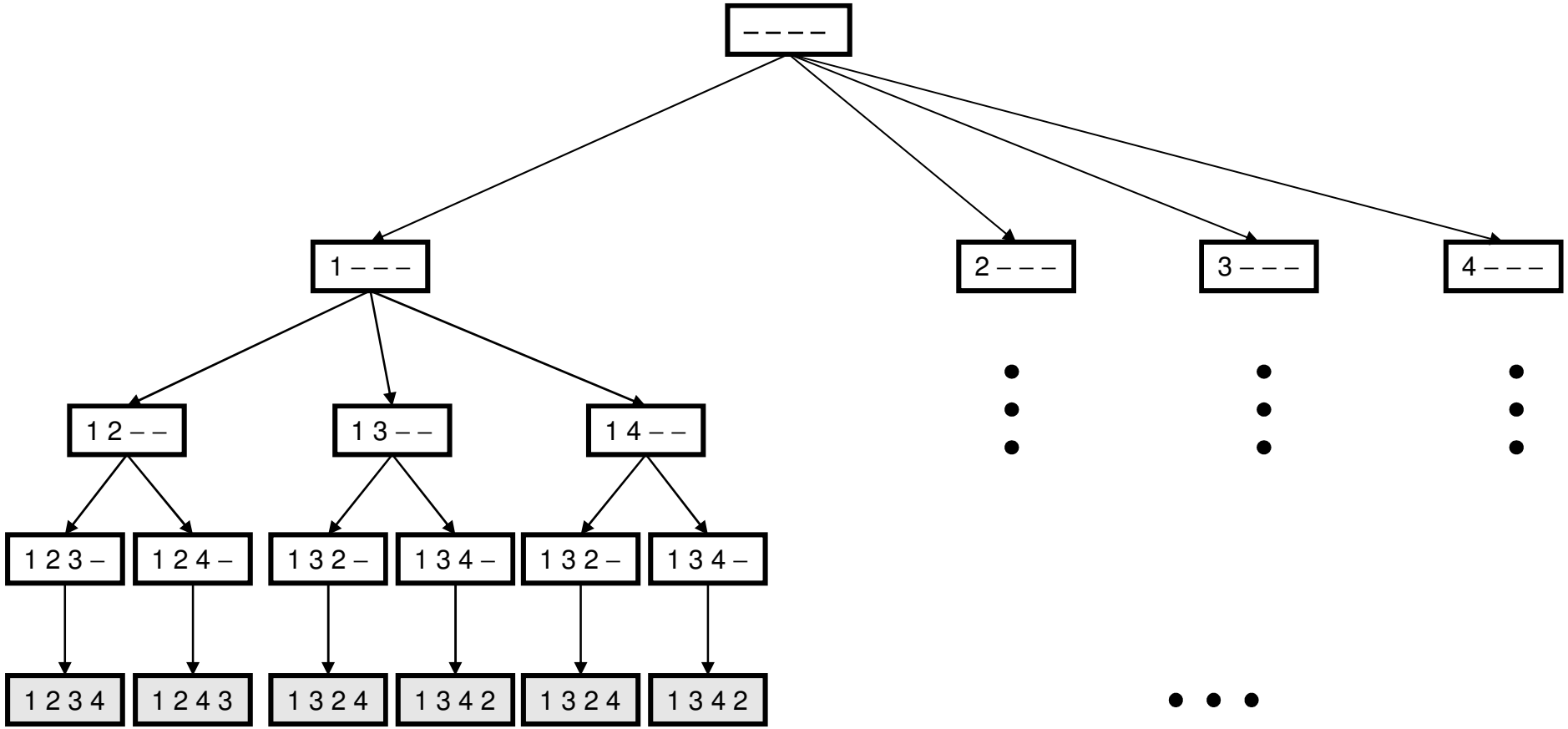
L'algoritmo branch-and-bound si prefigge di **esplorare parzialmente** l'insieme delle schedule ammissibili ma garantendo comunque un certificato di ottimalità alla terminazione

Albero di enumerazione

Al livello h dell'albero di enumerazione si fissa in tutti i modi possibili il job in posizione h

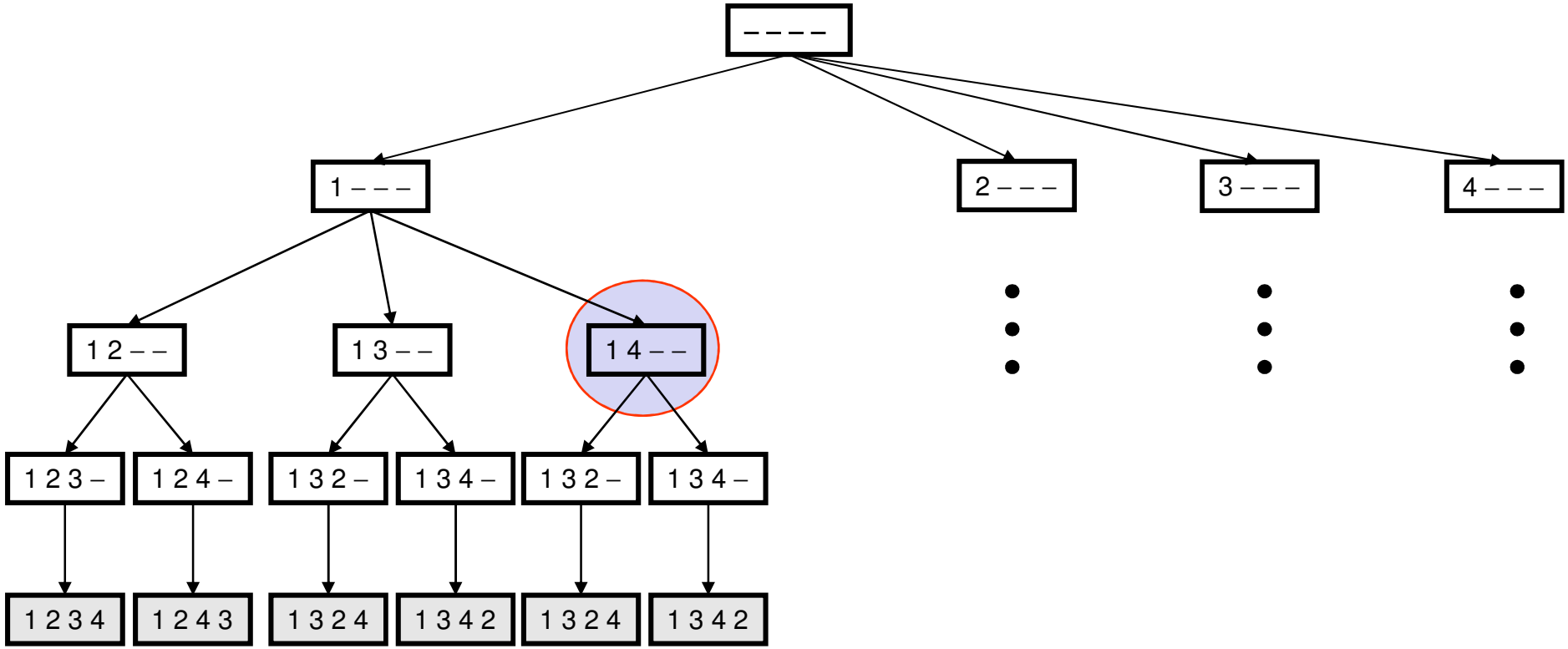


4 job, 1 macchina

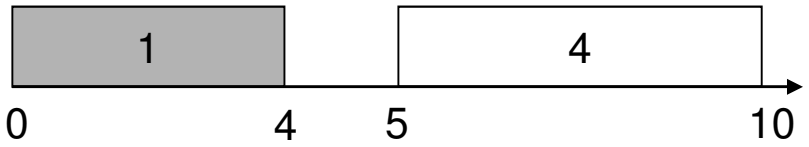


Le foglie corrispondono agli schedule ammissibili

4 job, 1 macchina



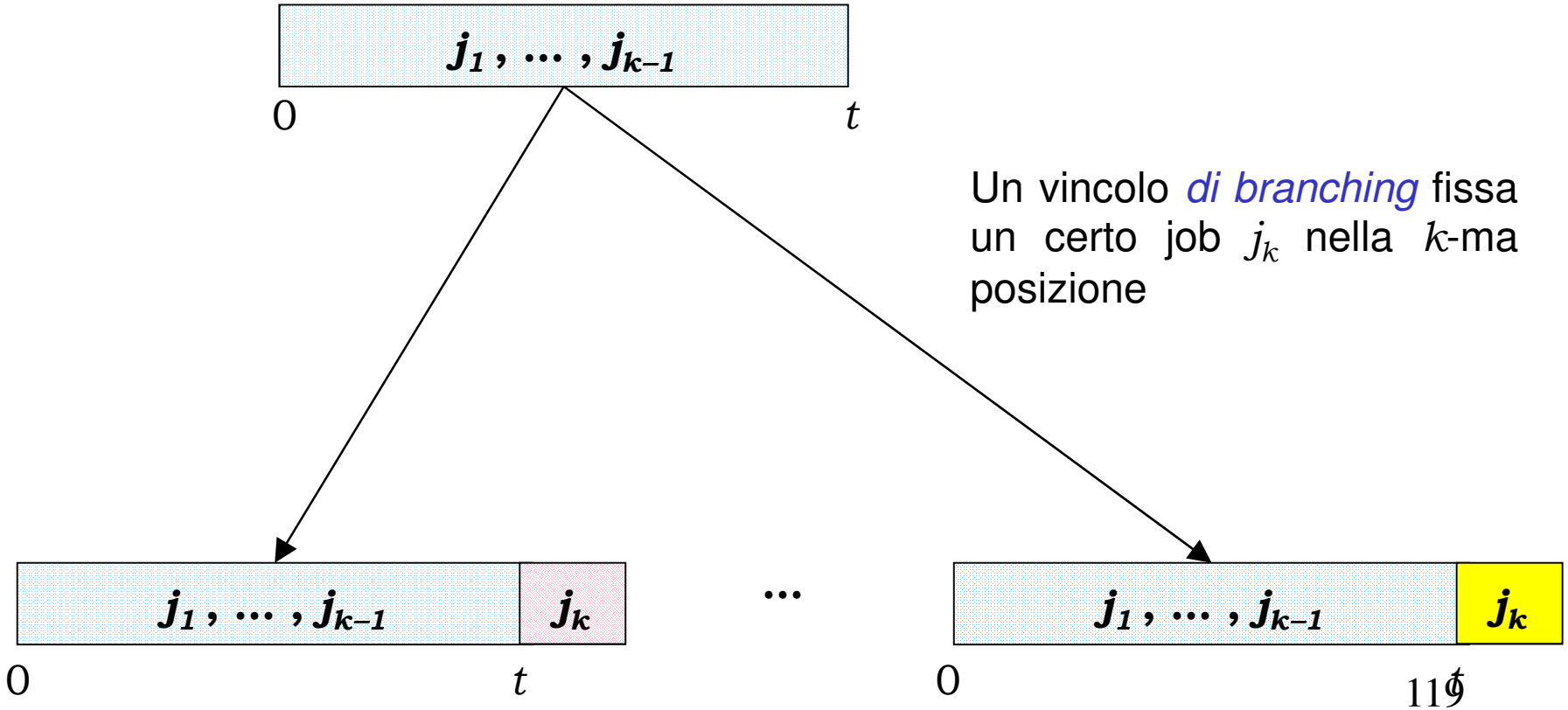
job	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



Ogni nodo non foglia corrisponde ad uno ***schedule parziale***

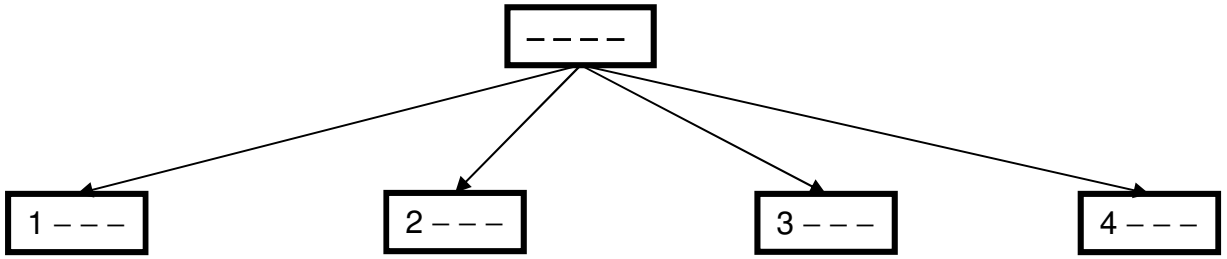
Vincoli di branching

Un sottoproblema $S(k-1)$ al livello $k-1$ contiene l'insieme delle schedule in cui le prime $k-1$ posizioni sono assegnate



Un vincolo *di branching* fissa un certo job j_k nella k-ma posizione

Decomposizione



Ad ogni branching si partiziona l'insieme S delle schedule ammissibile in sottoinsiemi S_i

Detto z^* il valore ottimo globale e z_i^* il valore ottimo del sottoproblema (S_i, γ), $i = 1,\ldots, p$.

Si ha quindi

$$z^* = \min_{i = 1,\ldots,p} z_i^*$$

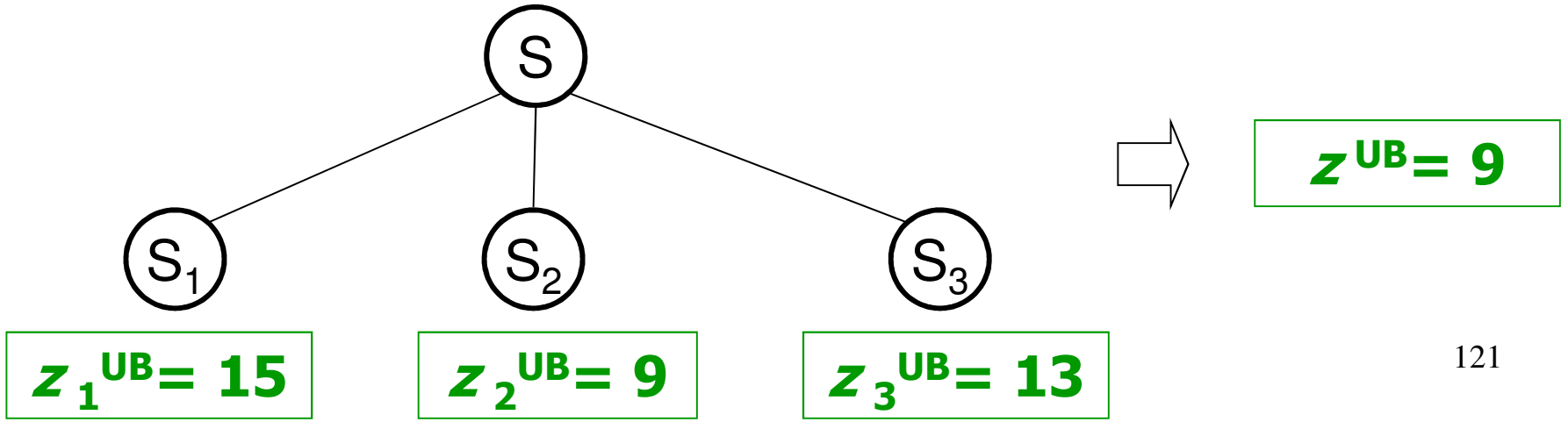
Limitazioni superiori di z^*

Sia $S = S_1 \cup S_2 \cup \ldots \cup S_p$ una partizione della regione ammissibile

Il valore di un qualunque schedule ammissibile per S_i è un *upper bound* UB_i per il valore ottimo z^*_i

Allora

$UB = \min_i UB_i$ è un upper bound per z^* (globale)



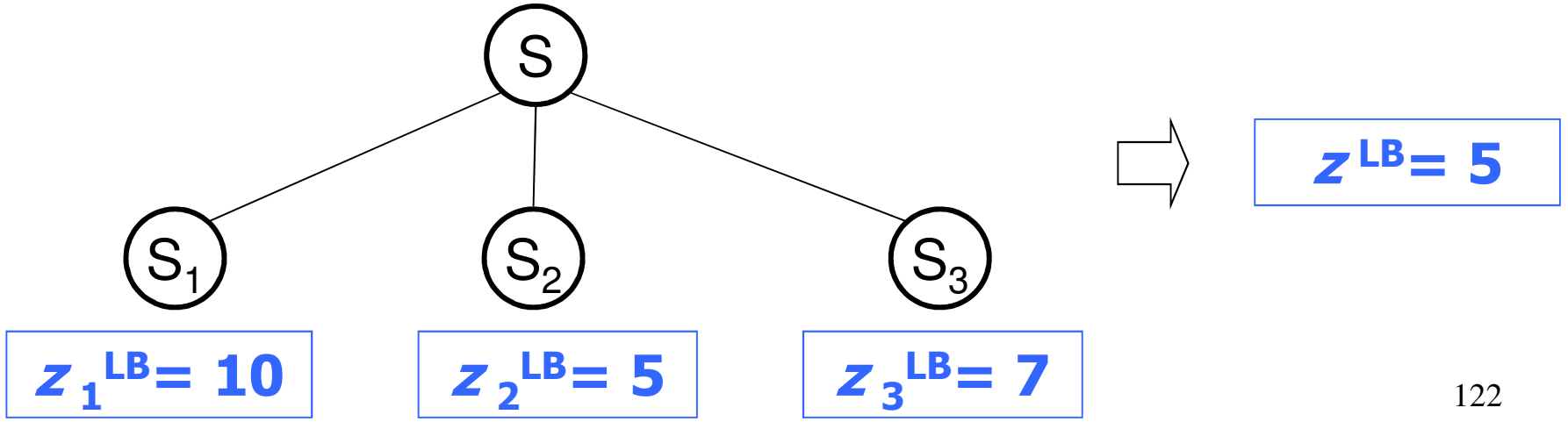
Limitazioni inferiori di z^*

Sia $S = S_1 \cup S_2 \cup \ldots \cup S_p$ una partizione della regione ammissibile

Se LB_i è un lower bound per il valore ottimo z^*_i

Allora,

$LB = \min_i LB_i$ è un lower bound per z^*



Calcolo di z_i^{LB}

Dato un problema di ottimizzazione $P=(z,S)$ (di minimo), si definisce ***rilassamento*** di P un nuovo problema RP=(w,Φ) tale che:

(i) $S \subseteq \Phi$

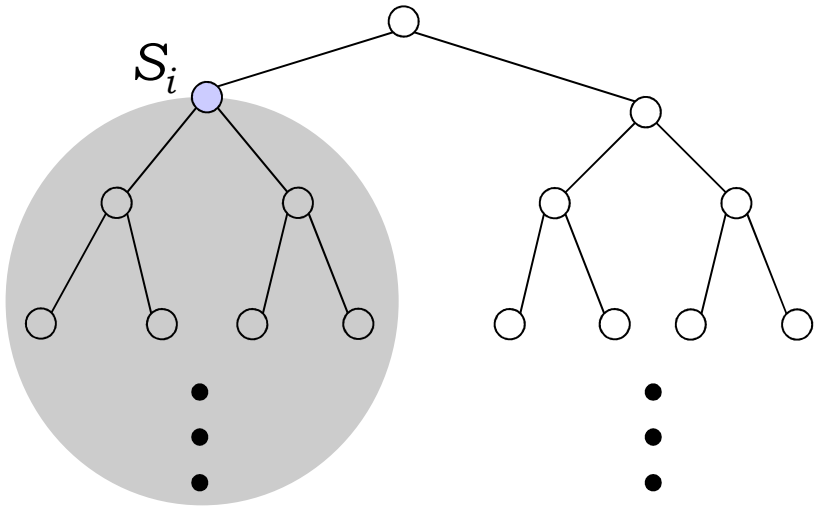
(ii) $\forall\ x \in S$ risulta $w(x) \leq z(x)$

- Il calcolo di LB_i si effettua risolvendo **in modo esatto** un opportuno rilassamento di (S_i, γ).
- La scelta del rilassamento si basa su due esigenze, spesso contrastanti:
  - Ottenere buone approssimazioni di z_i^*
  - Richiedere tempi di calcolo non elevati

Potatura

• Il metodo branch-and-bound si basa sull'idea di ridurre il più possibile il numero di sottoproblemi "valutati"

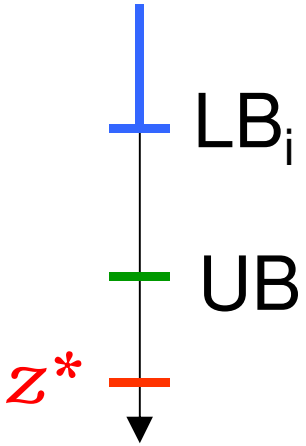
• Diversi criteri ci permettono di "potare un sottoproblema" (S_i, γ), cioè di NON procedere all'esplorazione del sottoalbero di radice S_i



Potatura per bound

- Sia UB un upper bound del valore ottimo z^* corrispondente al valore di uno schedule ammissibile *x*
- Sia LB_i **un** lower bound per il valore ottimo z^*_i

Se $LB_i \geq UB$



allora S_i non contiene soluzioni ammissibili migliori di *x* e il sottoproblema (S_i, γ) è **potato**

Potatura per ottimalità

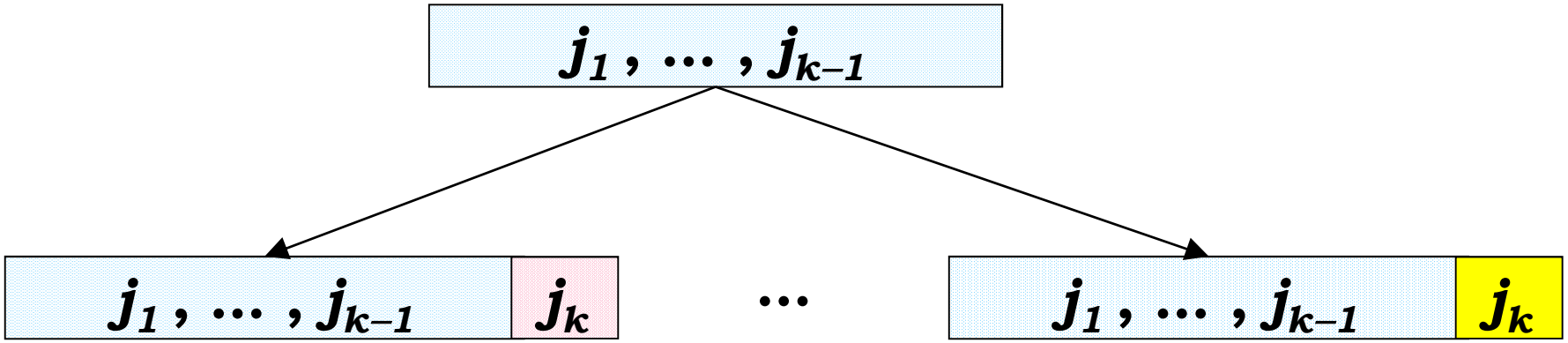
Può accadere che risolvendo il rilassamento si ottenga una soluzione ottima del sottoproblema. In altre parole, $LB_i = UB_i$

In questo caso, ovviamente, il sottoproblema non va ulteriormente suddiviso.

