Modelli di scheduling su macchina singola

1. Modello base

- Consideriamo il caso in cui *ogni job consiste di una sola operazione*. Poiché un insieme di job è partizionato in relazione alle macchine che devono eseguire le varie operazioni, allora *ogni singola macchina nello shop è indipendente dalle altre e può essere schedulata separatamente*.

- Consideriamo in prima istanza che i tempi di processamento siano costanti e noti e che i job siano tutti disponibili al tempo iniziale.

- Questo modello di base è importante sia per ragioni teoriche che pratiche.
  Dal punto di vista teorico questo problema è il più semplice e serve a capire i casi più complessi di job-shop. Dal punto di vista pratico questo modello rappresenta bene alcuni sistemi reali.

Modelli di scheduling su macchina singola

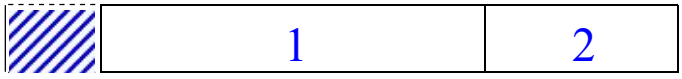
2. Schedule di permutazione

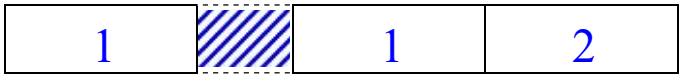
- In situazioni più complesse del modello di base sono possibili schedule con *preempzione* e con *idle-time*.
- Questo significa che per esempio nel caso di due job sono possibili le seguenti sequenze.

1) 

2) 

3) 

4) 

5) 

Nel caso del modello di base prima esposto si può dimostrare che:

Teorema: In un problema su *macchina singola* con *job tutti disponibili* $1||f$, con f *misura* di prestazione *regolare*, esiste almeno una *schedula ottima non preemptiva*, e *senza idle-time*.

- Questo risultato implica che è *sufficiente esaminare* soltanto le "*schedule di permutazione*" ossia quelle schedule che sono completamente specificate quando e' noto l'ordine (*sequenza*) di processamento dei job.

Modelli di scheduling su macchina singola

2. Schedule di permutazione (continua)

- Nel caso di scheduling di n job su macchina singola, le schedule di permutazione sono le $n!$ possibili permutazioni (sequenze) dei numeri $1, 2, \ldots, n$ di identificazione dei job.
  Data la sequenza e noti i tempi di processamento è possibile calcolare i tempi di attesa W_j e quindi ogni altra proprietà della schedula.

- In generale nelle schedule di permutazione si fa riferimento alla posizione k occupata da un job nella sequenza. Con $[k]$ si indica il *job che occupa la* k*-ma posizione*; $[3] = 7$ significa che il job 7 occupa la posizione 3; $p_{[k]}$ è il tempo di processamento del job in posizione k.
  Con questa notazione è ovvio che

$$C_{[1]} \leq C_{[2]} \leq \ldots \leq C_{[n]}$$

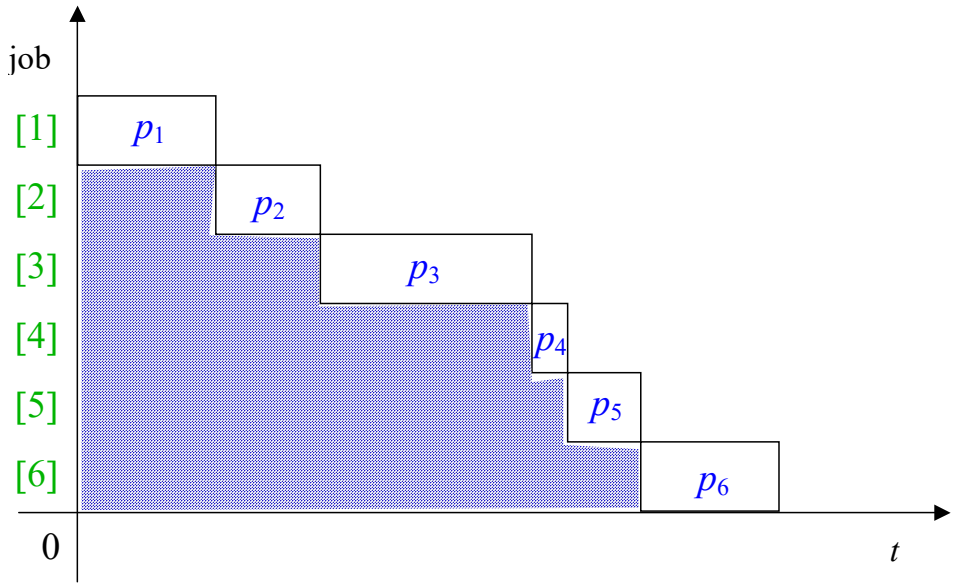
- E' opportuno sottolineare come nelle schedule di permutazione $F_{\max}$ è la somma dei p_i, $i = 1, 2, \ldots, n$.
  Il suo valore dipende pertanto dai tempi di processamento ed è lo stesso per qualsiasi delle $n!$ possibili sequenze.
  $F_{\max}$ quindi, così come altre misure ($C_{\max}$) sono *indipendenti dalla sequenza* e non possono essere assunti come criteri di ottimalità.

Modelli di scheduling su macchina singola

3.1. Regole di sequenziamento: (SPT)

A)Regola SPT

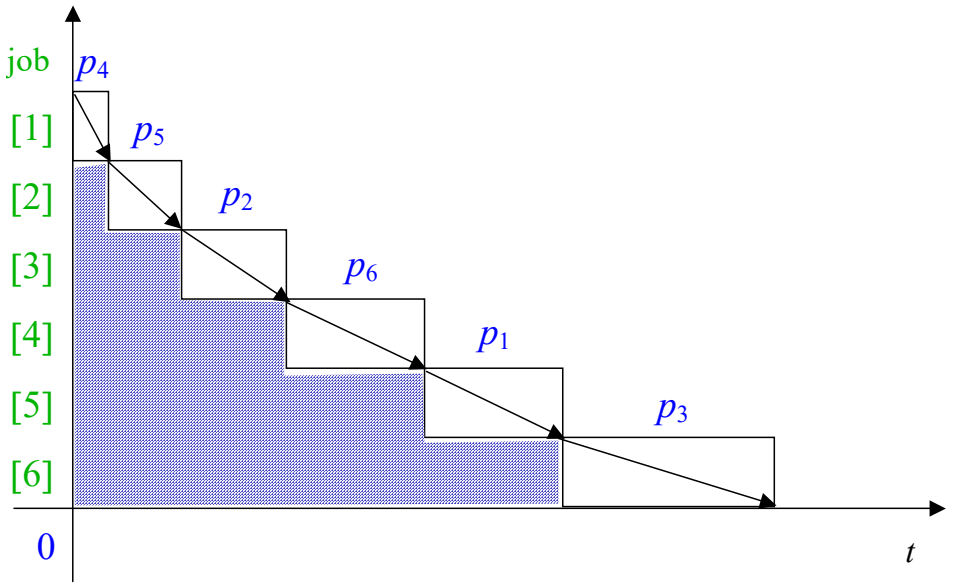
- Una sequenza di un problema su *macchina singola* può essere rappresentata come nella figura seguente



- L'area totale rappresenta la somma dei tempi di completamento dei job (tempo di completamento totale). Essa è composta da:
  - l'area rappresentata dai blocchi etichettati che è *indipendente* dalla sequenza dei job.
  - l'area tratteggiata che *dipende* dall'ordine dei blocchi etichettati.
- Caratterizzando ogni blocco etichettato con un vettore di pendenza $-1/p_i$, risulta che l'area tratteggiata è minima quando i blocchi sono ordinati in modo da formare con i vettori che li rappresentano una curva convessa come riportato nella figura seguente.

Modelli di scheduling su macchina singola

3.1. Regole di sequenziamento: (SPT) (continua)



- Questa configurazione che minimizza il tempo di completamento totale corrisponde ad ordinare i job secondo la regola *shortest-processing-time* (SPT) ossia

$$p_{[1]} \leq p_{[2]} \leq \ldots \leq p_{[n]}$$

- Più rigorosamente la regola SPT può essere provata in base al seguente

  Teorema (Smith 1956)

  Nel problema $1||\sum C_j$, il *tempo di completamento totale* e' *minimizzato* sequenziando i job in modo che

$$p_{[1]} \leq p_{[2]} \leq \ldots \leq p_{[n]}$$

  e *massimizzato* sequenziando i job in modo che

$$p_{[1]} \geq p_{[2]} \geq \ldots \geq p_{[n]}$$

  ossia in base alla regola *largest-processing-time* (LPT)

Modelli di scheduling su macchina singola

3.1. Regole di sequenziamento: (SPT) (continua)

Dimostrazione:

- Poiché basta considerare solo le schedule di permutazione il *tempo di completamento* del job $[k]$ è pari a

$$C_{[k]} = \sum_{i=1}^{k} p_{[i]}$$

- Il *tempo di completamento totale* di n job è allora dato da

$$\sum_{k=1}^{n} C_{[k]} = \sum_{k=1}^{n}\sum_{i=1}^{k} p_{[i]} = [(p_{[1]}) + (p_{[1]} + p_{[2]}) + \ldots + (p_{[1]} + p_{[2]} + \ldots + p_{[n]})] =$$

$$\sum_{i=1}^{n} (n - i + 1) p_{[i]}$$

- Poiché la *sequenza dei valori* $(n - i + 1)$ è *decrescente* la *minimizzazione* di $\sum C_j$ si ottiene *ordinando i job secondo valori non decrescenti di* $p_{[i]}$.
- Viceversa, poiché la somma di prodotti è massimizzata se entrambe le sequenze di numeri hanno lo stesso ordine, $\sum C_j$ è *massimizzata* da un *ordinamento non crescente dei valori di* $p_{[i]}$. c. v. d.

