Tema 5

Planificación y Asignación Optimizada de Recursos

CONTENTS

- Métodos exactos
- Aplicación a máquinas paralelas
- Heurísticas para problemas de secuenciación

Métodos exactos

- Vamos a estudiar procedimientos exactos,
  - Procedimientos que garantizan la obtención de la solución óptima en todos los casos
  - Suelen necesitar tiempos de computación que crecen exponencialmente con el tamaño del problema , por lo que sólo pueden utilizarse en instancias pequeñas
  - Desarrollar una formulación del problema y un método exacto ayuda a profundizar en la estructura del problema y a desarrollar buenas cotas.

Métodos exactos

- Todo modelo consta de 3 elementos:
  - Variables
  - Función objetivo
  - Restricciones

- Para los problemas de scheduling utilizaremos habitualmente variables 0-1

- La definición de las variables depende del problema
- En general, hay diferentes alternativas para definirlas

Formulación del problema 1 | | $\Sigma w_j C_j$

$$x_{jt} = \begin{cases} 1 & \text{si el job } j \text{ comienza en el instante } t \\ 0 & \text{en caso contrario} \end{cases} \qquad \left(C = \sum_{j=1}^{n} p_j\right)$$

$$Min \quad \sum_{j=1}^{n} \sum_{t=1}^{C} w_j (t + p_j - 1) x_{jt}$$

$$s.a \quad \sum_{t=1}^{C} x_{jt} = 1 \qquad \forall j = 1, \dots, n$$

$$\sum_{j=1}^{n} \sum_{s=\max\{t-p_j+1,0\}}^{t} x_{js} = 1 \qquad \forall t = 1, \dots, C$$

$$x_{jt} \in \{0,1\} \qquad \forall j,\ \forall t$$

Otra formulación del problema $1||\Sigma w_j C_j$

$$x_{jk} = \begin{cases} 1 & \text{si el job } j \text{ precede al job } k \\ 0 & \text{en caso contrario} \end{cases}$$

$$Min \quad \sum_{j=1}^{n}\sum_{k=1}^{n} w_j p_k x_{kj} + \sum_{j=1}^{n} w_j p_j$$

$$s.a \quad x_{jk} + x_{kj} = 1 \qquad \forall j,k \quad (j \neq k)$$

$$x_{jk} + x_{kl} + x_{lj} \leq 2 \qquad \forall j,k,l \quad (j \neq k, j \neq l, k \neq l)$$

$$x_{jk} \in \{0,1\} \qquad \forall j,\ \forall k \quad (x_{jj} = 0)$$

El problema $R||C_{max}$

Variables

- $X_{ij} \in \{0,1\}$ toma el valor 1 si el trabajo j se procesa en la máquina i,
- $C_{\max} \geq 0$ Makespan

$$\min \quad C_{\max} \tag{1}$$

$$\text{s.a.:} \sum_{j=1}^{n} p_{ij} X_{ij} \leq C_{\max}, \ \forall\ i \tag{2}$$

$$\sum_{i=1}^{m} X_{ij} = 1, \ \forall\ j \tag{3}$$

- Ejemplo de problema con 10 trabajos y 5 máquinas

Processing time Matrix		pij				
	Machines					
Jobs		1	2	3	4	5
1		47	15	38	10	38
2		23	20	39	46	24
3		38	10	26	36	40
4		27	27	29	27	17
5		12	31	35	40	49
6		22	47	31	23	21
7		27	36	39	16	16
8		19	24	29	44	13
9		17	31	27	13	26
10		15	31	39	20	41

- Ejercicio: Hacer en Excel

El problema $R||C_{max}$

- El solver de Excel no es capaz de resolver instancias mucho más grandes
- INSTALAR OPENSOLVER
- CPLEX

CPLEX para R | | C_{max}

```
/* PARAMETERS */
int N = ...; // número de trabajos
int M = ...; // número de máquinas

range NbN = 1..N;
range NbM = 1..M;
int p[NbM][NbN] = ...; // tiempos de proceso

/* VARIABLES */
dvar int+ Cmax; // Makespan
dvar boolean X[NbM][NbN];  // 1 si el trabajo j se asigna a la máquina i, 0 en caso contrario

/* OBJECTIVE */
minimize // minimizar makespan
  Cmax;

/* CONSTRAINTS */

subject to {
// Todos los trabajos se deben asignar exactamente una vez
forall(j in NbN)
  sum (i in NbM) ( X[i][j] ) == 1 ;

// Tiempos de completacióndnd e las máquinas
forall(i in NbM)
  sum (j in NbN) ( p[i][j]*X[i][j] ) <= Cmax ;
}
```


```
N = 10; // número de trabajos
M = 5; // número de máquinas

p =
[
[47 23 38 27 12 22 27 19 17 15]
[15 20 10 27 31 47 36 24 31 31]
[38 39 26 29 35 31 39 29 27 39]
[10 46 36 27 40 23 16 44 13 20]
[38 24 40 17 49 21 16 13 26 41]
];
```

Heurísticas

- Un buen algoritmo heurístico debe ser:

Eficiente, Eficaz y Robusto

- Los procedimientos para medir la calidad de un algoritmo son:
  - Comparación con la solución óptima
  - Comparación con una cota inferior o superior conocida
  - Comparación con un método exacto truncado
  - Comparación con otros heurísticos
  - Análisis del peor caso (worst case analysis)

¿Cómo se compara?