Se z = w, ogni soluzione ammissibile di RP che sia anche ammissibile per P, è anche ottima per P.

Calcolo di LB_i

- Rilassamento preemptivo: $1/r_j,prmp/L_{max}$
- la regola PEDD è ottima (ed efficiente)



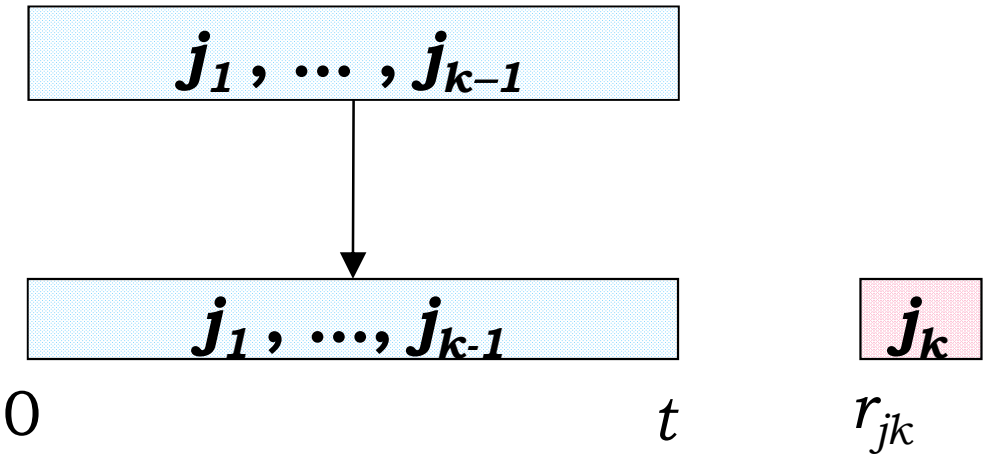
lower bound (livello k):

- eredita dal nodo padre
- ricalcola, essendo $LB_{k-1} \leq LB_k$

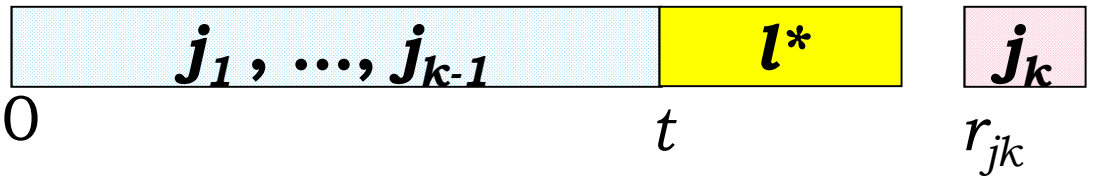
Dominanza fra sottoproblemi

Consideriamo uno schedule parziale al livello k-1: il sottoproblema $j_1, \ldots, j_{k-1}, \boldsymbol{j_k}$ deve essere generato solo se:

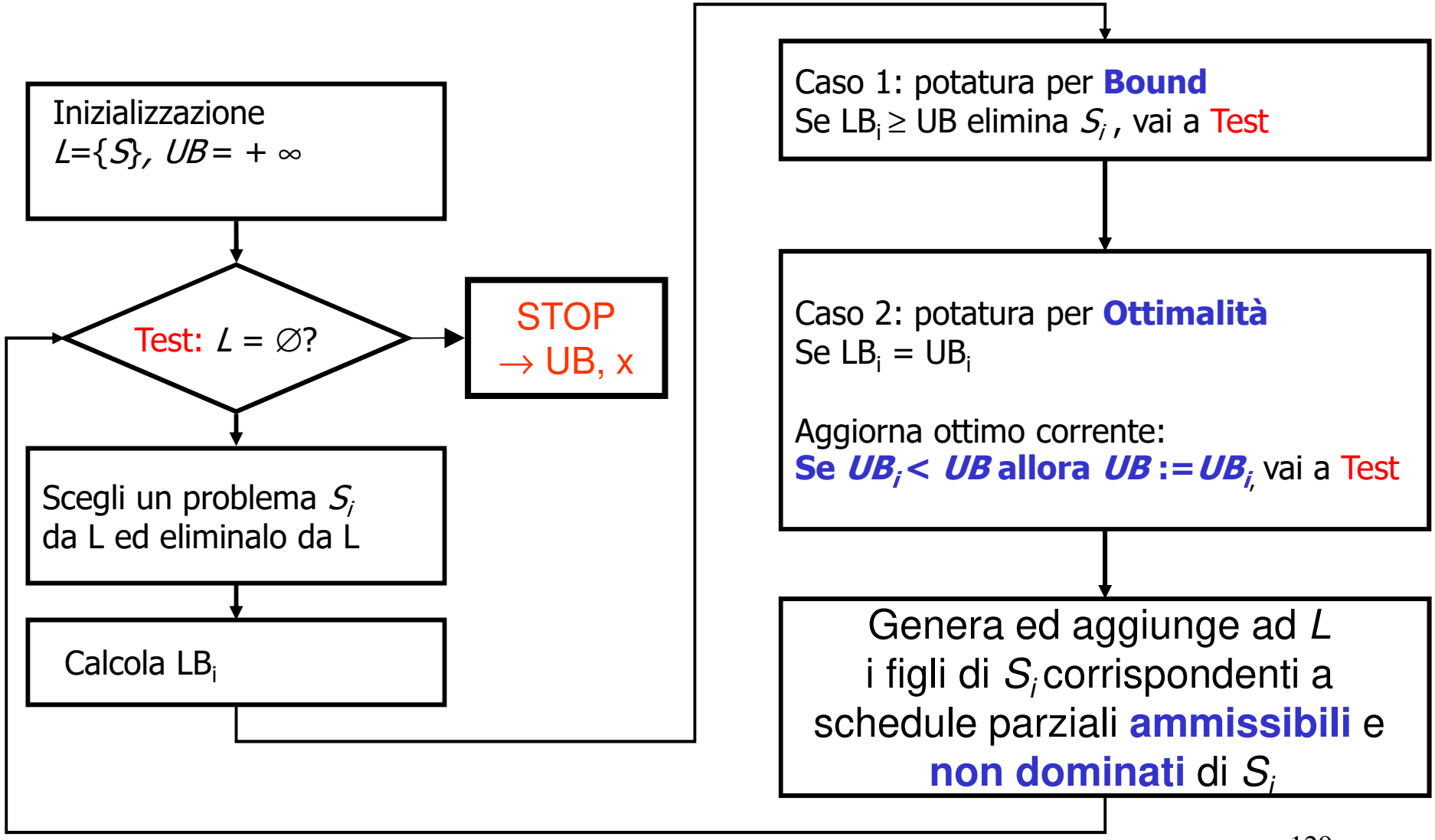
$$r_{j_k} < \min_{l \in J} (\max(t, r_l) + p_l) = \max(t, r_{l^*}) + p_{l^*}$$



altrimenti si otterrebbe un sottoproblema dominato da:

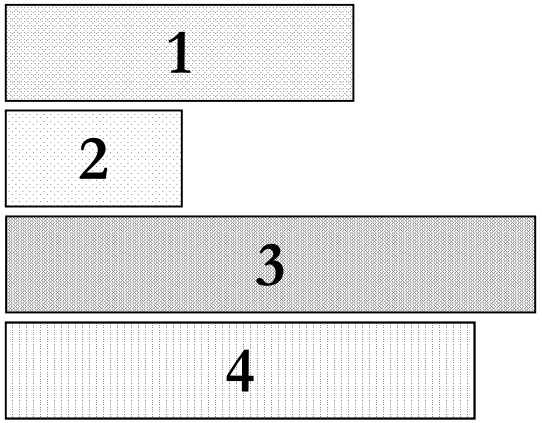


branch-and-bound (schema)

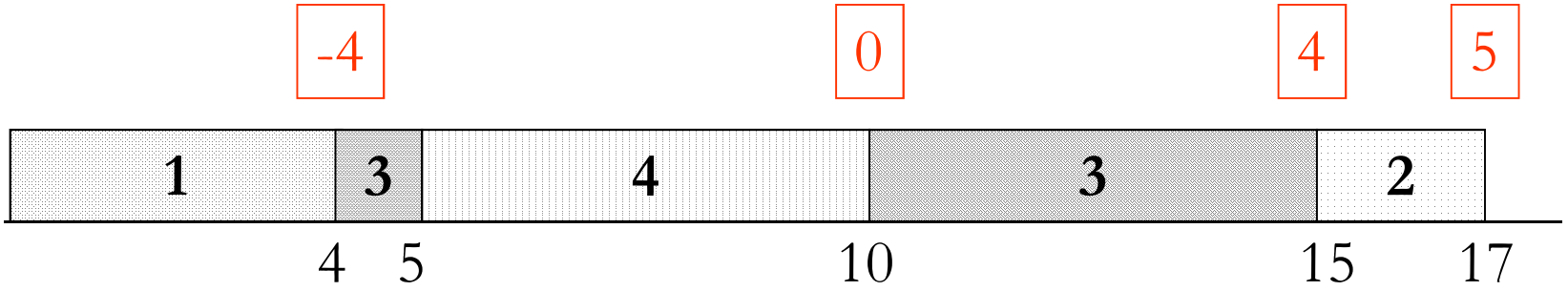


Esercizio ($1|r_j|L_{max}$)

job	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



nodo radice:



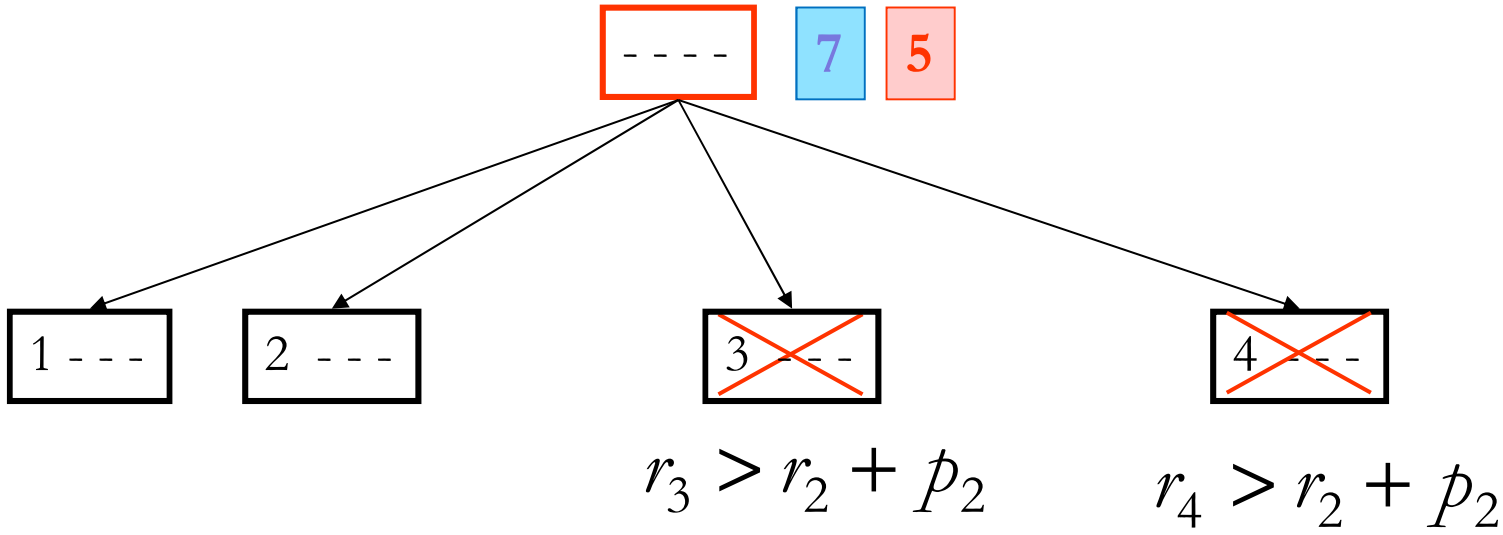
$z^{LB}(----) = 5$

Branching

job	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

ottimo corrente: (1,2,3,4)

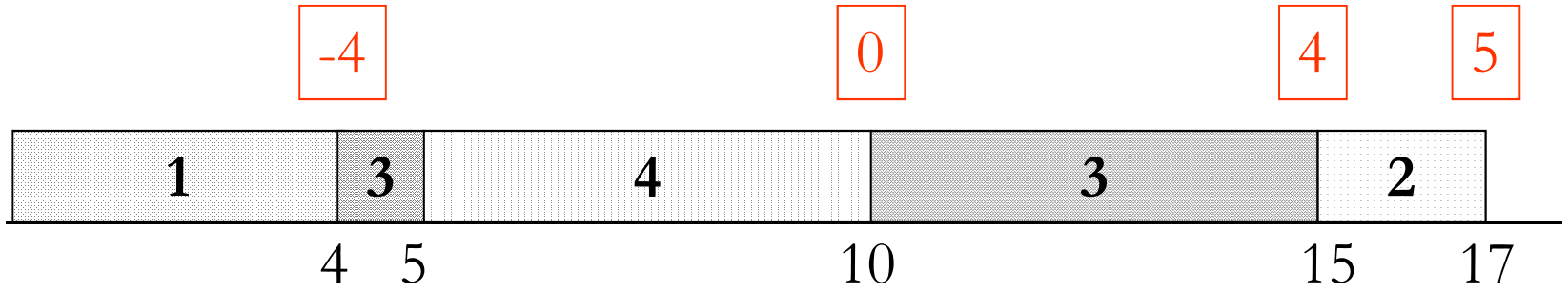
UB=7



Eliminati per dominanza

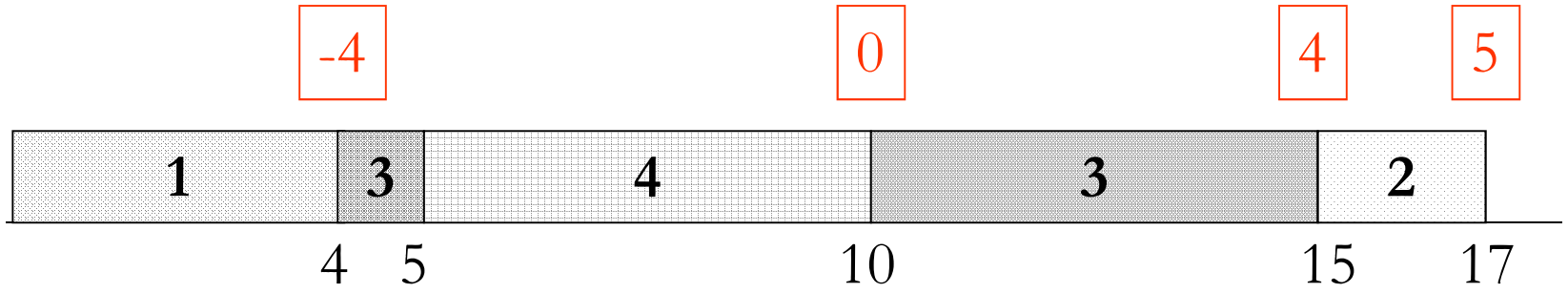
Valutazione di (1 - - -)

job	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



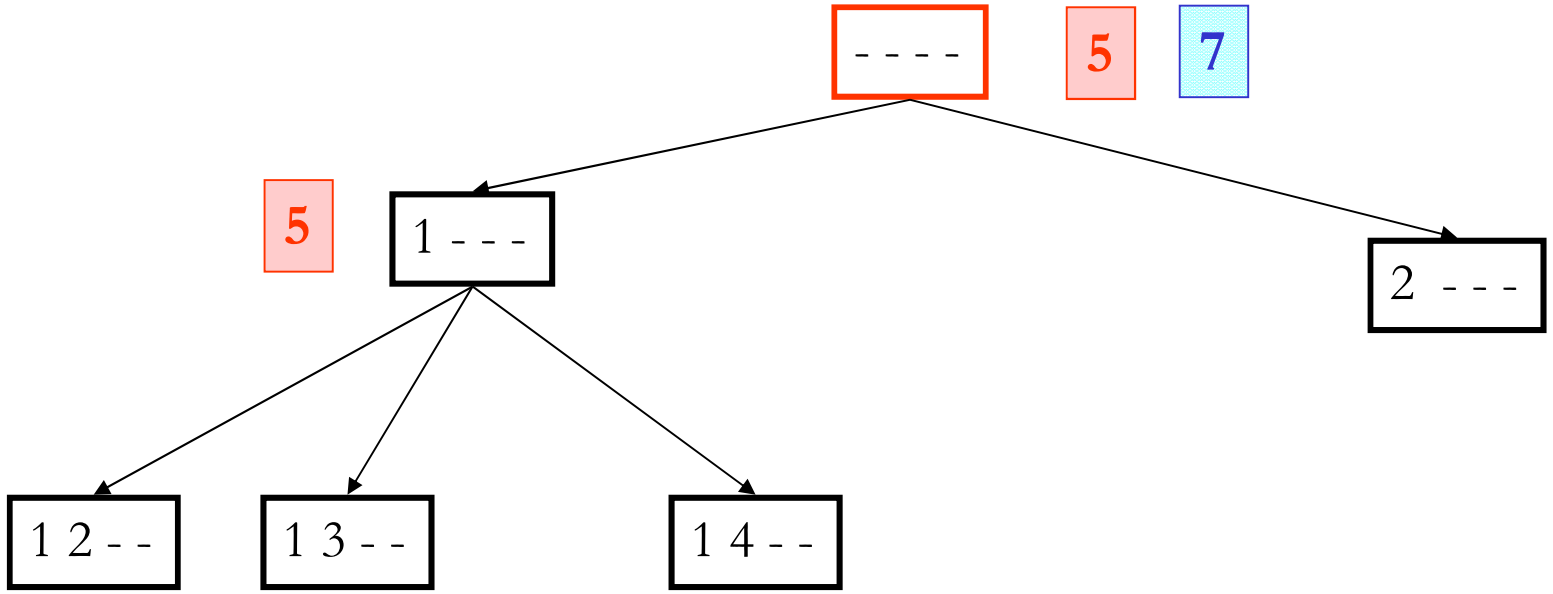
$z^{LB}(1,---) = 5$

Valutazione di (1 - - -)