- Si può dimostrare in generale che le stesse regole valgono anche se la funzione obiettivo è

$$\overline{C}^{\alpha} = \frac{\sum_{j=1}^{n} C_j^{\alpha}}{n} \qquad \text{per } \alpha > 0$$

Modelli di scheduling su macchina singola

3.1. Regole di sequenziamento: (SPT) (continua)

Osservazioni sulla regola SPT

- La regola SPT *ottimizza* anche le seguenti misure di prestazioni: $\bar{F}$, $\bar{W}$, ed $\bar{L}$.
- La regola SPT *non ottimizza* in generale alcune misure di prestazione che sono funzioni delle due-date.

  Se ad esempio consideriamo la seguente istanza:

job	p	d
a	1	3
b	2	2

  Le sequenze possibili sono ovviamente soltanto:

  (a, b) e (b, a)

- La regola SPT sceglie la sequenza (a, b) come ottima ma la sequenza (b, a) ha $L_{\max}$, $T_{\max}$, e $\bar{T}$ più piccole e un numero di job in ritardo minore.

- Nel caso in cui le *due-date* sono così strette che tutti i job sono in ritardo, la regola *SPT minimizza* $\bar{T} \equiv \bar{L}$ in ogni condizione e, nel caso in cui $d_1 = d_2 = \ldots = d_n = d$, anche il numero dei job in ritardo ($1|d_j = d|\sum U_j$).

Modelli di scheduling su macchina singola

3.2. Regole di sequenziamento: (EDD)

B) Regola EDD

Teorema (Jackson 1955)
Nei problemi $1||L_{\max}$ e $1||T_{\max}$, le misure $L_{\max}$ e $T_{\max}$ sono *minimizzate* sequenziando i job secondo *valori non decrescenti* delle *due-date* (regola *EDD* -*E*arliest *D*ue *D*ate).

$$d_{[1]} \leq d_{[2]} \leq \ldots \leq d_{[n]}$$

Dimostrazione:

- Senza perdita di generalità assumiamo che per ogni j si ha $p_j > 0$
- Sia S una schedula che differisce da una schedula *EDD* in quanto in una qualche posizione k si ha:

$$d_{[k]} > d_{[k+1]}$$

- E siano $i = [k]$ e $j = [k+1]$ gli identificatori dei job in posizione k-ma e $(k+1)$-ma in S, rispettivamente.

- Consideriamo ora una sequenza S' che differisce da S per il solo fatto di aver scambiato i e j di posizione.
  Le due sequenze sono le stesse per i primi $(k-1)$ job e gli ultimi $(n-k-1)$ per cui la *lateness massima* di questi $(n-2)$ job e' la *stessa*; sia L il suo valore.
  Le schedule *differiscono* quindi solo per le *lateness* di i e j.

Modelli di scheduling su macchina singola

3.2. Regole di sequenziamento: (EDD) (continua)

- Vogliamo dimostrare che:

$$\max\{L, L_i(S), L_j(S)\} \geq \max\{L, L_i(S'), L_j(S')\}$$

che vale all'uguaglianza se L domina gli altri termini in ciascuna espressione.

- Sia $t = \sum_{h=1}^{k-1} p_{[h]}$. Allora le relative *lateness*, sono:

in S: $L_i(S) = L_{[k]}(S) = t + p_{[k]} - d_{[k]} = t + p_i - d_i$

$L_j(S) = L_{[k+1]}(S) = t + p_{[k]} + p_{[k+1]} - d_{[k+1]} = t + p_i + p_j - d_j$

in S': $L_i(S') = L_{[k+1]}(S') = t + p_j + p_i - d_i$

$L_j(S') = L_{[k]}(S') = t + p_j - d_j$

- Ora $L_i(S) < L_j(S)$, perché per ipotesi $d_i > d_j$ (e $p_j > 0$); inoltre, per $L_j(S)$ valgono le seguenti relazioni:

$L_j(S) > L_i(S')$ perché $d_i > d_j$

$L_j(S) > L_j(S')$ perché $p_i > 0$

cioè

$$L_j(S) > \max\{L_i(S'), L_j(S')\}$$

per cui

$$\max\{L, L_i(S), L_j(S)\} \geq \max\{L, L_i(S'), L_j(S')\}$$

- Quindi ogni schedula S si può migliorare scambiando coppie di job adiacenti in modo da rispettare la regola EDD. c.v.d.
- Per quanto riguarda $T_{\max}$ la sua minimizzazione deriva direttamente da

$$T_{\max}(S) = \max\{0, L_{\max}(S)\} \geq \max\{0, L_{\max}(S')\} = T_{\max}(S')$$

Modelli di scheduling su macchina singola

3.3. Regole di sequenziamento: (MST)

C) Regola MST

Una seconda maniera di usare l'informazione sulle due-date è quella di considerare gli "*slack-time*".

- Si definisce "*slack-time*" del job i al tempo t l'espressione

$$d_i - (t + p_i) = d_i - t - p_i$$

- Poiché il job con il minimo slack time è quello che presumibilmente presenta il più alto rischio di ritardo esso dovrebbe avere la precedenza nella schedula.

- Poiché ad un certo istante della decisione t è comune a tutti i job la regola di sequenziamento è

$$(d_{[1]} - p_{[1]}) \leq (d_{[2]} - p_{[2]}) \leq \ldots \leq (d_{[n]} - p_{[n]})$$

Vale infatti il seguente

Teorema

Nei problemi $1||L_{\min}$ e $1||T_{\min}$, le misure $L_{\min}$ e $T_{\min}$ sono *massimizzate* sequenziando i job secondo *valori non decrescenti* degli *slack time* (regola *MST* -*M*inimum *S*lack *T*ime).

Modelli di scheduling su macchina singola

3.3. Regole di sequenziamento: (MST) (continua)

Dimostrazione:

- Consideriamo una schedula S che differisce da una schedula *MST* in quanto in una qualche posizione k si ha

$$(d_{[k]} - p_{[k]}) > (d_{[k+1]} - p_{[k+1]}).$$

- E siano $i = [k]$ e $j = [k+1]$, ripetendo lo stesso ragionamento del teorema precedente si ha:

$$L_i(S) < L_j(S') \quad \text{perché} \quad (d_i - p_i) > (d_j - p_j)$$
$$L_i(S) < L_i(S') \quad \text{perché} \quad p_j > 0$$

cioè

$$L_i(S) < \min\{L_j(S'), L_i(S')\}$$

- Se L è la *minima lateness* degli altri $(n-2)$ job della schedula si ottiene che

$$\min\{L, L_i(S), L_j(S)\} \leq \min\{L, L_i(S'), L_j(S')\}$$

- Perciò passando da S ad S', $L_{\min}$ non decresce e quindi S può essere migliorata. c.v.d.

- Analogamente

$$T_{\min}(S) = \max\{0, L_{\min}(S)\} \leq \max\{0, L_{\min}(S')\} = T_{\min}(S')$$

Conclusione. Riassumendo:

la regola *SPT minimizza* $\overline{F}$, $\overline{C}$, $\overline{W}$ e $\overline{L}$;
la regola *EDD minimizza* $L_{\max}$ e $T_{\max}$;
la regola *MST massimizza* $L_{\min}$ e $T_{\min}$.
Tutte operano in $O(n \log n)$.

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline

- Quando in un problema tutte le due-date possono essere soddisfatte (esiste almeno una sequenza dei job per cui $T_{\max} = 0$) può capitare che ci siano più sequenze in queste condizioni.

  In questo caso la scelta può essere fatta sulla base di un secondo criterio quale la minimizzazione di $\sum C_j$. Nel caso specifico le due-date assumono il ruolo di *deadline* in quanto vanno rispettate tassativamente.
  Consideriamo pertanto il problema $1|\text{deadline}|\sum C_j$, (o $1|L_{\max} \le 0|\sum C_j$) per il quale vale il seguente

Teorema (Smith 1956)

Se in un problema su macchina singola esiste una sequenza per cui $T_{\max} = 0$ (cioè $L_{\max} \le 0$), allora vi è un *ordinamento* dei job che *minimizza* $\sum C_j$ con il job k in ultima posizione *se e solo se*

a) $d_k \ge \sum_{j=1}^{n} p_j$; b) $p_k \ge p_i, \quad \forall\, i : d_i \ge \sum_{j=1}^{n} p_j$

Osservazione:

Il teorema dice che un job può essere in *ultima posizione se* questa circostanza *non provoca una tardiness diversa da zero* e se il job ha il *più grande tempo di processamento tra tutti quelli dell'insieme dei job che potrebbero stare nell'ultima posizione senza essere in ritardo*.

