UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI MATEMATICA "TULLIO-LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Algoritmi di scheduling e anomalie di timing nel multiprocessing

Relatore:

Prof. Marco Di Summa

Laureanda:

Agata Garbin

Matricola: 1192187

---

21 aprile 2022

Anno Accademico 2021/2022

Indice

Introduzione **5**

1 Introduzione allo scheduling **7**
1.1 Tipologie di problemi e notazioni 8
1.2 Vincoli e criteri di ottimalità 10
1.3 Nozioni fondamentali 11
1.4 Regole di scheduling 13

2 Algoritmi di scheduling **15**
2.1 Singola macchina soggetta a vincoli di precedenza 15
2.1.1 Algoritmo di Lawler 16
2.1.2 Scadenze e vincoli di precedenza 17
2.2 Minimizzare il numero di *late jobs* 18
2.2.1 Algoritmo di Moore 18
2.2.2 Sviluppo teorico dei risultati 20
2.3 Schedulazione preventiva con scadenze 24
2.3.1 Algoritmo di Sahni 26
2.4 Produzione a due fasi con tempi di installazione 27
2.4.1 Algoritmo di Johnson 30

3 Anomalie di timing nel multiprocessing **31**
3.1 Descrizione del sistema ed esempi di anomalie 31
3.2 Il *bound* generale 33
3.3 Il caso in cui $\prec$ è *vuoto* 35

A Algoritmi di Horn **43**
A.1 Minimizzare il ritardo massimo 43
A.2 Minimizzare il ritardo totale 44

Bibliografia **47**

Introduzione

A partire dagli anni '50 del secolo scorso, la teoria della schedulazione ha attirato sempre più l'attenzione degli studiosi. I problemi, prima solo teorici, hanno iniziato a trovare applicazione nell'industria manifatturiera, aumentando in complessità.

Schedulare significa assegnare attività a risorse nel tempo, dove le risorse sono processori, impiegati, macchine, mentre le attività possono essere considerate come fasi di un determinato progetto. Una schedulazione ci mostra un piano temporale per determinate attività e risponde alla domanda "Se tutto andrà bene, quando avverrà un determinato evento?". Supponiamo di essere interessati a sapere quando partirà un certo autobus dalla stazione. In questo caso, vorremmo conoscere il tempo d'inizio di una determinata attività (il viaggio del bus). Sapendo che il bus partirà subito dopo essere stato pulito e preparato, saremmo invece interessati al tempo di completamento delle attività che precedono la partenza del bus. In generale, intendiamo lo *scheduling* come il processo di creazione di schedulazioni. La teoria della schedulazione si occupa di ottimizzarlo.

Esistono vari metodi per la risoluzione di problemi di *scheduling* in base alla complessità della situazione considerata. L'obiettivo principale è ridurre i costi e i tempi necessari. Per pianificazioni nel lungo termine spesso si cerca di minimizzare i costi relativi agli investimenti per i materiali, il personale, la pubblicità, etc, mentre nel breve termine si considerano i costi relativi allo stoccaggio, al recupero delle materie prime necessarie e via dicendo. Per quanto riguarda i tempi nel breve termine si considerano generalmente le scadenze necessarie per soddisfare il cliente, i costi relativi al tempo necessario per preparare le macchine o al tempo in cui le macchine sono ferme (nel caso in cui esistano ad esempio dei vincoli di precedenza tra le varie attività).

Nel primo capitolo introdurremo le nozioni di base sulla teoria della schedulazione, facendo riferimento alle varie tipologie di problemi e ai relativi vincoli, ai criteri di ottimalità e alle regole principali per lo *scheduling*.

Nel secondo capitolo presenteremo una serie di algoritmi di *scheduling*, riguardanti in particolare i vincoli di precedenza, il controllo del numero di *late jobs*, la schedulazione preventiva, i tempi di installazione relativamente ad un problema a più fasi.

Nel terzo e ultimo capitolo vedremo infine alcuni risultati relativi alle anomalie temporali che si possono presentare in un problema con più processori. Vedremo un *bound* generale per tali anomalie e un caso particolare in cui le varie attività non sono vincolate ad un preciso ordinamento temporale.

Capitolo 1

Introduzione allo scheduling

Lo *scheduling*, citando la definizione proposta da Pinedo nel 1995,[1] è l'assegnazione di risorse limitate a determinate attività nel tempo. È un processo decisionale che ha l'obiettivo di ottimizzare uno o più "oggetti". Le varie componenti di un problema di scheduling sono i potenziali vincoli, le risorse, le funzioni obiettivo e naturalmente i lavori da processare.[2] Le entità da schedulare sono i lavori, le attività.

Se tutti i lavori consistono di una sola operazione parliamo di problemi *mono-operation*, altrimenti parliamo di problemi *multi-operation*. Le varie operazioni connesse ad un lavoro possono a loro volta essere collegate da vincoli di precedenza. In questo caso l'insieme di operazioni di un lavoro e i loro vincoli definiscono il cosiddetto *routing* del lavoro considerato.

Le "risorse" che eseguono le varie attività possono essere rinnovabili, come nel caso di macchine, processori, file, personale, etc., o non rinnovabili, come nel caso del denaro. Tra le risorse rinnovabili distinguiamo poi le risorse *disgiuntive*, che possono eseguire una operazione per volta, e le risorse *cumulative* che possono invece eseguire un numero (limitato) di operazioni simultaneamente.

Possiamo dividere i vari criteri secondo cui ottimizzare i diversi problemi di scheduling in due tipi: quelli relativi ai tempi e quelli relativi ai costi. Per quanto riguarda i tempi, troviamo criteri che misurano il tempo di completamento, altri che misurano il ritardo massimo dei lavori in relazione alla loro data di scadenza, etc. Per quanto riguarda i costi invece, troviamo criteri legati ai costi d'uso di una macchina o ai costi riguardanti i tempi d'attesa di una macchina tra un'operazione e l'altra, etc.

Esistono diverse aree di applicazione per i problemi di questo tipo. Li troviamo nei cosiddetti *sistemi di produzione flessibile*, formati da macchine controllate da computer, un sistema di trasporto automatizzato e un sistema di controllo; ma anche in ambienti chimici o elettrici, in cui diversi elementi che condividono la stessa area devono essere suddivisi in cisterne o spostati da un posto ad un altro. O ancora nella produzione di automobili, in cui determinati optional devono essere assemblati in vari tipi di veicoli e modelli, o nei sistemi informatici in cui diverse operazioni devono essere eseguite seguendo vincoli di sincronizzazione e scadenze.

[1] M. Pinedo. *Scheduling - Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs, 1995

[2] J. Carlier and P. Chretienne. *Problemes d'ordonnancement: modelisation /complexite / algorithmes.* Masson, Paris, 1988

Per poter trattare tutti i tipi di problemi di scheduling, data la vastità delle variabili in gioco, si ricorre ad una notazione semplice, ma formale. Denotiamo con n il numero di lavori da schedulare, con J_i (o anche T_i, da *task*) il lavoro i-esimo, con n_i il numero di operazioni del lavoro J_i, con $O_{i,j}$ l'operazione j-esima del lavoro J_i, con m il numero di macchine disponibili per il processamento dei lavori e con M_k la macchina k-esima. Infine chiamiamo schedulazione uno specifico ordinamento $(J_{i_1}, \ldots, J_{i_n})$ dell'insieme dei lavori da processare.

Nei prossimi capitoli, in cui analizzeremo vari problemi con i relativi algoritmi, verrà ripresa la notazione di base ed eventualmente ampliata con le informazioni necessarie alla discussione.

1.1 Tipologie di problemi e notazioni

Vediamo ora un primo modo di classificare i diversi problemi di scheduling.

Problemi di scheduling senza assegnazione

In questo tipo di problemi l'obiettivo è di trovare una sequenza per ogni macchina e un tempo d'inizio per ogni operazione. Esistono vari modelli:

- **macchina singola**: solo una macchina è disponibile per eseguire le operazioni. I lavori da processare sono necessariamente *mono-operation*.
- **flowshop** (F): più macchine sono disponibili per l'eleborazione dei lavori. La caratteristica di questo modello è che i lavori vengono processati dalle varie macchine tutti nello stesso ordine.
- **jobshop** (J): più macchine sono disponibili per l'elaborazione e ogni lavoro viene processato dalle varie macchine in un ordine preciso.
- **openshop** (O): più macchine sono disponibili per l'elaborazione e i lavori non seguono un ordine preciso per il processamento, ma possono essere assegnati alle macchine in qualsiasi ordine.
- **mixed shop** (X): più macchine sono disponibili e alcuni lavori seguono un ordine per il processamento, mentre altri sono liberi.

Problemi di scheduling e assegnazione in più fasi

In generale supponiamo che le macchine siano raggruppate in varie fasi ben definite e che ogni macchina appartenga ad una sola di queste. Le macchine di una stessa fase sono tutte in grado di eseguire le medesime operazioni. Il problema in questo caso è duplice: assegnare una macchina ad ogni operazione e sequenziare le varie operazioni nelle diverse macchine. Per ogni fase possiamo distinguere le seguenti configurazioni:

- le macchine sono identiche (P): un'operazione ha lo stesso tempo di processamento in tutte le macchine;

- le macchine sono uniformi (Q): il tempo di processamento di un'operazione $O_{i,j}$ nella macchina M_k è pari a $p_{i,j,k} = q_{i,j}/v_k$, dove $q_{i,j}$ è per esempio il numero di componenti di $O_{i,j}$ e v_k è il numero di componenti che M_k può processare per unità di tempo;

- le macchine sono scorrelate o indipendenti (R): il tempo di processamento di un'operazione $O_{i,j}$ nella macchina M_k è pari a $p_{i,j,k}$ ed è un dato del problema.

Problemi di scheduling più generali

Supponiamo in questo caso che ogni operazione abbia il suo insieme di macchine in cui può essere processata. Possiamo ancora una volta distinguere diversi casi:

- i lavori sono *mono-operation* (P/Q/R, "Multi Purpose Machines Scheduling Problems MPM SP" [Brucker[3]]);

- i lavori seguono un preciso ordine di processamento ("General Shop MPM SP");

- i lavori non seguono un ordine preciso ("Openshop MPM SP").

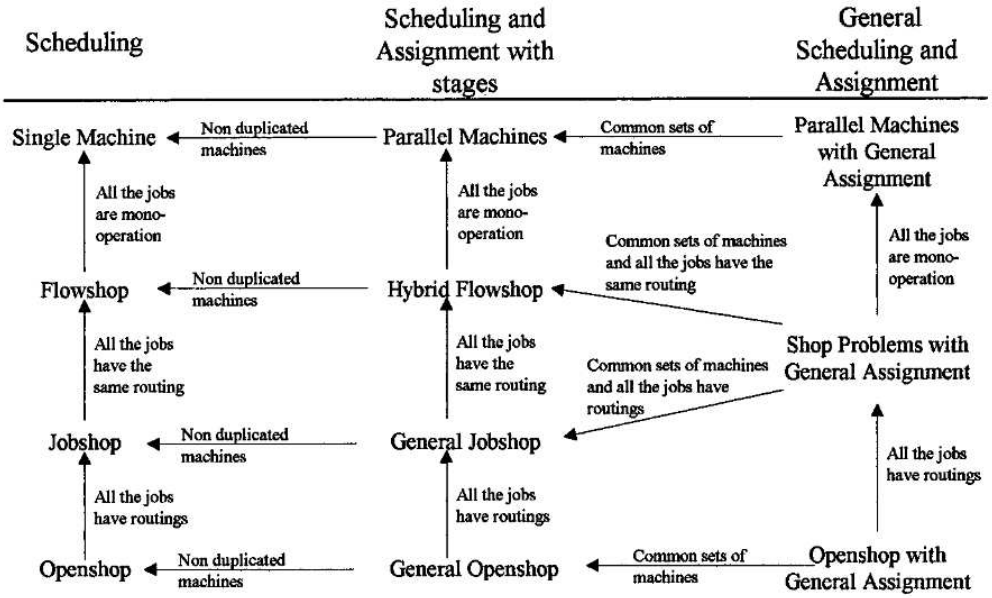


Figura 1.1: Una classificazione dei problemi di scheduling. In [1], p. 15.

La precedente classificazione basa la differenziazione dei problemi sul tipo di macchine e operazioni. Un altro modo di classificarli è basarsi sulle loro diverse caratteristiche, come segue.

1. *deterministici vs. stocastici.* Se tutte le caratteristiche del problema (tempo di processamento di ogni operazione, date di rilascio, etc.) sono dati conosciuti, si parla di problema deterministico. Se invece alcune caratteristiche sono assunte

[3]P. Brucker, *Scheduling Algorithms.* Springer, Berlin, 2004

come variabili random di una certa legge di probabilità, parliamo di problema stocastico.

2. *unitari vs. ripetitivi.* Se le operazioni sono cicliche, parliamo di problema ripetitivo. Se invece ogni operazione corrisponde ad un unico prodotto/risultato, parliamo di problema unitario.

3. *statici vs. dinamici.* Se tutti i dati del problema sono forniti allo stesso istante, parliamo di problema statico. Per alcuni problemi è possibile che con l'arrivo di nuove operazioni sia necessario ricalcolare e modificare in tempo reale la schedulazione precedentemente definita: parliamo in questo caso di problema dinamico.

La notazione più usata per riferirsi a problemi di scheduling è quella proposta da Graham nel 1979[4] ed è divisa in tre campi: $\alpha|\beta|\gamma$.

Il campo α si riferisce alla classificazione in Figura 1.1 e descrive la struttura dei problemi. Si divide a sua volta in due campi: $\alpha = \alpha_1\alpha_2$, e i valori di α_1 e α_2 si riferiscono alla configurazione delle macchine ed eventualmente al loro numero.

Il campo β contiene i vincoli espliciti del problema.

Il campo γ contiene il criterio (o i criteri) da ottimizzare.

Esistono estensioni più precise di questa notazione, convenienti per problemi specifici.

1.2 Vincoli e criteri di ottimalità

La soluzione di un problema di scheduling deve sempre soddisfare un certo numero di vincoli, siano essi impliciti o espliciti. In un problema flowshop per esempio è implicito che i lavori debbano essere processati seguendo un ordine prefissato, dunque un'operazione non può iniziare prima del termine della precedente. Altri tipi di vincoli possono riguardare le date di scadenza, che se imperative non è possibile superare, o i tempi di inizio dei singoli lavori.