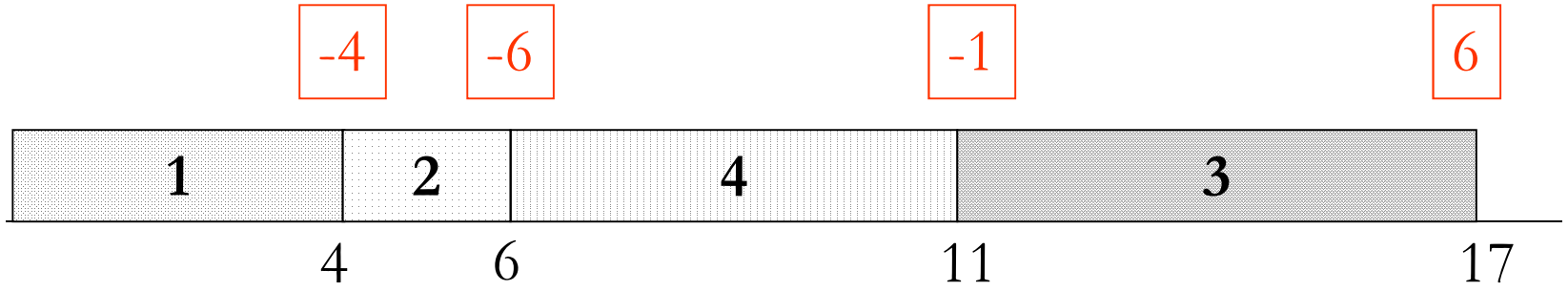
$z^{LB}(1,---) = 5$

la condizione di potatura per bound fallisce



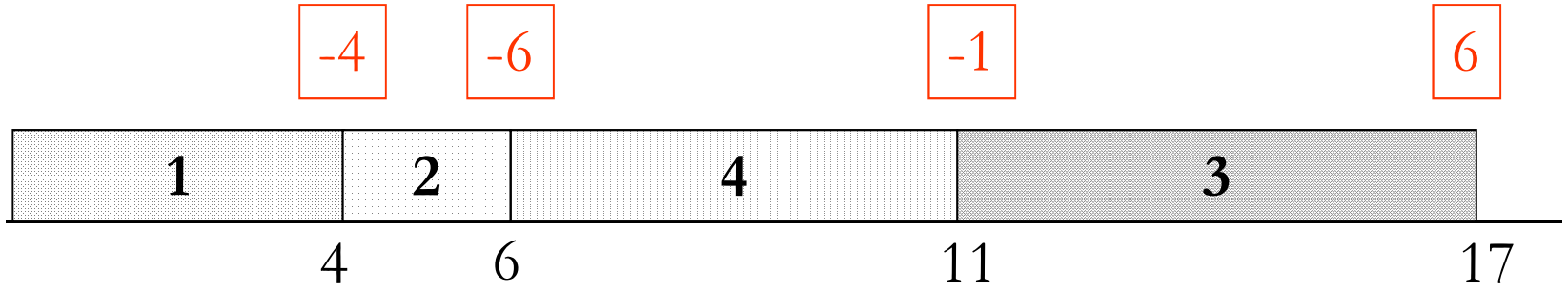
Valutazione di (1 2 - -)

job	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



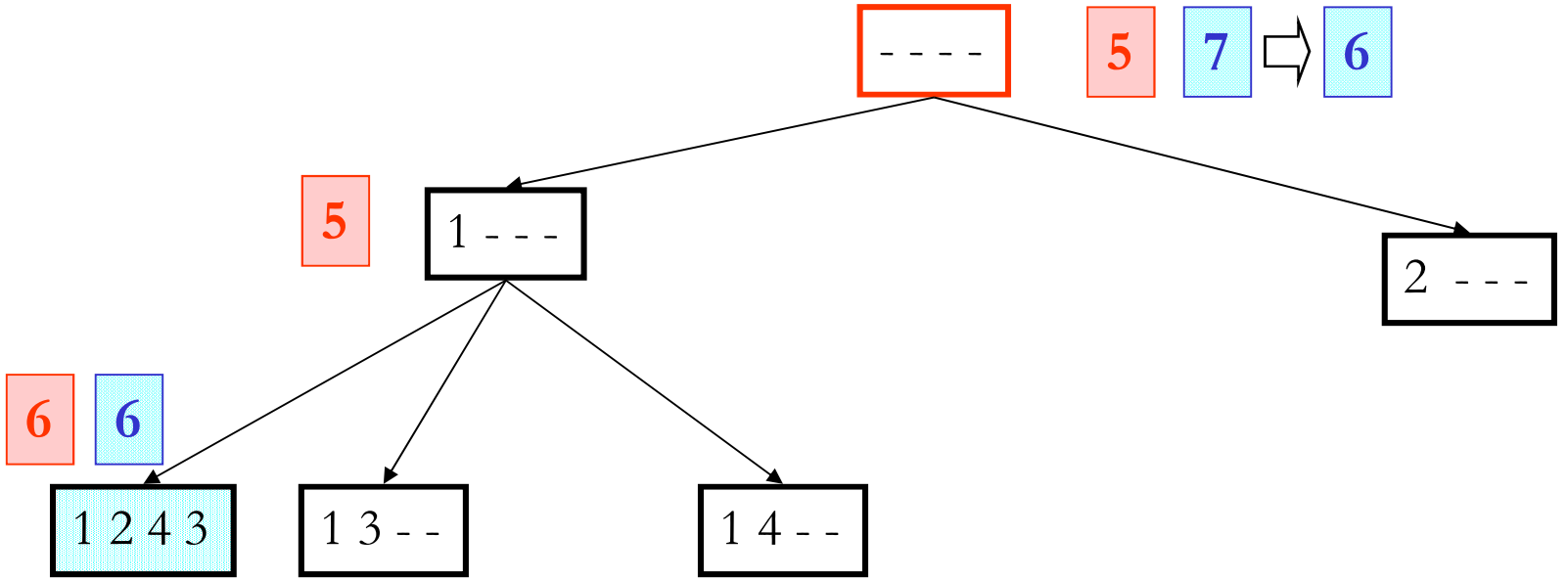
$z^{LB}(1,2\text{--}) = 6$ Soluzione ammissibile $\bar{z} = 6$

Valutazione di (1 2 - -)



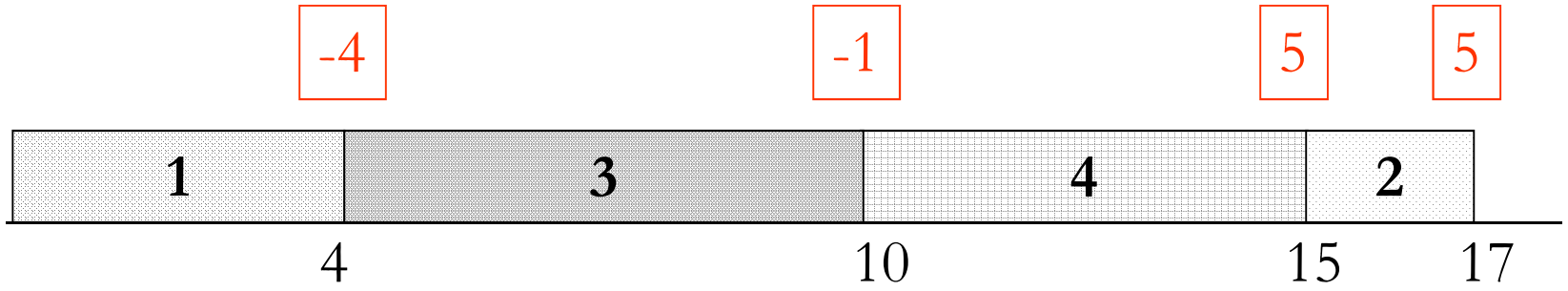
$z^{LB}(1,2--) = 6$

sottoproblema potato per ottimalità



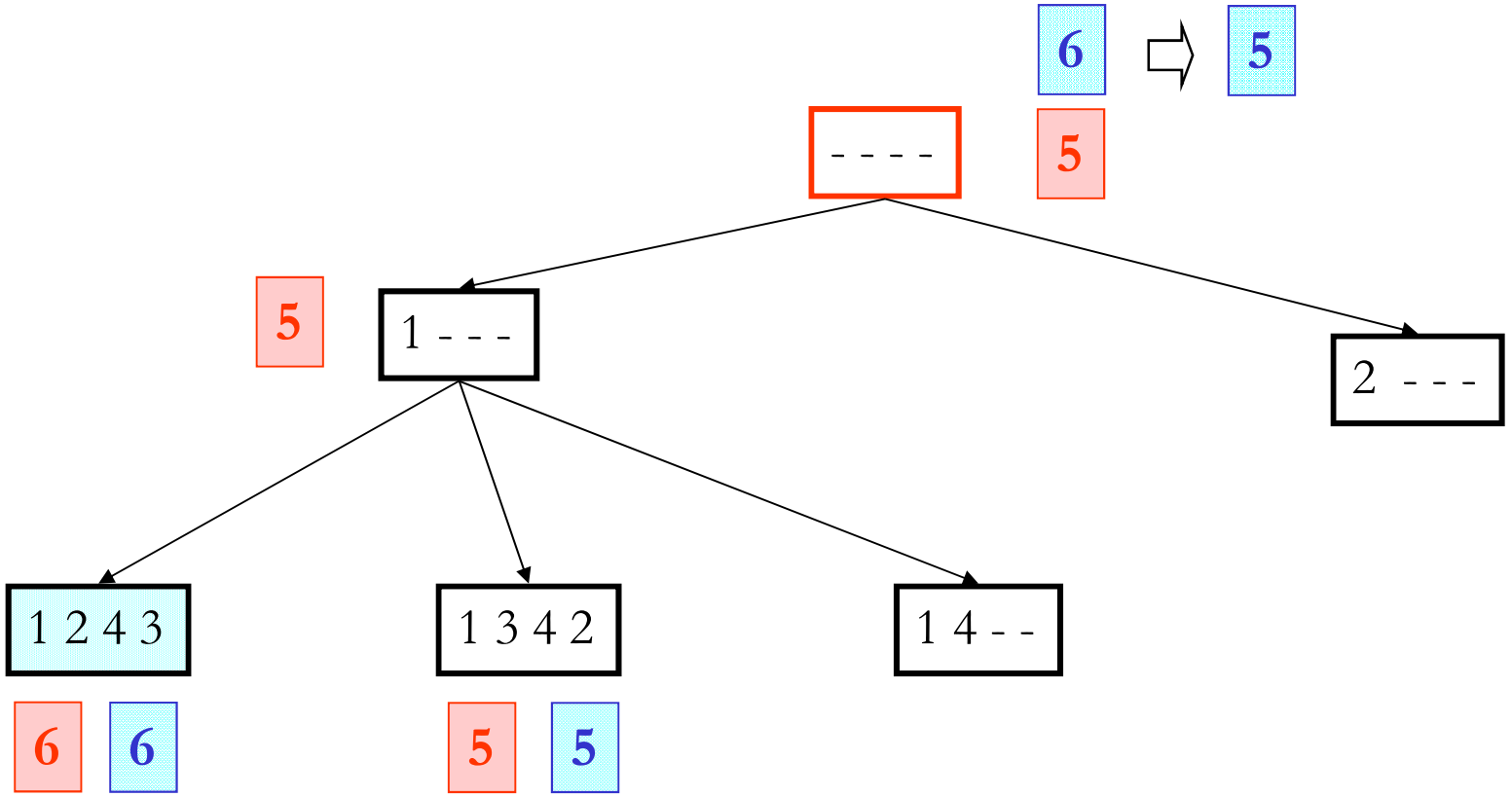
Valutazione di (1 3 - -)

job	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10



$z^{LB}(1,3--) = 5$ Soluzione ammissibile $\bar{z} = 5$

Arresto



Soluzione ammissibile di valore pari al lower bound globale: ottima

Scelte implementative

• Riassumendo, l'implementazione di un branch-and-bound consiste delle seguenti scelte:

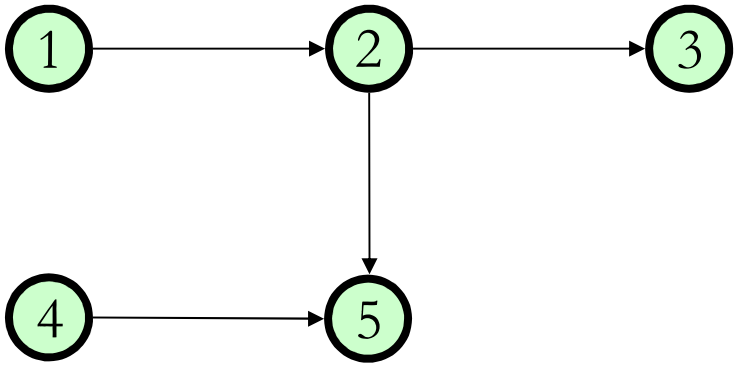
• regola di branching
• rilassamento
• regole di dominanza
• strategia di visita dell'albero di enumerazione (selezione del nuovo sottoproblema da valutare)
• euristiche (in particolare al nodo radice)

Esercizio

Calcolare la soluzione ottima del seguente problema $1/\text{prec}/\sum w_j C_j$
Regola di selezione del sottoproblema FIFO (visita in ampiezza)

job	1	2	3	4	5
w_j	1	4	1	6	3
p_j	3	7	1	5	4

Grafo delle precedenze

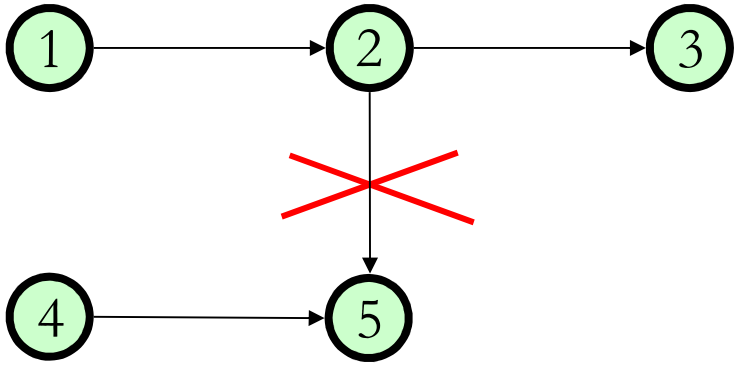


Scelta del rilassamento

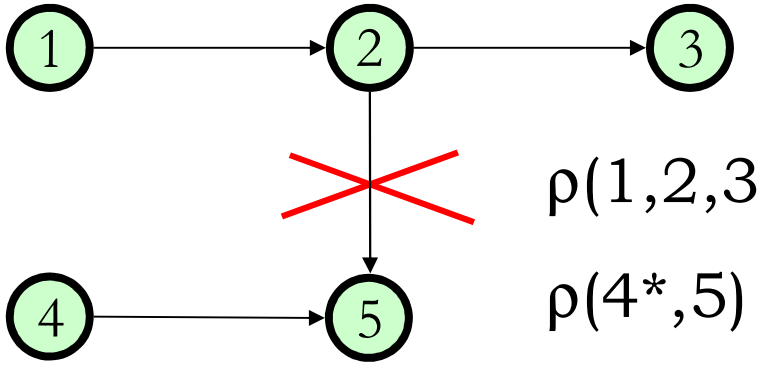
Eliminando archi dal grafo otteniamo rilassamenti del problema. Opzioni:

a. eliminare tutti gli archi costruendo un rilassamento in forma $1||\sum w_j C_j$ risolvibile con WSPT

b. escluderne alcuni in modo da ottenere un'istanza di $1|\text{chain}|\sum w_j C_j$ per cui conosciamo un algoritmo polinomiale.



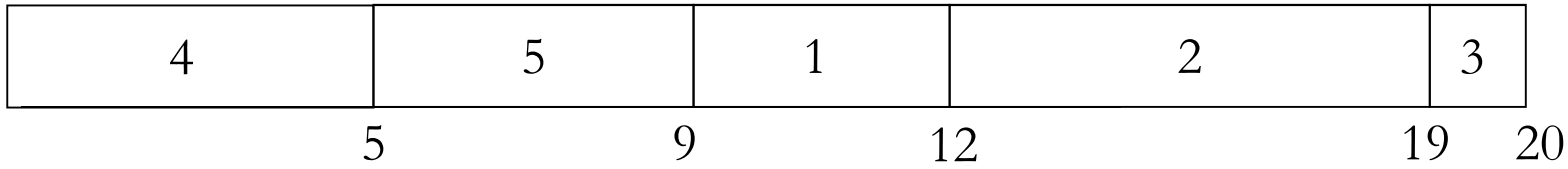
Valutazione di (-, -, -, -, -)



$\rho(1,2,3^*)=\max(1/3,5/10,6/11)=6/11$

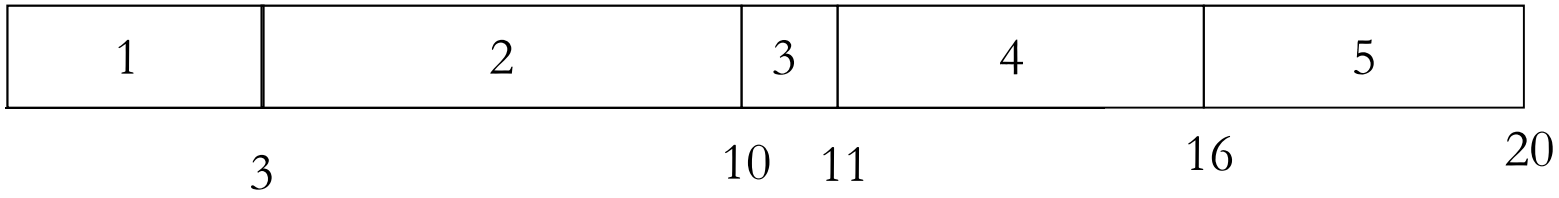
$\rho(4^*,5) = \max(6/5, 9/9)=6/5$

$\rho(5)=3/4$



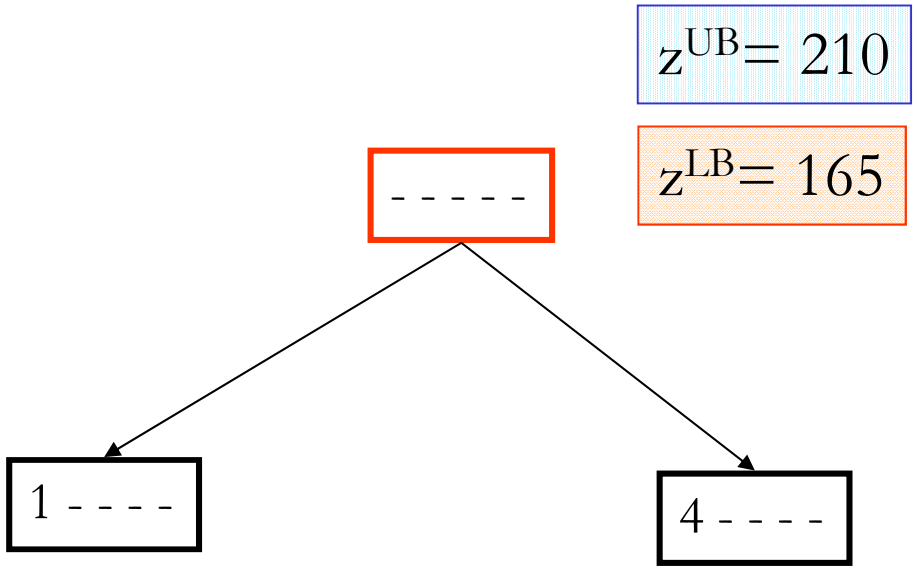
$z^{LB}= 165$

Soluzione del rilassamento chain: non ammissibile (5 precede 2)