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

Procedura ricorsiva:

- Sulla base del precedente teorema si può costruire la schedula completa che soddisfa le deadline e minimizza $\sum C_j$ nel modo seguente:
  - Si pone in posizione n il job con il *più grande* p_i tra quelli per cui le *due-date* sono *non superiori al tempo di completamento* del job in posizione n che è noto a priori in quanto uguale a $\sum_{i=1}^{n} p_i$.
  - Una volta fatta la scelta per $[n]$ si conosce il tempo di completamento del job in posizione $[n-1]$ perché

$$C_{[n-1]} = \sum_{i=1}^{n} p_i - p_{[n]}$$

  - Si prosegue quindi scegliendo il job $[n-1]$ con lo stesso criterio e si va avanti fino a costruire tutta la schedula.

- Questa procedura consente di risolvere il problema $1|\text{deadline}|\sum C_j$ nel caso in cui questo ammetta soluzione (esiste almeno una sequenza dei job per cui $T_{\max} = 0$).

- È possibile implementare tale procedura in modo che operi in $O(n \log n)$.

- La stessa procedura può essere impiegata per risolvere il problema $1|L_{\max} \leq \delta|\sum C_j$, visto che è equivalente a $1|L_{\max} \leq 0|\sum C_j$ con due-date $d'_j = d_j + \delta$.

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

Esempio: Si consideri la seguente istanza

job	1	2	3	4	5	6
p_i	4	7	1	3	2	5
d_i	24	21	8	5	10	23

Il sequenziamento *EDD* dà luogo a $T^*_{\max} = 0$ per cui esiste almeno una soluzione ammissibile.

Si può determinare la soluzione ottima applicando pertanto la procedura illustrata.

Poichè $\sum_{i=1}^{6} p_i = 22$ solo i job 1 e 6 hanno valori delle *due-date* ≥ 22. Poichè $p_6 > p_1$ il job 6 occupa la posizione 6.

L'istanza residua è con 5 job con $\sum_{i=1}^{5} p_i = 17$.

Pertanto i job 1 e 2 sono candidati alla quinta posizione.
Poiché $p_2 > p_1$ il job 2 viene scelto per la posizione 5.

Proseguendo in questo modo tra tutte le sequenze con $T_{\max} = 0$ si ottiene la sequenza (3, 4, 5, 1, 2, 6) di valore $\sum C_j = 60$.

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

- Estendiamo lo studio analizzando il problema *multi-obiettivo* $1||\sum C_j, L_{\max}$.

- Nei problemi multi-obiettivo, il concetto di soluzione ottima è sostituito da quello di *soluzione non-dominata* (Pareto-ottima, efficiente).

Def.

Dato un problema multi-obiettivo con m f.o. (da min.)

$$\min f_i(S),\ i = 1, \ldots, m$$
$$S \in X$$

la *soluzione* $S' \in X$ è *non-dominata* se $\nexists\ S \in X$: $f_i(S) \leq f_i(S')$, per ogni $i = 1, \ldots, m$, con almeno una delle disuguaglianze (es. h) valida in senso stretto ($f_h(S) < f_h(S')$).

- Risolvere un problema multi-obiettivo significa in generale determinare tutte le sue *soluzioni non-dominate*.

- Nel nostro problema (bi-obiettivo) $1||\sum C_j, L_{\max}$, se la schedula S' è non-dominata, è possibile trovare una schedula migliore per la prima f.o. solo a patto di peggiorare rispetto alla seconda (e viceversa).

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

- E' possibile determinare l'insieme delle soluzioni non-dominate di $1||\sum C_j, L_{\max}$ risolvendo diverse istanze del problema (sing. ob.) $1|L_{\max} \leq \delta|\sum C_j$. , con l'accortezza di schedulare prima il job con due-date inferiore nel caso di job con stesso tempo di processamento (regola per rompere la parità sui tempi di proc.).

- E' ovvio che per $\delta \geq \sum p_j$ il problema $1|L_{\max} \leq \delta|\sum C_j$ è equivalente a $1||\sum C_j$, per cui è inutile tentare di cercare soluzioni non-dominate con $L_{\max} > \sum p_j$.

Teorema

La schedula *SPT* S_{SPT} (con l'avvertenza di schedulare prima il job con due-date inferiore nel caso di job con stesso tempo di processamento) è *non-dominata*.

Dim.

Ovviamente la schedula S_{SPT} è la *migliore* rispetto a $\sum C_j$. Facciamo vedere che non è possibile trovare una schedula con $L_{\max} < L_{\max}(S_{SPT})$ senza aumentare $\sum C_j$. Per diminuire $L_{\max}(S_{SPT})$ dovremmo almeno scambiare di posto due job che hanno due-date differenti in modo da schedulare prima quello con due-date inferiore. Ma questo per come è costruita la schedula S_{SPT} (tenuto conto della regola in caso di parità) comporterebbe scambiare di posto job con differenti tempi di processamento con la conseguenza quindi di aumentare $\sum C_j$.

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

- Il teorema vale anche per la schedula ottima $S^*(\delta)$ di $1|L_{\max} \le \delta|\sum C_j$ (costruita tenendo conto della regola in caso di parità sui tempi di processamento).

- Cioè *non esiste* alcuna schedula S con:
  a) $\sum C_j(S) \le \sum C_j(S^*(\delta))$ e $L_{\max}(S) < L_{\max}(S^*(\delta))$, oppure
  b) $\sum C_j(S) < \sum C_j(S^*(\delta))$ e $L_{\max}(S) \le L_{\max}(S^*(\delta))$.

- Quindi per generare un'altra soluzione non-dominata occorre determinare la soluzione ottima di $1|L_{\max} \le L_{\max}(S^*(\delta)) - 1|\sum C_j$ che presenterà una diminuzione di $L_{\max}$ ma necessariamente un aumento di $\sum C_j$.

- Possiamo quindi usare il seguente algoritmo per generare tutte le schedule non-dominate di $1||\sum C_j, L_{\max}$.

Algoritmo per le schedule non-dominate

Step 0. Poni $\delta = \sum p_j$.

Step 1. Risolvi $1|L_{\max} \le \delta|\sum C_j$. Se non ammette soluzione STOP, altrimenti sia S^* la schedula ottima. (* S^* è non-dominata! *)

Step 2. Calcola $L_{\max}(S^*)$.

Step 3. Poni $\delta = L_{\max}(S^*) - 1$. Vai allo Step 1

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

Esempio: Si consideri la seguente istanza di $1||\sum C_j, L_{\max}$.

job	1	2	3	4	5
p_i	2	3	4	3	5
d_i	17	4	15	12	11

Iniziamo fissando $\delta = \sum p_j = 17$.
Risolviamo $1|L_{\max} \le \delta = 17|\sum C_j$. Le due-date d' sono:

job	1	2	3	4	5
d_i	34	21	32	29	28

Ovviamente per $\delta = 17$ è equivalente a $1||\sum C_j$. Applico quindi SPT (con regola per rompere la parità). Ottengo $S^*_1 = S_{SPT} = (1, 2, 4, 3, 5)$ con $\sum C_j(S^*_1) = 44$ e $L_{\max}(S^*_1) = 6$.

Fissiamo $\delta = L_{\max}(S^*_1) - 1 = 5$.
Risolviamo $1|L_{\max} \le \delta = 5|\sum C_j$. Le due-date d' sono:

job	1	2	3	4	5
d_i	22	9	20	17	16

Ottengo $S^*_2 = (1, 2, 4, 5, 3)$ con $\sum C_j(S^*_2) = 45$ e $L_{\max}(S^*_2) = 2$.

Fissiamo $\delta = L_{\max}(S^*_2) - 1 = 1$.
Risolviamo $1|L_{\max} \le \delta = 1|\sum C_j$. Le due-date d' sono:

job	1	2	3	4	5
d_i	18	5	16	13	12

Ottengo $S^*_3 = (2, 4, 5, 3, 1)$ con $\sum C_j(S^*_3) = 52$ e $L_{\max}(S^*_3) = 0$.

Modelli di scheduling su macchina singola

3.4. Regole di sequenziamento con deadline (continua)

Fissiamo $\delta = L_{max}(S^*_3) - 1 = -1$.

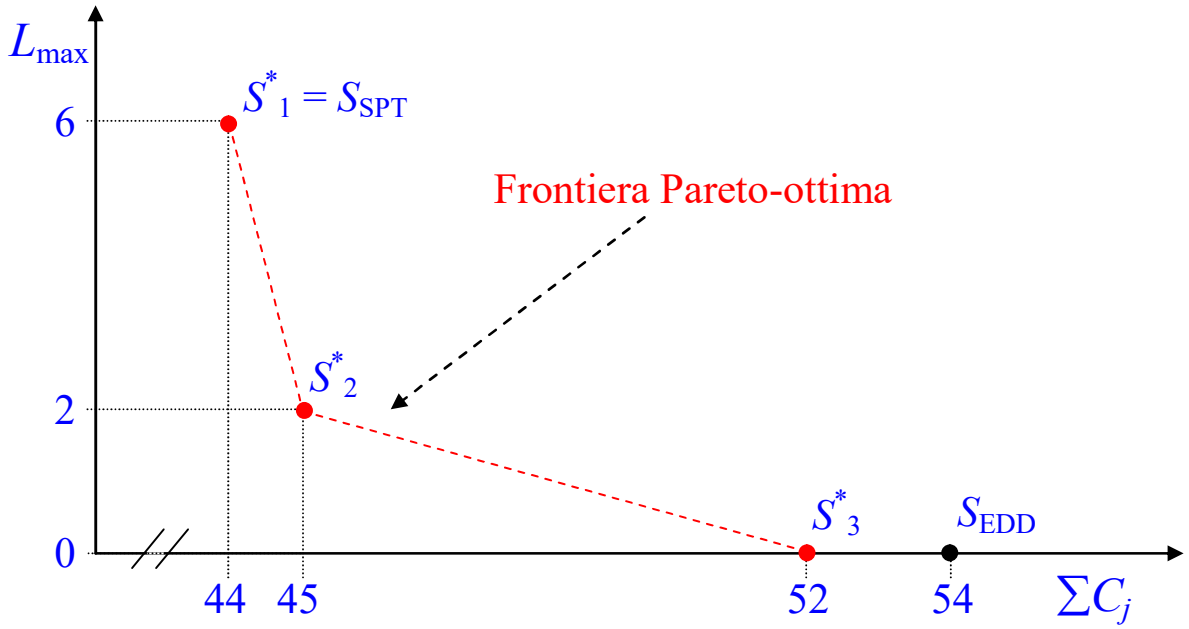
Risolviamo $1|L_{max} \leq \delta = -1|\sum C_j$. Le due-date d' sono:

job	1	2	3	4	5
d_i	16	3	14	11	10

Non ha soluzione. Stop.