Elenchiamo di seguito alcuni dei vincoli che maggiormente si incontrano in letteratura quando si trattano problemi di scheduling.

- **pmtn** indica che è autorizzata la prelazione. In questo caso è possibile interrompere forzatamente l'esecuzione di un'operazione per riprenderla, eventualmente in un'altra macchina, al termine del processamento dell'operazione prioritaria;

- **split** indica che è autorizzato lo *splitting*. In questo caso è possibile suddividere le operazioni in lotti più piccoli che saranno eseguiti in una o più macchine, eventualmente in contemporanea, per ridurre in generale i tempi di completamento;

- **prec** indica che le operazioni sono collegate da vincoli di precedenza;

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287-326, 1979

- **batch** indica che i lavori sono raggruppati in lotti;
- **no-wait** indica che le operazioni che compongono un lavoro si susseguono senza pause;
- **prmu** indica che le operazioni vengono eseguite dalle macchine nello stesso ordine;
- $\boldsymbol{d_i = d}$ o $\boldsymbol{p_i = p}$ indicano che tutte le date di scadenza o i tempi di processamento sono uguali.

Per decidere come organizzare una schedulazione poi, come accennato all'inizio del capitolo, possiamo avvalerci di un certo numero di criteri. Spesso la differenza tra criterio e vincolo è molto sottile e la scelta tra i due spetta al programmatore che inizia l'algoritmo. In generale, se un vincolo rappresenta qualcosa che necessariamente dobbiamo rispettare, ottimizzare un criterio permette invece di avere un certo grado di libertà nel trovare la soluzione. Inoltre se di fronte a un vincolo è possibile che una soluzione ammissibile non esista, quando si tratta di massimizzare o minimizzare un criterio questo problema non si pone, in quanto una soluzione possibile esiste sempre. Per alcuni tipi di problemi di scheduling non esiste un criterio da minimizzare. In questo caso ci troviamo necessariamente di fronte ad un problema decisionale: esiste una soluzione che rispetta i vincoli?

Possiamo infine classificare i criteri di ottimalità in due grandi famiglie: i criteri *minimax*, che rappresentano il valore massimo di un insieme di funzioni da minimizzare (ne vedremo un esempio nel prossimo capitolo, Sezione 2.1), e i criteri *minisum*, che rappresentano una somma di funzioni da minimizzare.

1.3 Nozioni fondamentali

Diciamo che un sottoinsieme di schedulazioni è *dominante* per un problema se e solo se, qualsiasi siano i dati del problema, in quell'insieme è contenuta una soluzione ottima.

Definizione 1.1. Sia S un insieme di soluzioni. Un criterio Z è *regolare* se e solo se Z è una funzione crescente del tempo di completamento dei lavori, cioè se e solo se:

$$\begin{aligned} &\forall x, y \in S, C_i(x) \leq C_i(y), \forall i = 1, \ldots, n, \\ &\Rightarrow Z(C_1(x), \ldots, C_n(x)) \leq Z(C_1(y), \ldots, C_n(y)) \end{aligned}$$

con C_i tempo di completamento del lavoro J_i.

Distinguiamo quattro classi di schedulazioni: non ritardate, attive, semi-attive e con inserimento di tempi di inattività. In particolare le schedulazioni con inserimento di tempi di inattività sono interessanti per quanto riguarda la minimizzazione di certi criteri non regolari.

Definizione 1.2. Una schedulazione appartiene alla classe delle schedulazioni con inserimento di tempi di inattività se e solo se prima dell'inizio di ogni operazione le macchine sono volontariamente lasciate inattive per un periodo di tempo non negativo.

Definizione 1.3. Sia $x \in S$ una schedulazione e sia S_x l'insieme di schedulazioni in cui le operazioni nelle varie macchine sono ordinate come in x. Una schedulazione x appartiene alla classe delle schedulazioni semi-attive se e solo se $\nexists y \in S_x$ tale che $C_i(y) \leq C_i(x), \forall i = 1, \ldots, n$, con almeno una disuguaglianza stretta.

La classe delle schedulazioni semi-attive è una sottoclasse del caso di schedulazioni con tempi di inattività. In generale il numero di schedulazioni possibili per un problema *jobshop* è infinito, poichè possiamo inserire una quantità arbitraria di tempo di inattività tra due operazioni adiacenti. Per rendere la schedulazione più utile e compatta, è possibile "aggiustare" il tempo d'inizio di ogni operazione eliminando il tempo di inattività superfluo senza però modificare la sequenza delle operazioni nella schedulazione. Data una sequenza per le operazioni in una macchina, tra tutte le schedulazioni possibili ne esiste solo una in cui non si possa modificare il tempo d'inizio di qualche operazione. L'insieme di tutte le schedulazioni (associate alle varie macchine del problema) in cui ciò *non* sia possibile è proprio l'insieme delle schedulazioni semi-attive. Questo insieme è dominante rispetto all'insieme di tutte le schedulazioni ed è possibile quindi considerare solo le schedulazioni semi-attive se vogliamo ottimizzare un problema generale. L'insieme delle schedulazioni semi-attive è finito, ma può avere una cardinalità molto grande. Per un problema *jobshop* classico, in cui ogni lavoro è composto da esattamente una operazione in ogni macchina, ognuna delle macchine disponbili deve processare n operazioni (con n il numero di lavori). Il numero di sequenze possibili è pari a $n!$ per ogni macchina, e se le sequenze in ogni macchina fossero indipendenti avremmo $(n!)^m$ schedulazioni semi-attive. Fortunatamente, la classe delle schedulazioni attive rappresenta un sottoinsieme dominante nell'insieme delle schedulazioni semi-attive.

Definizione 1.4. Una schedulazione $x \in S$ appartiene alla classe delle schedulazioni attive se e solo se $\nexists y \in S$ tale che $C_i(y) \leq C_i(x), \forall i = 1, \ldots, n$, con almeno una disuguaglianza stretta.

Una schedulazione attiva è anche semi-attiva. Nelle schedulazioni semiattive vorremmo modificare la sequenza originaria delle operazioni facendone iniziare alcune prima del previsto, senza ritardarne nessuna. L'insieme di tutte le schedulazioni in cui ciò *non* sia possibile corrisponde all'insieme delle schedulazioni attive.

In generale una schedulazione viene detta *attiva* se non è possibile schedulare un lavoro prima del previsto senza violare alcuni dei vincoli imposti dal problema; viene detta invece *semiattiva* se non è possibile schedulare un lavoro prima del previsto senza violare alcuni vincoli *o* cambiare l'ordine di processamento.

Lemma 1.1 (Baker). *Per problemi di ottimizzazione di criteri regolari l'insieme delle schedulazioni attive è dominante.*

Questo lemma implica che la ricerca di una soluzione ottima per l'ottimizzazione di un problema relativo ad un criterio regolare può limitarsi all'insieme delle schedulazioni attive. Per problemi *multicriterio* questo risultato resta valido se ottimiziamo vari criteri regolari, ma cessa di valere se almeno un criterio non lo è.

Definizione 1.5. Una schedulazione $x \in S$ appartiene alla classe delle schedulazioni non ritardate se e solo se nessuna operazione viene lasciata in attesa (non viene processata) anche se una macchina è disponibile per processarla.

Una schedulazione non ritardata è anche una schedulazione attiva.

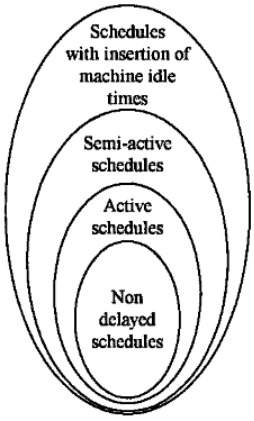


Figura 1.2: Inclusione tra le classi. In [1], p. 19.

1.4 Regole di scheduling

Elenchiamo di seguito alcune delle regole più incontrate in letteratura, spesso riferite a problemi applicati a situazioni industriali.

Regola SPT (*Shortest Processing Time first)*: ordina i lavori in ordine crescente rispetto al loro tempo di processamento. La regola opposta è la regola LPT (*Lowest Processing Time first).*

Regola WSPT (*Weighted Shortest Processing Time first)*: ordina i lavori in ordine crescente in base al rapporto p_i/w_i tra i tempi di processamento e i tempi di completamento pesati.

Regola EDD (*Earliest Due Date first)*: ordina i lavori in ordine crescente in base alla loro data di scadenza.

Regola EST (*Earliest Start Time first)*: ordina i lavori in ordine crescente in base al tempo d'inizio.

Per i problemi con macchine parallele queste regole devono essere leggermente modificate con l'aggiunta di un'ulteriore regola per l'assegnazione dei lavori alle macchine. Nel caso di macchine identiche possiamo usare la Regola FAM (*First Available Machine*) che assegna un lavoro alla prima macchina disponibile. In caso di macchine proporzionali possiamo usare invece la Regola FM (*Fastest Machine first*) che assegna un lavoro alla macchina più veloce disponibile per il processamento.

Se è autorizzata la prelazione, una regola applicabile è la Regola SRPT-FM (*Shortest Remaining Processing Time on Fastest Machine first*). Eccetera.

Capitolo 2

Algoritmi di scheduling

2.1 Singola macchina soggetta a vincoli di precedenza

Supponiamo che n lavori debbano essere processati da una singola macchina soggetta a determinati vincoli di precedenza, uno alla volta e senza interruzioni. Ad ogni lavoro J_j sono associati un tempo di processamento t_j e una funzione costo $c_j(t)$, monotona non descrescente, corrispondente al costo relativo al completamento del lavoro J_j al tempo t. Ci chiediamo come trovare una sequenza che minimizzi il massimo dei costi sostenuti.[1]

Teorema 2.1. *Sia S il sottoinsieme di lavori che possono essere processati per ultimi, cioè che non devono necessariamente precederne altri. Sia T la somma dei tempi di processamento di tutti i lavori. Sia infine J_k un lavoro in S tale che*

$$c_k(T) = \min_{J_j \in S}\{c_j(T)\}.$$

Allora esiste una sequenza ottima minmax, ovvero una sequenza che minimizza il massimo dei costi sostenuti, in cui il lavoro k-esimo è l'ultimo.

Dimostrazione. Consideriamo una qualsiasi sequenza π' in cui il lavoro $J_{k'} \neq J_k$, sia l'ultimo. Possiamo rappresentare graficamente una tale sequenza nel modo seguente, con A e B porzioni della sequenza formate dagli $n-2$ lavori diversi da J_k e $J_{k'}$:

π': | A | J_k | B | J_k' |

Supponiamo di modificare la sequenza π' per ottenere:

π: | A | B | $J_{k'}$ | J_k |

Se la sequenza π' soddisfa ai vincoli di precedenza, gli stessi vincoli sono soddisfatti anche dalla sequenza π, dato che né J_k né $J_{k'}$ devono precedere alcun lavoro. In particolare in π nessun lavoro viene completato più tardi che in π', a eccezione di J_k. Dalla monotonia della funzione dei costi segue che nessun costo tra quelli sostenuti può essere più elevato in π che in π', eccetto eventualmente il costo di

[1] I risultati presentati in questa sezione sono tratti da [3].

J_k. Per ipotesi, il lavoro J_k era stato scelto in modo che soddisfacesse la relazione $c_k(T) \leq c_{k'}(T)$. Segue allora che il massimo tra i costi sostenuti per la sequenza π non supera quello relativo alla sequenza π'. □

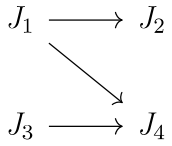
2.1.1 Algoritmo di Lawler

Dal teorema appena presentato discende in modo naturale un algoritmo per trovare una tale sequenza ottima minmax: si trovi un lavoro J_k che possa essere l'ultimo; si rimuova tale J_k dal problema e si trovi un lavoro che possa essere l'ultimo tra i rimanenti $n-1$ lavori e penultimo nella sequenza iniziale. Iterando la procedura si trova la sequenza cercata.

Poniamo $T = t_1 + t_2 + \ldots + t_n$ somma di tutti i tempi di processamento. Sia J_k il lavoro a cui è associato il costo $c_k(T)$ e sia L l'insieme dei lavori. Allora è possibile scrivere schematicamente l'algoritmo nel modo seguente [1]:

ALGORITMO DI LAWLER
$P = \emptyset$; *%insieme dei lavori già schedulati*
$T = \sum_{i=1}^{n} t_i$
For $i = n$ to 1:
$F = \{J_i \in L : J_i$ non ha successori in $L\}$
If $F = \emptyset$
Il problema non ha soluzione;
End If;
Sia $J_k \in F$ tale che $c_k(T) = \min_{J_j \in F}\{c_j(T)\}$.
$P = \{J_k\} \cup P$;
$L = L - \{J_k\}$;
$c_k = T$;
$T = T - t_k$;
End For;
Print P.

Esempio 2.1. Supponiamo di avere quattro lavori con i seguenti vincoli di precedenza:



Ad ogni lavoro è associato un tempo di processamento a_i, siano $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_4 = 1$, e una funzione costo c_i, $i = 1, \ldots, 4$. I lavori J_2 e J_4 possono essere scelti come ultimi nella sequenza, dato che non ne precedono altri. In ogni caso, qualsiasi sia l'ultimo lavoro nella sequenza, esso sarà completato al tempo $t = 6$. Confrontando l'andamento delle funzioni costo nel grafico in Figura 2.1 vediamo che $c_4(6) < c_2(6)$, dunque poniamo il lavoro J_4 alla fine della sequenza ottima. Dopo questa operazione, sia il lavoro J_2 che il lavoro J_3 sono eligibili per la penultima

posizione nella sequenza. In entrambi i casi, il lavoro in quella determinata posizione della sequenza sarà completato al tempo $t = 5$. Confrontiamo allora le funzioni costo dei due lavori al tempo $t = 5$ e vediamo che $c_3(5) < c_2(5)$. Posizioniamo dunque il lavoro J_3 in penultima posizione. Infine la scelta delle ultime due posizioni nella sequenza, per il lavoro J_1 e il lavoro J_2, è univocamente determinata dai vincoli di precedenza imposti inizialmente. Una sequenza ottima minmax sarà dunque $J_1 \to J_2 \to J_3 \to J_4$, e il costo massimo è sostenuto per il completamento del lavoro J_3, al tempo $t = 5$.

