

Parallel identical processor scheduling with weighted completion time

A column generation approach

Simone Cavana
219833{at}studenti.unimore.it

15 febbraio 2021

M = set di m macchine identiche

J = set di n job

p_j = tempo d'esecuzione di un job

w_j = peso di un job

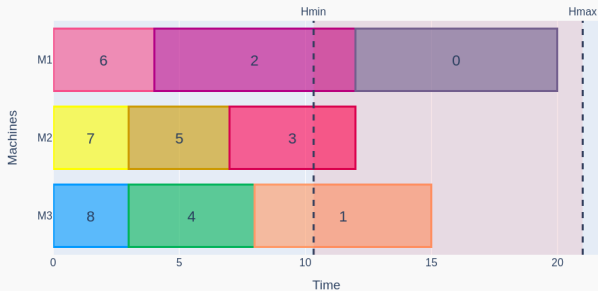
C_j = tempo di completamento di un job in una schedula

- ▶ Ogni macchina è disponibile dall'istante 0 e può elaborare al più un job per istante
- ▶ Non è ammessa preemption
- ▶ I job devono essere eseguiti in modo contiguo (no idle)

$P || \sum w_j C_j$

Esempio

job	w_j	p_j
1	1	8
2	4	7
3	7	8
4	5	5
5	5	5
6	6	4
7	7	4
8	7	3
9	9	3



- ▶ $H_{min} = (\sum_{j \in J} p_j - (m - 1)p_{max}) / m$
- ▶ $H_{max} = (\sum_{j \in J} p_j + (m - 1)p_{max}) / m$
- ▶ $p_{max} = \max_{j \in J} p_j$

Set-covering formulation

- ▶ Definiamo **schedula** $s \in S$, un insieme di job ammissibili assegnabile ad una qualsiasi macchina m
- ▶ L'**ordinamento di Smith** ci indica di effettuare un ordinamento in ordine decrescente sulla base del rapporto w_j/p_j per avere la condizione di ottimalità

$$a_{js} = \begin{cases} 1 & \text{se il job } j \text{ è assegnato alla schedula } s, \\ 0 & \text{altrimenti} \end{cases}$$

$$C_j(s) = \sum_{k=1}^j a_{ks} p_k$$

$$c_s = \sum_{j \in J} w_j a_{js} C_j(s) = \sum_{j \in J} w_j a_{js} \left[\sum_{k=1}^j a_{ks} p_k \right]$$

$$x_s = \begin{cases} 1 & \text{se la schedula } s \text{ è selezionata,} \\ 0 & \text{altrimenti} \end{cases}$$

Set-covering model

$$\begin{array}{ll}\min & \sum_{s \in S} c_s x_s \\ \text{s.t.} & \sum_{s \in S} x_s = m\end{array} \quad (1)$$

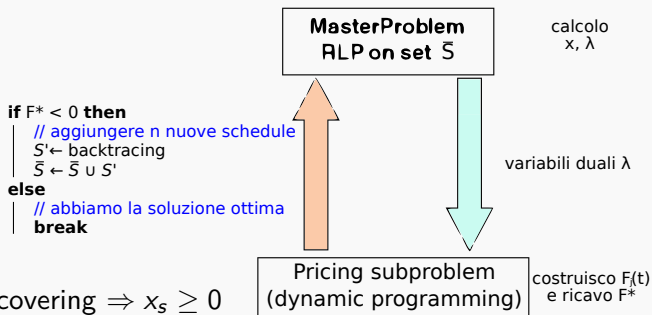
$$\sum_{s \in S} a_{js} x_s = 1, \quad j \in J \quad (2)$$

$$x_s \in \{0, 1\}, \quad s \in S \quad (3)$$

Costi ridotti (si ricavano dal duale):

$$\bar{c}_s = c_s - \sum_{j \in J} a_{js} \lambda_j = \sum_{j \in J} \left[w_j \left(\sum_{k=1}^j a_{ks} p_k \right) - \lambda_j \right] a_{js}$$

Column Generation Approach



- ▶ Rilasso set-covering $\Rightarrow x_s \geq 0$
- ▶ Risolvo RLP su \bar{S}
- ▶ Ricavo le variabili duali:
 - ▶ λ_0 è costante, relativa ad (1)
 - ▶ λ_j relative a (2)
- ▶ se $F^* < 0 \Rightarrow$ aggiungo le n schedule con costi ridotti più negativi tramite backtracing su $F_j(t)$

Pricing algorithm

Programmazione dinamica

$$P(j) = \sum_{k=1}^j p_k$$

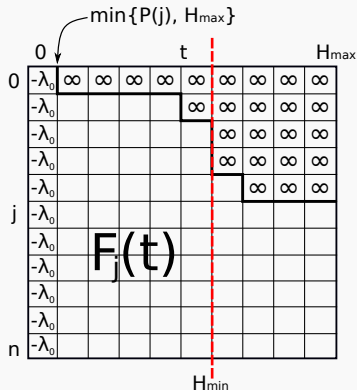
Inizializzazione:

$$F_j(t) = \begin{cases} -\lambda_0 & \text{se } j = 0, \text{ e } t = 0, \\ \infty & \text{altrimenti} \end{cases}$$

per i successivi step $j = 1, \dots, n$, $t = 0, \dots, \min\{P(j), H_{\max}\}$:

$$F_j(t) = \begin{cases} \min\{F_{j-1}(t), F_{j-1}(t - p_j) + w_j t - \lambda_j\} & \text{se } r_j + p_j \leq t \leq d_j \\ F_{j-1}(t) & \text{altrimenti} \end{cases}$$