Le schedule *non-dominate* sono 3: S^*_1, S^*_2, S^*_3.

La schedula *EDD* $S_{EDD} = (2, 5, 4, 3, 1)$ risulta invece dominata in quanto $L_{max}(S_{EDD}) = L_{max}(S^*_3) = 0$, ma $\sum C_j(S_{EDD}) = 54 > \sum C_j(S^*_3) = 52$.



Modelli di scheduling su macchina singola

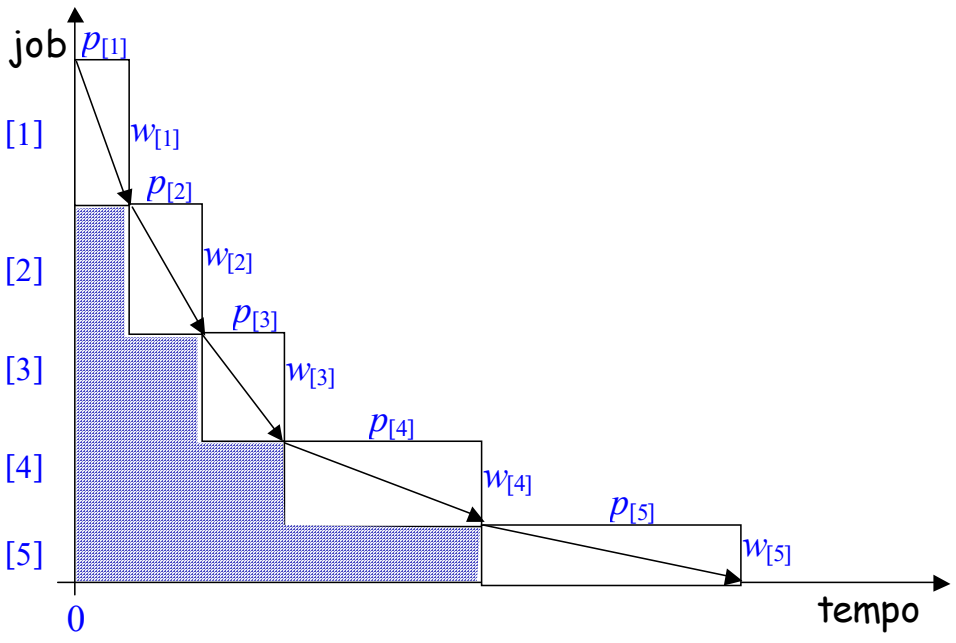
3.5. Sequenziamento con misure di prestazione pesate

- Quando i job non sono tutti ugualmente importanti ad ognuno di essi si associa un peso w_i che viene usato nella misura di prestazione.
- Il *flow-time medio pesato* è dato da

$$\overline{F}_w = \frac{1}{n}\sum_{i=1}^{n} w_i F_i \; ; \qquad \overline{F}'_w = \frac{\sum_{i=1}^{n} w_i F_i}{\sum_{i=1}^{n} w_i}$$

- Ricordiamo che minimizzare il flow-time medio pesato è equivalente a minimizzare il *tempo di completamento pesato totale* $\sum w_j C_j$ dei job.

- Volendo rappresentare una schedula, analogamente a quanto fatto nel caso non pesato, si può utilizzare la seguente figura in cui ogni rettangolo etichettato ha altezza proporzionale a w_i



Modelli di scheduling su macchina singola

3.5. Sequenziamento con misure di prestaz. pesate (cont.)

- Il *tempo di completamento pesato totale* $\sum w_j C_j$ dei job è rappresentato dall'area totale, di cui l'area dei blocchi è invariante con la sequenza mentre l'area tratteggiata è minimizzata ordinando i vettori rappresentativi (la cui pendenza è $-w_{[i]}/p_{[i]}$) in modo da formare una curva convessa.
  Ciò avviene ordinando i job secondo la regola

$$\frac{p_{[1]}}{w_{[1]}} \leq \frac{p_{[2]}}{w_{[2]}} \leq \ldots \leq \frac{p_{[n]}}{w_{[n]}}$$

- Più rigorosamente ciò può essere provato in base al seguente

Teorema (Smith 1956)

In un problema $1||\sum w_j C_j$, la minimizzazione di $\sum w_j C_j$ si ottiene sequenziando i job secondo la regola *Weighted Shortest Processing Time* (*WSPT*) ossia

$$\frac{p_{[1]}}{w_{[1]}} \leq \frac{p_{[2]}}{w_{[2]}} \leq \ldots \leq \frac{p_{[n]}}{w_{[n]}}$$

Modelli di scheduling su macchina singola

3.5. Sequenziamento con misure di prestaz. pesate (cont.)

Dimostrazione:

- Supponiamo per assurdo che esista una sequenza ottima S che non sia *WSPT*, e sia $z(S) = \sum_j w_j C_j(S)$.

- Pertanto in S ci devono essere almeno due job adiacenti, diciamo i e j, per i quali $p_i/w_i > p_j/w_j$.

- Assumiamo che il job i inizi al tempo t.

- Consideriamo la sequenza S' ottenuta da S scambiando i e j.

- $z(S') = z(S) - w_i(t+p_i) - w_j(t+p_i+p_j) + w_j(t+p_j) + w_i(t+p_j+p_i) =$
  $= z(S) + (w_i\, p_j - w_j\, p_i) < z(S)$, visto che $p_i/w_i > p_j/w_j$.

- Ciò contraddice l'ottimalità di S e prova il teorema.

c.v.d.

- La regola *WSPT* opera in $O(n \log n)$.

- La regola *WSPT* minimizza anche $\overline{C}_w$, $\overline{F}_w$, $\overline{L}_w$ e $\overline{W}_w$.

Modelli di scheduling su macchina singola

4.1 Altri problemi con due-date: min. numero job in ritardo

- *Nessuna* delle precedenti *regole risolve* in generale $1||\sum U_j$.

- Moore (1968) ha proposto il seguente algoritmo per $1||\sum U_j$:

  Fase 1. Costruisci una sequenza iniziale usando la regola EDD;
  Fase 2. Migliora la sequenza iniziale riarrangiando la posizione di alcuni job

- La Fase 2 opera nel seguente modo:
  2.1. Trova il primo job in ritardo nella sequenza corrente; sia ad esempio il job $[i]$. Se non ce n'è nessuno vai al passo 2.4;
  2.2. Rimuovi dalla sequenza corrente il job $[k]$ con tempo di processamento maggiore tra i job $[1]$, $[2]$, ..., $[i]$;
  2.3. Ritorna al passo 2.1;
  2.4. Costruisci una sequenza (ottima) aggiungendo alla sequenza corrente i job rimossi (in un qualsiasi ordine).

Modelli di scheduling su macchina singola

4.1 Altri problemi con due-date: min. numero job in ritardo

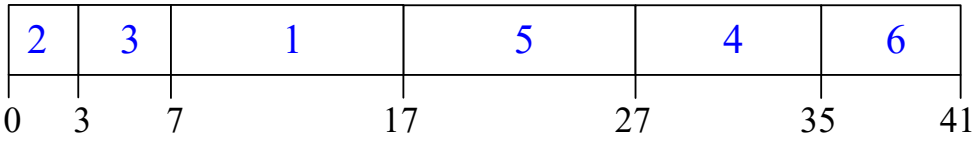
- Senza perdita di gener. consideriamo istanze con $p_j \le d_j$.

Esempio: Consideriamo la seguente istanza di $1||\sum U_j$.

job	1	2	3	4	5	6
d_j	15	6	9	23	20	30
p_j	10	3	4	8	10	6

- *Algoritmo di Moore*

Fase 1. Sequenza EDD iniziale: (2, 3, 1, 5, 4, 6)



$\sum U_j = 4$

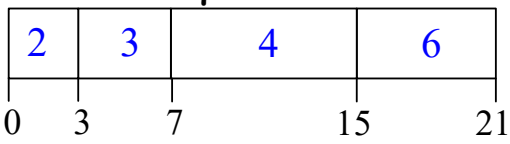
Fase 2.