RPD: desviación porcentual respecto a la mejor solución conocida

$$Percentage \quad Deviation = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100$$

Heurísticas

- Ya vimos que programar la producción de manera eficaz es un problema muy difícil
- No obstante es relativamente fácil de obtener un programa de producción **factible**
- Esto hace que muchas empresas se hayan conformado con soluciones obtenidas a partir de reglas de prioridad
- **Falsa creencia de que tener siempre las máquinas en operación asegura que no se pueden obtener mejores soluciones**

Reglas de prioridad

- Son reglas que proporcionan un orden en el que secuenciar las operaciones

- También llamadas reglas de despacho:

  - **Una regla de despacho es una regla que prioriza todos los trabajos u operaciones que están esperando para ser procesados en una máquina**

Reglas de prioridad

- Se suele trabajar al nivel de operación
- En cada momento (normalmente cuando queda libre una máquina) se selecciona una operación de acuerdo a una regla o prioridad y se le asigna una fecha de comienzo

Conjunto E	→	Conjunto P
Op. elegibles		Op. programadas

Reglas de prioridad

- Podemos clasificarlas en:
  - **Estáticas**: no dependen del tiempo en el que se está secuenciando ni del estado de la secuencia, sino sólo de características de los trabajos o de las máquinas
  - **Dinámicas**: dependen del tiempo (es decir, del estado de la secuencia)

  - **Locales**: usan información sólo de los trabajos o de las máquinas en un momento dado
  - **Globales**: usan información de todo el problema

El problema de una máquina: $1 \,||\, C_{max}$

- Máquina
  - Continuamente accesible

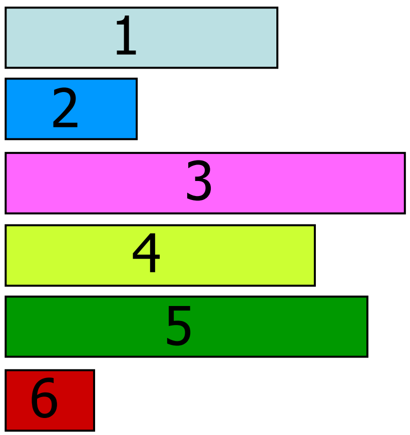
- Tareas (formadas por una sola operación)
  - Tiempo de proceso: p_j

- Objetivo
  - Minimizar el tiempo máximo de completación de las tareas: C_{max}

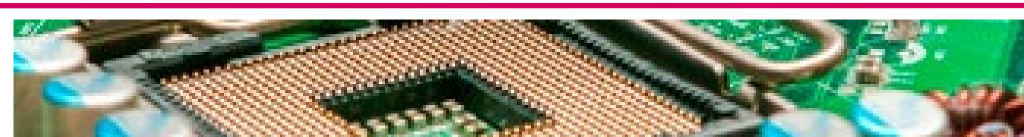
El problema de una máquina: $1 || C_{max}$

- Un ejemplo:

Job	p_j
1	6
2	3
3	9
4	7
5	8
6	2



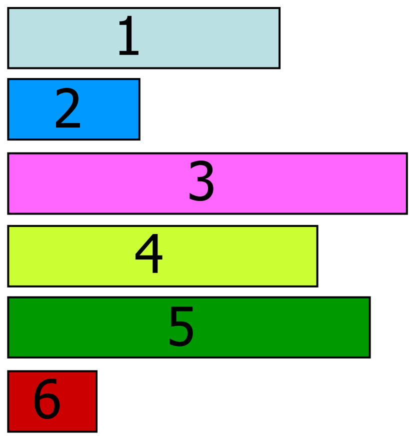
- Minimizar C_{max}



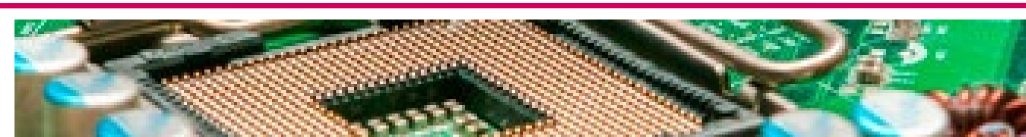
El problema de una máquina: $1 \mid\mid \sum C_j$

- El mismo ejemplo:

Job	p_j
1	6
2	3
3	9
4	7
5	8
6	2

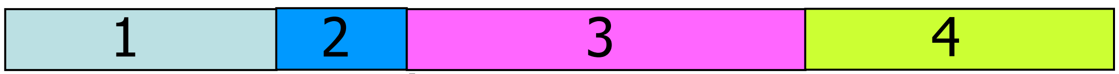


- ¿ Y si el objetivo es: $\sum C_j$?