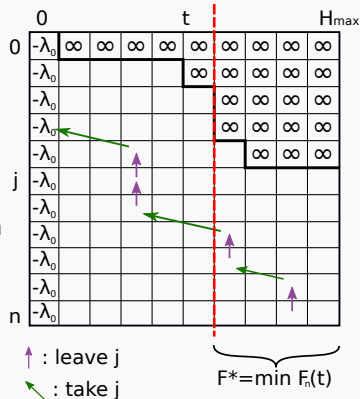
$$F^* = \min_{H_{\min} \leq t \leq H_{\max}} F_n(t)$$



Pricing algorithm

Backtracing

- ▶ $F_j(t)$ rappresenta i costi ridotti minimi dati dalla schedula composta dai job $\leq j$ che termina in t
- ▶ Le nuove schedule da inserire in \bar{S} sono quelle associate ad $F_n(t)$ più negative
- ▶ La strategia per risalire alla schedula dato il costo ridotto $F_j(t)$ è di ripercorrere a ritroso la matrice dove:
 - ▶ salgo in verticale se $F_j(t) == F_{j-1}(t)$ e non inserisco j in \bar{S}
 - ▶ salgo al job precedente e mi sposto nel tempo tanto quanto p_j , in modo da inserire j in \bar{S}



Randomized List Heuristic

Initialization

- ▶ $\bar{S} \rightarrow$ schedule iniziali
- ▶ $2000 \leq N \leq 5000$
- ▶ Estrazione basata su c_s
- ▶ NS migliorativo finale

Algorithm 1: Heuristic

Input: n, m, w, p, N

Output: \bar{S}

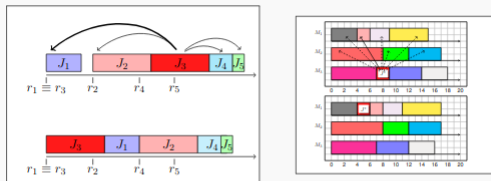
```
1  $\bar{S}, s \leftarrow \{\}$ 
2  $jobs \leftarrow \text{smith\_order}(n, w, p)$ 
3  $n\_iter \leftarrow 0$ 
4 while  $n\_iter < N$  do
5    $s \leftarrow \text{create\_rand\_sched}(jobs)$ 
6    $\bar{S} \leftarrow \bar{S} \cup s$ 
7    $n\_iter++$ 
8  $\bar{S} \leftarrow \text{extract\_best}(\bar{S}, 10)$ 
9 return  $\text{neighborhood\_search}(\bar{S})$ 
```

Randomized List Heuristic

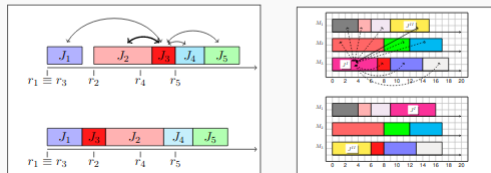
Neighborhood Search

- Insert: spostare un job da una macchina ad un altra
- Swap: scambiare due job schedulati su macchine differenti

► **Insert move:** move one “object” to another “position”



► **SWAP move:** exchange two “objects”



Integralità della soluzione

Branch & Bound

- ▶ Caso speciale frazionario che rispetta **Teorema 1**
- ▶ Bound dato da x^* , soluzione ottima di RLP

1. Branching sugli **intervalli d'esecuzione**



2. Branching sull'**immediato successore**



Benchmark

- ▶ 15 istanze “semplici” **fornite** da Barnes & Brennan
- ▶ Istanze generate **random** con:
 - ▶ m estratto da $[3,4,5]$
 - ▶ n da $[20, 30, 40, 50]$
 - ▶ tempi d'esecuzione e pesi:
 1. p_j ricavati da $[1, 10]$, w_j da $[10, 100]$
 2. sia p_j che w_j estratti da una distribuzione normale fra $[1, 100]$
 3. sia p_j che w_j estratti da una distribuzione normale fra $[1, 20]$

Risultati

Barnes & Brennan benchmark

Risultati

Random benchmark

Conclusioni

Bibliografia

- ▶ Marjan van den Akker, Han Hoogeveen, and Steef van de Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872, 1999.
- ▶ Marjan van den Akker, Han Hoogeveen, and Steef van de Velde. Applying Column Generation to Machine Scheduling, pages 303–330. Springer US, Boston, MA, 2005.
- ▶ J. Wesley Barnes and J. J. Brennan. An improved algorithm for scheduling jobs on identical machines. *AIIE Transactions*, 9(1):25–31, 1977.