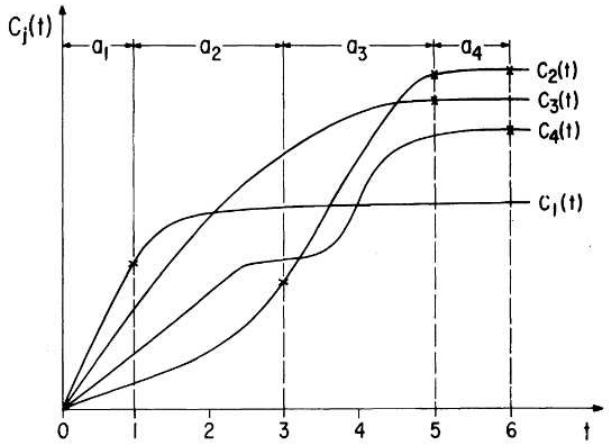


Figura 2.1: Grafico delle funzioni $c_i(t)$. In [3], p. 545.

2.1.2 Scadenze e vincoli di precedenza

Supponiamo che ad ogni lavoro sia associata, invece che una funzione costo $c_j(t)$, una scadenza d_j. Il problema è ora quello di trovare, se esiste, una sequenza, soggetta a dei dati vincoli di precedenza, che porti all'esecuzione di ogni lavoro entro la propria scadenza. Per risolverlo, ad ogni lavoro viene associata una nuova scadenza $\hat{d}_j \leq d_j$ e i lavori sono ordinati secondo la crescita di $\hat{d}_j$.[2] Definiamo $\hat{d}_j$ nel modo seguente: sia ρ una relazione di ordine parziale tale per cui se $J_i \rho J_j$ allora il lavoro J_i deve precedere il lavoro J_j; assumiamo che i lavori sia numerati in modo che $J_i \rho J_j$ implichi $i \leq j$. Allora per $j = 1, 2, \ldots, n$, poniamo $\hat{d}_j = \min\{d_k | J_j \rho J_k\} + j\epsilon$, con ϵ molto piccolo. Notiamo che $J_i \rho J_j$ implica $\hat{d}_i < \hat{d}_j$.

La soluzione di questo problema è indipendente dal tempo di processamento dei lavori. Inoltre il calcolo suggerisce la seguente regola *"first-to-last"*:

Ordinare i lavori dal primo all'ultimo, scegliendo come successivo un lavoro che abbia un successore con la scadenza più breve possibile, nell'insieme dei lavori disponibili (cioè senza predecessori non ancora scelti).

Lo stesso problema può anche essere analizzato come problema di minmax: sia assegnato ad ogni lavoro J_j una funzione costo $c_j(t)$. Seguendo l'algoritmo in 2.1.1,

[2]Per una dimostrazione completa si veda: E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77-84, 1969.

scopriamo che una sequenza ottima minmax è ottenuta dalla seguente regola *"last-to-first"*:

Ordinare i lavori dall'ultimo al primo, scegliendo come successivo un lavoro con la scadenza più breve possibile, nell'insieme dei lavori disponibili (cioè senza predecessori non ancora scelti).

Anche in questo caso notiamo che la sequenza non dipende dai tempi di processamento. Se questa sequenza non va a buon fine, nessun'altra sequenza lo fa. Osserviamo infine che entrambe le regole servono a minimizzare il ritardo massimo dei lavori, nel caso non sia possibile completare tutti i lavori in tempo.

Alcuni risultati su schedulazioni in cui ogni lavoro termina entro la propria scadenza e minimizzazione del ritardo complessivo saranno presentati nella Sezione 2.3 e nell'Appendice A.

2.2 Minimizzare il numero di *late jobs*

Il problema presentato in questa sezione è di sequenziare n lavori $J_1, \ldots, J_n$ a cui sono associati tempi di processamento noti, $t_1, \ldots, t_n$, e date di scadenza $d_1, \ldots, d_n$, tramite un singolo impianto di produzione in modo da minimizzare il numero di *late jobs*, ovvero, letteralmente, il numero di lavori in ritardo rispetto alla loro data di scadenza prevista.[3] I tempi di processamento, in cui sono inclusi i tempi di montaggio e smontaggio, sono indipendenti dalla sequenza. Assumiamo che gli n lavori siano disponibili per la produzione durante il periodo di schedulazione e che l'impianto di produzione operi continuativamente fino a che tutti i lavori sono completati. Inoltre non siano ammessi *lot-splitting*[4] (la produzione di un lavoro procede quindi dall'inizio alla fine senza interruzioni).

Assumiamo poi che sia possibile completare ogni lavoro entro la sua data di scadenza a patto di iniziare immediatamente, ovvero $\forall i, t_i \leq d_i$. Se ci trovassimo in una situazione diversa da questa, ridurremmo la dimensione del problema eliminando i lavori che non soddisfano a questa condizione.

2.2.1 Algoritmo di Moore

L'algoritmo consiste in una regola decisionale per la suddivisione dell'insieme dei lavori in due sottoinsiemi. Se i lavori nel primo sottoinsieme sono sequenziati in base alla loro data di scadenza e sono seguiti dai lavori del secondo sottinsieme, in qualsiasi ordine, la sequenza che viene a crearsi è ottima. In particolare:

Step 1: Ordiniamo l'insieme dei lavori in base al tempo di processamento e consideriamo come sequenza corrente la sequenza risultante $(J_{i_1} \ldots J_{i_n})$ con $t_{i_1} \leq \ldots \leq t_{i_n}$.

Step 2: Usando la sequenza trovata, troviamo il primo *late job*, sia J_{i_q}, e passiamo allo Step 3. Se non esiste un tale lavoro, l'algoritmo termina con una schedulazione ottima ottenuta ordinando i lavori della sequenza corrente in base alla loro data di scadenza, seguiti dai lavori scartati, in qualsiasi ordine.

[3] I risultati presentati in questa sezione sono tratti da [5].

[4] Cfr. *split/splitting*, Sezione 1.2.

Step 3: Riordiniamo i lavori $J_{i_1}, \ldots, J_{i_q}$ in base alla loro data di scadenza. Otteniamo la sequenza $J_{l_1} \ldots J_{l_q}$, $J_{i_{q+1}} \ldots J_{i_n}$, con $d_{l_1} \leq \ldots \leq d_{l_q}$. A questo punto ci sono due casi da considerare:

1) Se tutti i lavori nella sequenza $J_{l_1}, \ldots, J_{l_q}$ sono in anticipo (non *late job*), consideriamo come sequenza completa la sequenza corrente, e torniamo allo Step 2.

2) Altrimenti scartiamo il lavoro J_{i_q} e lo rimuoviamo dalla sequenza $J_{l_1} \ldots J_{l_q}$, $J_{i_{q+1}} \ldots J_{i_n}$. Infine applichiamo lo Step 2 alla sequenza così ottenuta.

Siano T l'insieme dei lavori da schedulare, $S = (J_1, \ldots, J_n)$ e L il numero di *late jobs*. Assumiamo $d_1 \leq \ldots \leq d_n$ e sia C_i il costo di completamento del lavoro J_i. Allora un modo schematico per visualizzare l'algoritmo è il seguente [1]:

ALGORITMO DI MOORE

$Rit = \emptyset$;
While ($\exists J_l \in S$ tale che $C_l > d_l$):
  Sia k tale che $C_{S[k]} > d_{S[k]}$ e $\forall i < k$, $C_{S[i]} \leq d_{S[i]}$;
  Sia j tale che $j \leq k$ e $t_{S[j]} = \max_{i=1,\ldots,k}(P_{S[i]})$;
  $S = S - \{J_j\}$;
  $Rit = Rit \cup \{J_j\}$;
End While;
$L = \lvert Rit \rvert$;
Print $S \cup Rit$ and L.

Esempio 2.2. Per visualizzare al meglio l'algoritmo, pensiamo ad un esempio numerico:

Lavoro	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
Tempo di processamento	10	6	3	1	4	8	7	6
Scadenza	35	20	11	8	6	25	28	9

Step 1: Ordiniamo i lavori in base al tempo di processamento più corto. Otteniamo:

Lavoro	J_4	J_3	J_5	J_2	J_8	J_7	J_6	J_1
Tempo di processamento	1	3	4	6	6	7	8	10
Scadenza	8	11	6	20	9	28	25	35
Tempo di completamento	1	4	8	-	-	-	-	-

Step 2: Dalla tabella, osserviamo che il lavoro J_5 è il primo *late job*. Proseguiamo quindi con lo Step 3.

Step 3: Riordiniamo i primi tre lavori in base alla loro scadenza. Otteniamo:

Lavoro	J_5	J_4	J_3	J_2	J_8	J_7	J_6	J_1
Tempo di processamento	4	1	3	6	6	7	8	10
Scadenza	6	8	11	20	9	28	25	35
Tempo di completamento	4	5	8	14	20	-	-	-

Dalla tabella si vede che i primi lavori sono ora in anticipo (terminano prima della loro data di scadenza). Possiamo allora tornare allo Step 2 utilizzando la sequenza in tabella come sequenza principale.

Step 2: Il primo *late job* è ora J_8.

Step 3: Ordiniamo i primi cinque lavori in base alla scadenza. La sequenza che si trova è la seguente:

Lavoro	J_5	J_4	J_8	J_3	J_2	J_7	J_6	J_1
Tempo di processamento	4	1	6	3	6	7	8	10
Scadenza	6	8	9	11	20	28	25	35
Tempo di completamento	4	5	11	14	20	-	-	-

Dato che sia J_8 che J_3 sono *late*, scartiamo il lavoro J_8 e troviamo una nuova sequenza:

	Sequenza corrente							Lavori scartati
Lavoro	J_5	J_4	J_3	J_2	J_7	J_6	J_1	J_8
Tempo di processamento	4	1	3	6	7	8	10	6
Scadenza	6	8	11	20	28	25	35	9
Tempo di completamento	4	5	8	14	20	29	-	-

Step 2: Il primo *late job* è ora J_6.

Step 3: Riordiniamo i primi sei lavori in base alla scadenza. Otteniamo:

	Sequenza corrente							Lavori scartati
Lavoro	J_5	J_4	J_3	J_2	J_6	J_7	J_1	J_8
Tempo di processamento	4	1	3	6	8	7	10	6
Scadenza	6	8	11	20	25	28	35	9
Tempo di completamento	4	5	8	14	22	29	-	-

Dato che J_7 è *late*, scartiamo J_6 e otteniamo una nuova sequenza:

	Sequenza corrente						Lavori scartati	
Lavoro	J_5	J_4	J_3	J_2	J_7	J_1	J_8	J_6
Tempo di processamento	4	1	3	6	7	10	6	8
Scadenza	6	8	11	20	28	35	9	25
Tempo di completamento	4	5	8	14	21	31	-	-

Step 2: Tutti i lavori nella sequenza che stiamo considerando sono in anticipo. L'algoritmo quindi termina con la sequenza trovata $(J_5, J_4, J_3, J_2, J_7, J_1, J_8, J_6)$ come schedulazione ottima dei lavori.

A questo punto non è necessario riordinare i lavori in base alla loro scadenza (facendolo si troverebbe comunque una sequenza ottima).

2.2.2 Sviluppo teorico dei risultati

Per ogni schedulazione S, definiamo due insiemi E e L nel modo seguente:

$$J_i \in E \leftrightarrow C_i \leq d_i,$$

$$J_i \in L \leftrightarrow C_i > d_i$$

con C_i e d_i rispettivamente tempo di completamento e scadenza del lavoro J_i.

Definizione 2.1. Data una schedulazione S, sia A l'insieme ordinato definito ordinando gli elementi dell'insieme E in base al loro ordine in S.

Analogamente sia R l'insieme ordinato definito ordinando gli elementi dell'insieme L in base al loro ordine in S.

Infine sia π una permutazione arbitraria dell'insieme R e sia P l'insieme ordinato risultante.

Lemma 2.1. *Ogni schedulazione ottima S di un insieme J di lavori, $J = \{J_1, \ldots, J_n\}$, con tempi di processamento dati $t_1, \ldots, t_n$, e scadenze $d_1, \ldots, d_n$, è equivalente alla schedulazione $S^* = (A, R)$ e ad ogni altra schedulazione nella forma $S^{**} = (A, P)$.*

Dimostrazione. Supponiamo che $S = (J_{i_1}, \ldots, J_{i_n})$ non sia nella forma (A, R). Allora in S devono esistere due lavori adiacenti, J_{i_k} e $J_{i_{k+1}}$, tali che $C_{i_k} > d_{i_k}$ e $C_{i_{k+1}} \leq d_{i_{k+1}}$.

Definiamo allora nuova schedulazione ottima S_1 scambiando J_{i_k} e $J_{i_{k+1}}$, e S_1 avrà $A_1 = A$ e $R_1 = R$. Se S_1 è nella forma (A, R) abbiamo finito. Altrimenti operiamo su S_1 per ottenere S_2, *etc.* Questo processo termina in un numero finito di step e restituisce una schedulazione ottima S_q con $A_q = A$, $R_q = R$, e $S_q = (A, R) = S^*$. □

Lemma 2.2. *Data una schedulazione ottima S nella forma (A, R), è possibile definire una nuova schedulazione ottima $S_D = (A_D, R)$ riordinando l'insieme A in base alla scadenza di ogni lavoro.*

Questo lemma discende direttamente dal seguente:

Lemma 2.3 (Lemma di Jackson)**.** *Dato un insieme di lavori $\{J_1, \ldots, J_n\}$ con tempi di processamento $t_1, \ldots, t_n$ e scadenze $d_1, \ldots, d_n$, esiste una schedulazione senza late jobs se e solo se non sono presenti late jobs nella schedulazione ottenuta ordinando i lavori in base alla loro scadenza.*

Dato che tutti i lavori nell'insieme A sono puntuali (terminano entro la loro scadenza), lo saranno anche se riordinati in base alla scadenza.