El problema de una máquina: $1 || \sum C_j$

- ¿Cómo minimizar $\sum C_j$?



$C_1 = p_1$

$C_2 = p_1 + p_2$

$C_3 = p_1 + p_2 + p_3$

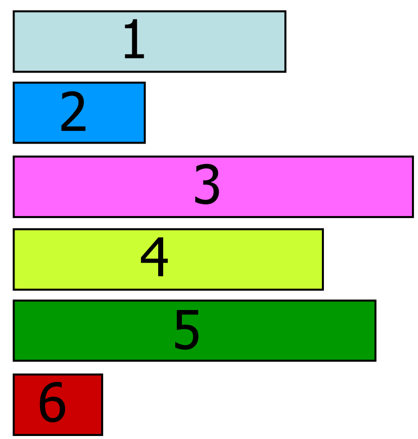
$C_4 = p_1 + p_2 + p_3 + p_4$

$$C_1 + C_2 + C_3 + C_4 = 4p_1 + 3p_2 + 2p_3 + p_4$$

SPT (shortest processing time)

Solución del ejemplo con $\sum C_j$

Job	p_j
1	6
2	3
3	9
4	7
5	8
6	2

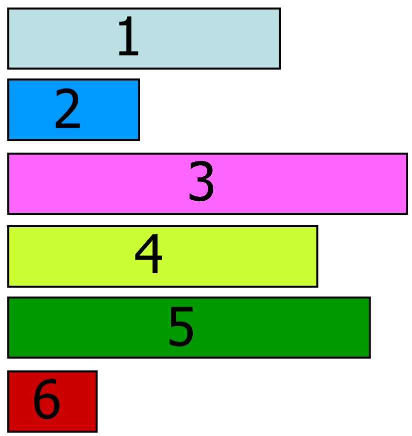


2 5 11 18 26 35

El problema de una máquina: $1||\sum w_j C_j$

- El mismo ejemplo:

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



- ¿ Y si el objetivo es: $\sum w_j C_j$?

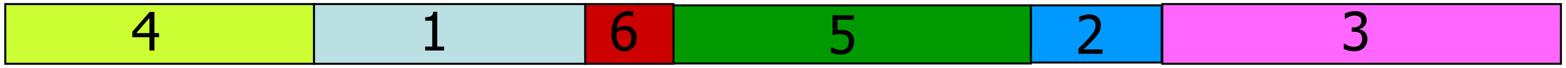
El problema de una máquina: $1||\sum w_j C_j$

- ¿Cómo minimizar $\sum w_j C_j$?

Job	p_j	w_j	p_j/wj
1	6	3	**2**
2	3	1	**3**
3	9	2	**4.5**
4	7	4	**1.75**
5	8	3	**2.66**
6	2	1	**2**

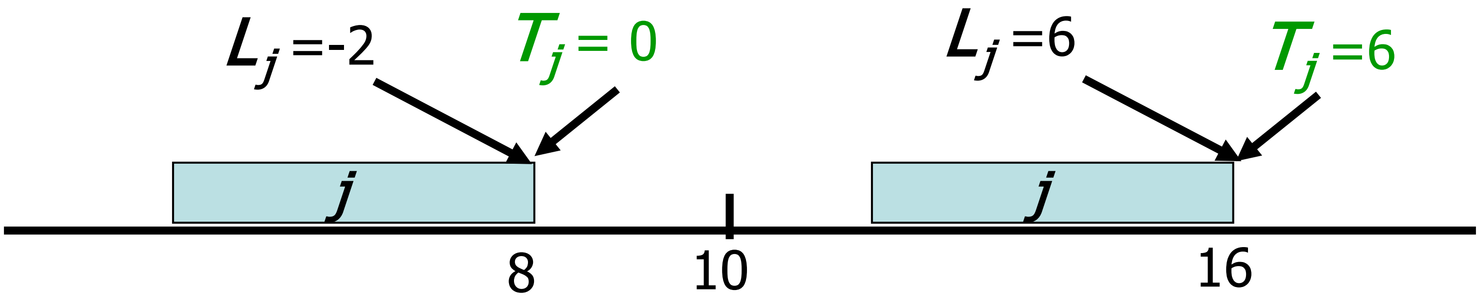
Regla WSPT (Regla de Smith):