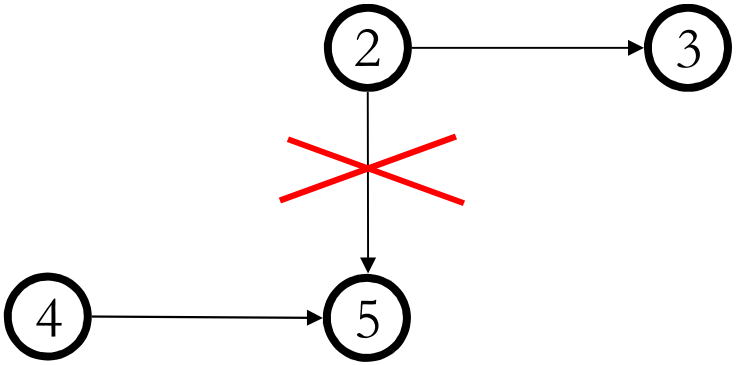
Soluzione iniziale – UB =210

Branching al nodo radice



Lista dei problemi candidati: {(1, - - - -), (4, - - - -)}

Valutazione di (1, -, -, -, -)



job	1	2	3	4	5
w_j	1	4	1	6	3
p_j	3	7	1	5	4

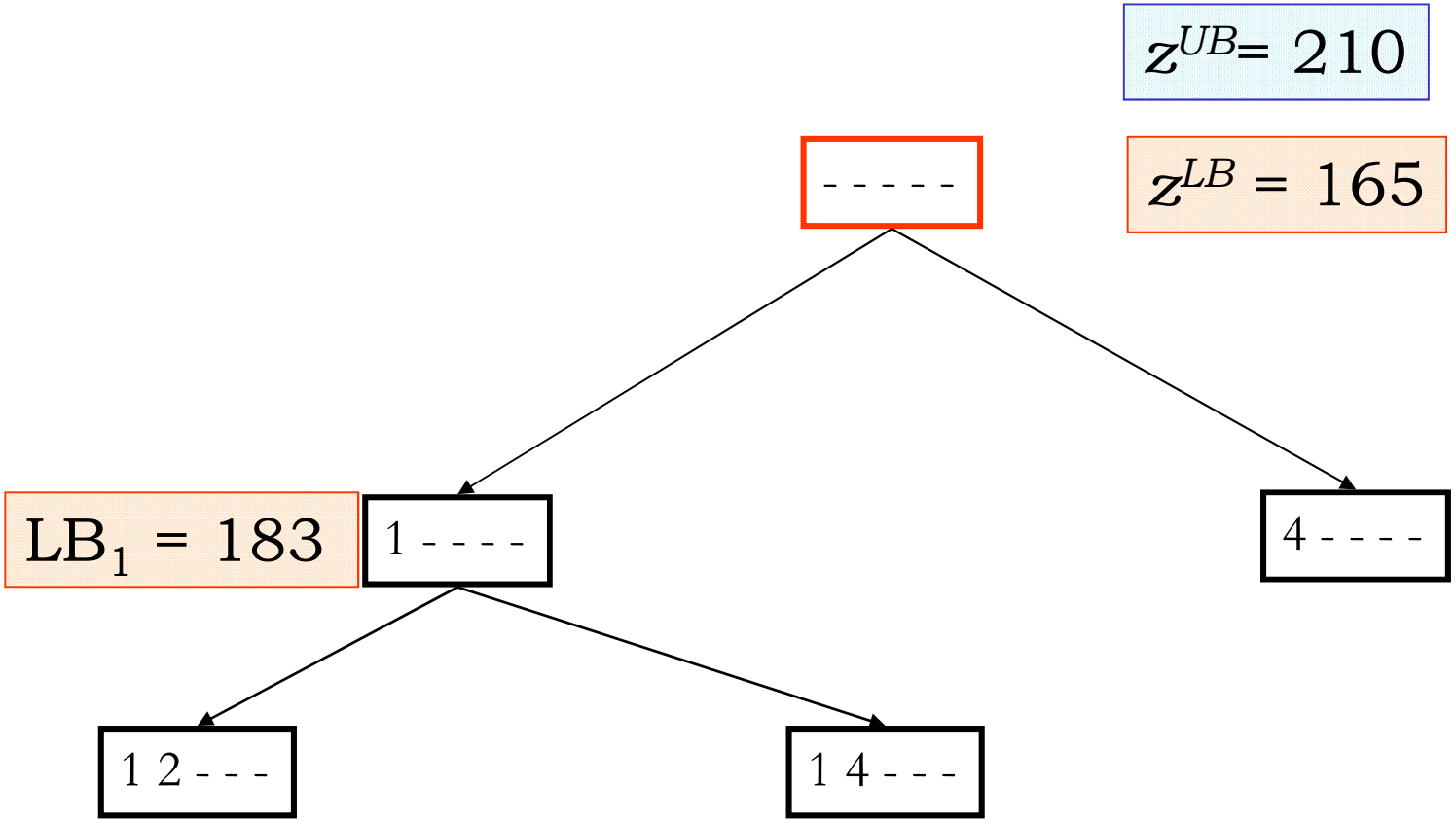
$\rho(2,3^*) = \max(4/7, 5/8)=5/8$

$\rho(4^*,5) = \max(6/5, 9/9)=6/5$

$\rho(5) = 3/4$

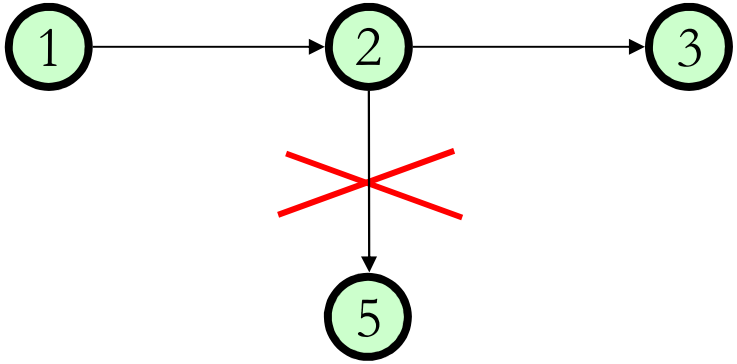
$LB_1 = 183$ 5 precede 2: soluzione non ammissibile

Branching



Lista dei problemi candidati: {(1, 2, - - -), (1, 4, - - -), (4, - - -)}

Valutazione di (4, -, -, -, -)



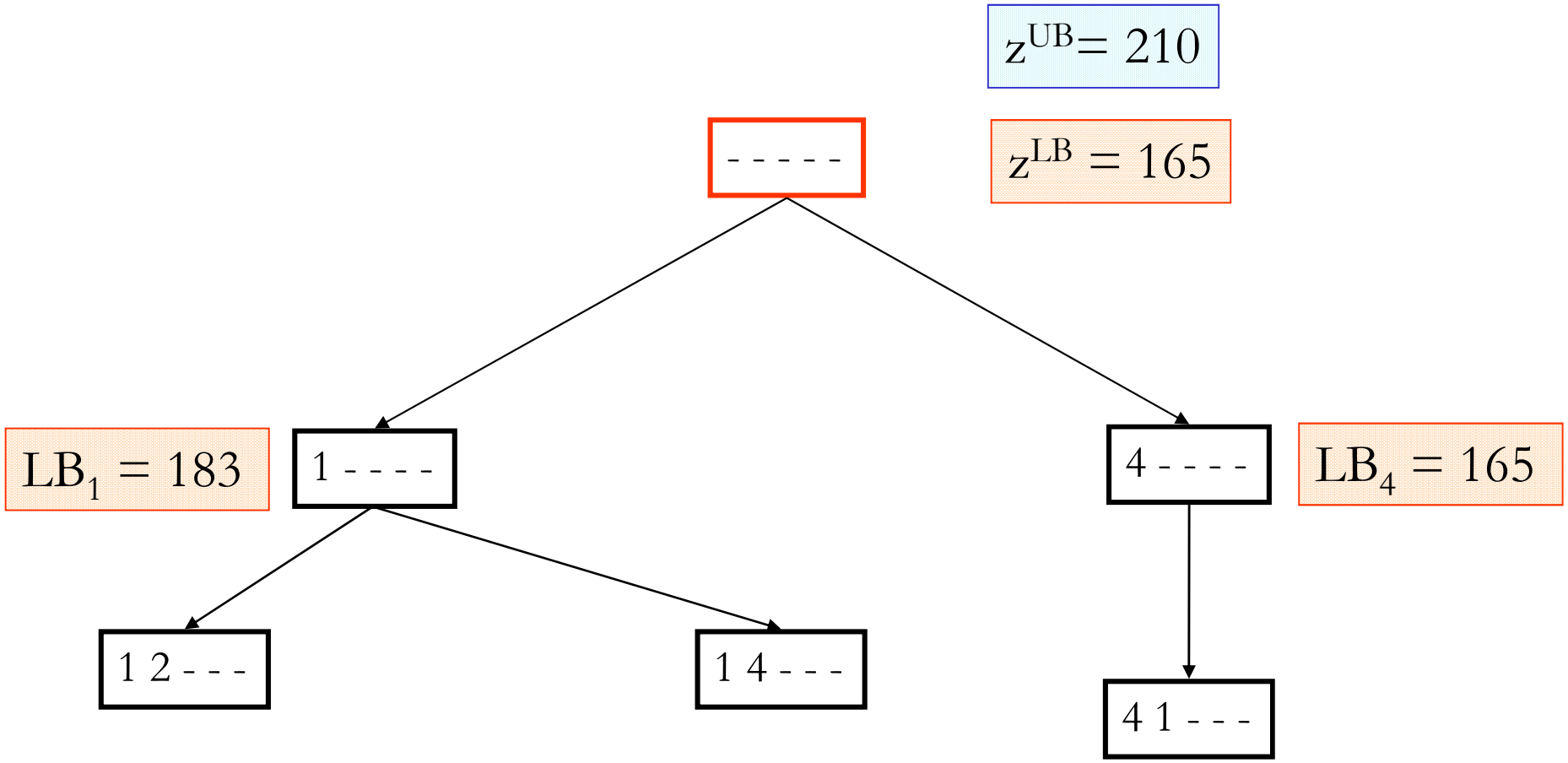
$\rho(1, 2,3^*) = \max(1/3, 5/10, 6/11)=6/11$

$\rho(5) = 3/4$

$LB_4 = 165$

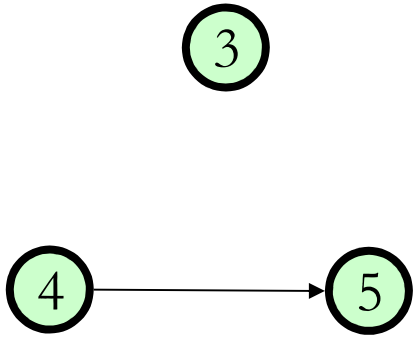
5 precede 2: soluzione non ammissibile

Branching



Lista dei problemi candidati: $\{(1, 2, - - -), (1, 4, - - -), (4, 1_{146-})\}$

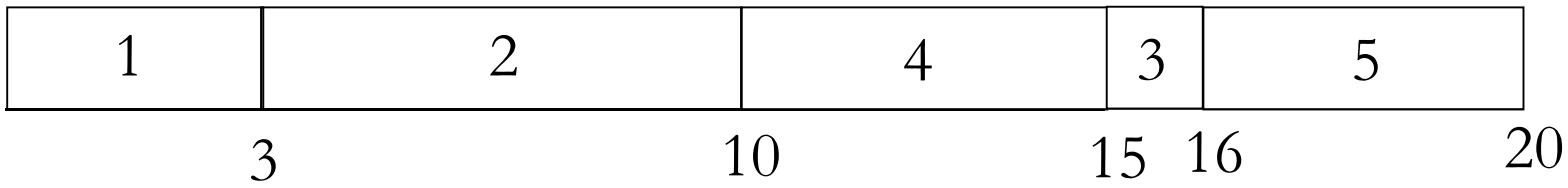
Valutazione di (1, 2, -, -, -)



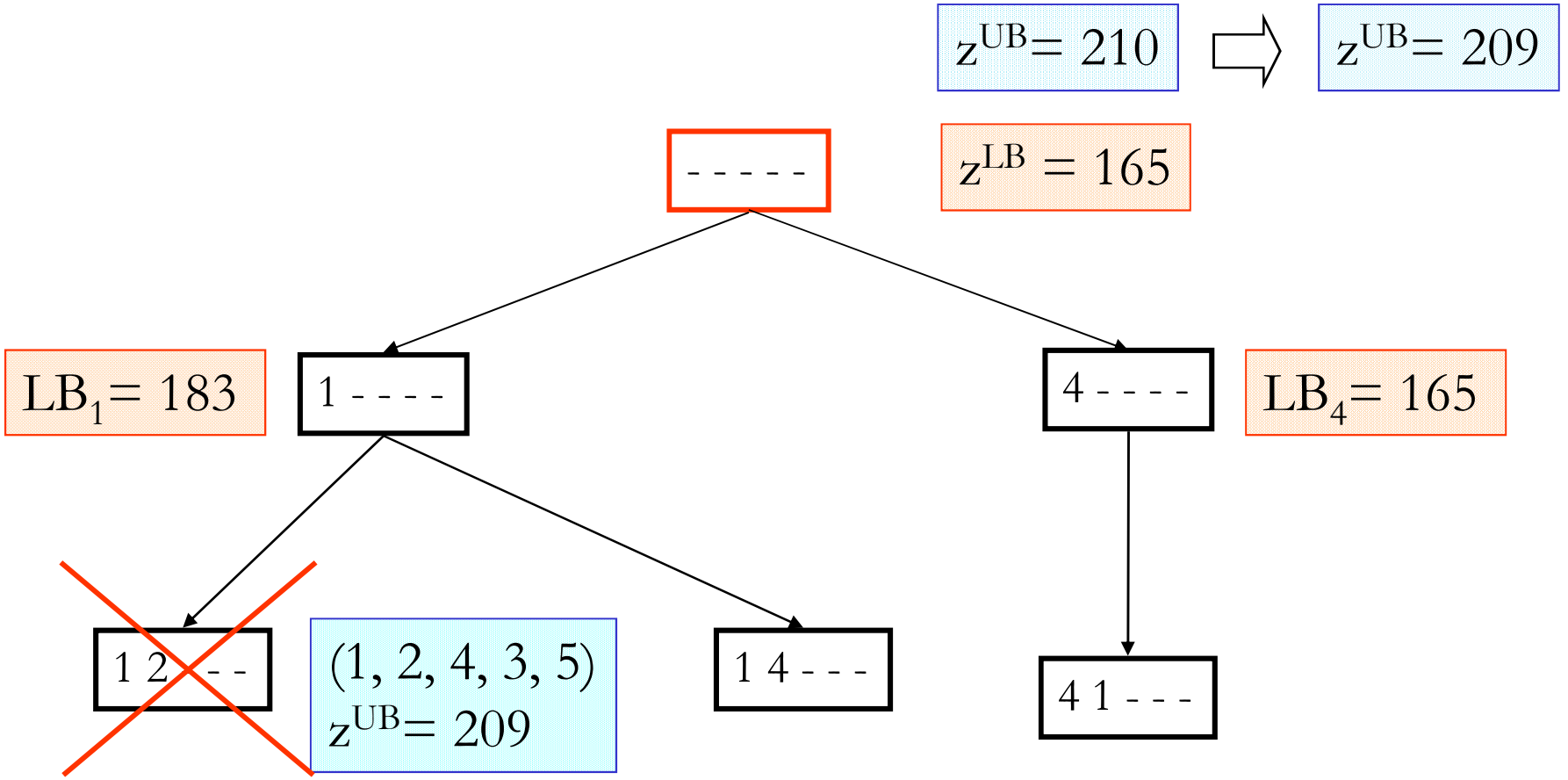
$\rho(3) = \max(1/1) = 1$

$\rho(4^*,5) = \max(6/5, 9/9) = 6/5$

Soluzione del rilassamento chain: ammissibile



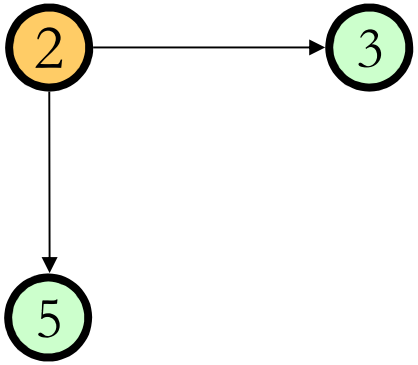
aggiornamento dell'ottimo corrente $z^{LB} = 209 = z^{UB}$



potato per ottimalità

Lista dei problemi candidati: {(1, 4, -, - -), (4, 1, -, - -)}

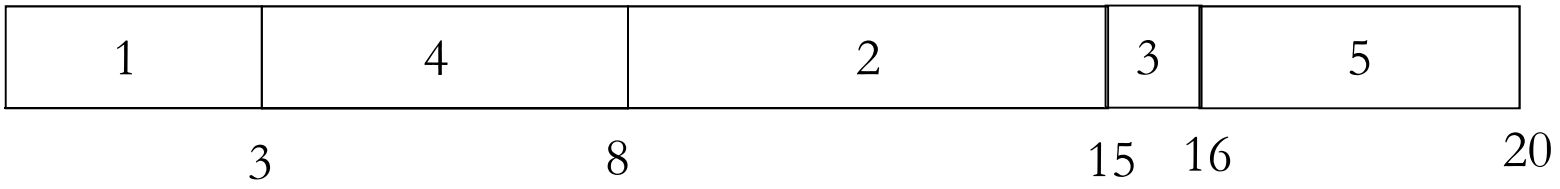
Valutazione di (1, 4, -, -, -)



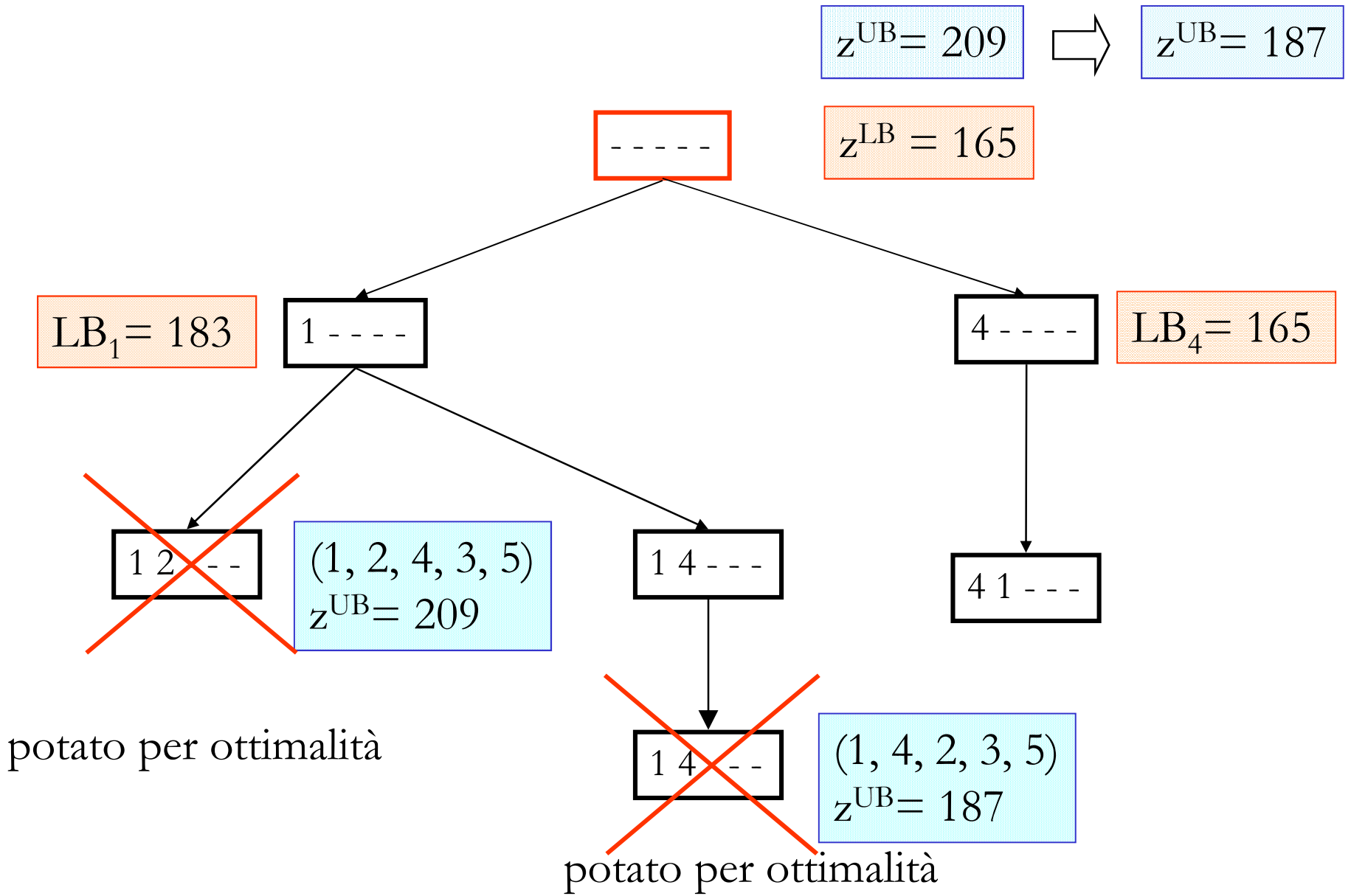
Solo il job 2 è schedulabile:
l'unico figlio è (1 4 2 - -)

Valutazione di (1, 4, 2, -, -)

$\rho(3) = \max(1/1) = 1$ $\rho(5) = 3/4$

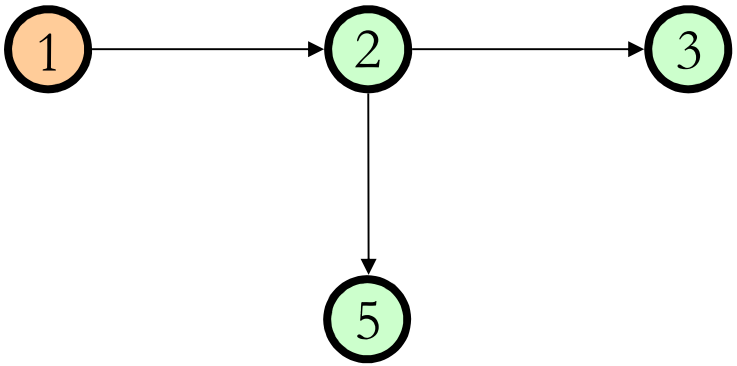


Soluzione del rilassamento chain: ammissibile L= 187



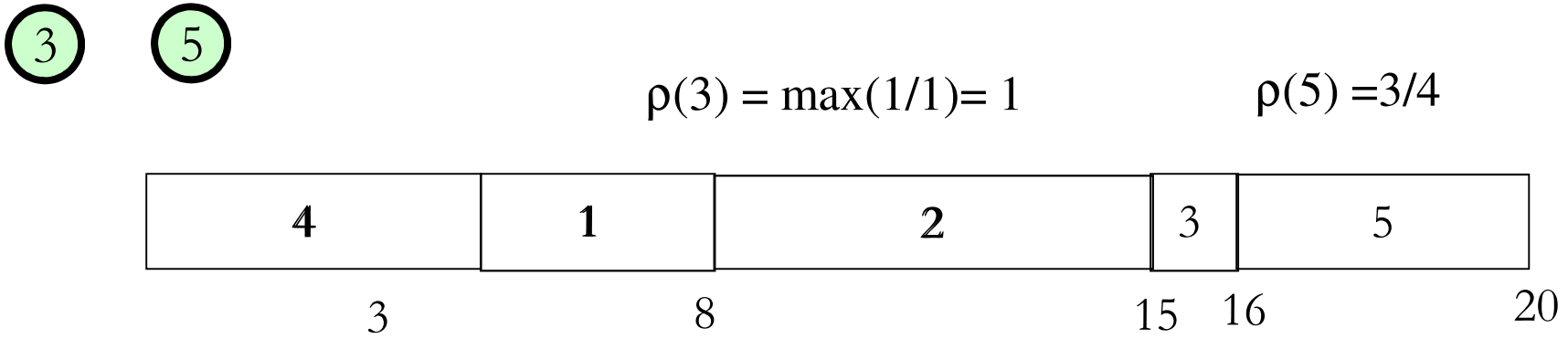
Lista dei problemi candidati: {(4, 1, - - -)}