Questi lemmi mostrano che a qualsiasi schedulazione S è associata una schedulazione ottima $S_D = (A_D, P)$. Di conseguenza il problema originale può essere riformulato come un problema di produrre una schedulazione nella forma (A_D, P) in cui la cardinalità dell'insieme E_D associato a S_D sia massima.

Lemma 2.4. *Supponiamo che esista una schedulazione ottima S per un insieme J di lavori, con L non vuoto (ricordiamo che $J_i \in L \leftrightarrow C_i > d_i$). Sia J^* un sottoinsieme qualsiasi di L. Allora per ogni schedulazione ottima $S' = (A', R')$ dell'insieme di lavori $J' = J - J^*$, è possibile definire una schedulazione ottima $S'' = (A'', P'')$ dell'insieme J, ponendo $A'' = A'$ e $L'' = J^* \cup L'$.*

Dimostrazione. Assumiamo, senza perdita di generalità, che S sia nella forma (A, R) e consideriamo una schedulazione ottima $S' = (A', R')$ per l'insieme di lavori J'. Se la cardinalità dell'insieme E', $|E'|$, eguaglia la cardinalita dell'insieme E, $|E|$, abbiamo finito.

Supponiamo quindi che $|E'| \neq |E|$.

a) Se $|E'| < |E|$ possiamo definire una nuova schedulazione ottima $S_1 = (A_1, P_1)$ per l'insieme di lavori J ponendo $A_1 = A$ e $L_1 = L - J^*$. Ma allora $|E_1| = |E| > |E'|$, quindi S' non è una schedulazione ottima per l'insieme J'. Contraddizione.

b) Se $|E'| > |E|$ possiamo deifinire una nuova sequenza ottima $S_2 = (A_2, P_2)$ per l'insieme J ponendo $A_2 = A'$ e $L_2 = L' \cup J^*$. Ma $|E_2| = |E'| > |E|$, quindi S non è una schedulazione ottima per J. Contraddizione.

Di conseguenza $|E'| = |E|$. □

Il problema è ora di sviluppare un algoritmo che per una certa schedulazione ottima S_d con almeno un *late job* trovi un lavoro $J_i \in L$. Sarà possibile ottenere una schedulazione ottima per l'insieme originario di lavori J applicando ripetutamente l'algoritmo ed eliminando i lavori trovati. Il processo terminerà quando, dopo aver eliminato l'insieme di lavori $J^* = (J_{i_1} \dots J_{i_q})$, l'algoritmo non riuscirà a trovare un lavoro utile nell'insieme $J - J^*$. Una schedulazione ottima per l'insieme di partenza J sarà definita ponendo $E = J - J^*$, $L = J^*$; dunque $S = (A_D, P)$, con A_D insieme ordinato ottenuto riordinando E in base alla scadenza di ogni lavoro.

Si può dimostrare che la schedulazione ottenuta è ottima applicando ripetutamente il Lemma 2.4.

Algoritmo di selezione

Sia dato un insieme di lavori $J = \{J_1, \dots, J_n\}$ con tempi di processamento $t_1, \dots, t_n$ e scadenze $d_1, \dots, d_n$. L'algoritmo che segue troverà un lavoro $J_i \in J$ tale che $J_i \in L$, se un tale lavoro esiste.

Definiamo la sequenza di partenza $(J_1 \dots J_n)$ assumendo che $t_1 \leq t_2 \leq \dots \leq t_n$.

Start: Trovare il primo *late job* J_q nella sequenza corrente, dove $q > 1$. Se un tale lavoro non esiste, la sequenza non ha *late jobs* e l'algoritmo termina. Riordinare i precedenti $q - 1$ lavori in base alla loro scadenza per ottenere una sequenza nella forma:

$$(J_{i_1} \dots J_{i_{q-1}}, J_q \dots J_n)$$

con $t_q \leq \dots \leq t_n$, $d_{i_1} \leq \dots \leq d_{i_{q-1}}$ e $t_q \geq t_{i_j}$ per $j = 1, \dots, q-1$. Inserire J_q nella sequenza $(J_{i_1} \dots J_{i_{q-1}})$ in base alla scadenza, per ottenere la sequenza:

$$(J_{i_1} \dots J_{i_k}, J_{i_{k+1}} \dots J_{i_{q-1}}, J_{q+1} \dots J_n)$$

con $d_{i_1} \leq \dots \leq d_{i_k} \leq d_q \leq d_{i_{k+1}} \leq \dots \leq d_{i_{q-1}}$.

Ci sono ora tre casi da considerare:

1) Se tutti i lavori nella sottosequenza $(J_{i_1} \dots J_{i_k}, J_{i_{k+1}} \dots J_{i_{q-1}})$ sono in anticipo, ricominciare l'analisi dall'inizio utilizzando come sequenza la sequenza appena definita.

2) Se J_q è un *late job*, allora $J_q \in L$ per qualche schedulazione ottima dei lavori nella sequenza corrente.

Dimostrazione. Consideriamo una schedulazione ottima nella forma $S_D = (A_D, P)$ per l'insieme dei lavori della sequenza corrente. Se $J_q \in P$, abbiamo finito. Assumiamo quindi che $J_q \in A_D$, che S_D sia del tipo $S_D = (J_{l_1} \dots J_{l_r} \dots J_{l_t}, P)$ e che $d_{l_1} \leq \dots \leq d_{l_r} < d_{l_{r+1}} \leq \dots \leq d_{l_t}$, con $J_{l_r} = J_q$.

Nella sequenza corrente $t_q \leq \ldots \leq t_n$ e $t_{i_j} \leq t_q$ per $j = 1, \ldots, q-1$. La sommatoria dei tempi di processamento per i lavori nell'insieme $\{J_{i_1} \ldots J_{i_k}\}$ è quindi minima nella classe di tutti i sottoinsiemi di k lavori della sequenza corrente aventi scadenza minore o uguale a d_q. Dato che J_q è un *late job* nella sequenza $(J_{i_1} \ldots J_{i_k}, J_q)$, allora sarà un *late job* anche in tutte le sequenze ordinate in base alla scadenza in cui J_q è il $k+1$-esimo lavoro. Ma J_{l_r} è in anticipo nella schedulazione S_D e dunque r deve essere strettamente minore di $k+1$. Inoltre $\{J_{l_{r+1}} \ldots J_{l_t}\} \cap \{J_{i_1} \ldots J_{i_k}\} = \emptyset$, dato che $d_{i_j} \leq d_q$ per $j = 1, \ldots, k$ e $d_q < d_{l_m}$ per $m = r+1, \ldots, t$. Quindi perché un elemento di $\{J_{i_1} \ldots J_{i_k}\}$ sia in orario nella schedulazione S_D, deve necessariamente appartenere all'insieme $\{J_{l_1} \ldots J_{l_{r-1}}\}$. Di conseguenza ci sono lavori in $\{J_{i_1} \ldots J_{i_k}\}$ che appartengono all'insieme L relativo a S_D perché $r < k+1$. A questo punto è allora possibile definire una nuova schedulazione ottima S_D^*: ordiniamo l'insieme di lavori $\{J_{i_1} \ldots J_{i_k}\}$ in base al minor tempo di completamento e selezioniamo i primi r lavori dell'insieme. Sostituiamo la sequenza iniziale $(J_{l_1} \ldots J_{l_r})$ nella schedulazione S_D con questo insieme di lavori ordinati in base alla loro scadenza e chiamiamo S_D^* la nuova schedulazione. Il lavoro $J_{l_r} = J_q$ appartiene ora all'insieme L^* associato alla nuova schedulazione ottima S_D^*. □

3) Se J_q non è un *late job*, ma almeno uno dei lavori nella sequenza $(J_{i_{k+1}} \ldots J_{i_{q-1}})$ lo è, allora $J_q \in L$ per qualche schedulazione ottima dell'insieme dei lavori nella sequenza corrente.

Dimostrazione. Come nella dimostrazione del punto 2), assumiamo di avere una schedulazione ottima $S_D = (J_{l_1} \ldots J_{l_r} \ldots J_{l_t}, P)$ e che $d_{l_1} \leq \ldots \leq d_{l_r} < d_{l_{r+1}} \leq \ldots \leq d_{l_t}$, con $J_{l_r} = J_q$.

a) Se $r < k+1$, possiamo definire una schedulazione ottima S_D^* come al punto 2), dove $J_q \in L^*$.

b) Se $r \geq k+1$, le argomentazioni sviluppate al punto 2) possono essere utilizzate per far vedere che il tempo di completamento di $J_{l_r} = J_q$ nella sequenza S_D è maggiore o uguale al suo tempo di completamento nella sequenza $(J_{i_1} \ldots J_{i_k}, J_q)$. Inoltre i lavori nell'insieme $\{J_{i_{k+1}} \ldots J_{i_{q-1}}\}$ devono seguire J_q nella schedulazione S_D, nella quale $(J_{l_1} \ldots J_{l_r} \ldots J_{l_t})$ è ordinata in base alle scadenze di ciascun lavoro. Pertanto dato che almeno uno dei lavori nell'insieme $\{J_{i_{k+1}} \ldots J_{i_{q-1}}\}$ era un *late job* nella sequenza di partenza, deve essercene almeno uno, sia J_{i_m}, che appartiene all'insieme L relativo a S_D. Ma $t_{i_m} \leq t_q$ e $d_{i_m} \geq d_q$. Quindi J_q può essere rimosso da A_D e sostituito da J_{i_m} a formare una nuova schedulazione ottima S_D^* in cui $J_q \in L^*$. □

Questo processo continua e ogni lavoro successivo o viene accettato nella sottosequenza iniziale ordinata in base alle scadenze di ciascun lavoro, o viene scartato. L'algoritmo termina con una sequenza $(J_{a_1} \ldots J_{a_t})$ di lavori in anticipo, ordinati in base alla scadenza di ognuno, e un insieme di lavori $\{J_{a_{t+1}} \ldots J_{a_n}\}$ che sono stati scartati. Una schedulazione ottima per il problema iniziale sarà

$$S_D = (J_{a_1} \ldots J_{a_t}, P)$$

con P associato all'insieme $\{J_{a_{t+1}} \ldots J_{a_n}\}$.

2.3 Schedulazione preventiva con scadenze

Consideriamo un problema di schedulazione in cui n lavori indipendenti devono essere programmati in m macchine identiche parallele (processori).[5] Ad ogni lavoro J_i siano associati una data di rilascio $r_i \geq 0$, una data di scadenza d_i e un tempo di processamento $t_i \geq 0$. Il lavoro J_i non può essere processato prima della sua data di rilascio r_i e deve essere completato entro la sua scadenza d_i. Assumiamo che $r_i + t_i \leq d_i$, $1 \leq i \leq n$. Assumiamo inoltre che sia concessa la prelazione (parleremo dunque di schedulazione *preventiva*), ma che non ci siano costi associati ad essa, e che nessun lavoro possa essere processato da più di una macchina contemporaneamente.

In questa sezione parleremo di schedulazione-DD per un insieme J di n lavori per riferirci ad una schedulazione in cui ogni lavoro è completato entro la sua data di scadenza (*Due Date*).

Il nostro obiettivo è di sviluppare un algoritmo per ottenere una schedulazione-DD preventiva nel caso in cui o tutti i lavori hanno la stessa data di rilascio, o tutti i lavori hanno la stessa data di scadenza.

Nel cercare un algoritmo che restituisca una schedulazione-DD, si possono distinguere tre tipi di algoritmi: *online*, *quasi online* e *offline*.

Definizione 2.2. Sia J un insieme di n lavori con k tempi di rilascio distinti. Un algoritmo *online* è un algoritmo che costruisce una schedulazione-DD da un tempo 0 ad un tempo τ qualsiasi senza conoscere i lavori rilasciati dal tempo τ in poi, né i tempi di rilascio rimanenti. Un algoritmo *quasi online* è un algoritmo che costruisce la schedulazione dal tempo 0 al tempo r_i senza conoscere i lavori rilasciati in r_i, r_{i+1}, ..., $1 \leq i \leq k$. L'unica informazione in più disponibile per un algoritmo *quasi online* rispetto ad uno *online* è la data di rilascio successiva a r_i (ma non il lavoro ad essa associata). Un algoritmo *offline*, infine, conosce al tempo 0 i tempi di ogni lavoro, compresi tempi di rilascio e scadenze.

Dalla definizione segue che se non esistono algoritmi *offline* per ottenere una schedulazione-DD preventiva, non possono essercene nemmeno *quasi online*. Inoltre se non esistono algoritmi *quasi online*, non esistono neanche algoritmi *online*.

Il lemma seguente prova che non esistono algoritmi *quasi online*, e dunque neanche *online*, che restituiscano una schedulazione-DD preventiva quando $m > 1$.

Lemma 2.5. *In presenza di due o più macchine parallele non esiste alcun algoritmo quasi online per costruire una schedulazione-DD.*

Dimostrazione. Supponiamo che $m = 2$ e al tempo 0 siano rilasciati tre lavori A, B e C, con $t_A = t_B = 5$, $t_C = 10$, $d_A = 5$, $d_B = 15$ e $d_C = 24$. Supponiamo che il tempo di rilascio successivo a questi sia 5. Ogni algoritmo quasi-online dovrà costruire la schedulazione tra 0 e 5 prima di conoscere date di scadenza e tempi di processamento dei lavori che saranno rilasciati al tempo $t = 5$. Inoltre in ogni schedulazione-DD, il lavoro A dovrà essere programmato tra 0 e 5. Assumiamo senza perdita di generalità che sia programmato nella macchina *M1*. Assumiamo poi che il lavoro B sia programmato nella seconda macchina, *M2*, in un intervallo Δ, $0 \leq \Delta \leq 5$, e che il lavoro C sia nella stessa macchina, nell'intervallo successivo

[5] I risultati presentati in questa sezione sono tratti da [6].

di ampiezza $5 - \Delta$, come in Figura 2.2(a). Se $\Delta = 0$, assumiamo che al tempo 5 siano rilasciati i lavori D e E, con tempi $t_D = t_E = 6$ e scadenze $d_D = d_E = 11$. La schedulazione-DD deve essere ora come in figura 2.2(b). Appare evidente che non c'è modo per il lavoro B di finire entro la sua scadenza. D'altra parte, se $\Delta = 5$ tutti i lavori potrebbero essere processati, assegnandoli alla macchina opportuna, entro la loro scadenza (Figura 2.2(c)).

