Permutation Flow Shop Problem

Luca Di Marco
0333083
luca.dimarco.01@students.uniroma2.eu

August 30, 2023

Abstract

Questo documento ha l'obiettivo di descrivere la differenza di implementazione e di risultati ottenuti per il problema di permutation flow shop nei casi di modello a vincoli di posizione, modello a vincoli di precedenza e modello euristico.

1 Introduzione

Il Permutation Flow Shop Problem (PFSP) è un tipo di problema di programmazione nell'ambito della produzione: riguarda l'organizzazione di una sequenza di job da eseguire attraverso una serie di macchine in modo tale da minimizzare il makespan, cioé il tempo necessario per completare tutti i job. I job vengono processati in maniera sequenziale attraverso le varie macchine (un solo job per volta) ed ogni macchina lavora sulla stessa sequenza di job. Nel caso di studio successivo vengono prese in considerazione M macchine e N job. Nelle simulazioni, invece, il numero di macchine è fissato a tre. Di seguito vengono mostrati i vari modelli implementati ed i successivi risultati sperimentali.

2 Modello MIP con vincoli posizionali

2.1 Modello

Le variabili sono:

$$x_{k,j} = \begin{cases} 1, & \text{se job } j \text{ è assegnato alla posizione } k \text{ in } \pi \\ 0, & \text{altrimenti} \end{cases} \qquad k, j = 1, ..., n$$

$$C_{i,j} \in \mathbb{R}_{\geq 0} \quad \forall i \in M, \forall j \in J$$

La funzione obiettivo è:

$$min(C_{max})$$

dove

$$C_{max} = max_{1 \leq j \leq n}\{C_{m,\pi_j}\}$$

Indichiamo con:

- $p_{m,j}$ = tempo di processamento job j su macchina m;
- $C_{i,k}$ = tempo di completamento del job nella pos. k nella macchina i;

I vincoli, dunque, sono:

$$\sum_{j=1}^{n} x_{k,j} = 1 \quad k = 1, ..., n$$

$$\sum_{k=1}^{n} x_{k,j} = 1 \quad j = 1, ..., n$$

$$C_{1,1} \geq \sum_{j=1}^{n} x_{1,j} * p_{1,j} \quad j = 1, ..., n$$

$$C_{1,k} \geq C_{1,k-1} + \sum_{j=1}^{n} x_{k,j} * p_{1,j} \quad k = 2, ..., n$$

$$C_{i,k} \geq C_{i-1,k} + \sum_{j=1}^{n} x_{k,j} * p_{i,j} \quad i = 2, ..., m \quad k = 1, ..., n$$

$$C_{i,k} \geq C_{i,k-1} + \sum_{j=1}^{n} x_{k,j} * p_{i,j} \quad i = 1, ..., m \quad k = 2, ..., n$$

$$C_{i,k} \geq 0 \quad i = 1, ..., m \quad k = 1, ..., n$$

$$x_{k,j} \in \{0, 1\} \quad j, k = 1, ..., n$$

$$C_{max} \geq C_{m,j} \quad j \in J$$

3 Modello MIP con vincoli di precedenza

3.1 Modello

Le variabili sono:

$$x_{j,j'} = \begin{cases} 1, & \text{se job } j \text{ precede job } j' \text{ in } \pi \\ 0, & \text{altrimenti} \end{cases} \qquad 1 \leq j < j' \leq n$$

$$C_{i,j} \in \mathbb{R}_{\geq 0} \quad \forall i \in M, \forall j \in J$$

La funzione obiettivo è identica alla precedente:

$$min(C_{max})$$

dove

$$C_{max} = max_{1 \leq j \leq n}\{C_{m,j}\}$$

Indichiamo con:

- $p_{i,j}$ = tempo di processamento job j su macchina i;
- $C_{i,j}$ = tempo di completamento del job j nella macchina i;