$[i] = 1$; $[k] = 1$; rimuovo job $[k] = 1$;

sequenza corrente: (2, 3, 5, 4, 6)

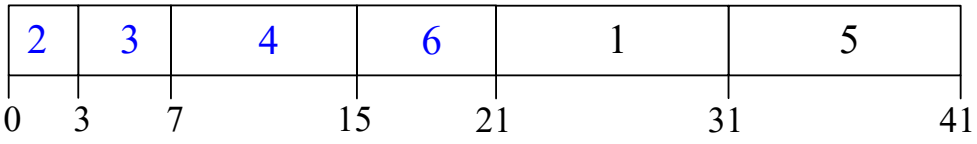
$[i] = 4$; $[k] = 5$; rimuovo job $[k] = 5$;

sequenza corrente: (2, 3, 4, 6)



nessun job in ritardo nella seq. corrente;

sequenza ottima: (2, 3, 4, 6; 1, 5)



$\sum U_j = 2$

Modelli di scheduling su macchina singola

4.1 Altri problemi con due-date: min. numero job in ritardo

- Versione *iterativa* dell'algoritmo di Moore:
  - Ad ogni iterazione si aggiunge in coda alla sequenza parziale il job con due-date più piccola;
  - Nel caso in cui il job aggiunto non termini entro la due-date si elimina dalla nuova sequenza parziale il job con tempo di processamento più lungo.
  - Al termine si aggiungono i job rimossi.
- Entrambe le versioni operano in $O(n^2)$.
- La seconda versione permette di provare l'ottimalità dell'algoritmo di Moore.

Teorema

- L'*Algoritmo di Moore* trova una soluzione ottima per $1||\sum U_j$

Dim:

- Senza perdita di generalità assumiamo $d_1 \leq d_2 \leq \ldots \leq d_n$.
- Sia S_k la sottosequenza della sequenza $(1, 2, \ldots, k)$ di job che soddisfa le seguenti due condizioni:
  1. S_k è formata dal massimo numero di job, diciamo N_k, completati entro la due-date, tra i job in $(1, 2, \ldots, k)$;
  2. Tra tutte le sottosequenze di $(1, 2, \ldots, k)$ per cui vale la (1), i job di S_k hanno il più piccolo tempo di processamento totale.
- Si noti che la sequenza $(S_n \oplus J \setminus S_n)$ è ottima.

Modelli di scheduling su macchina singola

4.1 Altri problemi con due-date: min. numero job in ritardo

- Si dimostra per induzione che l'algoritmo di Moore determina la sottosequenza S_n.
  a. E' semplice mostrare che l'algoritmo costruisce $S_1 = (1)$ se $p_1 \leq d_1$, altrimenti $S_1 = \varnothing$, soddisfacendo la (1) e la (2).
  b. Quindi supponendo che l'algoritmo ha determinato S_k, mostriamo che determina S_{k+1}.

- Occorre esaminare 2 casi:

  i. job $k+1$ è aggiunto a S_k ed è completato entro la due-date. Quindi per la sottosequenza $(S_k \oplus \{k+1\})$ vale la (1); inoltre siccome $(S_k \oplus \{k+1\})$ è sottosequenza di $(1, 2, \ldots, k+1)$ e per S_k vale la (2), questo assicura che la (2) continua a valere anche per $(S_k \oplus \{k+1\})$ che quindi può essere vista come S_{k+1}.

  ii. $k+1$ è aggiunto a S_k ma non è completato entro la due-date. Siccome per S_k valgono la (1) e la (2), ne segue che $N_{k+1} = N_k$ e quindi per la sottosequenza $(S_k \oplus \{k+1\})$ vale la (1); inoltre, rimuovendo da $(S_k \oplus \{k+1\})$ il job più lungo, sia questo il job j', non si altera il numero di job completati entro la due-date (in quanto $k+1$ terminerebbe in tempo visto che si avrebbe $C_{k+1} \leq d_k \leq d_{k+1}$ nel caso in cui il job eliminato j' fosse diverso da $k+1$), e, avendo rimosso il job j' più lungo, la sottosequenza $(S_k \oplus \{k+1\})\backslash\{j'\}$ rispetta anche la (2) che quindi può essere assunta come S_{k+1}. cvd.

- Il problema pesato $1||\sum w_j U_j$ è *NP-hard in senso ordinario*.
- Anche il caso particolare $1|d_j=d|\sum w_j U_j$ è *NP-hard in senso ordinario* in quanto è equivalente al KNAPSACK-(0/1).

Modelli di scheduling su macchina singola

4.2. Altri problemi con due-date: min. total tardiness

- *Nessuna* delle precedenti *regole minimizza* in generale $\overline{T}$.

- Si può dimostrare che $1||\sum T_j$ è *NP-hard in senso ordinario*, e che si può risolvere con un algoritmo di programmazione dinamica in tempo pseudo-polinomiale.

- Nel caso particolare in cui le due-date sono *concordi* con i tempi di processamento, cioè
  $$p_i \le p_j \quad \Rightarrow \quad d_i \le d_j$$
  il problema $1||\sum T_j$ è *risolvibile* in tempo polinomiale con la regola *SPT*.

- La generalizzazione del problema della minimizzazione della total tardiness con pesi w_j differenti fra job, cioè il problema $1||\sum w_j T_j$, si può dimostrare invece che è *NP-hard in senso forte*.

- Nel caso particolare di due-date e pesi *concordi* con i tempi di processamento, cioè
  $$p_i \le p_j \quad \Rightarrow \quad d_i \le d_j \text{ e } w_i \ge w_j$$
  il problema $1||\sum w_j T_j$ è *risolvibile* in tempo polinomiale con la regola *SPT*.

Modelli di scheduling su macchina singola

5. Tempi di set-up dipendenti dalla sequenza

- In molti casi reali il tempo di set-up di un job dipende dal job che lo procede sulla macchina.

  Esempio: produzione di colori diversi con una stessa macchina. E' intuitivo che ogni volta che si passa dalla produzione di un colore ad un altro la macchina deve essere pulita e che il tempo per eseguire questa operazione è strettamente legato ai colori che si succedono.
  Per questo caso di seguito sono indicati in ore i tempi di pulitura in una particolare configurazione:

precede \ segue	BIANCO	GIALLO	ROSSO	BLU
BIANCO	0	1	2	3
GIALLO	6	0	1	2
ROSSO	8	6	0	1
BLU	10	8	6	0

I cicli di produzione sono ad esempio:

sequenze	tempo di changeover
B - G - R - Bl - B	13
B - R - Bl - G - B	17
B - Bl - R - G - B	21
B - R - G - Bl - B	20

Modelli di scheduling su macchina singola

5. Tempi di set-up dipendenti dalla sequenza (continua)

- Poichè in questi casi i tempi di "*set-up*" non possono essere inglobati nei tempi di processamento, occorre introdurre una matrice $\{s_{ij}\}$ il cui generico termine s_{ij} rappresenta il tempo di "*changeover*" dal job i al job i.
- Spesso è conveniente introdurre un job *fittizio* (il job 0 di durata nulla) che occupa la posizione 0 nella sequenza per rappresentare lo stato preliminare "*idle*" della macchina.
- In questo modo il termine $s_{[0],[1]}$, con il job 0 in posizione 0 ($[0] \equiv 0$), rappresenterebbe il *tempo richiesto per portare la macchina da uno stato iniziale di non utilizzazione allo stato "ready" per il processamento del job* $[1]$.
- In questo modello il *tempo totale* necessario per il processamento di tutti i job non è costante come nel caso precedente ma *dipende dall'ordine dei job*.
- In particolare i*tempi di completamento dei job* sono dati:

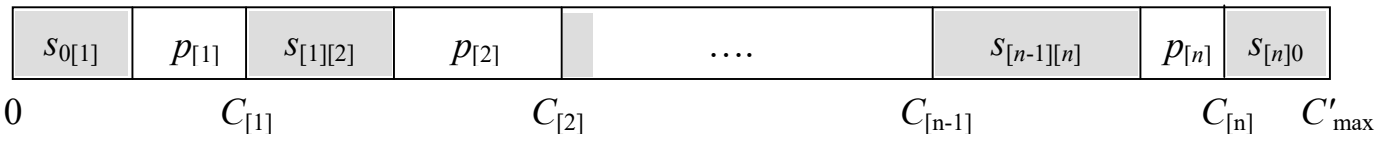
$$
\begin{aligned}
C_{[1]} &= s_{[0],[1]} + p_{[1]} \\
C_{[2]} &= C_{[1]} + s_{[1],[2]} + p_{[2]} \\
&\dots \\
C_{[i]} &= C_{[i\text{-}1]} + s_{[i\text{-}1],[i]} + p_{[i]}
\end{aligned}
$$

Modelli di scheduling su macchina singola

5. Tempi di set-up dipendenti dalla sequenza (continua)

- Ne consegue che il makespan C_{max} e' dato da:

$$C_{max} = C_{[n]} = \sum_{i=1}^{n} s_{[i-1],[i]} + \sum_{i=1}^{n} p_{[i]}$$



Poiché $\sum_{i=1}^{n} p_{[i]} = \text{cost}$, C_{max} è minimizzato *minimizzando* la *somma dei tempi di set-up*

- Si *associ* ad ogni *job* una *città* e supponiamo che le *distanze (temporali) fra le città corrispondano ai tempi di set-up fra i job*, allora il problema $1|s_{ij}|C_{max}$ (o $1|\text{seq-dep}|C_{max}$) è equivalente al problema del commesso viaggiatore (*TSP*) con $(n+1)$ città, dovendo considerare lo stato iniziale come il job 0 che assume il ruolo della città (deposito) 0, ed assumendo s_{ij} pari al tempo per andare dalla città i alla città j (p_j è il tempo di servizio (visita) della città j). Sia $c_{ij} = s_{ij} + p_j$ il costo per andare da i a j.