Sia allora $\Delta \neq 0$. Assumiamo che i lavori D e E vengano rilasciati al tempo 5, con $t_D = t_E = 5$ e $d_D = d_E = 10$. Assumiamo inoltre che al tempo 15 vengano rilasciati i lavori F e G, con $t_F = t_G = 9$ e $d_F = d_G = 24$. Ogni schedulazione-DD deve allora programmare i lavori D, E, F e G come in Figura 2.2(d). A questo punto è evidente che non c'è modo di completare il lavoro C entro la sua scadenza. Se però avessimo scelto $\Delta = 0$, mantenendo i tempi di rilascio scelti, avremmo potuto costruire una schedulazione-DD opportuna (Figura 2.2(e)).

Per $m > 2$ è possibile usare lo stesso esempio introducendo $m - 2$ lavori rilasciati al tempo 0, con scadenza e tempo per l'elaborazione pari a 24. □

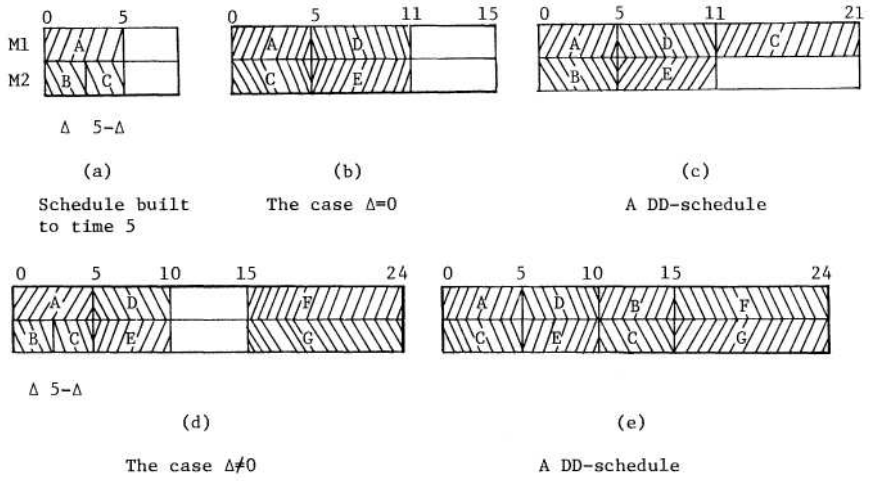


Figura 2.2: Esempio per la dimostrazione del Lemma 2.5. In [6], p. 927.

Per il caso $m = 1$, Horn[6] [7] presenta un algoritmo *quasi online* di complessità $O(n \log n)$. In particolare l'algoritmo proposto costruisce una schedulazione-DD con non più di $n - 1$ prelazioni.

Lemma 2.6. *Per il caso $m = 1$ esiste un insieme J di n lavori tale che ogni schedulazione-DD ad esso associata abbia almeno $n - 1$ prelazioni.*

Dimostrazione. Consideriamo l'insieme dei lavori $\{J_1, \ldots, J_n\}$, con $(r_1, t_1, d_1) = (0, n, 2n - 1)$, $(r_i, t_i, d_i) = (2i - 3, 1, 2i - 2)$, $2 \leq i \leq n$. In ogni schedulazione-DD ogni lavoro J_i deve essere schedulato tra il tempo $2i - 3$ e il tempo $2i - 2$, $2 \leq i \leq n$. L'unico modo di completare il lavoro J_1 è di schedularlo negli intervalli $(2j, 2j + 1)$, $0 \leq j < n$ e questo introduce $n - 1$ prelazioni. □

[6]Cfr. Appendice A

In questo modo otteniamo un algoritmo che costruisce una schedulazione-DD per insiemi di lavori in cui ogni lavoro ha la stesso tempo di rilascio.

L'algoritmo che propone Sahni ha complessità $O(n \log mn)$ e costruisce schedulazioni con al massimo $n-2$ prelazioni.

2.3.1 Algoritmo di Sahni

Quando tutti i lavori sono disponibili per essere processati al tempo 0 è possibile costruire una schedulazione-DD, se esiste, considerando i lavori in base alla scadenze in ordine non decrescente.

Sia J un insieme di n lavori e siano δ_i, $1 \leq i \leq k$, k scadenze distinte. Poniamo inoltre $\delta_0 = 0$. Sia n_i il numero di lavori con scadenza δ_i, $1 \leq i \leq k$, $\sum n_i = n$, e assumiamo $n_i \geq 1$. L'algoritmo proposto da Sahni schedula gli n lavori in k passi. Al passo i saranno schedulati tutti i lavori con scadenza δ_i. Per ogni lavoro J_r con scadenza δ_i si procede nel modo seguente:

1. Sia $RPT(j)$ il tempo di processamento disponibile nella macchina j-esima dal tempo 0 al tempo δ_i, $1 \leq j \leq m$.

2. Se t_r è maggiore di $RPT(j)$ per ogni j, allora non è possibile programmare il lavoro J_r in modo da rispettare la sua scadenza e l'algoritmo termina.

3. Se t_r è minore o uguale a $RPT(j)$ per ogni j, con $RPT(j) \neq 0$, allora il lavoro J_j viene schedulato nella macchina con il minimo RPT diverso da zero. L'algoritmo procede poi a schedulare il lavoro successivo.

4. Se non ci troviamo né nel caso 2 né nel caso 3, determiniamo la macchina x con RPT massimo tale che $t_r \geq RPT(x)$ e la macchina y con RPT minimo, diverso da zero, tale che $t_r < RPT(y)$. Osserviamo che una tale macchina y potrebbe non esistere. Se y esiste, allora il lavoro J_r viene schedulato in modo che utilizzi tutto l'RPT di x e prenda da y il tempo aggiuntivo necessario. Se invece y non esiste, dobbiamo avere $t_r = RPT(x)$ e scheduliamo J_r in modo da utilizzare tutto l'RPT della macchina x. L'algoritmo procede poi con la schedulazione dei lavori rimanenti.

Introduciamo C_j^M, tempo di completamento dell'ultimo lavoro nella macchina M_j, $\forall j = 1, \ldots, m$. Possiamo visualizzare l'algoritmo come segue [1]:

ALGORITMO DI SAHNI

Assumiamo $d_1 \geq \ldots \geq d_n$;
$C_j^M = 0$, $\forall j = 0, \ldots, m$
For $i = 1$ to n:
  % Scheduliamo il lavoro J_i
  Sia $L = \{j\colon C_j^M < d_i\}$
  If $((L = \emptyset)$ or $(d_i - \min_{j \in L}(C_j^M) < t_i))$
    Non esistono schedulazioni possibili;
  End If;
  Sia $k \in L$ tale che $C_k^M = \max_{j \in L}(C_k^M)$;
  If $(d_i - C_k^M \geq t_i)$
    % Il lavoro J_i *è schedulato completamente in* M_j
    $C_k^M = C_k^M + t_i$;
  Else:
    % Il lavoro J_i *è processato in più di una macchina*
    Sia $L_1 = \{j \in L\colon\ d_i - C_j^M < t_i\}$;
    Sia $L_2 = \{j \in L\colon\ d_i - C_j^M \geq t_i\}$;
    Sia $\alpha \in L_2$ tale che $C_\alpha^M = \max_{j \in L_2}(C_j^M)$;
    Sia $\beta \in L_1$ tale che $C_\beta^M = \min_{j \in L_1}(C_j^M)$;
    $C_\alpha^M = C_\alpha^M + (p_i - d_i + C_\beta^M)$;
    $C_\beta^M = d_i$;
  End If;
End For;
Stampa la schedulazione trovata.

2.4 Produzione a due fasi con tempi di installazione

Consideriamo di avere n lavori da schedulare ciascuno prima nella macchina M_1 e poi nella macchina M_2; ogni macchina può processare al massimo un lavoro in un dato istante temporale.[7]

Consideriamo $2n$ costanti positive arbitrarie A_i, B_i, $i = 1, 2, \ldots, n$, dove con A_i indichiamo la somma tra il tempo di installazione e il tempo di completamento del lavoro i-esimo nella macchina M_1 e con B_i la stessa somma relativamente alla macchina M_2. Cerchiamo una schedulazione ottima che minimizzi il tempo totale necessario per il processamento dei lavori.

Lemma 2.7. *La sequenza di produzione di una macchina può essere costruita allo stesso modo dell'altra senza perdita di tempo.*

Dimostrazione. Consideriamo la scala temporale per il processamento dei lavori in ogni macchina e posizioniamo gli elementi di A e B in modo arbitrario, rispettando le ipotesi del problema: l'inizio di B_j non può precedere la fine di A_j. Se l'ordine degli elementi in A è diverso da quello dei rispettivi elementi in B, posizioniamo gli elementi in scala come segue:

[7] I risultati presentati in questa sezione sono tratti da [8].

M_1	A_i	A_k	A_j			
M_2				B_j	B_k	B_i

Ora, senza perdita di tempo, possiamo ordinare i lavori nella macchina M_1 nello stesso ordine in cui si trovano in M_2 tramite scambi successivi, considerando coppie consecutive di lavori J_i con A_i e B_i in posizioni diverse tra M_1 ed M_2 e partendo da sinistra. Simmetricamente, possiamo ordinare i lavori in M_2 affinché seguano lo stesso ordine presente in M_1.

Dato che ora l'ordine in M_1 e l'ordine in M_2 concidono, possiamo iniziare il processamento di ogni lavoro il prima possibile per minimizzare il tempo totale. In particolare nella macchina M_1 non ci sono ritardi (non ci sono perdite di tempo). □

Sia ora X_i il tempo di inattività per la macchina M_2 immediatamente prima che il lavoro i-esimo venga in essa processato.

Se consideriamo per esempio la sequenza $S = 1, 2, \ldots, n$, nelle due macchine abbiamo la seguente scala temporale:

M_1	A_1		A_2		A_3		A_4	
M_2	X_1	B_1		X_2	B_2	X_3	B_3	X_4	B_4

Abbiamo:

$$\begin{aligned}
X_1 &= A_1 \\
X_2 &= \max(A_1 + A_2 - B_1 - X_1, 0) \\
X_1 + X_2 &= \max(A_1 + A_2 - B_1, A_1) \\
X_3 &= \max(\sum_{i=1}^{3} A_i - \sum_{i=1}^{2} B_i - \sum_{i=1}^{2} X_i, 0) \\
\sum_{i=1}^{3} X_i &= \max(\sum_{i=1}^{3} A_i - \sum_{i=1}^{2} B_i, \sum_{i=1}^{2} X_i) \\
&= \max(\sum_{i=1}^{3} A_i - \sum_{i=1}^{2} B_i, \sum_{i=1}^{2} A_i - B_1, A_1)
\end{aligned}$$

In generale, $\sum_{i=1}^{n} X_i = \max_{1 \le u \le n} K_u$, con $K_u = \sum_{i=1}^{u} A_i - \sum_{i=1}^{u-1} B_i$.

Sia $F(S) = \max_{1 \le u \le n} K_u$. Vorremmo una sequenza S^* tale che $F(S^*) \le F(S_O)$ per qualsiasi S_O.

Consideriamo la sequenza S' formata scambiando il j-esimo elemento in S con l'elemento $j+1$-esimo. Allora si ha:

$$F(S') = \max_{1 \le u \le n} K'_u$$

dove

$$\begin{aligned}
&K'_u = \sum_{i=1}^{u} A'_i - \sum_{i=i}^{u} B'_i, \; A'_i = A_i, \quad \text{per} \quad i \neq j, j+1 \\
&A'_j = A_{j+1}, \; B'_j = B_{j+1}, \; A'_{j+1} = A_j, \; B'_{j+1} = B_j
\end{aligned}$$

Dunque $K'_u = K_u$ se $u \neq j, j+1$, e in particolare $F(S') = F(S)$ a meno che $\max(K_j, K_{j+1}) \neq \max(K'_j, K'_{j+1})$.

Teorema 2.2. *La seguente regola fornisce un ordinamento ottimale della sequenza: L'elemento j precede l'elemento $j+1$ se*

$$\max(K_j, K_{j+1}) < \max(K'_j, K'_{j+1}). \tag{2.1}$$

In caso di uguaglianza entrambi gli ordinamenti sono ottimali, purchè rispettino le preferenze sugli elementi (cfr. Caso 4, Lemma 2.8).

Sottraendo $\sum_{i=1}^{j+1} A_i - \sum_{i=1}^{j+1} B_i$ da entrambi i termini in (2.1), la relazione diventa:

$$\max(-B_j, -A_{j+1}) < \max(-B_{j+1}, -A_j),$$

cioè:

$$\min(A_j, B_{j+1}) < \min(A_{j+1}, B_j). \tag{2.2}$$

Questo ordinamento è transitivo (come vediamo nel lemma che segue) e fornisce quindi una sequenza S^* unica a meno di alcuni elementi 'indifferenti' (se $A_k = B_k$, J_k è indifferente per gli altri elementi nel senso che A_k e B_k stanno in relazione ad essi in ugual modo).

Lemma 2.8. *La relazione (2.2) è transitiva.*

Dimostrazione. Supponiamo $\min(A_1, B_2) \leq \min(A_2, B_1)$ e $\min(A_2, B_3) \leq \min(A_3, B_2)$. Allora $\min(A_1, B_3) \leq \min(A_3, B_1)$, a meno che J_2 sia indifferente per J_1 e J_3. Infatti:

1. Se $A_1 \leq B_2, A_2, B_1$ e $A_2 \leq B_3, A_3, B_2$, allora $A_1 \leq A_2 \leq A_3$ e $A_1 \leq B_1$. Quindi $A_1 \leq \min(A_3, B_1)$.

2. Se $B_2 \leq A_1, A_2, B_1$ e $B_3 \leq A_2, A_3, B_2$, allora $B_3 \leq B_2 \leq B_1$ e $B_3 \leq A_3$. Quindi $B_3 \leq \min(A_3, B_1)$.

3. Se $A_1 \leq B_2, A_2, B_1$ e $B_3 \leq A_2, A_3, B_2$, allora $A_1 \leq B_1$ e $B_3 \leq A_3$. Quindi $\min(A_1, B_3) \leq \min(A_3, B_1)$.