Valutazione di (4, 1, -, -, -): fixing

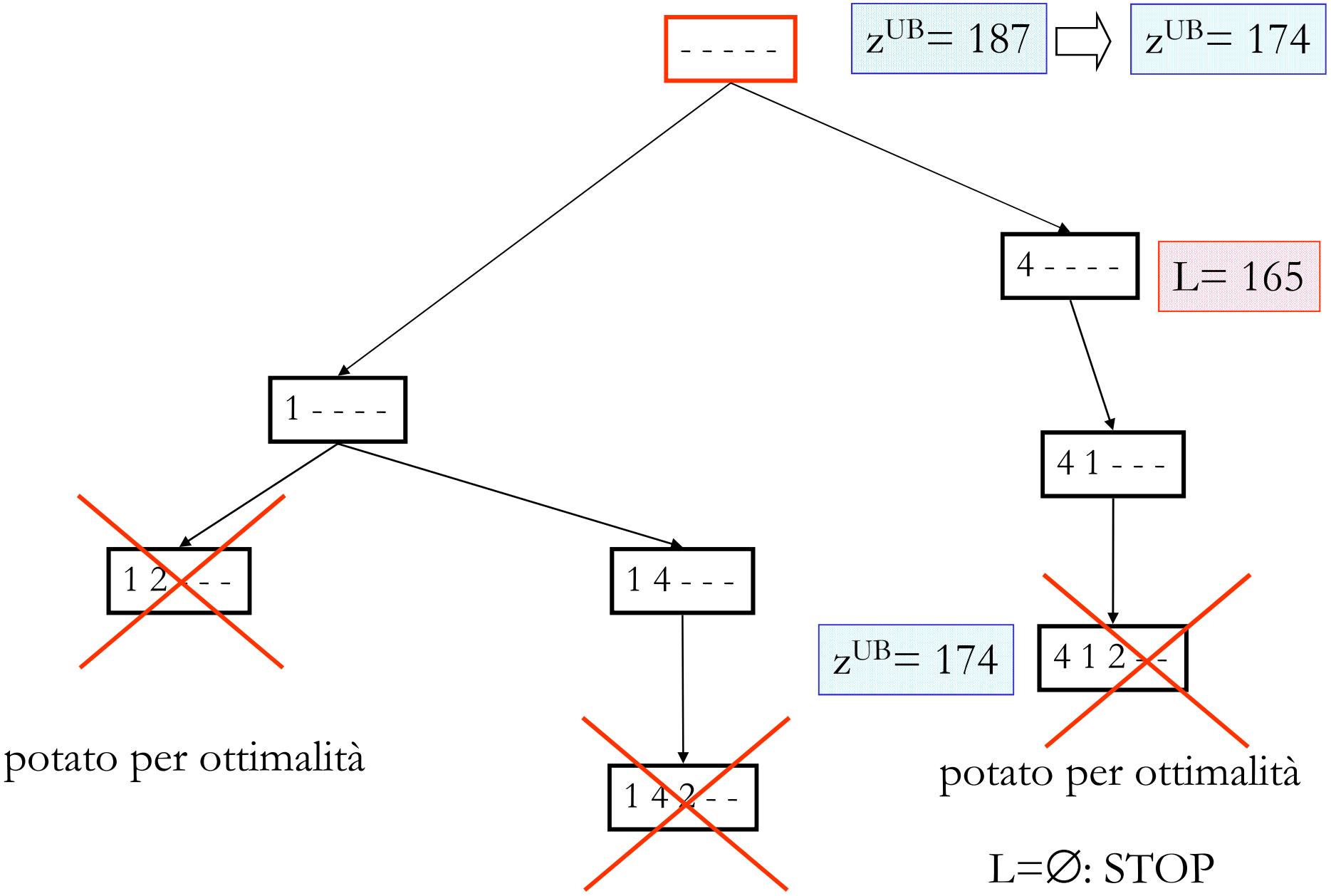


Solo il job 1 è schedulabile:
l'unico figlio è (4 1 - - -)

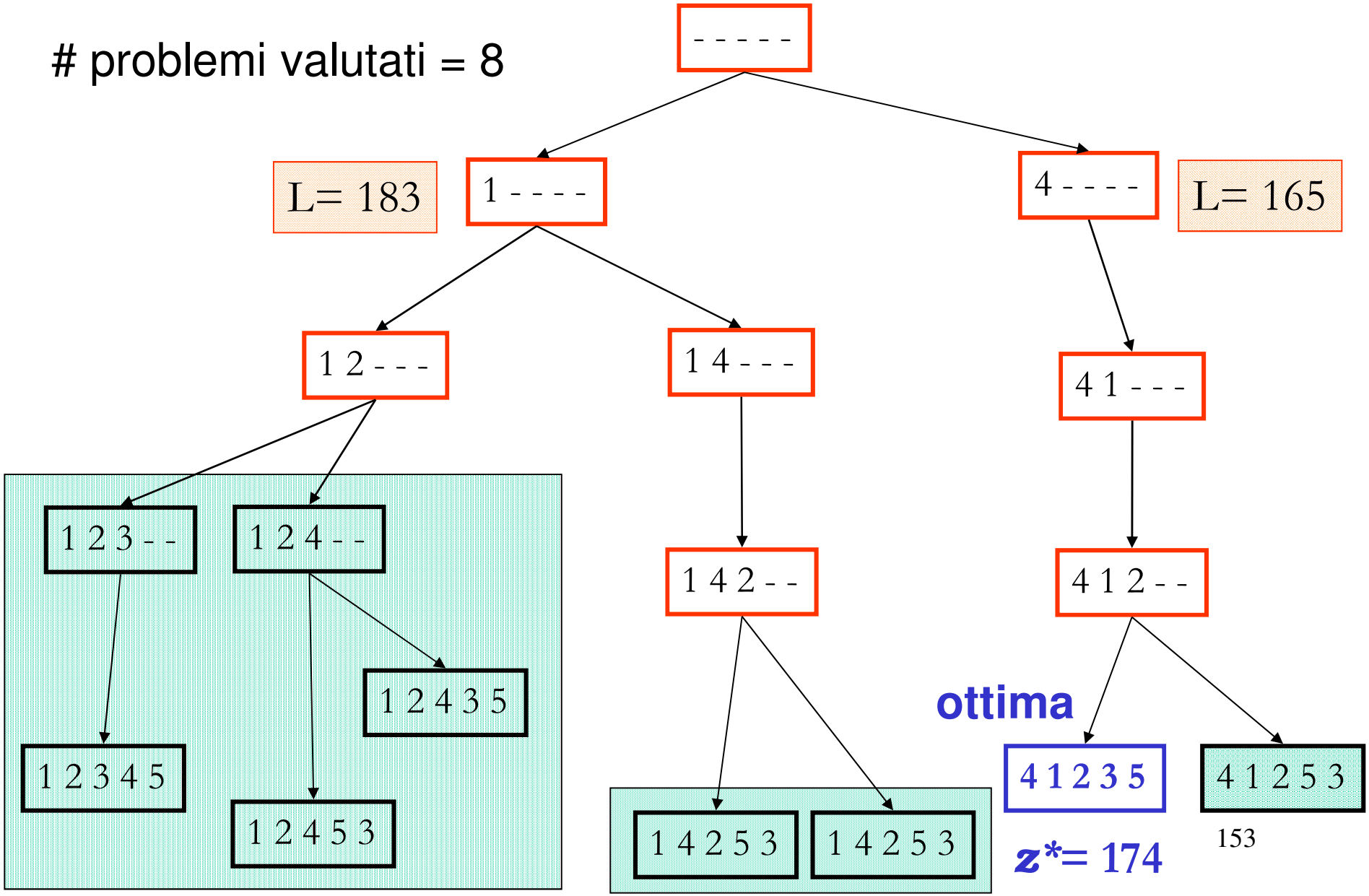
Valutazione di (4, 1, 2, -, -)



Soluzione del rilassamento chain: ammissibile L= 174

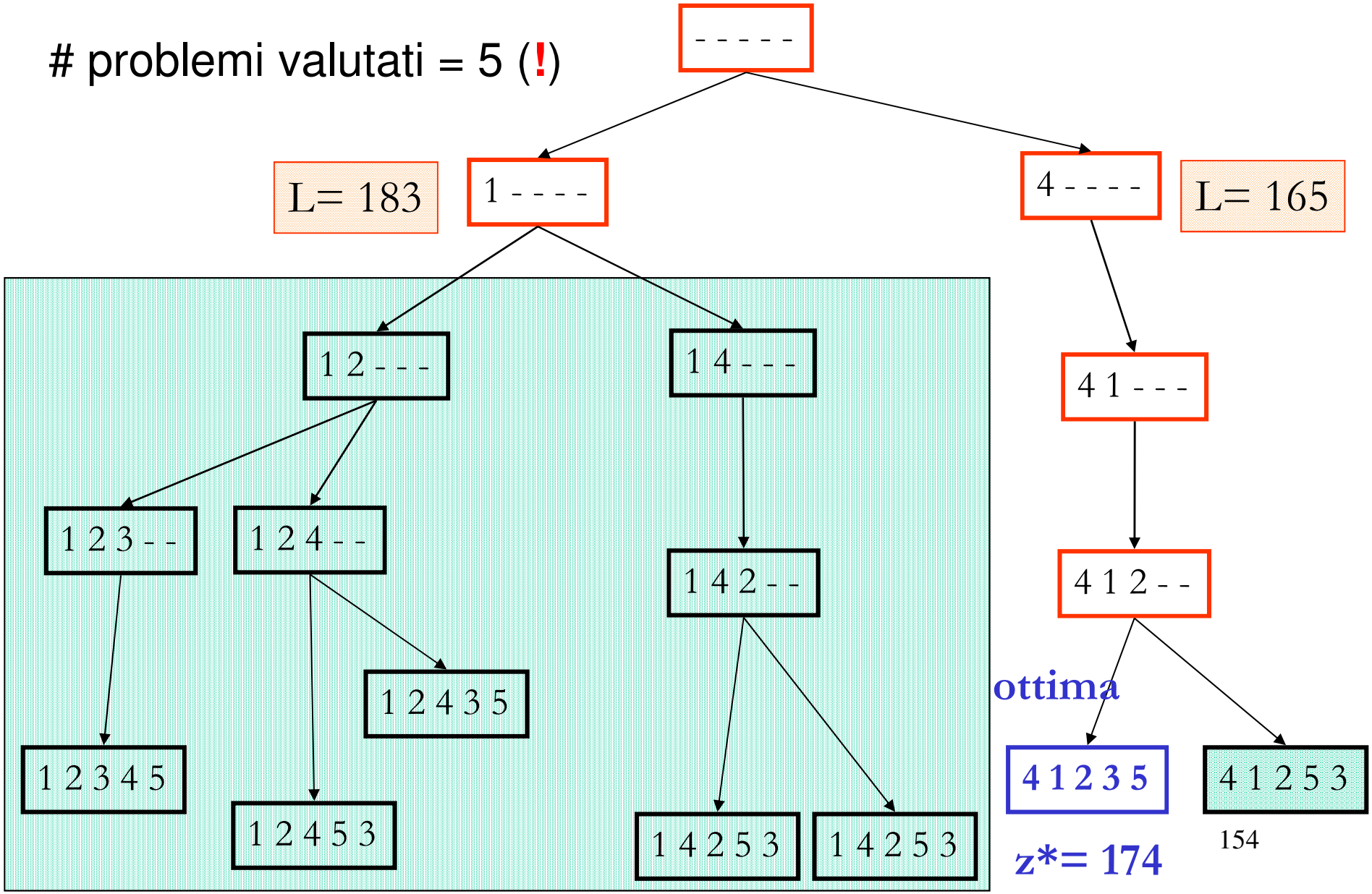

- - - - -
z^{UB}= 187
z^{UB}= 174
4 - - - -
L= 165
1 - - - -
4 1 - - -
1 2 - - -
1 4 - - -
z^{UB}= 174
4 1 2 - -
potato per ottimalità
1 4 2 - -
potato per ottimalità
potato per ottimalità
L=∅: STOP

scelta del sottoproblema: FIFO (visita in ampiezza)

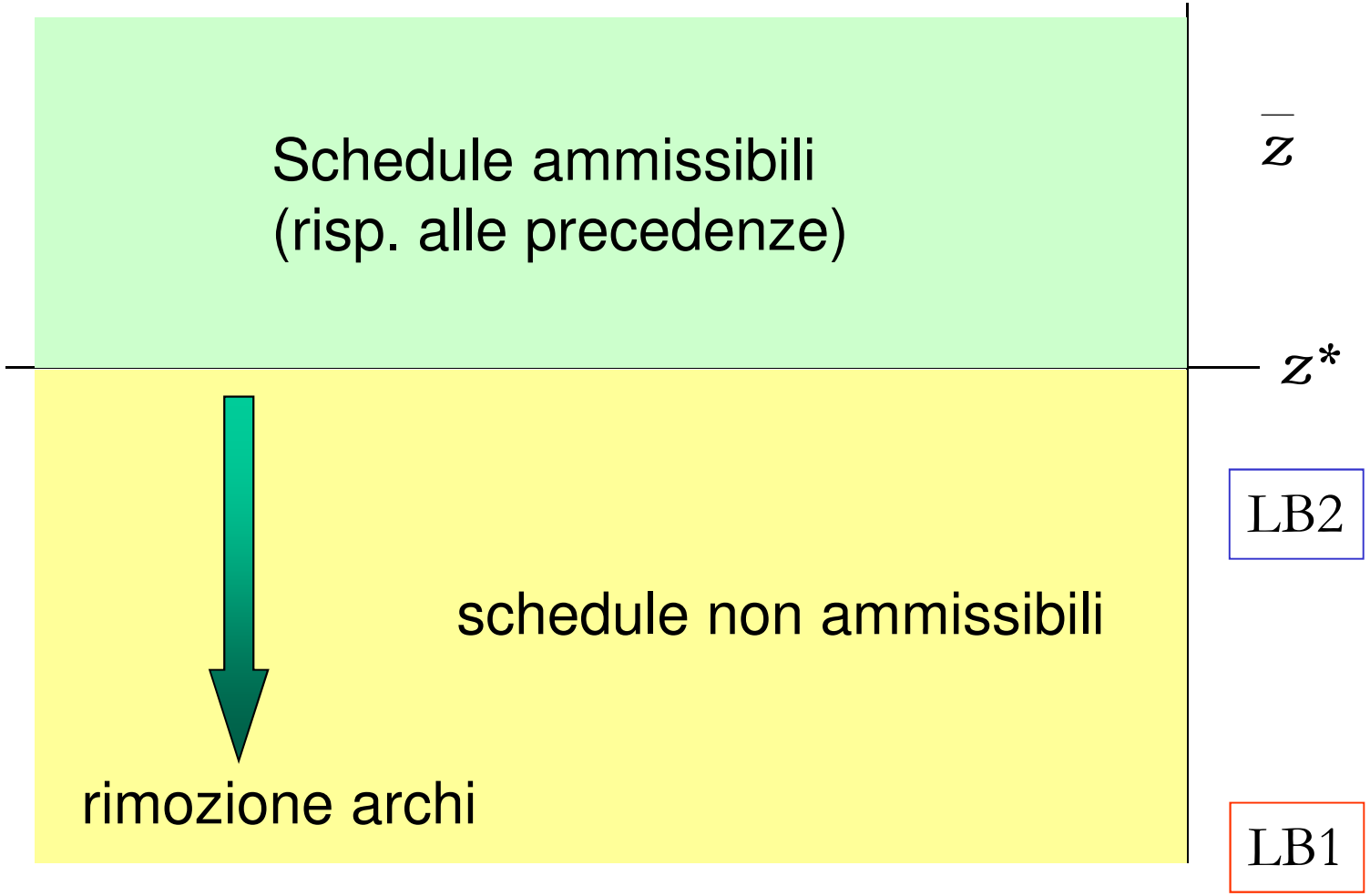


scelta del sottoproblema: least lower bound

problemi valutati = 5 (!)



Gerarchia di rilassamenti



LB1 = eliminati i vincoli di precedenza O(n log n)

LB2 = eliminato solo l'arco (2,5) $O(n^2)$

Relazioni fra funzioni obiettivo

Definizione. Due funzioni obiettivo si dicono **equivalenti** se uno schedule ottimo per la prima lo è anche per la seconda e viceversa.

Esempio. $1//\Sigma C_j$. e $1//\Sigma L_j$. sono equivalenti

Infatti, $L_j = C_j - d_j$

quindi $\Sigma L_j = \Sigma C_j - \Sigma d_j$

ma l'ultimo termine è costante e non dipende dallo schedule

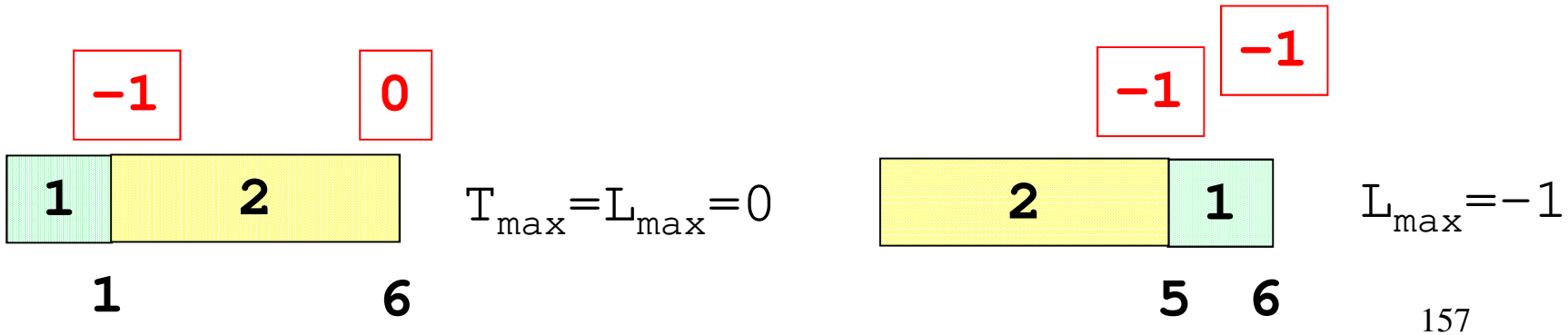
L_{max} vs. T_{max}

Uno schedule che minimizza L_{max} minimizza anche T_{max} infatti,

$T_{max} = max \{max(0, L_1), \ldots, max(0, L_n)\} = max\{0, L_1, \ldots, L_n\} = max \{0, L_{max}\}$.

non vale il veceversa:

job	1	2
p_j	1	5
d_j	7	6



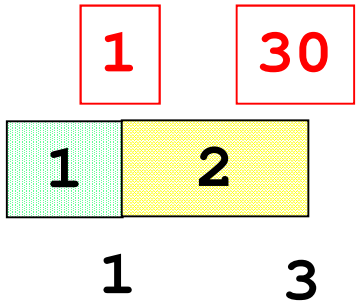
Osservazione

Sia P un generico problema e RP un suo rilassamento.

- se P ed RP hanno la stessa funzione obiettivo, allora qualunque soluzione ottima di RP che sia anche ammissibile per P è ottima per P;

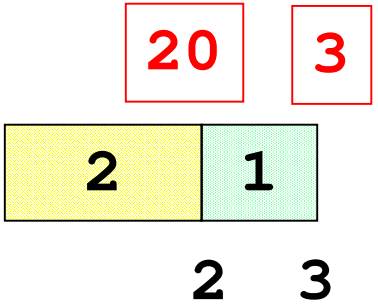
- al contrario, se P ed RP hanno funzioni obiettivo diverse questo non è vero: consideriamo la seguente istanza di $1//\Sigma w_j C_j$ ed il suo rilassamento $1//\Sigma C_j$.

job	1	2
p_j	1	2
w_j	1	10



$C_1+C_2=4$*

$w_1C_1+w_2C_2=31$



$C_1+C_2=5$

$w_1C_1+w_2C_2=23$*

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

job in tempo (EDD)	job in ritardo (qualsiasi ordine)

Algoritmo di *Moore*

- Costruisce lo schedule in avanti
- 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. Sia j^* il job in J^c con la minima due date:

   $J := J \cup \{j^*\}$; $J^c := J^c \setminus \{j^*\}$

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

   sia k^* il job in J più lungo:

   $J := J \setminus \{k^*\}$ $J^d := J^d \cup \{k^*\}$

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 $\{1,\ldots,k\}$ che soddisfa le seguenti proprietà:

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

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

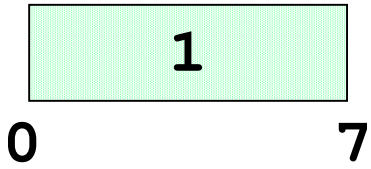
Caso b. Il job $k+1$ è aggiunto all'insieme J_k ed è completato in ritardo. Dato che J_k soddisfa (1) e (2) deve essere $N_k = N_{k+1}$.