- Poiché la matrice S dei set-up è in generale asimmetrica, il *TSP*, a cui il problema di sequenziamento è riondotto, è del tutto generale (in particolare *asimmetrico*), quindi $1|s_{ij}|C_{max}$ è *NP-hard in senso forte*.

Modelli di scheduling su macchina singola

4. Tempi di set-up dipendenti dalla sequenza (continua)

- Se la misura di prestazione è $\sum C_j$ abbiamo:

$$\sum C_j = (s_{[0],[1]} + p_{[1]}) +$$
$$(s_{[0],[1]} + p_{[1]}) + (s_{[1],[2]} + p_{[2]}) +$$
$$\dots$$
$$(s_{[0],[1]} + p_{[1]}) + (s_{[1],[2]} + p_{[2]}) + \dots + (s_{[ni-1],[n]} + p_{[n]}) =$$
$$= n(s_{[0],[1]} + p_{[1]}) + (n-1)(s_{[1],[2]} + p_{[2]}) + \dots + (s_{[n-1],[n]} + p_{[n]}).$$

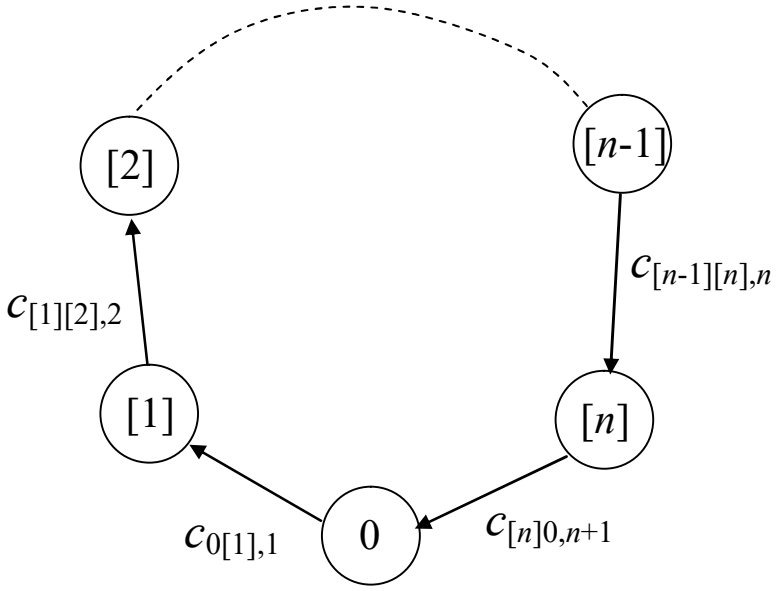
- Il problema $1|s_{ij}|\sum C_j$ è ancora *NP-hard in senso forte* ed è equivalente al *Cumulative-TSP* (o *Delivery Man Problem*) in cui il costo di percorrenza del primo arco del tour $c_{[0],[1]}$ è pesato n volte, quello del secondo arco è pesato $(n-1)$ volte, …, quella del penultimo arco è pesato 1 volta e quella dell'ultimo arco, $(n+1)$-esimo, è pesato 0 volte, dove l'arco (i,j) ha costo $c_{ij} = (s_{ij} + p_j)$.

- Il *C-TSP* è un particolare caso del ***Time Dependent-TSP***.

Modelli di scheduling su macchina singola

4. Tempi di set-up dipendenti dalla sequenza (continua)

- Nel *TD-TSP* il costo di percorrenza di un arco (i, j) è funzione anche della posizione in cui l'arco compare nel ciclo hamiltoniano.
  $c_{ij,k}$: costo dell'arco (i, j) se è il k-mo arco del ciclo.



- Nel caso del *C-TSP*: $c_{ij,k} = (n - k + 1)c_{ij}$.

- Il *TD-TSP* è formulabile come *Problema dello Assegnamento Quadratico* (*QAP*)

- *Definizione del QAP*
  Dati n dipartimenti e n siti, e noto il costo d_{ikjl} dell'assegnamento del dip. i al sito k e del dip. j al sito l, il problema consiste nell'assegnare ciascun dipartimento esattamente ad un sito in modo tale che ciascun sito accolga esattamente un dipartimento, minimizzando il costo totale degli assegnamenti.

Modelli di scheduling su macchina singola

4. Tempi di set-up dipendenti dalla sequenza (continua)

- *Formulazione del QAP*

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{l=1}^{n} d_{ikjl}\ x_{ik} \cdot x_{jl}$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \text{per ogni dip. } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \text{per ogni sito } k = 1, \ldots, n$$

$$x_{ik} \in \{0, 1\}, \quad i = 1, \ldots, n; \quad k = 1, \ldots, n$$

- Per esempio, se il costo dell'assegnamento è associato alla movimentazione delle merci tra coppie di dipartimenti allora $d_{ikjl} = w_{ij} \cdot \delta_{kl}$, dove w_{ij} è il volume delle merci scambiate tra il dip. i e il dip. j e δ_{kl} è il costo unitario di movimentazione che dipende dalla coppia dei siti k ed l che ospitano rispettivamente i dip. i e j.

- Il *TD-TSP* è equivalente al *QAP*, assumendo $d_{ikjl} = 0$, tranne che:

  $d_{i1i1} = c_{0i,1}$ e $d_{inin} = c_{i0,n+1}$, per ogni città $i = 1, \ldots, n$;
  $d_{i(k-1)jk} = c_{ij,k}$, per ogni coppia di città $i, j = 1, \ldots, n$ $(i \neq j)$ e per ogni posizione $k = 2, \ldots, n$.

  N.B.: E' possibile formulare anche il *TSP* come *QAP* considerando ovviamente il *TSP* come caso particolare del *TD-TSP* (assumendo cioè $c_{ij,k} = c_{ij}$) e poi formulando *TD-TSP* come *QAP* in base a quanto sopra riportato.

Modelli di scheduling su macchina singola

4. Tempi di set-up dipendenti dalla sequenza (continua)

- *Altra (migliore) formulazione del TD-TSP*

$$\min \sum_{j=1}^{n} c_{0j,1}\, x_{j1} + \sum_{i=1}^{n} \sum_{j=1,\neq i}^{n} \sum_{k=2}^{n} c_{ijk}\, y_{ijk} + \sum_{i=1}^{n} c_{i0,n+1}\, x_{in}$$

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

$$\sum_{j=1}^{n} x_{jk} = 1, \qquad k = 1, \ldots, n \qquad (*)$$

$$x_{ik} = \sum_{j=1,\neq i}^{n} y_{ij,k+1}, \quad i = 1, \ldots, n;\ k = 1, \ldots, n-1$$

$$x_{jk} = \sum_{i=1,\neq j}^{n} y_{ij,k}, \quad j = 1, \ldots, n;\ k = 2, \ldots, n$$

$$x_{ik} \in \{0, 1\}, \qquad i = 1, \ldots, n;\ k = 1, \ldots, n$$

$$y_{ijk} \in \{0, 1\}, \qquad i = 1, \ldots, n;\ j \neq i = 1, \ldots, n;\ k = 2, \ldots, n$$

- Le variabili $x_{ik} \in \{0, 1\}$, $i, k = 1, \ldots, n$, sono le variabili di assegnamento città (job)/posizione nel tour.
- Le variabili $y_{ijk} \in \{0, 1\}$, $i, j = 1, \ldots, n$ $(i \neq j)$ e $k = 2, \ldots, n$, rappresentano la percorrenza o meno dell'arco (i, j) come k-mo arco del tour.
- Si dimostra che il vincolo (*) è ridondante.

N.B.: Anche in tal caso è possibile formulare il *TSP* come caso particolare del *TD-TSP* (assumendo cioè $c_{ij,k} = c_{ij}$).

Modelli di scheduling su macchina singola

5. *Sequenziamento con "ready time"*

- Se i job non sono tutti simultaneamente disponibili per il processamento ($r_i \neq 0$) le "*schedule di permutazione*" non sono più sufficienti per trovare la soluzione ottima.
- Occorre infatti considerare anche la possibilità di "*preempzione*" e di "*idle-time*".
- In tal caso una sequenza di n numeri interi non è sufficiente per descrivere una schedula di n job poiché un singolo job può comparire più volte nella schedula ogni volta con tempi diversi.

- Esaminiamo le differenti *tipologie* di "*preempzione*" in relazione alla situazione in cui il job interrotto si viene a trovare quando ritorna alla macchina.

  Esaminiamo due casi estremi:

  (i) *Preempt-resume*. Il *job interrotto viene di nuovo processato per il tempo residuo* senza la necessità di un extra-lavoro della macchina.

  (ii) *Preempt-repeat*. Il *job interrotto* perde il beneficio del processamento parziale e *viene riprocessato integralmente ogni volta che viene riconsiderato*.