4. Se $B_2 \leq A_1, A_2, B_1$ e $A_2 \leq B_3, A_3, B_2$, allora $A_2 = B_2$, cioè J_2 è indifferente per J_1 e J_3. In questo caso l'elemento J_1 potrebbe o meno precedere l'elemento J_3, ma ciò non contraddice la transitività dato che prima ordiniamo J_1 e J_3 e poi posizioniamo J_2 in un punto qualsiasi.

□

Come volevamo quindi, $F(S^*) \leq F(S_O)$ per qualsiasi sequenza S_O, dato che S^* può essere ottenuta da S_O tramite scambi ripetuti tra elementi consecutivi, in base a (2.2), e ogni scambio porta ad un valore di F minore o uguale al precedente.

2.4.1 Algoritmo di Johnson

Usando la relazione (2.2), è possibile ordinare gli elementi in modo semplice ed efficace seguendo alcuni step.

1. Ordiniamo gli elementi A_i e B_i, $i = 1, 2, \ldots, n$ in due colonne.

i	A_i	B_i
1	A_1	B_1
2	A_2	B_2
.	.	.
.	.	.
n	A_n	B_n

2. Cerchiamo tra tutti i tempi $A_1, \ldots, A_n, B_1, \ldots, B_n$ quello minore.

3. Se il tempo trovato in 2. è associato alla prima macchina, posizioniamo per primo l'elemento J_i corrispondente al tempo A_i scelto.

4. Se il tempo trovato in 2. è associato alla seconda macchina, posizioniamo per ultimo l'elemento J_i corrispondente al tempo B_i scelto.

5. Cancelliamo dalla tabella i tempi A_i e B_i corrispondenti all'elemento J_i.

6. Ripetiamo gli step per i $2n - 2$ elementi rimasti in tabella.

7. In caso di parità, per ragioni di definitezza, consideriamo per primo l'elemento col pedice più piccolo. In caso di parità tra A_i e B_i, scegliamo A_i per procedere con gli step.

Esempio 2.3. Consideriamo i tempi $A_1, \ldots, A_5, B_1, \ldots, B_5$ definiti come segue:

i	A_i	B_i
1	4	5
2	4	1
3	30	4
4	6	30
5	2	3

Seguendo gli step precedentemente descritti troviamo la sequenza ottima $(5, 1, 4, 3, 2)$. Il ritardo totale per la sequenza è di 4 unità e il tempo totale è di 47 unità. Ribaltando l'ordine degli elementi si troverebbe un tempo totale pari a 78 unità che corrisponde al caso pessimo.

Possiamo visualizzare l'algoritmo nel modo seguente:

ALGORITMO DI JOHNSON
Sia T l'insieme dei lavori da schedulare; Sia $U = \{J_i \in T : A_i < B_i\}$ Sia $V = \{J_i \in T : A_i \geq B_i\}$ Ordinare gli elementi J_i di U in ordine crescente rispetto ad A_i; Ordinare gli elementi J_i di V in ordine decrescente rispetto ad B_i; $S = U \cup V$; Print S.

Capitolo 3

Anomalie di timing nel multiprocessing

Un sistema multiprocessore è un sistema di eleborazione in cui più processori lavorano parallelamente in modo, in generale, da ridurre i tempi di processamento. In un sistema di questo tipo è possibile che si presentino delle anoamalie: aumentando il numero di processori per esempio potrebbe aumentare il tempo totale necessario per eseguire tutte le attività richieste.

In questo capitolo presentiamo un modello di sistema multiprocessore e determiniamo quanto può essere influenzato, al massimo, il tempo totale per il processamento delle attività previste.[1]

3.1 Descrizione del sistema ed esempi di anomalie

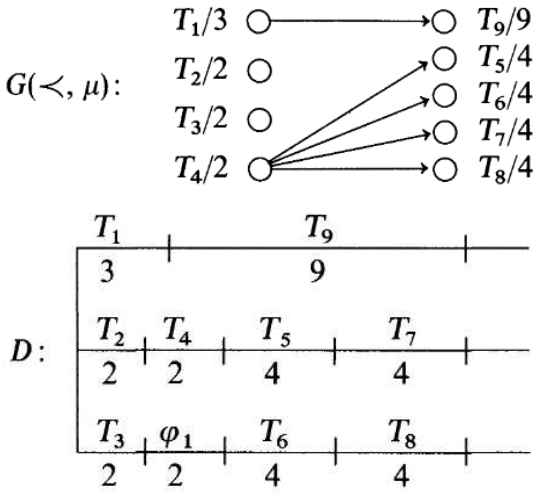
Supponiamo di avere n macchine identiche M_i, $i = 1, \ldots, n$, e un insieme di lavori $T = \{T_1, \ldots, T_r\}$ che devono essere eseguiti dalle macchine M_i. In T siano dati un ordinamento parziale $\prec$ e una funzione $\mu : T \to (0, \infty)$. Da quando una macchina M_i inizia a processare un lavoro T_j, essa continuerà senza interruzioni fino al completamento di tale attività, impiegando $\mu(T_j)$ unità di tempo (consideriamo le $\mu(T_j)$ unità di tempo come un intervallo semiaperto $[t, t + \mu(T_j))$). Per rispettare l'ordinamento parziale su T, se $T_i \prec T_j$ allora T_j non può essere processato prima del termine del processamento di T_i. Supponiamo infine di avere una sequenza $L = (T_{i_1}, \ldots, T_{i_r})$ formata da tutti i lavori in T ordinati in base a una *lista di priorità*. La macchina M_i esegue il lavoro T_j nel modo seguente: al tempo 0 tutti i processori analizzano la lista L dall'inizio cercando un lavoro T_i pronto per essere processato, cioè tale per cui in L non ci siano T_j con $T_j \prec T_i$. Il primo lavoro T_j pronto viene processato da M_i, che continua ad elaborarlo per un tempo $\mu(T_j)$. In generale, non appena una macchina M_i conclude il processamento di un lavoro, inizia il processamento del primo lavoro pronto nella lista L. In caso di assenza di un tale lavoro, M_i diventa inattiva (esegue un lavoro *vuoto* che indichiamo con φ_k). M_i resta inattiva fino a che un'altra macchina M_j completa il processamento di un lavoro, per poi analizzare L alla ricerca di un nuovo T_k da processare. Nel caso di più macchine pronte al processamento di un nuovo lavoro, assegnamo il primo lavoro pronto alla macchina con indice

[1] Tutti i risultati e le figure presenti in questo capitolo sono tratti da [9].

minore. Indichiamo con ω il tempo minimo necessario per eseguire tutti i lavori in T.

Vediamo un esempio in cui risulta chiaro il procedimento utilizzato dalle macchine per l'elaborazione di tutti i lavori ed evidenziamo alcune anomalie associate ad esso. Indichiamo l'ordinamento parziale $\prec$ in T e la funzione μ tramite un grafo orientato $G(\prec, \mu)$ in cui i vertici corrispondono ai lavori T_i e l'arco (T_i, T_j) appartiene al grafo se $T_i \prec T_j$. Etichettiamo ogni vertice con l'indicazione $T_j/\mu(T_j)$. Rappresentiamo inoltre l'attività di ogni macchina M_i tramite un diagramma D formato da n linee orizzontali, a rappresentare l'asse delle ascisse che inizia dal tempo 0, in cui sono indicati i lavori (con i rispettivi tempi) processati da M_i.

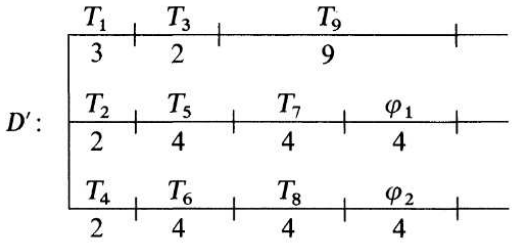
Esempio 3.1. $n = 3$, $L = (T_1, T_2, \ldots, T_9)$.



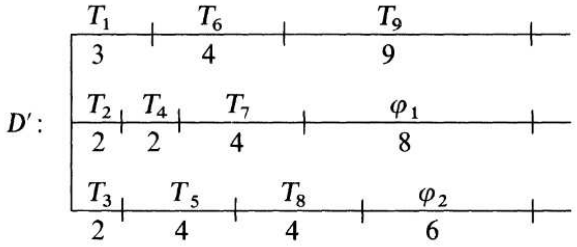
Per ogni intervallo in D indichiamo il lavoro T_i che viene processato durante la sua durata (sopra), e la sua durata (sotto).

Nell'esempio che stiamo considerando, abbiamo $\omega = 12$. Per definizione, ω è funzione di L, μ, $\prec$ ed n. Modifichiamo il valore di questi quattro parametri e studiamone l'effetto nella variazione di ω.

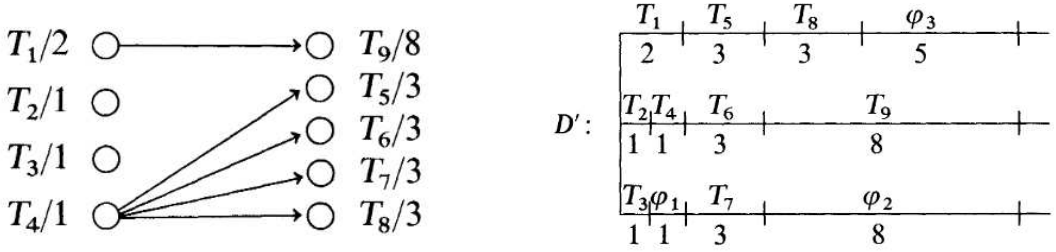
1. Sostituiamo L con $L' = (T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$ e lasciamo gli altri tre parametri invariati. Ora $\omega' = \omega'(L', \mu, \prec, n) = 14$.



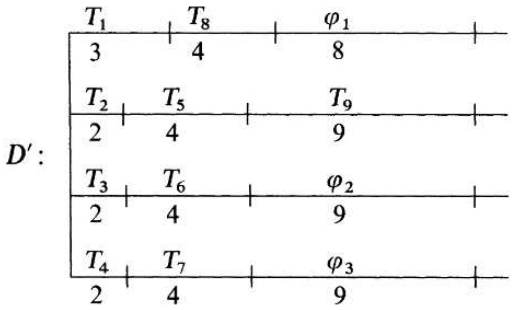
2. Modifichiamo $\prec$ per ottenere $\prec'$ rimuovendo $T_4 \to T_5$ e $T_4 \to T_6$. In questo caso $\omega' = \omega'(L, \mu, \prec', n) = 16$.



3. Diminuiamo μ per ottenere $\mu'(T_i) = \mu'(T_i) - 1 \; \forall i$. In questo caso anche $G(\prec, \mu)$ viene modificato, e $\omega' = \omega'(L, \mu', \prec, n) = 13$.



4. Aumentiamo n da 3 a $4 = n'$. Ora $\omega' = \omega'(L, \mu, \prec, n') = 15$.



Contrariamente a quanto si potrebbe immaginare, rendere meno stringente $\prec$, diminuire μ o aumentare ν fa in ogni caso aumentare ω.

Presentiamo ora un *bound* entro il quale "si muove" ω. Il *bound* presentato è ottimo, nel senso che non esiste una funzione nelle stesse variabili che sia migliore per il problema dato.

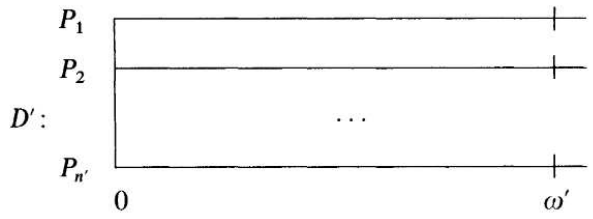
3.2 Il *bound* generale

Supponiamo di avere un insieme T di attività (o lavori) che vorremmo eseguire due volte distinte. Associati alla prima esecuzione abbiamo la funzione μ, l'ordine parziale $\prec$, la lista di priorità L e un insieme di n macchine identiche $M_i, i = 1, \ldots, n$. Associati alla seconda esecuzione abbiamo invece la funzione $\mu' \leq \mu$, l'ordine parziale $\prec' \subseteq \prec$, la lista di priorità L' e n' macchine identiche $M_i, i = 1, \ldots, n'$. Indichiamo con ω e ω' i rispettivi tempi finali.

Teorema 3.1.

$$\frac{\omega'}{\omega} \leq 1 + \frac{n-1}{n'}$$

Dimostrazione. Consideriamo il diagramma D' che si ottiene eseguendo i lavori T_i con i parametri descritti sopra.



Possiamo dividere i tempi nell'intervallo $[0, \omega')$ in due sottoinsiemi A e B, dove A contiene tutti i tempi in cui tutte le macchine stanno processando un lavoro di T, mentre B contiene i tempi in cui almeno una macchina è inattiva (ma non tutte le macchine lo sono). Sia A che B sono unione disgiunta di intervalli temporali semiaperti. Sia T_{j_1} un lavoro che in D' termina al tempo ω' e sia $S(T_{j_1})$ il tempo d'inizio dell'esecuzione di T_{j_1}.

Se $S(T_{j_1}) \in B$ e non è un punto di frontiera per l'insieme B, allora per come è stato definito B esiste una macchina M_i che per qualche $\varepsilon > 0$ rimane inattiva in $[S(T_{j_1} - \varepsilon, S(T_{j_1}))$. Viene spontaneo chiedersi come mai il processamento di T_{j_1} non sia iniziato prima nonostante ci fosse una macchina M_i inattiva prima e durante il tempo $S(T_{j_1})$. L'unica possibilità è che ci sia un lavoro T_{j_2} in D' tale che $T_{j_2} \prec' T_{j_1}$ che viene completato al tempo $S(T_{j_1})$ (in questo caso necessariamente T_{j_1} non può iniziare prima del tempo $S(T_{j_1})$).