I vincoli sono:

$$C_{i,j'} \geq p_{i,j'} + C_{i,j} - M * (1 - x_{j,j'}) \quad i = 1, ..., m \quad 1 \leq j < j' \leq n$$

$$C_{i,j} \geq p_{i,j} + C_{i,j'} - M * x_{j,j'} \quad i = 1, ..., m \quad 1 \leq j < j' \leq n$$

$$C_{i,j} \geq p_{i,j} \quad i \in M, j \in J$$

$$C_{max} \geq C_{m,j} \quad j \in J$$

$$C_{i,j} \geq C_{i-1,j} + p_{i,j} \quad j \in J \quad i = 2, ..., m$$

$$x_{j,j'} \in \{0, 1\} \quad j, j' \in J \quad 1 \leq j < j' \leq n$$

$$C_{i,j} \geq 0 \quad \forall i \in M \quad \forall j \in J$$

4 Modello euristico

4.1 Algoritmo NEH

L'algoritmo euristico NEH è implementato come segue:

1. INPUT: tempi di processamento di ogni job su ogni macchina $p_{j,1}, ..., p_{j,m}$;
2. Valuto il tempo di processamento di ogni job come: $T(j) = \sum_{i=1}^{m} p_{j,i}$;
3. Costruisco la sequenza $\pi = (\pi_1, ..., \pi_n)$ ordinando i job in ordine decrescente secondo $T(j)$;
4. Schedulo i primi due job π_1 e π_2 e trovo la migliore sequenza tra i due;
5. FOR j=3 TO n DO: inserisco π_j in tutte le posizioni della sequenza attualmente costruita e scelgo la migliore che minimizza C_{max} nell'utltima macchina e aggiorno π_{NEH} (sequenza ottima) END FOR;
6. OUTPUT: π_{NEH}.

5 Risultati

5.1 Svolgimento dei test

I test sono stati eseguiti su una macchina MacBook Air M1 con 8 GB di RAM e 8 Core CPU. Il solver scelto per l'esecuzione dei test è Gurobi. Le tabelle si compongono di tre run differenti, ognuna con una istanza diversa generata mediante il file *createinstance.py*: in ogni run, i tre diversi modelli condividono la stessa istanza per poter confrontare i valori.
Le istanze vengono generate modificando il numero ed i tempi di processamento dei diversi job. Le macchine, invece, sono costanti e pari a tre.

- Nella cartella *Modello1* è possibile trovare il file che tratta l'implementazione del modello posizionale in Python;
- Nella cartella *Modello2* è possibile trovare il file che tratta l'implementazione del modello a precedenze in Python;
- Nella cartella *Euristico* è possibile trovare il file che tratta l'implementazione del modello euristico in Python.

I file sopra descritti sono scaricabili al seguente collegamento: code here.

5.2 Risultati delle simulazioni

Di seguito vengono mostrati i risultati delle varie simulazioni. Ad ogni istanza viene imposto un tempo limite di 600s di esecuzione. Le istanze vengono modificate cambiando i tempi di processamento uniformi dei vari job. I dati nelle tabelle sono ordinati come segue: C_{max}/tempo esecuzione/GAP.

5.3 Discussione dei risultati

Dai risultati di seguito illustrati, è possibile notare come il modello posizionale restituisca valori ottimi in tempi molto brevi, con una minima di 0.1s ad una massima di 0.35s. In tutte le simulazioni il GAP rimane allo 0%, ottenendo quindi il risultato ottimo. Il modello con vincoli di precedenza ha buone prestazioni in presenza di pochi job (massimo 10) ma al cresecere del numero di job (da 15 a 40) non riesce a calcolare una soluzione ottima con un GAP dello 0% nel tempo limite di 600s: la soluzione da lui trovata è, comunque, nella maggioranza dei casi identica alla soluzione ottima trovata dal modello posizionale e solo alcune volte si discosta di poco. Il GAP rimane, però, tra il 20% e l'80%. Da tenere conto la scelta del big M nel modello a precedenze: per avere un modello funzionante non deve essere minore del tempo di completamento dell'ultimo job sull'ultima macchina. Per ottenere le migliori performance, quindi, il valore del big M deve essere modificato per ogni diverso quantitativo di job, così da non inserirlo nè troppo grande (rallentando così il solver) nè troppo piccolo, rendendo inconsistente il modello da risolvere e creando una infeasibility. L'algoritmo euristico, come potevamo aspettarci, fornisce risultati vicini all'ottimo ma spesso non migliori dei precedenti due modelli, ma in un tempo minore. In alcune casi, però, riesce a restituire valori identici all'ottimo calcolato dal modello posizionale.