Modelli di scheduling su macchina singola

5. Sequenziamento con "ready time" (continua)

- Una disciplina intermedia di "preempzione" è quella per cui si ha:
  - *Preempt-resume* *per il tempo di processamento vero e proprio*
  - *Preempt-repeat* *per il tempo di set-up*

- Per spiegare l'*utilità* della "*preempzione*" consideriamo il seguente esempio:

job	1	2
r_i	0	1
p_i	4	1

a) In *assenza di preempzione* la schedula $(1, 2)$ ha

$$\overline{F} = (F_{[1]} + F_{[2]})/2 = (4 + 4)/2 = 4$$

b) Se si ammette una *interruzione preempt-resume* del job 1 quando arriva il job 2 allora si ha

$$\overline{F} = (1 + 5)/2 = 3$$

c) Se si ammette una *interruzione preempt-repeat* del job 1 quando arriva il job 2 allora si ha

$$\overline{F} = (1 + 6)/2 = 3.5$$

Modelli di scheduling su macchina singola

5. Sequenziamento con "ready time" (continua)

Osservazione 1

La situazione con *ready time* ed una disciplina *preempt - resume* può essere gestita con la stessa strategia usata nel caso di arrivi simultanei.
Infatti *al momento dell'arrivo di un job* si può rivedere la *decisione*, considerando soltanto i *job presenti* in quel momento nello shop con tempi di processamento pari ai *tempi di processamento residui* dei job stessi.

a) Nel caso in cui l'obiettivo sia la minimizzazione di $\overline{F}$, la scelta migliore in quel momento è data dalla regola *SPT*.

b) Nel caso in cui l'obiettivo sia la minimizzazione di $L_{\max}$, la scelta migliore in quel momento è data dalla regola *EDD*.

Osservazione 2

Pertanto nel caso *prempt-resume* la regola generale nel sottocaso a) è la *versione "interrompibile"* della *SPT*, cioè la *Shortest Remaining Processing Time* (*SRPT*) *rule*; nel sottocaso b) è la *versione "interrompibile"* della *EDD*, cioè la *Preemptive Earliest Due Date* (*PEDD*) *rule*.

La *SRPT risolve* il problema $1|\, r_j,\, \text{pmtn}|\, \Sigma\, C_j$.

La *PEDD risolve* il problema $1|\, r_j,\, \text{pmtn}|\, L_{\max}$.

Non è necessario conoscere in anticipo i job in arrivo. Se questi sono noti la complessità è $O(n \log n)$.

Modelli di scheduling su macchina singola

5. Sequenziamento con "ready time" (continua)

Osservazione 3

La disciplina *preempt-repeat differisce* dalla disciplina *preempt-resume* anche in relazione alla circostanza che la *conoscenza anticipata del job in arrivo ha un effetto sulla schedula*.

Se infatti ci si aspetta l'arrivo di un job che provoca un'interruzione del job in esecuzione, potrebbe risultare più conveniente aspettare e lasciare la macchina libera per un certo tempo.

Nel precedente esempio la sequenza $(2,\ 1)$ senza interruzioni ha $\overline{F} = 3{,}5$ con una "idle time" iniziale della macchina pari ad 1.

Note conclusive sulla complessità computazionale

- $1|\, r_j|\, \Sigma\, w_j\, C_j$ — NP-hard in senso forte, anche nel caso con $w_j = 1$
- $1|\, r_j,\, \text{pmtn}|\, \Sigma\, C_j$ — trattabile (SRPT rule)
- $1|\, r_j,\, \text{pmtn}|\, \Sigma\, w_j\, C_j$ — NP-hard in senso forte
- $1|\, r_j\,|\, L_{\max}$ — NP-hard in senso forte
- $1|\, r_j,\, \text{pmtn}|\, L_{\max}$ — trattabile (PEDD rule)

Modelli di scheduling su macchina singola

6. Sequenziamento con precedenze fra i job

- In molti casi per ragioni tecnologiche o di altro tipo esistono *relazioni di precedenza tra i job* da processare.

(i) *Stringhe di job*

- Assumiamo che gli n *job* di un insieme siano *raggruppati* in k *stringhe* con $n_1, n_2, \ldots, n_k$ job corrispondenti.
- Supponiamo anche che i *job* delle varie *stringhe* siano *fissati*, che sia *fissato* il loro *ordine* e che ogni stringa debba essere processata integralmente (*stringa non-preemptiva*).
- Il problema da risolvere è quello di decidere *quale è il migliore* dei $k!$ modi in cui le k stringhe possono essere sequenziate.
- Definiamo pertanto:

  p_{ij} *tempo* di *processamento* del *job* j-mo della *stringa* i-ma.

  C_{ij} *tempo di completamento* del *job* j-mo della *stringa* i-ma.

  p_i' *tempo* di *processamento* della *stringa* i-ma, pari a $\sum_{j=1}^{n_i} p_{ij}$.

  C_i' *tempo di completamento* della *stringa* i-ma, cioè il tempo di completamento C_{i,n_i} dell'ultimo job della stringa.

Modelli di scheduling su macchina singola

6. Sequenziamento con precedenze fra i job (continua)

(i) Stringhe di job

- La *regola* di sequenziamento che *minimizza* $\Sigma C'_i$ è la regola *SPT* applicata alle *stringhe* ossia

$$p'_{[1]} \leq p'_{[2]} \leq \ldots \leq p'_{[k]}$$

- Se la funzione da minimizzare è il tempo di completamento totale pesato delle stringhe con il peso della stringa i pari al numero n_i dei suoi job,

$$\sum_{i=1}^{k} n_i C'_i$$

allora *l'ordinamento ottimo* è dato dalla regola *WSPT* applicata alle *stringhe* ossia

$$\frac{p'_{[1]}}{n_{[1]}} \leq \frac{p'_{[2]}}{n_{[2]}} \leq \ldots \leq \frac{p'_{[k]}}{n_{[k]}}$$

- Facciamo vedere che questa *regola* è *valida* anche per *minimizzare* ΣC_j cioè il *tempo di completamento totale* dell'insieme dei *job*.

Teorema

In un problema di scheduling in cui vi sono k stringhe non preemptive di job con l'obiettivo di minimizzare il tempo di completamento totale dei job $(1|\text{string}|\Sigma C_j)$, la soluzione ottima si ottiene ordinando le stringhe secondo la seguente regola:

$$\frac{p'_{[1]}}{n_{[1]}} \leq \frac{p'_{[2]}}{n_{[2]}} \leq \ldots \leq \frac{p'_{[k]}}{n_{[k]}}$$

Modelli di scheduling su macchina singola

6. *Sequenziamento con precedenze fra i job (continua)*

(i) *Stringhe di job*

Dimostrazione

Sia h_{ij} il tempo che intercorre tra il completamento del job j-mo della stringa i-ma ed il completamento di tutta la stringa.

Ovviamente $h_{i,n_i} = 0$ e $h_{ij} = \sum_{q=j+1}^{n_i} p_{iq}$, $j = 1, 2, \ldots, n_{i-1}$.

Per ogni job, h_{ij} è una costante data dalla definizione del problema, ossia dai tempi di processamento e dall'ordine dei job entro la stringa. Valgono quindi:

$$C_{ij} = C'_i - h_{ij}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{n_i} C_{ij} = \sum_{i=1}^{k} n_i C'_i - \sum_{i=1}^{k}\sum_{j=1}^{n_i} h_{ij}$$

L'ultimo termine è una costante che non dipende dall'ordinamento preassegnato delle stringhe. Il primo termine invece è minimizzato dall'ordinamento non decrescente dell'enunciato.

Modelli di scheduling su macchina singola

6. *Sequenziamento con precedenze fra i job (continua)*

(ii) *Catene di job*

- Esaminiamo il caso di relazioni di precedenza (diretta o meno) tra i job che non implichino l'esistenza di stringhe non preemptive.
- Per capire la differenza consideriamo un insieme di tre job: $\{a, b, c\}$.

a) In assenza di vincoli di precedenza le possibili sequenze sono $3!$, cioè 6:

1) a, b, c; 2) a, c, b; 3) b, a, c;
4) c, a, b; 5) b, c, a; 6) c, b, a.

b) Se b e c costituissero la stringa (b, c) allora le sequenze possibili sarebbero solo 2:

1) a, b, c; 2) b, c, a.

c) Se invece si richiedesse soltanto che $b < c$ allora le possibili sequenze sarebbero 3:

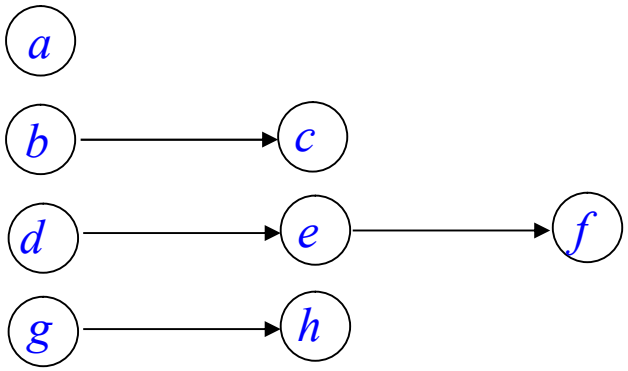
1) a, b, c; 2) b, a, c; 3) b, c, a;

Modelli di scheduling su macchina singola

6. *Sequenziamento con precedenze fra i job (continua)*

(ii) *Catene di job*

- Consideriamo ora il caso di *vincoli di precedenza* che *partizionano* gli n job *in* k *catene*, circostanza che si verifica se ogni job ha al più un predecessore (diretto) ed un successore (diretto) come nel seguente esempio



Teorema

In un problema di scheduling in cui le relazioni di precedenza sono tali da partizionare i job in k sottoinsiemi disgiunti nei quali l'ordine è specificato (*catene*) ma che possono essere interrotte tra i job, ed il cui obiettivo è minimizzare il tempo di flusso medio ($1|\text{chain}|\Sigma C_j$), la soluzione ottima si ottiene ordinando le stringhe secondo la seguente procedura:

Modelli di scheduling su macchina singola

6. Sequenziamento con precedenze fra i job (continua)

(ii) *Catene di job*

1) Per ogni job j nella catena i si calcola $x_{ij} = \dfrac{\sum_{h=1}^{j} p_{ih}}{j}$
2) Per ogni catena i si calcola $y_i(h_i) = \min \{x_{i1}, \ldots, x_{in_i}\}$ ove h_i è il valore del secondo indice del $\min \{\ldots\}$
3) Sia I la catena per cui vale $y_I(h_I) \leq y_i(h_i), \quad \forall\ i$, e si pongono i primi h_I job all'inizio della sequenza sulla macchina.
4) Si eliminano i primi h_I job della catena I e si ricalcola y_I.
5) Si ripetono i passi 3) e 4) finché tutti i job sono sequenziati.