Quindi, aggiungere il job $k+1$ a J_k non aumenta il numero di job in tempo. Ma aggiungere $k+1$ 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.

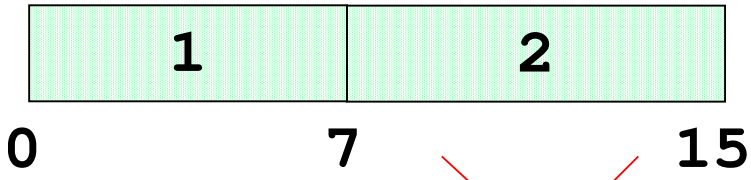
Esempio

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

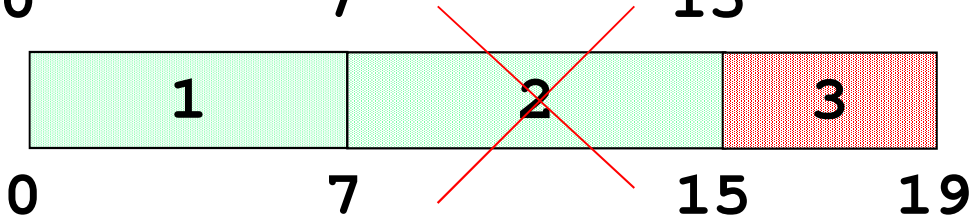
iter 1.

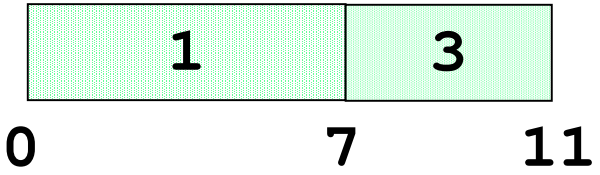


iter 2.



iter 3.

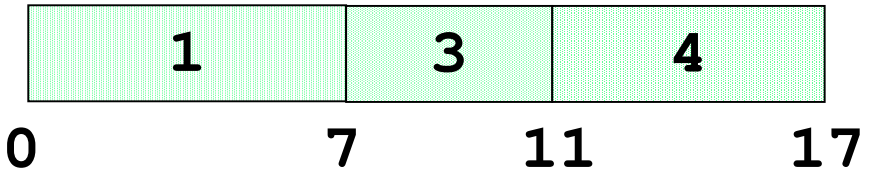




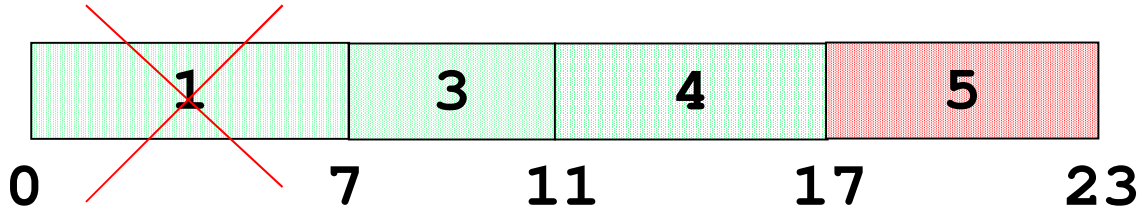
Esempio

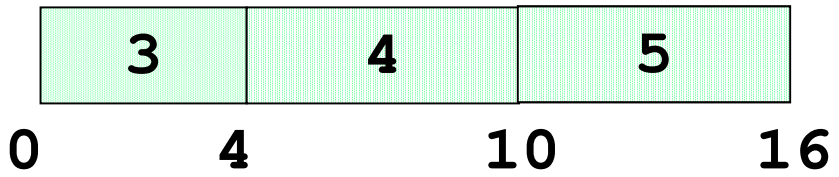
job	1	2	3	4	5
p_j	7	8	4	6	6
d_j	9	17	18	19	21

iter 4.

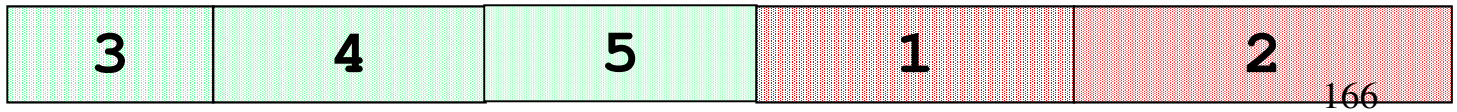


iter 5.





Soluzione



1.4 Problemi min-sum con due date (Tardiness totale ΣT_j)

Problema dello zaino binario

Zaino di capacità $C \in Z_+$

Insieme $N = \{1,\ldots,n\}$ di oggetti. Per ogni oggetto:

ingombro $a_j \in Z_+$ profitto $p_j \in Z_+$

Problema: scegliere un insieme di oggetti di profitto massimo tale che l'ingombro totale non ecceda la capacità.

Sia S una soluzione ottima e S′ una soluzione ottenuta da S eliminando l'elemento i.

S′ è sol. ottima per il problema $N' = N \setminus \{i\}$ e $C' = C - a_i$

Altrimenti, detta S′′ una sol ottima, $S^* = S'' \cup \{i\}$ sarebbe ammissibile e migliore di S, contraddizione.

Formula ricorsiva

P(i, λ) = massimo profitto ottenibile con i primi i oggetti ed uno zaino di capacità λ

consideriamo l'elemento i e valutiamo le due possibilità:

i è tenuto nello zaino: $P(i, \lambda) = P(i-1, \lambda - a_i) + p_i$

i è scartato: $P(i, \lambda) = P(i-1, \lambda)$,

Quindi:

$$P(i, \lambda) = \max(P(i-1, \lambda - a_i) + p_i , P(i-1, \lambda))$$

il problema consiste nel determinare $P(n, C)$

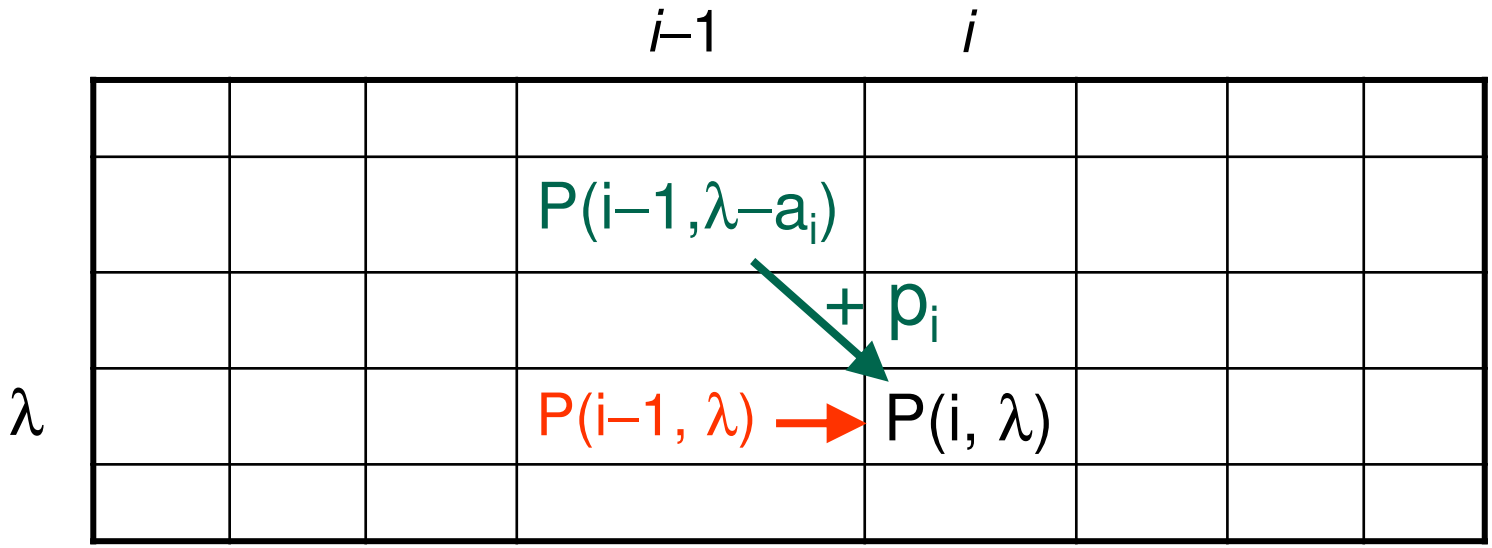
Divide et impera vs.PD

il metodo ***divide et impera*** procede per decomposizioni successive (top-down) e combina successivamente le soluzioni dei sottoproblemi per ottenere la soluzione del problema originario.

È possibile (anzi, frequente) che molti sottoproblemi siano uguali, quindi si esegue molto lavoro inutile!

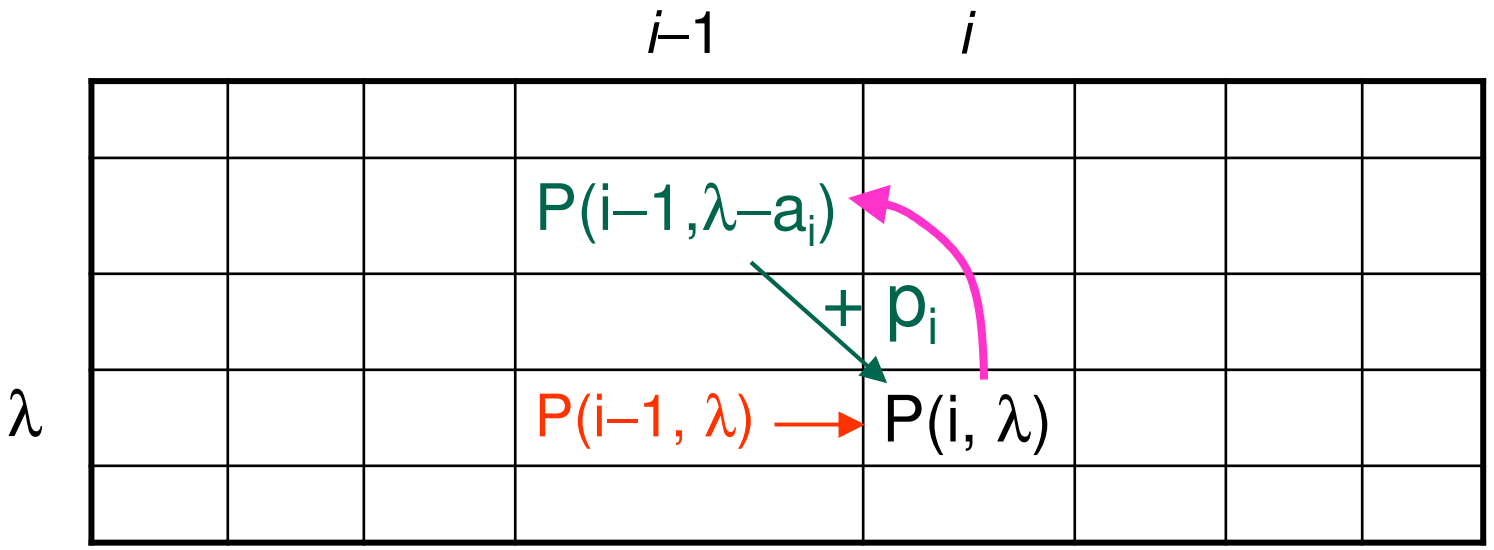
La ***programmazione dinamica*** risolve i sottoproblemi in modo bottom-up, memorizzando le soluzioni in una tabella e utilizzandole ogni volta che servono alla soluzione di un problema più grande

Complessità spaziale



- ogni valore P(i, λ) è calcolato solo con elementi della colonna precedente
- Quindi, per calcolare il valore ottimo, è sufficiente memorizzare la colonna precedente ⇒ spazio O(C)

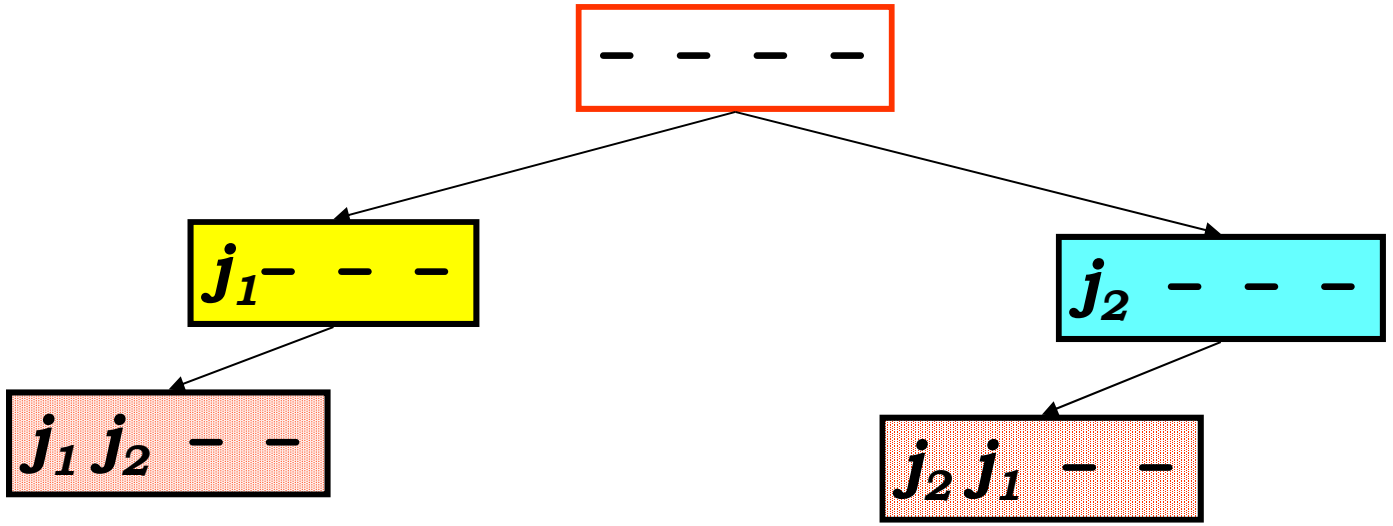
Ricostruire la soluzione



- Per ciascun valore calcolato serve un puntatore all'elemento della riga precedente da cui è derivato
- Quindi, è necessario mantenere l'intera tabella $\Rightarrow$ spazio O(nC)

Programmazione dinamica nello scheduling

Motivazione: Il branch-and-bound valuta diversi sottoproblemi in cui lo schedule parziale contiene gli stessi job (!)



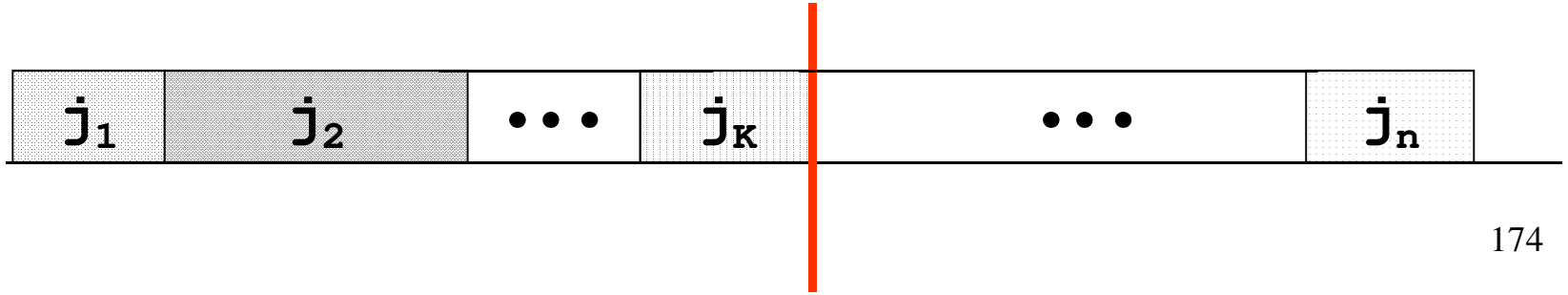
Ereditarietà della struttura ottima

Consideriamo un problema del tipo:

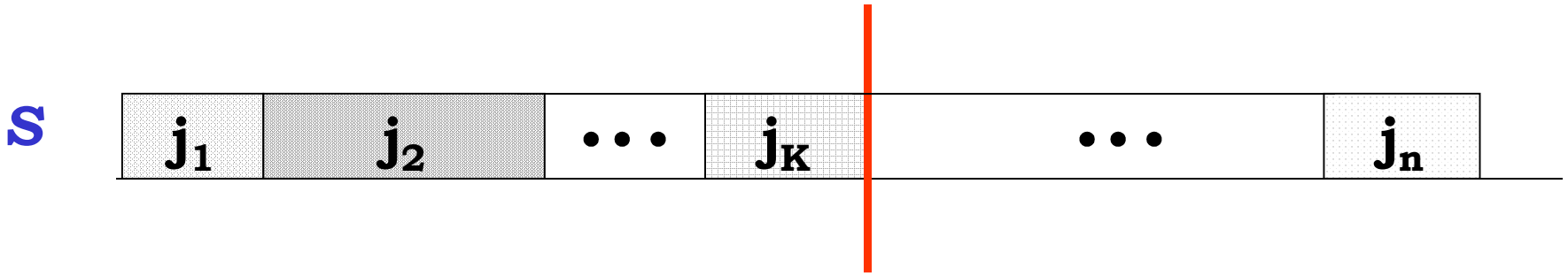
$$1//\Sigma_{j=1,\ldots,n}\ h_j(C_j)$$

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

Ereditarietà: in uno schedule ottimo i primi K job (per un qualunque $K \in \{1,\ldots,n\}$) formano uno schedule ottimo per il sottoproblema composto esclusivamente da tali job.



Ereditarietà della struttura ottima



per un qualunque $K \in \{1, \ldots, n\}$ il valore della funzione obiettivo ha la seguente forma:

$$\sum_{k=1}^{n} h_{jk}(C_{jk}) = \sum_{k=1}^{K} h_{jk}(C_{jk}) + \sum_{k=K+1}^{n} h_{jk}(C_{jk}) = A + B$$

Consideriamo il problema composto dai job $\{j_1,\ldots,j_K\}$: la sequenza $(j_1,\ldots,j_K)$ è ottima. Altrimenti, potremmo ridurre A senza modificare B, contraddicendo l'ottimalità di ***S***.

Metodo di Held e Karp

$z(J)$ valore ottimo del problema in cui l'insieme dei job è J.

Se $J = \{ j \}$ allora $z(J) = h_j(p_j)$

Se $|J| > 1$, l'ultimo job è completato al tempo Σp_j ed i primi $|J| - 1$ job devono essere schedulati in modo ottimo nel sottoproblema associato.
Quindi:

$$z(J) = \min_{j \in J} \left(z(J \setminus \{j\}) + h_j \left(\sum_{k \in J} p_k \right) \right)$$

Metodo di Held e Karp

Per risolvere la formula ricorsiva

$$z(J) = \min_{j \in J} \left(z(J \setminus \{j\}) + h_j \left(\sum_{k \in J} p_k \right) \right)$$

L'algoritmo procede in modo *bottom-up* iniziando a calcolare le soluzioni ottime dei problemi con un solo job, quindi utilizzando queste per risolvere i problemi con due job, e via di seguito.