Table 1: 10 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	59/0.03s/0%	59/1.23s/0%	60/0.0009s
2	57/0.01s/0%	57/0.28s/0%	57/0.0009s
3	54/0.01s/0%	54/0.61s/0%	56/0.0009s

Table 2: 10 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	222/0.01s/0%	222/7.11s/0%	224/0.0009s
2	228/0.03s/0%	228/6.50s/0%	228/0.0009s
3	252/0.03s/0%	252/18.53s/0%	255/0.0009s

Table 3: 15 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	91/0.01s/0%	91/**600s**/23%	91/0.0026s
2	97/0.02s/0%	97/**600s**/34%	99/0.0025s
3	92/0.01s/0%	92/**600s**/32%	92/0.0026s

Table 4: 15 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	336/0.22s/0%	337/**600s**/30%	336/0.0026s
2	340/0.02s/0%	340/**600s**/40%	342/0.0026s
3	373/0.02s/0%	373/**600s**/39%	375/0.0024s

Table 5: 20 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	113/0.02s/0%	113/**600s**/38%	116/0.0057s
2	116/0.06s/0%	116/**600s**/40%	119/0.0057s
3	102/0.05s/0%	102/**600s**/39%	102/0.0054s

Table 6: 20 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	481/0.02s/0%	481/**600s**/57%	485/0.0058s
2	408/0.04s/0%	408/**600s**/53%	410/0.0058s
3	456/0.02s/0%	456/**600s**/47%	456/0.0057s

Table 7: 25 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	149/0.05s/0%	149/**600s**/67%	149/0.0096s
2	145/0.06s/0%	145/**600s**/60%	149/0.0086s
3	159/0.03s/0%	159/**600s**/63%	159/0.0096s

Table 8: 25 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	555/0.05s/0%	555/**600s**/62%	557/0.0011s
2	560/0.04s/0%	560/**600s**/62%	561/0.009s
3	566/0.04s/0%	566/**600s**/63%	568/0.001s

Table 9: 30 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	168/0.26s/0%	169/**600s**/70%	169/0.0176s
2	173/0.04s/0%	173/**600s**/71%	173/0.0167s
3	186/0.35s/0%	187/**600s**/69%	186/0.0157s

Table 10: 30 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	656/0.07s/0%	656/**600s**/65%	657/0.017s
2	621/0.07s/0%	623/**600s**/65%	622/0.017s
3	692/0.05s/0%	692/**600s**/64%	694/0.017s

Table 11: 35 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	211/0.07s/0%	211/**600s**/76%	211/0.027s
2	208/0.05s/0%	208/**600s**/73%	208/0.025s
3	210/0.06s/0%	210/**600s**/74%	212/0.025s

Table 12: 35 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	785/0.05s/0%	785/**600s**/66%	786/0.025s
2	735/0.09s/0%	735/**600s**/67%	735/0.025s
3	774/0.05s/0%	774/**600s**/68%	774/0.026s

Table 13: 40 job con p_{ij} Unif(1,10)

Run	Posizionale	Precedenze	Euristico
1	243/0.09s/0%	243/**600s**/84%	243/0.036s
2	244/0.09s/0%	244/**600s**/81%	245/0.036s
3	226/0.10s/0%	226/**600s**/79%	229/0.036s

Table 14: 40 job con p_{ij} Unif(10,30)

Run	Posizionale	Precedenze	Euristico
1	891/0.09s/0%	892/**600s**/70%	891/0.037s
2	858/0.07s/0%	858/**600s**/69%	860/0.039s
3	851/0.17s/0%	856/**600s**/69%	851/0.036s

6 Conclusioni

Valutando i risultati ottenuti è stato possibile capire come il modello più performante, anche in presenza di molti job, risulta essere il modello posizionale. Il modello con vincoli di precedenza rimane un modello valido ma con problemi di velocità in presenza di un gran numero di job e con l'ulteriore svantaggio del dover gestire al meglio il valore del big M, non richiesto invece nel modello posizionale. Il modello euristico fornisce comunque risultati validi in breve tempo, anche se spesso relativamente lontani dall'ottimo calcolato dal solver.