Dimostrazione

La procedura produce e ordina $K \geq k$ sottocatene, e, non considerando la seconda sottocatena di una data catena finché alla prima non è assegnata una posizione nella sequenza, assicura il rispetto delle precedenze esistenti nella catena originaria.

Assumendo che queste sottocatene siano stringhe e quindi assumendo che non possano essere interrotte, allora, in base al teorema di ordinamento ottimo delle stringhe per la minimizzazione di ΣC_j (secondo cui l'ordinamento ottimo delle stringhe è quello non decrescente dei rapporti tra il tempo di processamento della stringa e il numero dei job della stringa stessa) si deduce che la procedura fornisce un sequenziamento ottimo.

D'altra parte non c'è nessun vantaggio ad interrompere le sottocatene costruite dalla procedura, che pertanto si comportano come le stringhe.

Modelli di scheduling su macchina singola

6. Sequenziamento con precedenze fra i job (continua)

(iii) Relazioni di precedenza generali

- Supponiamo che le relazioni di precedenza fra i job siano del tutto generali.

- Consideriamo il caso abbastanza generico in cui la valutazione di ciascun job j sia effettuata attraverso un'arbitraria funzione di costo $\gamma_j(C_j)$ che sia *non decrescente* all'aumentare di C_j.

- Il problema che vogliamo risolvere è $1|\text{prec}|\ \max_j\{\gamma_j(C_j)\}$. Si noti che la funzione obiettivo $\max_j\{\gamma_j(C_j)\}$ è *regolare*.

- Ad esempio $L_{\max}$ è un caso particolare di $\max_j\{\gamma_j(C_j)\}$, ponendo $\gamma_j(C_j) = L_j = C_j - d_j$; analogamente per $T_{\max}$.

- Lawler (1973) ha proposto il seguente algoritmo di sequenziamento che opera costruendo la sequenza dal fondo:

Algoritmo di Lawler

$\hat{C} = \Sigma_j\, p_j;\ \hat{J} = J$

for $i = n$ to 1 do

$X = \{j \in \hat{J} \mid Succ(j) \cap \hat{J} = \varnothing\}$

$[i] = j^* = \text{argmin}_{j \in X}\ \{\gamma_j(\hat{C})\}$

$\hat{J} = \hat{J} \setminus \{j^*\}$

$\hat{C} = \hat{C} - p_{j^*}$

Modelli di scheduling su macchina singola

6. Sequenziamento con precedenze fra i job (continua)

(iii) *Relazioni di precedenza generali*

Teorema

- L'*Algoritmo di Lawler* trova una soluzione ottima per $1|\text{prec}|\ \max_j\{\gamma_j(C_j)\}$

Dim:

- Tra tutti i job j che non hanno successori ($Succ(j) = \varnothing$), sia $j^* = \text{argmin}_j\{\gamma_j(\Sigma_j\ p_j)\}$. Dimostriamo che nella sequenza ottima j^* occupa l'ultima posizione.
- Si consideri una sequenza ammissibile S con $k \neq j^*$ come ultimo job; sia ad esempio $S = (A,\ j^*,\ B,\ k)$, dove A e B rappresentano due sottosequenze.
- Consideriamo la sequenza S' ottenuta da S spostando j^* in ultima posizione; cioè $S' = (A, B, k, j^*)$.
- Anche S' è ammissibile in quanto per ipotesi j^* non ha successori.
- Inoltre, passando da S a S' la schedulazione dei job nella sottosequenza A non varia, mentre quelli della sottosequenza B e il job k subiscono una riduzione del loro tempo di completamento pari a p_{j^*}. Il job j^* è l'unico a subire un aumento del tempo di completamento; tuttavia, siccome $\gamma_{j^*}(\Sigma_j\ p_j) \leq \gamma_k(\Sigma_j\ p_j)$, il costo associato a j^* in S' non è più grande del costo di k in S, e quindi $\max_j\{\gamma_j(C_j)\}$ non aumenta passando da S a S'.
- Pertanto esiste senz'altro una sequenza ottima con j^* in ultima posizione. Ripetendo la discussione sull'insieme dei job che si ottiene cancellando j^*, e così via, si dimostra il teorema. c.v.d

Modelli di scheduling su macchina singola

6. Sequenziamento con precedenze fra i job (continua)

(iii) *Relazioni di precedenza generali*

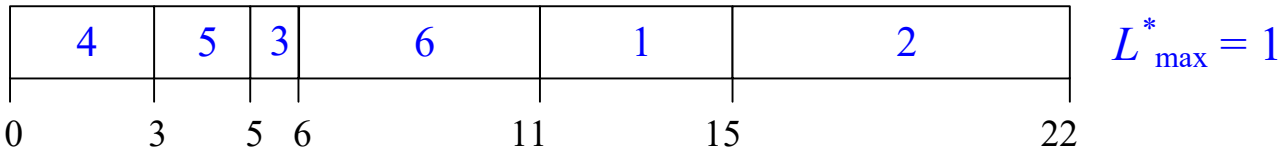
- L'*Algoritmo di Lawler* opera in tempo $O(n^2)$, in quanto esegue n iterazioni e ad ogni iterazione confronta tra loro $O(n)$ valori.

Esempio: Consideriamo la seguente istanza del problema $1|\text{prec}|L_{\max}$

job	1	2	3	4	5	6
d_i	24	21	8	5	10	23
p_i	4	7	1	3	2	5

con le precedenze: $1 < 2;\ 3 < 1;\ 4 < 3, 6;\ 5 < 3, 6;\ 6 < 1$.

Applichiamo l'algoritmo di Lawler, con $\gamma_j(\hat{C}) = \hat{C} - d_j$. Il job 2 è l'unico a non avere successori; poniamo il job 2 in posizione 6. Il job 1 è l'unico a non avere successori tra i job non sequenziati; poniamo il job 1 in posizione 5. I job 3 e 6 sono gli unici a non avere successori tra i job non sequenziati; poniamo il job 6 in posizione 4 visto che $\gamma_6(\hat{C}) \le \gamma_3(\hat{C})$. Il job 3 è l'unico a non avere successori tra i job non sequenziati; poniamo il job 3 in posizione 3. I job 4 e 5 sono gli unici a non avere successori tra i job non sequenziati; poniamo il job 5 in posizione 2 visto che $\gamma_5(\hat{C}) \le \gamma_4(\hat{C})$. Resta infine solo il job 4 che poniamo in posizione 1.



$L^*_{\max} = 1$

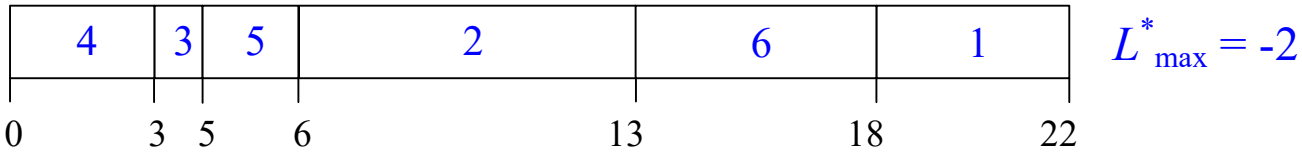
Modelli di scheduling su macchina singola

6. *Sequenziamento con precedenze fra i job (continua)*

(iii) *Relazioni di precedenza generali*

- Applicato al problema $1||L_{\max}$, visto che non ci sono precedenze fra i job, l'algoritmo di Lawler può essere semplificato, dovendo selezionare ogni volta il job con costo $\gamma_j(\hat{C}) = \hat{C} - d_j$ più piccolo tra i job non ancora sequenziati (in fondo alla sequenza). Ciò corrisponde a determinare una sequenza che letta da sinistra verso destra è la sequenza EDD. Ovviamente in tal caso l'algoritmo può determinare la soluzione in $O(n \log n)$ supponendo di ordinare preventivamente i job in ordine non decrescente della data di consegna.

- Riconsiderando l'istanza del precedente esempio però per il problema $1||L_{\max}$, l'applicazione dell'algoritmo di Lawler avrebbe prodotto la sequenza EDD.



$L^*_{\max} = -2$