Supponiamo ora che $S(T_{j_1}) \in A$ o $S(T_{j_1}) \neq 0$ sia un punto di frontiera per l'insieme B ed esista $x < S(T_{j_1})$ tale che $x \in B$. Sia $x_1 = sup\{x : x < S(T_{j_1}) \; e \; x \in B\}$. Per come sono stati definiti A e B, vediamo che $x_1 \in A$ e, per qualche macchina M_i e qualche $\varepsilon > 0$, M_i è inattiva nell'intervallo temporale $[x_1 - \varepsilon, x_1)$. Ci chiediamo di nuovo come mai il processamento di T_{j_1} non sia iniziato durante questo intervallo, prima di $S(T_{j_1})$. L'unica risposta possibile è che in questo intervallo fosse processato un altro lavoro T_{j_2} da cui T_{j_1} dipendeva rispetto all'ordinamento sull'insieme T. Se non fosse questo il caso, allora sicuramente sarebbe iniziato il processamento di T_{j_1} o di qualche suo predecessore.

In entrambi i casi ($S(T_{j_1}) \in A$ o B), abbiamo visto che o esiste un lavoro T_{j_2}, completato al tempo $F(T_{j_2})$, tale che $T_{j_2} \prec' T_{j_1}$ e $y \in [F(T_{j_2}), S(T_{j_1}))$ implica $y \in A$, oppure $x < S(T_{j_1})$ implica $x \in A$ o $x < 0$. Possiamo ripetere questa costruzione induttivamente, formando $T_{j_3}, T_{j_4}, \ldots$, etc., fino ad arrivare ad avere un lavoro T_{j_m} per cui $x < S(T_{j_m})$ implichi $x \in A$ o $x < 0$.

Abbiamo quindi dimostrato l'esistenza di una catena di lavori

$$T_{j_m} \prec' T_{j_{m-1}} \prec' \ldots \prec' T_{j_n} \prec' T_{j_1} \tag{3.1}$$

in D' tale che ad ogni tempo $T \in B$ esiste un T_{j_k} che sta venendo processato. La cosa importante da notare è che

$$\sum_{\varphi'_i \in D'} \mu'(\varphi'_i) \leq (n'-1) \sum_{k=1}^{m} \mu'(T_{j_k}), \tag{3.2}$$

dove la somma a sinistra è fatta su tutti i *lavori vuoti* φ'_i in D'. Ma la catena (3.1) e l'ipotesi $\prec' \subseteq \prec$ implicano

$$T_{j_m} \prec T_{j_{m-1}} \prec \ldots \prec T_{j_n} \prec T_{j_1}. \tag{3.3}$$

Quindi

$$\omega \geq \sum_{k=1}^{m} \mu(T_{j_k}) \geq \sum_{k=1}^{m} \mu'(T_{j_k}). \tag{3.4}$$

Di conseguenza, per (3.2) e (3.4), abbiamo:

$$\begin{aligned} \omega' &= \frac{1}{n'}\left\{ \sum_{T_k \in T} \mu'(T_k) + \sum_{\varphi'_i \in D'} \mu'(\varphi'_i) \right\} \\ &\leq \frac{1}{n'}(n\omega + (n'-1)\omega). \end{aligned} \tag{3.5}$$

Infine da questo otteniamo

$$\frac{\omega'}{\omega} \leq 1 + \frac{n-1}{n'}. \tag{3.6}$$

□

Si può provare che questo *bound* è il migliore possibile. Si vede infatti che per $n = n'$, si può arrivare ad avere $\omega'/\omega = 2 - 1/n$ variando uno tra i parametri $L, \mu, \prec$. Si può notare inoltre che per $n = 1$, ω' non supera mai ω, mentre per $n > 1$ ω' può essere maggiore di ω anche con un n' grande.

Può sembrare ragionevole lavorare con una lista di priorità L *dinamica*, al posto della lista fissa considerata finora. Pensiamo per esempio di agire come segue: ogni volta in cui una macchina è libera (cioè non sta processando alcun lavoro), essa inizia immediatamente ad eseguire il lavoro disponibile attualmente "a capo" della catena di lavori non ancora processati più lunga (la somma dei tempi di processamento della catena è massima). Supponiamo di avere un tempo totale per finire di processare tutti i lavori pari a ω_L. Denotiamo con ω_0 il tempo finale totale minore possibile. Il *bound* migliore per il rapporto tra ω_L e ω_0 è dato da

$$\frac{\omega_L}{\omega_0} \leq 2 - \frac{2}{n+1},$$

che è solo un piccolo miglioramento rispetto al teorema precedente.

Esiste però un caso per cui è possibile migliorare significativamente il risultato, pur usando algoritmi che non richiedono un grosso sforzo computazionale, come vediamo nel prossimo paragrafo.

3.3 Il caso in cui $\prec$ è *vuoto*

Supponiamo di avere un insieme $T = \{T_1, \ldots, T_r\}$ di lavori, una funzione (temporale) $\mu : T \to (0, \infty)$ ed n processori (o macchine). Cerchiamo un algoritmo per ottenere un tempo finale totale ω tale che il rapporto ω/ω_0, con ω_0 il minimo tempo finale

possibile, sia molto vicino ad 1. In questo caso, dato che $\prec$ non incide sulla sequenza dei lavori, una macchina libera inizia sempre ad eseguire il lavoro disponibile con tempo necessario al completamento più lungo. Sia ω_L il tempo finale totale per questo algoritmo. Vale allora il seguente teorema.

Teorema 3.2.

$$\frac{\omega_L}{\omega_0} \leq \frac{4}{3} - \frac{1}{3n}, \tag{3.7}$$

e questo bound è il migliore possibile.

Dimostrazione. Assumiamo, per assurdo, che esistano un insieme di lavori $T = \{T_1, \ldots, T_r\}$ e una funzione $\mu : T \to (0, \infty)$ che contraddicono (3.7). Denotiamo $\mu(T_i)$ con α_i e rinumeriamo i lavori T_i così da avere

$$\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_r. \tag{3.8}$$

Assumiamo $n \geq 2$ (per $n = 1$ il teorema continua a valere) e che r sia minimo.

Notiamo per prima cosa che, per come abbiamo definito ω_L, l'ordine in cui vengono eseguiti i vari lavori corrisponde ad usare la lista di priorità $L = (T_1, T_2, \ldots, T_r)$.

Supponiamo che T_m sia il lavoro con l'ultimo tempo di completamento (che dovrà essere necessariamente pari a ω_L), con $m < r$. Se consideriamo l'insieme troncato $T' = \{T_1, \ldots, T_m\}$ con la lista di priorità $L' = (T_1, \ldots, T_m)$, vediamo che il tempo di esecuzione ω' di T', rispettando L', è proprio ω_L. D'altra parte per il valore ottimo ω'_0 di T', si ha che $\omega'_0 \leq \omega_0$, dove ω_0 indica il tempo ottimo per l'insieme T. Dunque:

$$\frac{\omega'}{\omega_0} \geq \frac{\omega_L}{\omega_0} > \frac{4}{3} - \frac{1}{3n}, \tag{3.9}$$

e l'insieme T' forma un controesempio "più piccolo" per il teorema, ma questo contraddice l'ipotesi di minimalità su r. Assumiamo allora che T_r sia l'unico lavoro che termina al tempo ω_L.

È immediato vedere che

$$\omega_0 \geq \frac{1}{n} \sum_{i=1}^{r} \alpha_i. \tag{3.10}$$

Inoltre, se denotiamo con τ il tempo in cui T_r inizia ad essere processato, abbiamo

$$\sum_{i=1}^{r-1} \alpha_i \geq n\tau, \qquad \omega_L = \tau + \alpha_r \tag{3.11}$$

dato che nel modello che stiamo considerando nessuna macchina è inattiva prima dell'avvio di T_r. Allora valgono le seguenti relazioni:

$$\begin{aligned} \frac{\omega_L}{\omega_0} = \frac{\tau + \alpha_r}{\omega_0} &\leq \frac{\alpha_r}{\omega_0} + \frac{1}{n\omega_0} \sum_{i=1}^{r-1} \alpha_i \\ &= \frac{(n-1)\alpha_r}{n\omega_0} + \frac{1}{n\omega_0} \sum_{i=1}^{r} \alpha_i \\ &\leq \frac{(n-1)\alpha_r}{n\omega_0} + 1. \end{aligned}$$

Ora, dato che T contraddice (3.7), abbiamo

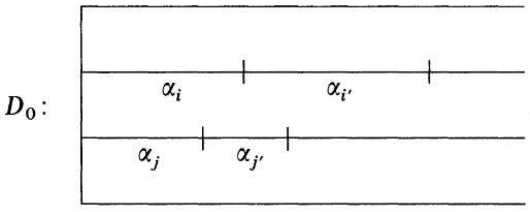
$$1 + \frac{(n-1)\alpha_r}{n\omega_0} \geq \frac{\omega_L}{\omega_0} > \frac{4}{3} - \frac{1}{3n}$$
$$\frac{(n-1)\alpha_r}{n\omega_0} > \frac{1}{3} - \frac{1}{3n} = \frac{n-1}{3n},$$

cioè, infine,

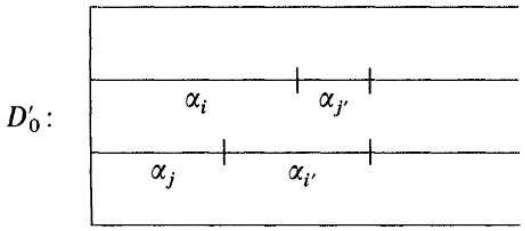
$$\alpha_r > \frac{\omega_0}{3}. \tag{3.12}$$

Quindi se (3.7) è falsa, allora in una soluzione ottima, con diagramma temporale D_0, nessuna macchina può processare più di due lavori.

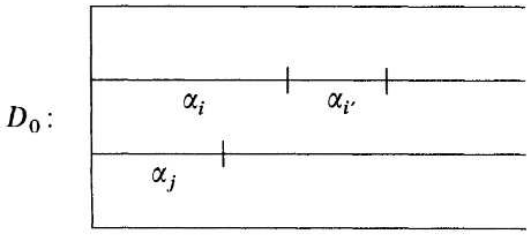
Supponiamo di avere un diagramma D_0 con la configurazione seguente, con $\alpha_i > \alpha_j$, $\alpha_{i'} > \alpha_{j'}$.



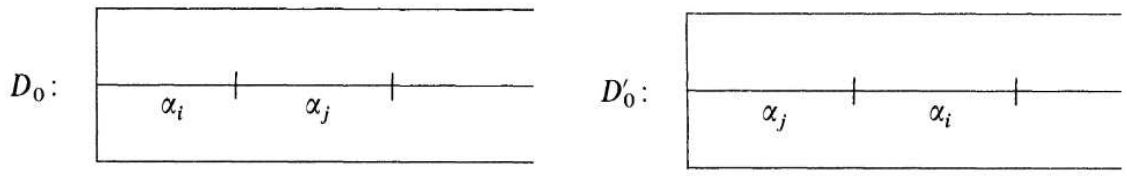
Se scambiamo $\alpha_{i'}$ e $\alpha_{j'}$ a formare D_0', allora in D_0' il nuovo tempo finale ω' soddisfa $\omega' \leq \omega_0$, cioè questo unico scambio non causa un aumento di ω_0.



Allo stesso modo, avendo una configurazione in D_0 come nella figura seguente, con $\alpha_i > \alpha_j$, spostare α_i nella linea con α_j non causerebbe un aumento di ω_0.



Chiamiamo queste due operazioni di modifica di D_0 *operazioni di Tipo 1.* Chiamiamo invece *operazione di Tipo 2* una modifica di D_0 (Figura 3.3(e)), con $\alpha_i < \alpha_j$, nella forma D_0' (Figura 3.3(f)). Anche questa operazione non cambia il valore di ω_0.



(e) (f)

Figura 3.1: Operazione di Tipo 2

Definiamo, per ogni diagramma temporale D, una funzione $\varphi(D)$ come segue: sia F_i il minimo tempo t per cui per ogni $t' \geq t$ la macchina M_i sia inattiva in D. Allora:

$$\varphi(D) = \sum_{1 \leq i < j \leq n} |F_i - F_j|.$$

Si può vedere che:

1) Se D' è ottenuto da D tramite un'operazione di Tipo 1, allora $\varphi(D') < \varphi(D)$;

2) Se D' è ottenuto da D tramite un'operazione di Tipo 2, allora $\varphi(D') = \varphi(D)$.

Supponiamo ora di partire da un diagramma D_0 e applicare tutte le possibili operazioni di Tipo 1 e 2 fino ad ottenere un diagramma D^* che non abbia configurazioni interne tali da permettere altre operazioni di Tipo 1 e 2 su di esso. L'esistenza di un tale D^* segue dal fatto che:

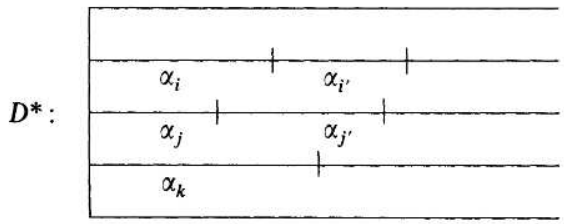
-esistono un numero finito di modi per collocare gli r lavori nelle n macchine disponibili,

-tra due operazioni di Tipo 1 possiamo applicare solo un numero finito di operazioni di Tipo 2,

-per 1), in generale possiamo applicare solo un numero finito di operazioni di Tipo 1.