Esempio: $1//\Sigma T_j$

job	1	2	3	4
p_j	8	6	10	7
d_j	14	9	16	16

Soluzioni per i quattro insiemi di un singolo elemento:

$$z(\{j\}) = T_j = max\,(0,\, p_j - d_j)$$

J	{1}	{2}	{3}	{4}
$p_j - d_j$	–6	–3	–6	–9
z (J)	0	0	0	0

Esempio: insiemi di due elementi

Soluzioni per i sei insiemi di due elementi:

$$z(\{j1,j2\}) =$$
$$= min(z(\{j_1\})+(p_{j1}+p_{j2})-d_{j2},\ z(\{j_2\})+(p_{j1}+p_{j2})-d_{j1})$$

J	{1,2}		{1,3}		{1,4}		{2,3}		{2,4}		{3,4}	
$C(J)$	14		18		15		16		13		17	
last	1	2	1	3	1	4	2	3	2	4	3	4
$T_j(C(J))$	0	5	4	2	1	0	7	0	4	0	1	1
$z(\{J\setminus\{j\}\})$ $+T_j(C(J))$	0	5	4	2	1	0	7	0	4	0	1	1
min	*			*		*		*		*	*	
$z(J)$	0		2		0		0		0		1791	

Esempio: insiemi di tre elementi

Soluzioni per i quattro insiemi di tre elementi. Esempio: $J=\{1, 2, 3\}$, $C(J) = p_1+p_2+p_3$

$$z(\{j1, j2, j3\})=min(z(\{j_1, j_2\})+C(J)-d_{j3},$$
$$z(\{j_1, j_3\})+C(J)-d_{j2}, z(\{j_2, j_3\})+C(J)-d_{j1})$$

J	{1,2,3}			{1,2,4}			{1,3,4}			{2,3,4}		
$C(J)$	24			21			25			23		
Last	1	2	3	1	2	4	1	3	4	2	3	4
$T_j(C(J))$	10	15	8	7	12	5	11	9	9	14	7	7
$z(\{J\setminus\{j\}\}+T_j(C(J)))$	10	17	8	7	12	5	12	9	11	15	7	7
min			*			*		*			*	
$z(J)$	8			5			9			17		

Esempio: insieme di quattro elementi

J	$\{1,2,3,4\}$			
$C(J)$	31			
Last	1	2	3	4
$T_j(C(J))$	17	22	15	15
$z(\{J\setminus\{j\}\}+T_j(C(j)))$	24	31	20	23
Min			*	
$z(J)$	20			

Ricostruzione della soluzione ottima:
$\{1,2,3,4\} \Rightarrow$ job 3 schedulato per ultimo;
$\{1,2,4\} \Rightarrow$ job 4 schedulato per ultimo;
$\{1,2\} \Rightarrow$ job 1 schedulato per ultimo;

(2,1,4,3) soluzione ottima

Complessità

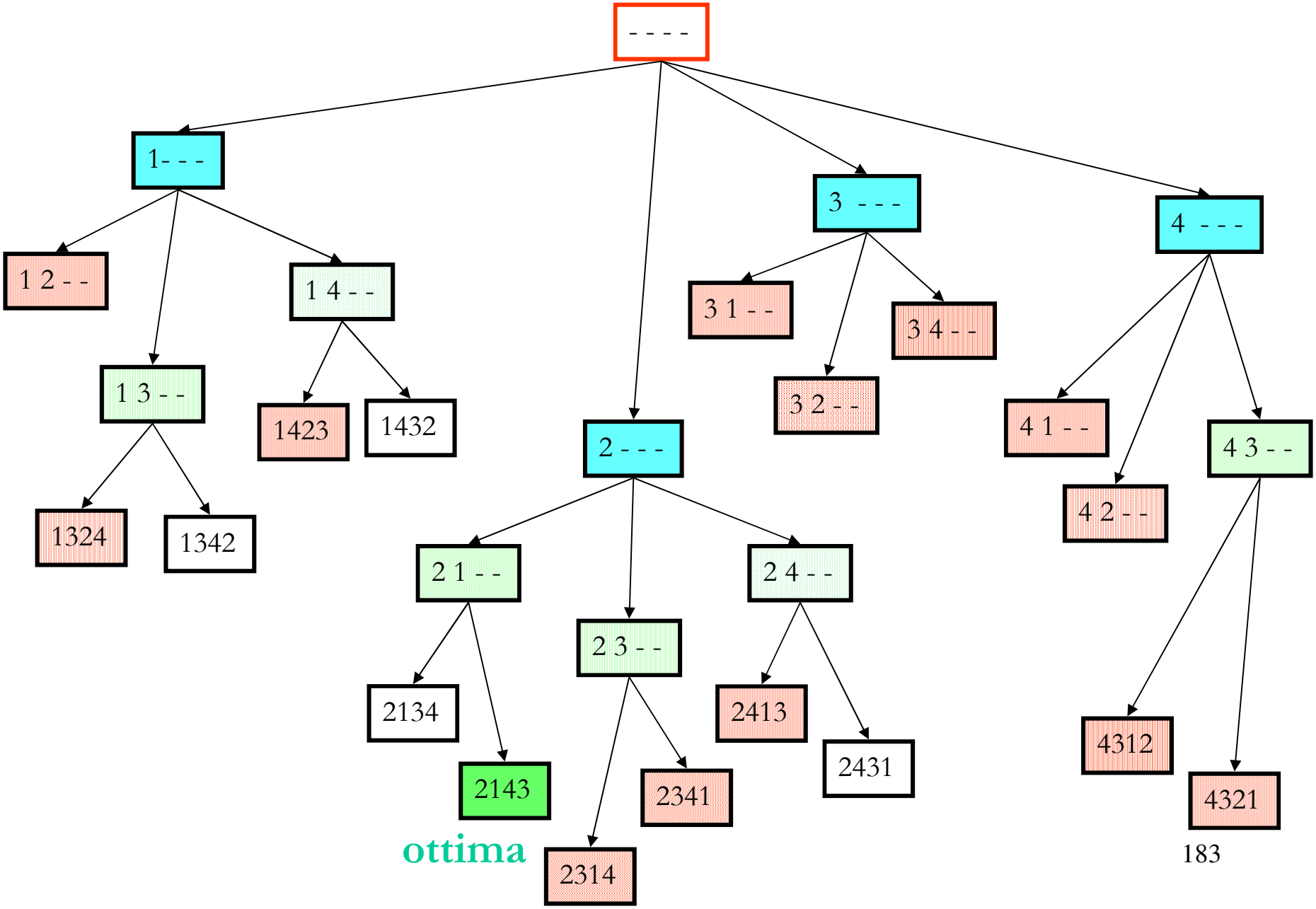
Tempo: numero di sottoproblemi valutati

$$\binom{n}{1}+\binom{n}{2}+\binom{n}{3}+\ldots+\binom{n}{n}$$

$$\Rightarrow \mathbf{O}(2^n)$$

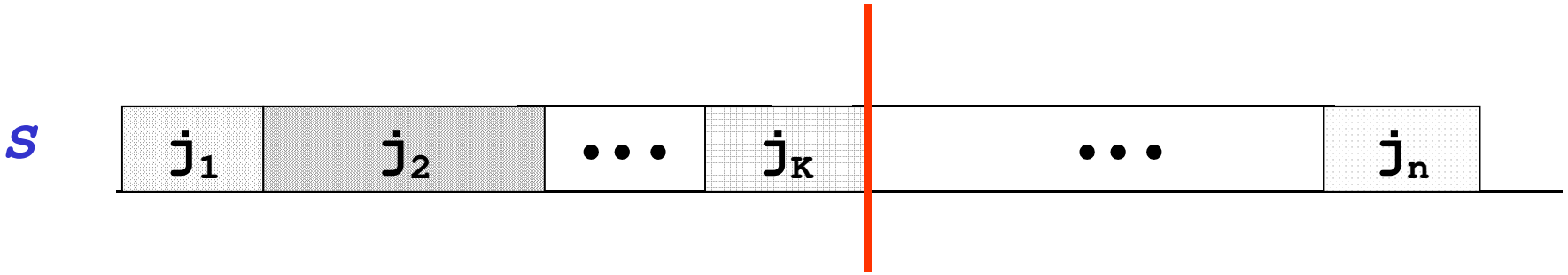
Spazio: 2^n * da memorizzare.

DP vs. branch&bound



Applicabilità I

Il principio di ereditarietà della struttura ottima si basa sul fatto che un risequenziamento dei job $(j_1,\ldots,j_K)$ non cambia il termine B.

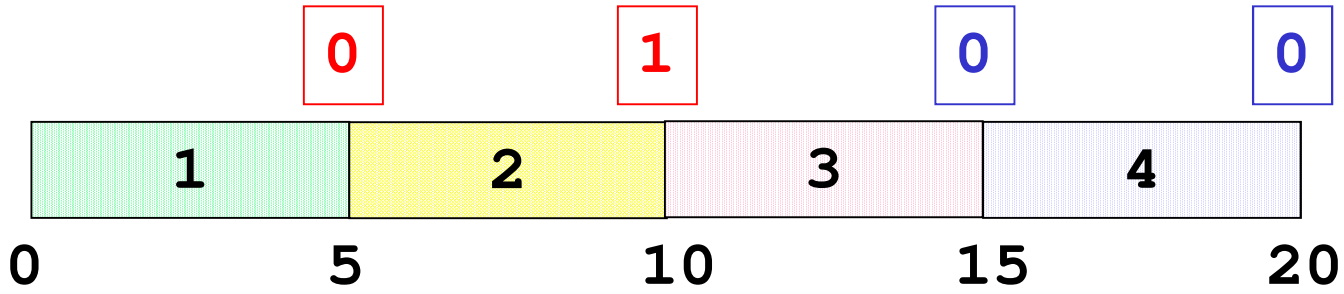


$$\sum_{k=1}^{n} h_{jk}(C_{jk}) = \sum_{k=1}^{K} h_{jk}(C_{jk}) + \sum_{k=K+1}^{n} h_{jk}(C_{jk}) = A + B$$

Mantenendo la stessa struttura della funzione obiettivo, questo non è vero se le release date sono non tutte nulle.

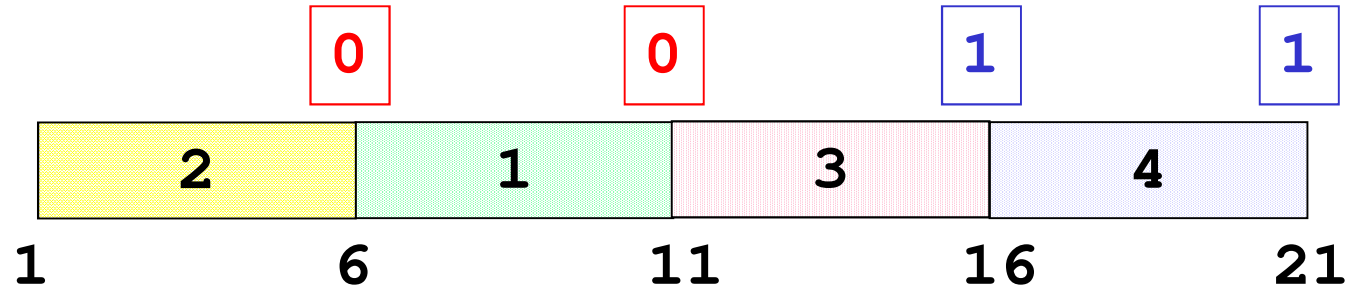
Controesempio

job	1	2	3	4
p_j	5	5	5	5
r_j	0	1	9	15
d_j	11	9	15	20



$\Sigma T_j = 1$

$T_1+T_2=1$



$\Sigma T_j = 2$

$T_1+T_2=0$

Applicabilità II

Il metodo di Held e Karp si applica anche a problemi min-max.
Ad esempio se consideriamo $1//L_{max}$ la ricorsione diventa:

$$z(J) = \min_{j \in J} \left(\max \left(z(J \setminus \{j\}), \sum_{j \in J} p_j - d_j \right) \right)$$

$1//\Sigma T_j$
Tardiness totale

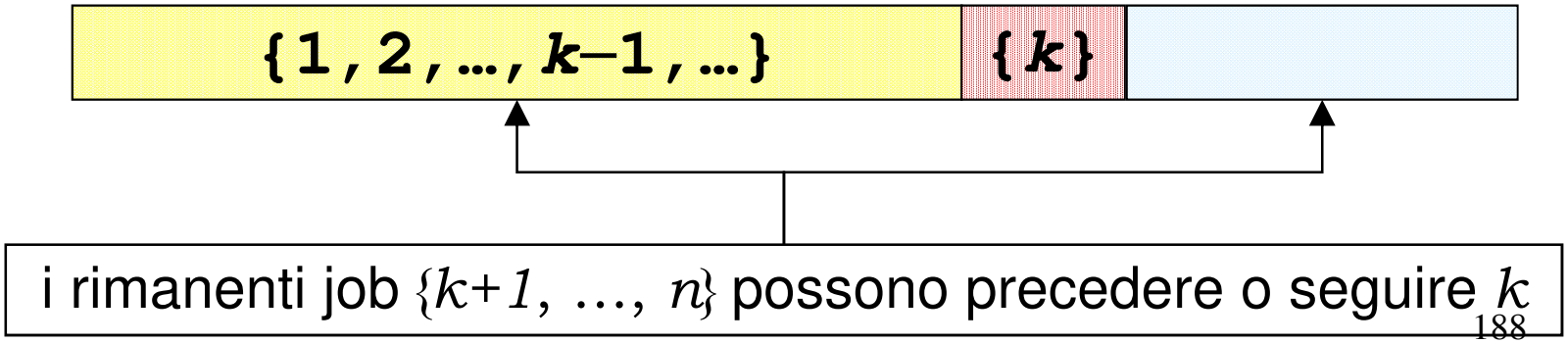
algoritmo pseudo-polinomiale

Struttura della sequenza ottima

Lemma 1.10 Se $p_j \leq p_k$ e $d_j \leq d_k$, esiste una sequenza ottima in cui j precede k.

Ipotesi: $d_1 \leq \ldots \leq d_n$, $p_k = \max(p_1, \ldots, p_n)$

Quindi, esiste una sequenza ottima in cui i job $\{1, \ldots, k-1\}$ precedono, in un qualche ordine, il job k.



Analisi di sensibilità

Prendiamo un job k qualunque e poniamo:

$$d_k = \max(d_k, C_k^{'})$$

Dove C_k' è il **massimo tempo di completamento del job k in una (qualsiasi) soluzione ottima**.

Otteniamo 2 istanze, entrambe con n job e tempi di processamento $p_1, \ldots, p_n$

I1] $d_1, \ldots, d_n$

I2] $d_1, \ldots, d_{k-1}, \max(d_k, C_k'), d_{k+1}, \ldots, d_n$

Analisi di sensibilità

***I1*]** $d_1,\ldots,d_n$

***I2*]** $d_1,\ldots,d_{k-1}$, $max(d_k,C_k')$,d_{k+1} ,…,d_n

Lemma 1.11
Ogni sequenza ottima per I2 è ottima per I1

Dimostrazione. Siano:

- S' uno schedule ottimo di I1 in cui k è completato al tempo C_k'

- S'' uno schedule ottimo di I2 e C_k'' il relativo tempo di completamento di k.

Dimostrazione I

Siano:

- $V'(S)$ la tardiness totale della sequenza S rispetto ad $\mathrm{I}1$
- $V''(S)$ la tardiness totale della sequenza S rispetto ad $\mathrm{I}2$

Valutiamo le due sequenze S' e S'' rispetto a $\mathrm{I}1$ e $\mathrm{I}2$.
Per costruzione:

1. $V'(S') = V''(S') + A_k$
2. $V'(S'') = V''(S'') + B_k$

Dimostriamo il teorema facendo vedere che:

- $V''(S') \geq V''(S'')$ e
- $A_k \geq B_k$

Dimostrazione II: consideriamo S'

Se $C_k' \leq d_k$, allora I1 e I2 coincidono ed il lemma è provato;
altrimenti: la due date di k in I2 è proprio C_k', cioè coincide col suo tempo di completamento in S'.
Quindi:

$$1.\ V'(S') = V''(S') + \boldsymbol{A_k}$$

$$\boldsymbol{A_k = C_k' - d_k}$$

Dimostrazione III: consideriamo S"

2. $V'(S'') = V''(S'') + \boldsymbol{B_k}$

consideriamo la sequenza S'', in cui il job k è completato in C_k''

$$B_k = T_k' - T_k'' = max(0, C_k'' - d_k) - max(0, C_k'' - C_k')$$

2 casi:

$B_k = max(0,\ C_k'' - d_k)$ se $C_k'' \leq C_k'$

$B_k = C_k'' - d_k - C_k'' + C_k' = C_k' - d_k$ se $C_k'' > C_k'$

quindi:

$$\boldsymbol{B_k = max(0,\ min(C_k'',\ C_k')\ - d_k)}$$

Dimostrazione IV

1. $V'(S') = V''(S') + A_k$
2. $V'(S'') = V''(S'') + B_k$

S´] $A_k = C_k' - d_k$

S''] $B_k = max(0, min(C_k'', C_k') - d_k)$

quindi: $A_k \geq B_k$

Essendo S'' ottima per I2, deve essere $V''(S') \geq V''(S'')$
da cui:

$$V'(S') \geq V'(S'')$$

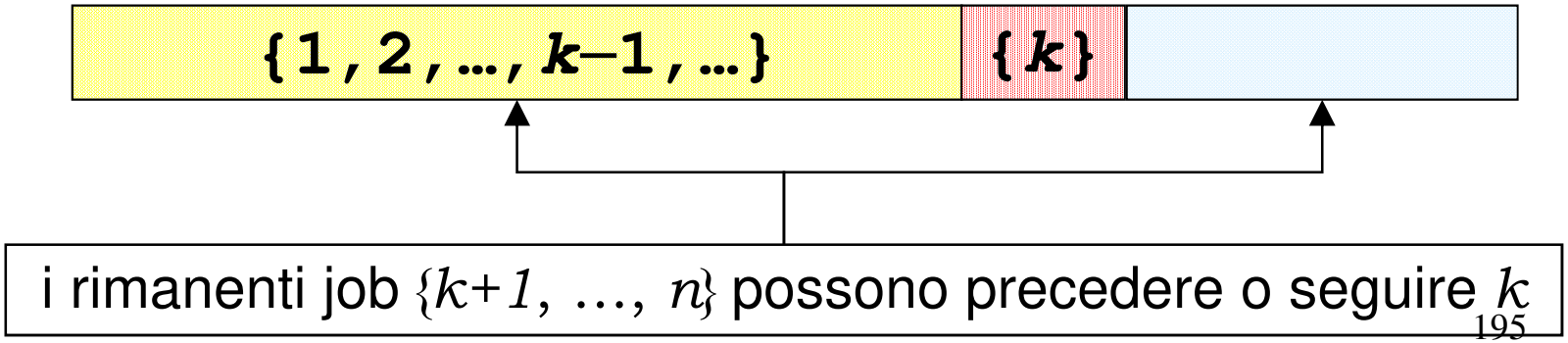
□

Struttura della sequenza ottima

Lemma 1.10 Se $p_j \leq p_k$ e $d_j \leq d_k$, esiste una sequenza ottima in cui j precede k.

Ipotesi: $d_1 \leq \ldots \leq d_n$, $p_k = \max(p_1, \ldots, p_n)$

Quindi, esiste una sequenza ottima in cui i job $\{1, \ldots, k-1\}$ precedono, in un qualche ordine, il job k.