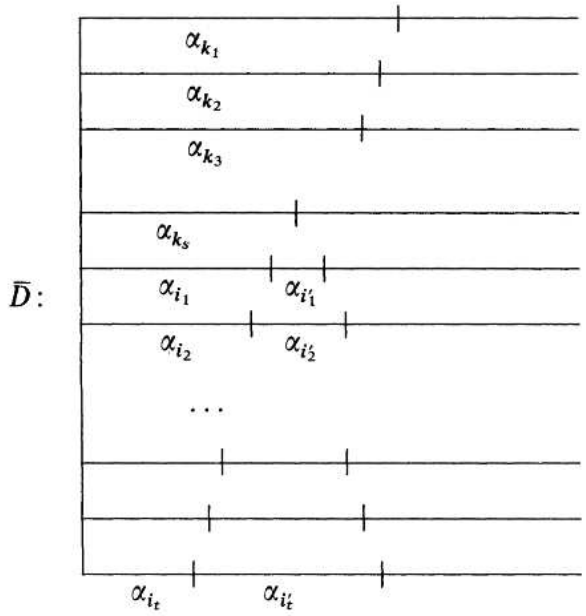
Dunque per ogni configurazione possibile in D^* della forma seguente, abbiamo che

$$\alpha_i > \alpha_{j'} \quad \text{implica} \quad \alpha_{j'} \geq \alpha_{i'}, \quad \alpha_i \leq \alpha_k \quad \text{e} \quad \alpha_i > \alpha_{i'}. \tag{3.13}$$



Possiamo allora portare D^* nella forma $\bar{D}$
dove

$$\alpha_{k_1} \geq \alpha_{k_2} \geq \ldots \geq \alpha_{k_s},$$
$$\alpha_{i_1} \geq \alpha_{i_2} \geq \ldots \geq \alpha_{i_t},$$
$$\alpha_{i'_1} \leq \alpha_{i'_2} \leq \ldots \leq \alpha_{i'_t} \quad \text{per (3.13).}$$

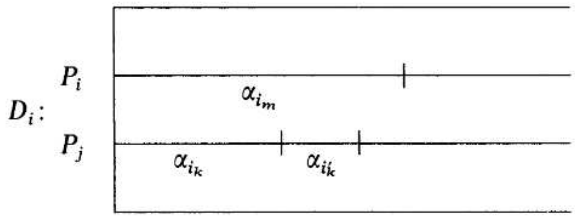


Ma (3.13) implica anche $\alpha_{k_s} \geq \alpha_{i_1}$ e $\alpha_{i_t} \geq \alpha_{i_{t'}}$. Combinando queste relazioni otteniamo

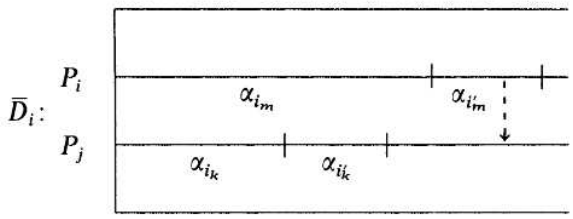
$$\alpha_{k_1} \geq \ldots \geq \alpha_{k_s} \geq \alpha_{i_1} \geq \ldots \geq \alpha_{i_t} \geq \alpha_{i'_t} \geq \ldots \geq \alpha_{i'_1}. \tag{3.14}$$

Dato che nessuna delle operazioni applicate in D_0 causa un aumento di ω_0, per l'ottimalità di ω_0 anche il tempo totale di completamento di $\bar{D}$ sarà pari a ω_0.

Osserviamo che il diagramma $\bar{D}$ è molto simile, a patto di rinominare i lavori con uguale lunghezza, al diagramma D_L ottenuto tenendo conto della lista $(T_1, \ldots, T_r)$ ordinata in base alla lunghezza decrescente dei lavori. In effetti D_L può differire da $\bar{D}$ solo nell'assegnamento dei lavori $T_{i'_e}$. Una differenza tra i due cioè può verificarsi solo se per una coppia di lavori $T_{i_k}, T_{i'_k}$ abbiamo $\alpha_{i_k} + \alpha_{i'_k} \leq \alpha_{i_m}$, per qualche $m < k$.



In questo caso in D_L il lavoro $T_{i'_m}$ di lunghezza (temporale) $\alpha_{i'_m}$ potrebbe essere assegnato alla macchina P_j invece che a P_i. Se questa situazione fosse possibile, allora in $\bar{D}$ avremmo quanto segue in figura, e sarebbe possibile spostare $\alpha_{i'_m}$ da P_i a P_j, mantenendo un tempo di completamento pari a ω_0.



Viene però a crearsi una contraddizione, dato che per (3.12) ogni macchina non può processare più di due lavori.

Concludiamo allora che D_L e $\bar{D}$ sono "isomorfi" e $\omega_0 = \omega_L$. Questo però contraddice l'ipotesi di assurdo ($\omega_L/\omega_0 > 4/3 - 1/(3n)$) e prova la validità del *bound* fornito dal teorema.

Resta da dimostrare ora che il *bound* proposto è il migliore possibile.

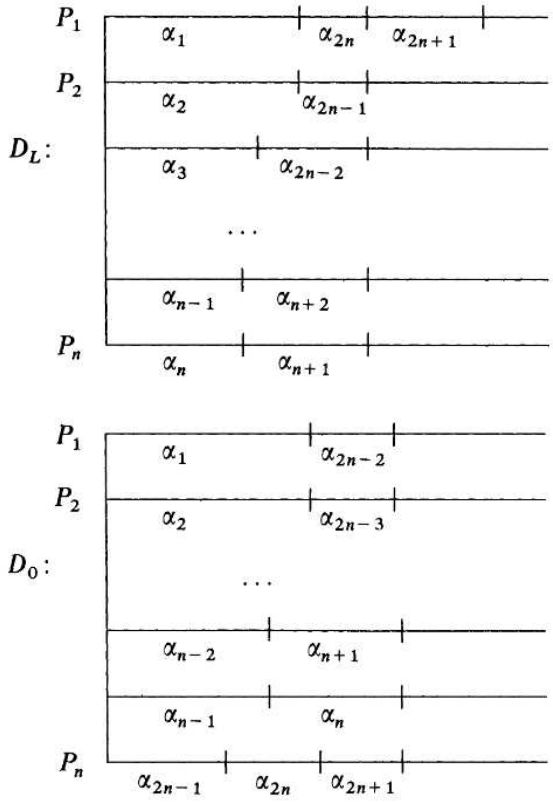
Consideriamo il seguente insieme di lunghezze α_i di lavori:

$$(\alpha_1, \alpha_2, \ldots, \alpha_r) = (2n-1, 2n-1, 2n-2, 2n-2, \ldots, n+1, n+1, n, n, n),$$

dove $r = 2n+1$. In particolare abbiamo

$$\alpha_k = 2n - \left[\frac{k+1}{2}\right], \quad k = 1, \ldots, 2n \qquad \text{e} \qquad \alpha_{2n+1} = n.$$

In questo caso allora i diagrammi D_L e D_0 sono nella forma seguente, e $\omega_L = 4n-1$, $\omega_0 = 3n$.



Dunque

$$\frac{\omega_L}{\omega_0} = \frac{4}{3} - \frac{1}{3n}$$

e questo *bound* è il migliore possibile. □

Appendice A

Algoritmi di Horn

Presentiamo alcuni algoritmi proposti da Horn [7] per la minimizzazione del ritardo massimo e del ritardo totale nel caso di una macchina singola relativamente alla schedulazione di lavori che necessitano di una sola operazione. Ad ogni lavoro J_i associamo un tempo di rilascio r_i, una data di scadenza d_i e un tempo di completamento t_i.

A.1 Minimizzare il ritardo massimo

L'algoritmo che segue opera nel caso in cui ogni lavoro è disponibile al tempo r_i, con scadenza d_i. Supponiamo che gli r_i siano in generale diversi tra loro.

Algoritmo 1. Sia E il sottoinsieme di lavori disponibili all'istante A_1 più piccolo. Sia A_2 l'istante successivo ad A_1 in cui altri lavori sono disponibili (eventualmente $A_2 = \infty$ se tutti i lavori sono disponibili all'instante A_1). Sia J_i il lavoro in E con data di scadenza d_i minore delle scadenze degli altri lavori in E. Sia $L_1 = \min\{t_i, A_2 - A_1\}$. Assegnamo il lavoro J_i all'intervallo $[A_1, A_1 + L_1]$. Se J_i non viene completato, riduciamo t_i di L_1, altrimenti eliminiamo J_i dall'insieme di lavori. Ripetiamo questa operazione per tutti i lavori rimanenti fissando un tempo minimo $A_1 + L_1$ di disponibilità dei lavori. Continuiamo fino a che tutti i lavori sono stati processati dall'algoritmo.

Dimostriamo che l'algoritmo minimizza il ritardo massimo come richiesto.

Dimostrazione. Indichiamo con $I \to L$ il fatto che l'intervallo L segue direttamente l'intervallo I, cioè in particolare $\max I = \min L$. Supponiamo che S sia una schedulazione ottima per i lavori e che in essa l'intervallo $[A_1, A_1 + L_1]$ sia composto dagli intervalli $I_1, I_2, \ldots, I_m$, con $I_1 \to \ldots \to I_m$, dove ogni I_j rappresenta un intervallo in cui un lavoro viene o non viene processato. Se nell'intervallo I_1 sta venendo processato un lavoro J_k diverso dal lavoro J_i scelto dall'algoritmo, e non il lavoro J_i, allora $J_k \in E$. Modifichiamo la schedulazione in modo che durante l'intervallo I_1 sia processato J_i ($|I_1| \leq L_1 \leq t_i$) processando il lavoro J_k invece di J_i in un qualsiasi altro intervallo in S di lunghezza massima $|I_1|$ in cui veniva precedentemente processato J_i. Dato che la scadenza del lavoro J_i è tanto presto quanto almeno quella

di J_k (infatti $J_k \in E$), è chiaro che una tale modifica non incide sul ritardo massimo generale della schedulazione.

Osservazione A.1. Se I_1 è un periodo di inattività per il lavoro J_i, spostiamo $|I_1|$ unità di tempo utile per il lavoro J_i in I_1.

Ripetiamo il procedimento per tutti i sottointervalli $I_2, I_3, \ldots, I_n$ in S sostituendo ogni periodo di processamento di un lavoro diverso da J_k, o un periodo di tempo di inattività, con un periodo di processamento di J_i. Anche in questo caso, per la stessa ragione, il ritardo massimo complessivo non è aumentato. In questo modo viene a crearsi una nuova schedulazione S_1 in cui il lavoro J_i viene processato nell'intervallo $[A_1, A_1 + L_1]$.

Risulta evidente che non considerando $[A_1, A_1 + L_1]$ e riducendo t_i a $t_i - L_1$, possiamo ripetere lo stesso procedimento in $[A_1 + L_1, \infty]$ applicando l'algoritmo ripetutamente, senza aumentare il ritardo massimo, così da ottenere una schedulazione ottima per il problema considerato. □

A.2 Minimizzare il ritardo totale

In questo caso supponiamo che non siano date scadenze ai lavori. Vogliamo minimizzare $\sum_i (f_i - r_i)$, dove f_i indica l'istante in cui viene completato il lavoro J_i. Se tutti i lavori sono disponibli al tempo 0 il problema è facilmente risolvibile schedulando i lavori in ordine crescente rispetto ai tempi di completamento. Supponiamo di avere dei tempi di rilascio r_i generici e siano ammessi *job-splitting*.

Algoritmo 2. Sia E il sottoinsieme di lavori disponibili all'istante minimo A_1. Sia A_2 l'istante successivo ad A_1 in cui altri lavori sono disponibili ($A_2 = \infty$ se tutti i lavori sono disponibili all'instante A_1). Sia J_i il lavoro in E con tempo di completamento t_i più piccolo. Sia $L_1 = \min\{t_i, A_2 - A_1\}$. Assegnamo il lavoro J_i all'intervallo $[A_1, A_1 + L_1]$. Se J_i non viene completato, riduciamo t_i di L_1, altrimenti eliminiamo J_i dall'insieme di lavori. Ripetiamo questa operazione per tutti i lavori rimanenti fissando un tempo minimo $A_1 + L_1$ di disponibilità dei lavori. Continuiamo fino a che tutti i lavori sono stati processati dall'algoritmo.

Dimostriamo che l'algoritmo minimizza il ritardo totale.

Dimostrazione. Come nella prova dell'Algoritmo 1, sia S una schedulazione ottima per i lavori e in essa l'intervallo $[A_1, A_1 + L_1]$ sia composto dagli intervalli $I_1, I_2, \ldots, I_m$, con $I_1 \to \ldots \to I_m$, dove ogni I_j rappresenta un intervallo in cui un lavoro viene o non viene processato. Se nell'intervallo I_1 sta venendo processato un lavoro J_k diverso dal lavoro J_i scelto dall'algoritmo, e non il lavoro J_i, allora $J_k \in E$. Consideriamo tutti gli intervalli in S tali che durante ognuno di essi viene processato J_i o J_k; denotiamo tali intervalli con $I_1 = P_1 < P_2 < \ldots < P_r$, dove scrivendo $P_{j-1} < P_j$ intendiamo che P_{j-1} precede P_j. Modifichiamo la sequenza in modo che nelle prime t_i unità di tempo in $\bigcup P_j$, determinate prendendo una sottosequenza $P_1, P_2, \ldots$ ed eventualmente un intervallo finale parziale (consideriamo

solo una parte dell'intervallo) così che la lunghezza totale sia t_i, venga processato il lavoro J_i e nelle ultime t_k unità di tempo ($\sum_j |P_j| = t_i + t_k$) venga processato J_k.

In questa nuova schedulazione S', il lavoro che termina più tardi tra J_i e J_k, sia J_k, termina alla fine di P_r, diciamo al tempo T_r, esattamente quando sarebbe terminato il primo tra J_i e J_k nella schedulazione iniziale S. Il lavoro che termina prima, J_i, nella nuova schedulazione S' termina a qualche tempo T', non più tardi del tempo T in cui sarebbero terminati J_i o J_k in S, dato che $t_i \leq d_k$ e il lavoro J_i è sempre stato processato il prima possibile in S'. Il ritardo totale per J_i e J_k in S', $(T_r - r_k) + (T' - r_i)$, è dunque piccolo almeno tanto quanto il ritardo totale in S, $(T_r - r_k) + (T - r_i)$ o $(T_r - r_i) + (T - r_k)$ in base a quale lavoro termina prima in S. In S' quindi non aumenta il ritardo totale.

Come nell'algoritmo precedente, ripetute modifiche nella schedulazione assegnano tutto l'intervallo $[A_1, A_1 + L_1]$ al processamento di J_i senza aumentare il ritardo totale, e iterazioni simili mostrano che l'ordine di processamento dato dall'algoritmo in generale non aumenta il ritardo totale. Dunque la schedulazione è ottima per il problema considerato. □

Bibliografia

[1] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria Scheduling. Theory, Models and Algorithms.* Springer, 2006.

[2] Kenneth R. Baker and Dan Trietsch. *Principles of sequencing and scheduling.* John Wiley & Sons Inc., 2009.

[3] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.

[4] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

[5] J. M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

[6] S. Sahni. Preemptive scheduling with due dates. *Operations Research*, 27(5):925–934, 1979.

[7] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

[8] S. M. Johnson. Optimal two- and three-stage production schedules with setup times includes. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[9] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.