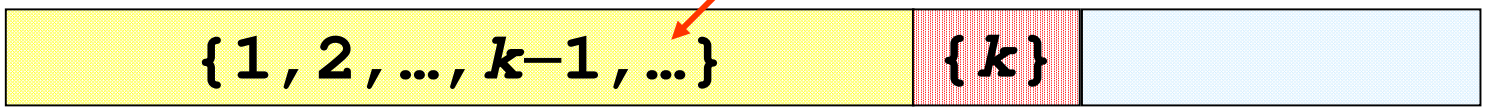


Posizione dei job $\{k+1, \ldots, n\}$

Lemma 1.12

Esiste un intero δ, $0 \leq \delta \leq n - k$, tale che esiste una sequenza ottima S in cui il job k è preceduto da tutti i job j con $j \leq k + \delta$ e seguito da tutti i job j per cui $j > k + \delta$

$$1, \ldots, k-1, k, \mathbf{k+1}, \ldots, \mathbf{k+\delta}, \ldots, n$$



Posizione dei job $\{k+1, \ldots, n\}$

Sia C_k' il massimo tempo di completamento del job k in una soluzione ottima dell'istanza $I1$

Sia S'' uno schedule ottimo di $I2$ tale da soddisfare Lemma 1.10

Sia C_k'' il tempo di completamento del job k in S''

Per il Lemma 1.10, S'' è ottima anche per I1, quindi

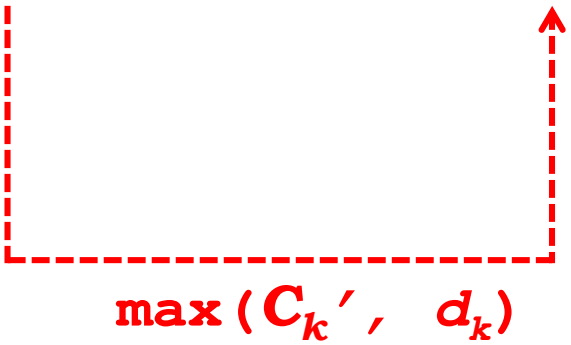
$$C_k'' \leq \max(C_k', d_k)$$

Dimostrazione I

Ragioniamo sulla soluzione S'', sfruttando la perturbazione della due date di k (ed il Lemma 1.10)

$$1, \ldots, k-1, k, k+1, \ldots, k+\delta, \ldots, n$$

$$d_1, \ldots, d_{k-1}, d_k, d_{k+1}, \ldots, d_{k+\delta}, \ldots, d_n$$

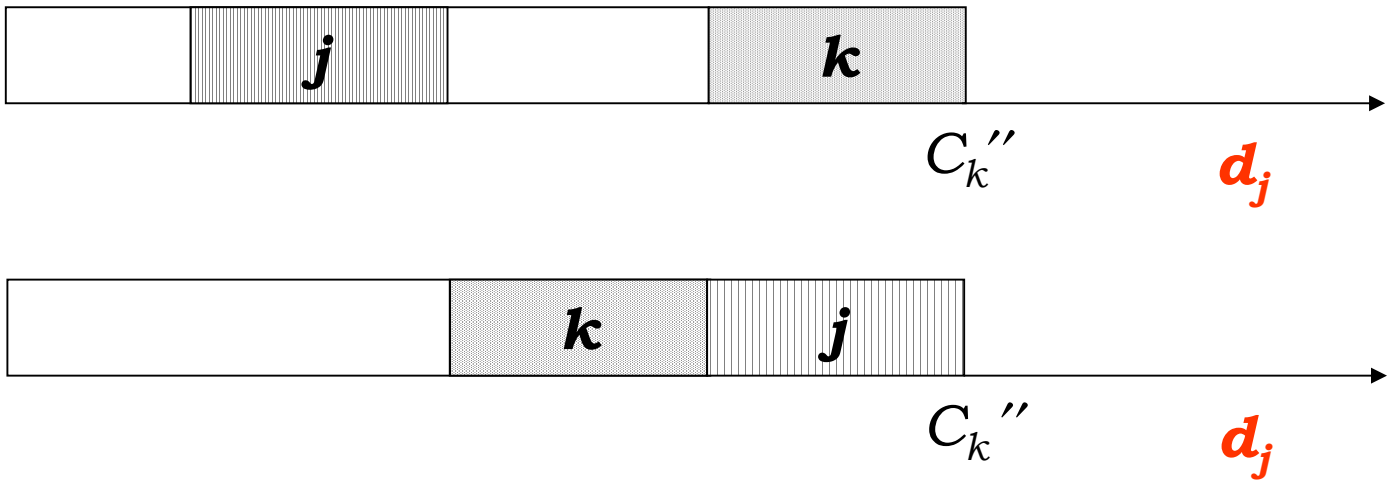


Dimostrazione I

Affermazione 1. Si può assumere che k non sia preceduto in ***S''*** da alcun job j con

$$d_j \geq max(C_k', d_k) \geq C_k''$$

Infatti, lo swap:



produrrebbe una sequenza ammissibile non peggiore di quella di partenza

Dimostrazione II

Affermazione 2.
il job k in I2 ha una due date pari a $max(C_k', d_k)$. Quindi, per il Lemma 1.10, deve essere preceduto in $\boldsymbol{S''}$ da tutti i job j con

$$d_j \leq max(C_k', d_k).$$

Quindi: scegliendo δ come il più grande intero tale che

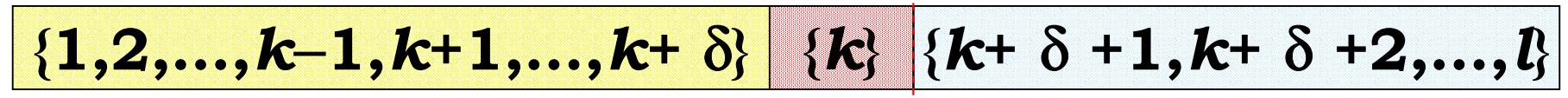
$$\boldsymbol{d_{k+\delta} \leq max(C_k', d_k)}$$

Ed essendo S'' ottima per I1, la prova è completa

Programmazione Dinamica

Insieme di job $\{1, \ldots, l\}$ che inizia all'istante t.
Dal Lemma 2, esiste un δ, $\mathbf{0 \le \delta \le l - k,}$ per cui uno schedule ottimo è dato dalla concatenazione di tre insiemi di job:

1. in qualche ordine | 2 | 3. in qualche ordine



t

$$C_k(\delta) = t + \Sigma_{j \le k+\delta} p_j$$

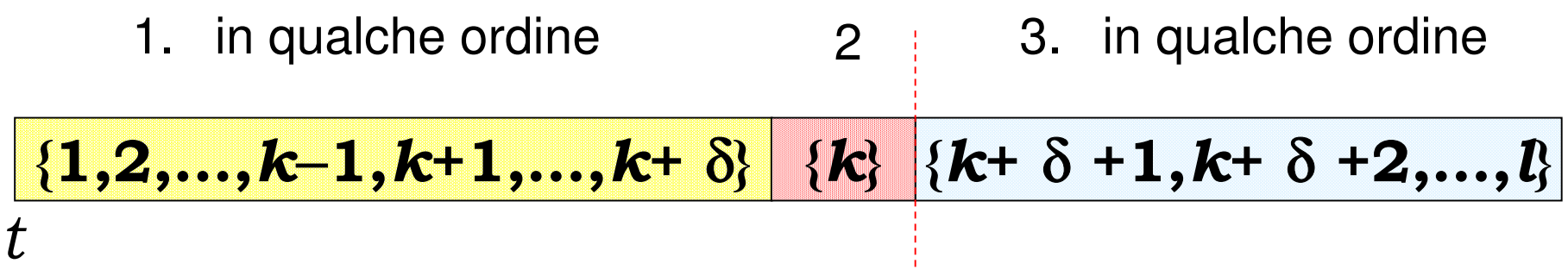
tempo di completamento di k

Affinché l'intera sequenza sia ottima devono essere ottime le sottosequenze relative agli insiemi 1 e 3 $\Rightarrow$ **ricorsione**

Programmazione Dinamica

Il Lemma 2, purtroppo non fornisce un metodo di calcolo di δ (perché?), ma solo un certificato di esistenza.

Tuttavia, $\mathbf{0 \le \delta \le l - k,}$ quindi, possiamo enumerare tutti i possibili valori di δ e scegliere quello che produce la sol. migliore



$$C_k(\delta) = t + \Sigma_{j \le k+\delta} p_j$$

tempo di completamento di k

Struttura del sottoproblema

Definiamo

$$J(j, l, s) \subseteq \{j, j+1, \ldots, l-1, l\}$$

l'insieme di job di durata minore o uguale a p_s. L'insieme non contiene il job s anche se $j \leq s \leq l$.

$V(J(j, l, s), t)$ tardiness totale dell'insieme $J(j, l, s)$ in una soluzione ottima quando l'insieme è iniziato al tempo t

condizioni al contorno:

$$V(\varnothing, t) = 0,$$
$$V(\{j\}, t) = \max(0, t + p_j - d_j)$$

Ricorsione

$$V(J(j,\ l,\ s),\ t) = min_{0 \le \delta \le l-k}\{V(J(j,\ k+\delta,\ k),\ t) +$$

$$+ max(0,\ C_k(\delta) - d_k) + V(J(k+\delta+1,\ l,\ k),\ C_k(\delta))\}$$

in cui:

$$p_k = \max(p_r |\ r \in J(j,\ l,\ s))$$

valore ottimo:

$$V(\{1,\ \ldots,\ n\},\ 0)$$

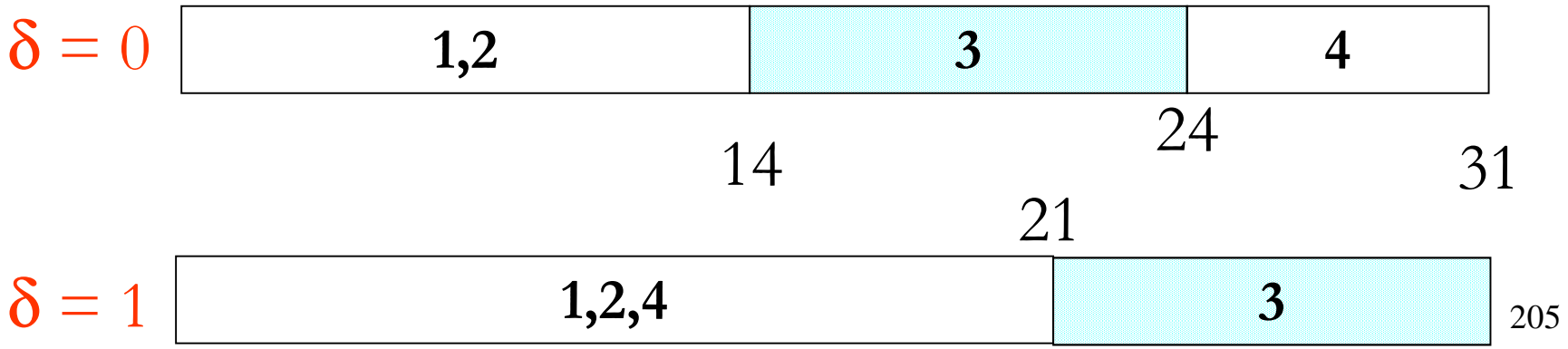
Esempio

job	1'	2'	3	4
p_j	8	6	10	7
d_j	14	9	16	16

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

V({1, 2, 3, 4}, 0) = min **(**

δ= 0 V(*J*(1, 3, 3), 0) + max (0, 24 – 16) + V(*J*(4, 4, 3), 24),
δ = 1 V(*J*(1, 4, 3), 0) + max (0, 31 – 16) **)**



Esempio

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

$J(1, 3, 3) = \{1,2\}$

$k = 2$

$J(1, 3, 3)$ è risolto essendo $d_1 < d_2$:

$V(J(1, 3, 3), 0) = \max(0, 6 - 9) + \max(0, 14 - 14) = 0$

$J(4, 4, 3) = \{4\}$

$V(J(4, 4, 3), 24) = \max(0, 31 - 16) = 15$

Esempio

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

V({1, 2, 3, 4}, 0) = min **(**

δ= 0 0 + max (0, 24 – 16) + 15,
δ = 1 V(*J*(1, 4, 3), 0) + max (0, 31 – 16) **)**

⇒ V({1, 2, 3, 4}, 0) = min (23, V(*J*(1, 4, 3), 0) + 15)

Esempio

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

$J(1, 4, 3) = \{1,2,4\}$

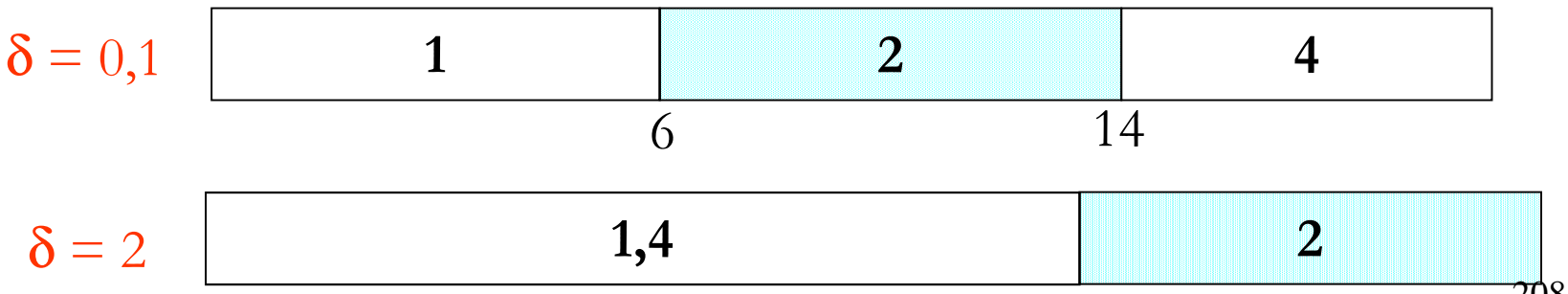
k = 2

V(J(1, 4, 3), 0) = min(

δ= 0 V(J(1, 2, 2), 0) + max (0, 14 – 14) + V(J(3, 4, 2), 14),

δ=1 V(J(1, 3, 2), 0) + max (0, 14 – 14) + V(J(4, 4, 2), 14),

δ=2 V(J(1, 4, 2), 0) + max (0, 21 – 14))



Esempio

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

V(J(1, 4, 3), 0) = min(
δ= 0 V(J(1, 2, 2), 0) + max (0, 14 – 14) + V(J(3, 4, 2), 14),
δ=1 V(J(1, 3, 2), 0) + max (0, 14 – 14) + V(J(4, 4, 2), 14),
δ=2 V(J(1, 4, 2), 0) + max (0, 21 – 14))

δ=0:

J(1, 2, 2) = {1} → V(J(1, 2, 2), 0) = max (0, 6 – 9) = 0
J(3, 4, 2) = {4} → V(J(3, 4, 2), 14) = max (0, 21 – 16) = 5

Esempio

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

δ=1:

$J(1, 3, 2) = \{1\} \rightarrow V(J(1, 3, 2), 0) = \max(0, 6 - 9) = 0$

$J(4, 4, 2) = \{4\} \rightarrow V(J(4, 4, 2), 14) = \max(0, 21 - 16) = 5$

δ=2:

$J(1, 4, 2) = \{1,4\} \rightarrow V(J(1, 4, 2), 0) = \max(0, 6 - 9)$

$+ \max(0, 13 - 16) = 0$

Esempio

job	1'	2'	3	4
p_j	8	6	10	7
d_j	14	9	16	16

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

V(J(1, 4, 3), 0) = min(
* $\delta = 0$, $\delta = 1$ 0 + 0 + 5,
$\delta = 2$ 0 + 7) = 5

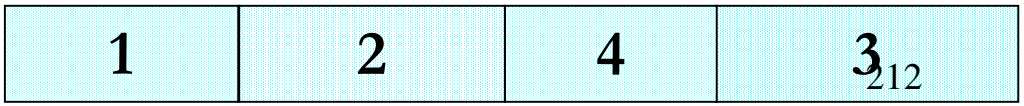
Esempio

job	1	2	3	4
p_j	6	8	10	7
d_j	9	14	16	16

V({1, 2, 3, 4}, 0) = min ((0 + 8 + 15), **V(*J*(1, 4, 3), 0)** + 15)

δ = 0 23,
*δ = 1 20) = 20

V(*J*(1, 4, 3), 0) = min(
* δ = 0, δ = 1 0 + 0 + 5,
δ= 2 0 + 7) = 5



Complessità

Tempo:
$O(n^3)$ sottoinsiemi $J(j, l, k)$ e Σp_j possibili istanti t. Quindi, il numero di equazioni ricorsive da risolvere è pari a $O(n^3 \Sigma p_j)$. Ciascuna di esse richiede un tempo $O(n)$.

Quindi, l'algoritmo richiede tempo $\boldsymbol{O(n^4 \Sigma p_j)}$

Tardiness totale pesata

$$1//\Sigma w_j T_j$$

$1//\Sigma w_j T_j$

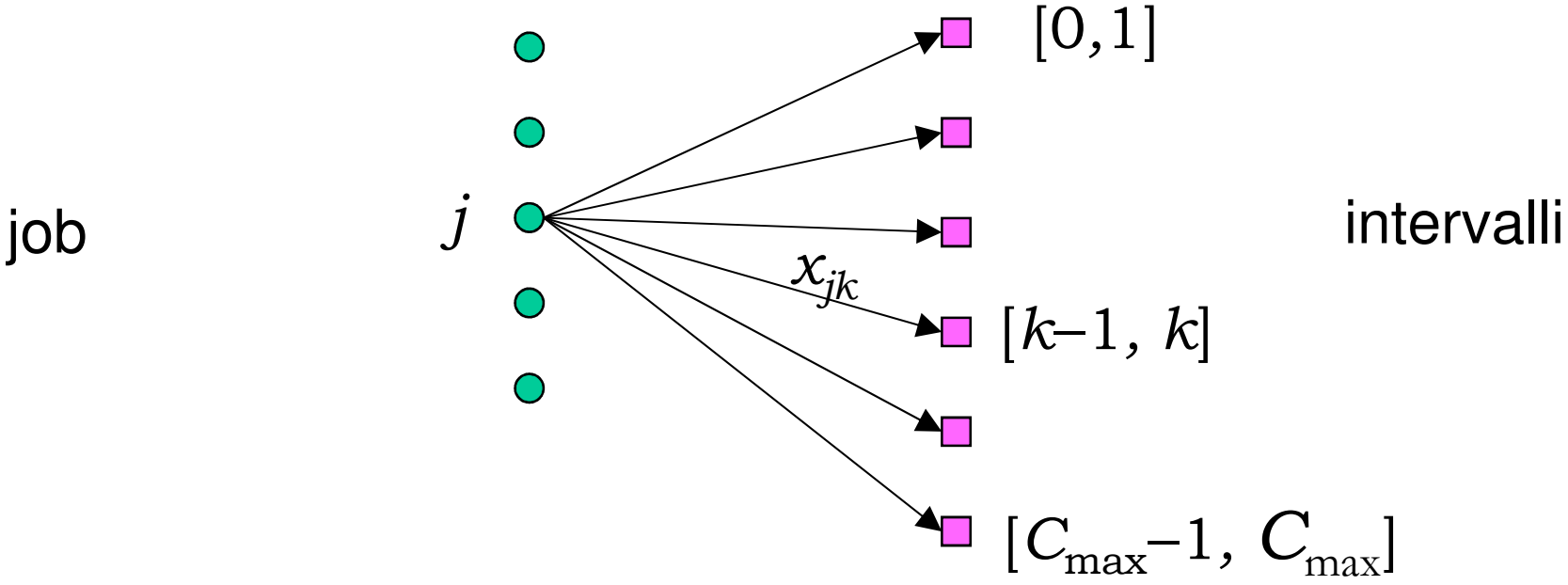
Teorema. Il problema $1//\Sigma w_j T_j$ è NP-hard in senso forte.

Lemma. Se $p_j \leq p_k$ e $d_j \leq d_k$, e $w_j \geq w_k$ allora esiste una sequenza ottima in cui j precede k.

Rilassamento preemptivo: ciascun job j di durata p_j è suddiviso in p_j job di durata unitaria.

Variabili decisionali:
$x_{jk}=1$ se una unità del job j è eseguita nell'intervallo $[k-1,k]$ e 0 altrimenti

$1/prmp/\Sigma w_j T_j$



$$\sum_{k=1}^{C_{\max}} x_{jk} = p_j, j = 1,\ldots,n$$

$$\sum_{j=1}^{n} x_{jk} = 1, k = 1,\ldots,C_{\max}$$

$1/prmp/\Sigma w_j T_j$: formulazione PL0-1

$$\max \sum_{j=1}^{n} \sum_{k=1}^{C_{\max}} c_{jk} x_{jk}$$

occorre determinare c_{jk} tali da rappresentare la tardiness totale (o un suo rilassamento)

$$\sum_{k=1}^{C_{\max}} x_{jk} = p_j, \quad j = 1,\ldots,n$$

$$\sum_{j=1}^{n} x_{jk} = 1, \quad k = 1,\ldots,C_{\max}$$

$$x_{jk} \in \{0,1\}, \qquad j = 1,\ldots,n; k = 1,\ldots,C_{\max}$$

problema dei trasporti

Richiamo: Problema dei trasporti

s località sorgente e t destinazioni. Ciascuna sorgente i$\in$ {1,...,s} dispone di d_i unità di prodotto ed ogni destinazione $j \in$ {1,...,t} ne richiede almeno r_j unità.

Il costo unitario di trasporto da i a j è c_{ij}.

Determinare una politica di trasporto di costo minimo.

$$\min \sum_{i=1}^{s} \sum_{j=1}^{t} c_{ij} x_{ij}$$

x_{ij} quantità trasportata da i a j

$$\sum_{j=1}^{t} x_{ij} = d_i \qquad i \in \{1,\ldots,s\}$$

$$\sum_{i=1}^{s} x_{ij} = r_j \qquad j \in \{1,\ldots,t\}$$

$$x_{ij} \geq 0, \quad \text{interi}$$

Coefficienti c_{jk}

Determinare c_{jk} tali che, per ogni schedule non preemptivo, risulti:

$$\sum w_j T_j \geq \sum_{j=1}^{n} \sum_{k=1}^{C_{\max}} c_{jk} x_{jk}$$

Definiamo i coefficienti di costo in modo che:

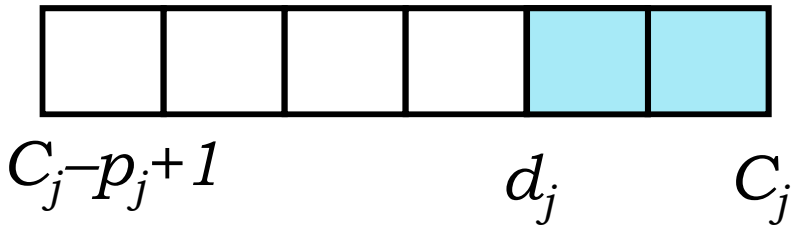
$$\sum_{k=l-p_j+1}^{l} c_{jk} \leq w_j \max(l - d_j, 0)$$

per $j = 1,\ldots,n$; $l = 1,\ldots,C_{max}$

Funzione obiettivo

Allora, per una qualunque soluzione <u>non preemptiva</u> risulta, per ogni *j*:

$$\sum_{k=1}^{C_{\max}} c_{jk} x_{jk} = \sum_{k=C_j - p_j + 1}^{C_j} c_{jk} \leq w_j \max(C_j - d_j, 0)$$



Possibile scelta:

$$c_{jk} = \begin{cases} 0, \ per \ k \leq d_j \\ w_j, \text{ altrimenti} \end{cases}$$