Ordenar por pj/wj creciente



Objetivos ligados a fechas de entrega

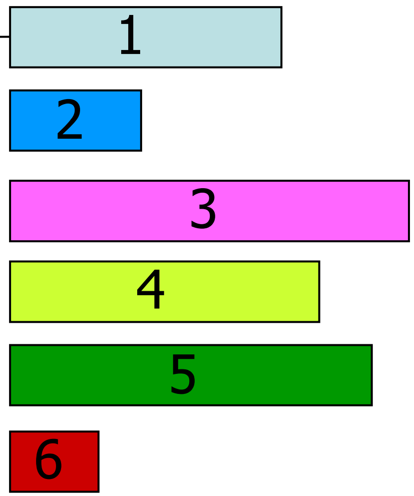
- Fecha de entrega (due date) : d_j
- Job 1: $p_j = 8$, $d_j = 10$
- **Medidas de ajuste a la fecha de entrega:**
  - **Retraso (lateness)** $L_j = C_j - d_j$
  - **Tardanza (tardiness)** $T_j = \max\{0,\ C_j - d_j\}$



Minimizando L_{max}

- El mismo ejemplo, con fechas de entrega:

Job	p_j	d_j
1	6	3
2	3	7
3	9	12
4	7	10
5	8	20
6	2	29



Regla EDD (Regla de Jackson)

Ordenar por d_j creciente

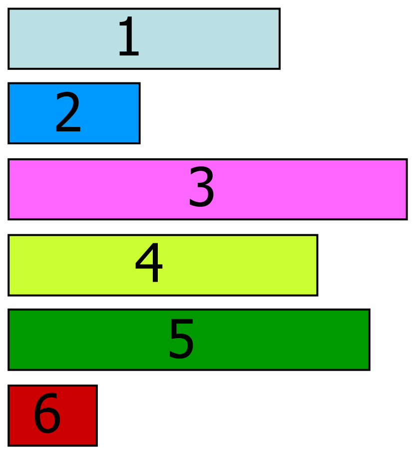
- Solución óptima:

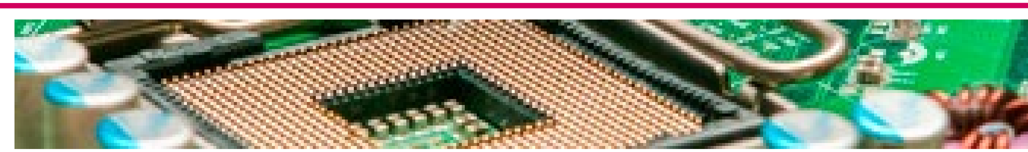
6 9 16 25 33 35

3 1 2 4 13 13 6

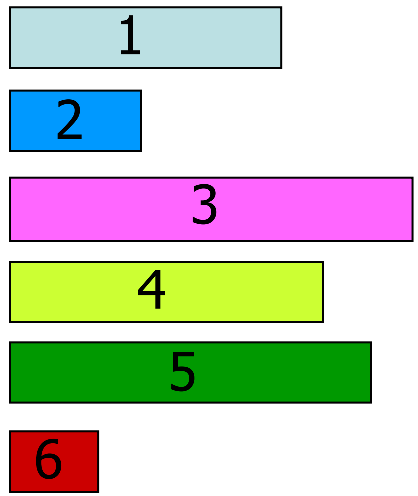
Máquinas en paralelo

Job	p_j
1	6
2	3
3	9
4	7
5	8
6	2





Job	p_j
1	6
2	3
3	9
4	7
5	8
6	2



Regla LPT

Ordenar por p_j decreciente

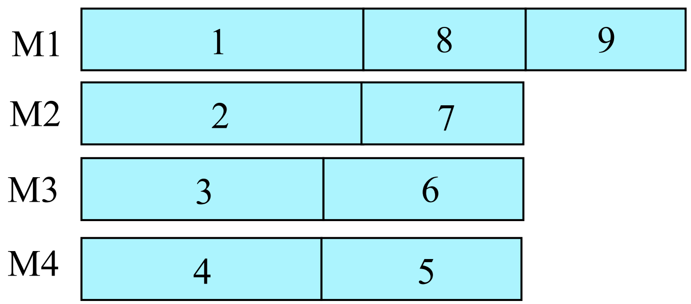
Máquinas en paralelo: *P||Cmax*

La regla LPT no garantiza optimalidad:

- m = 4

Jobs	1	2	3	4	5	6	7	8	9
p_j	7	7	6	6	5	5	4	4	4

LPT



$C_{max}(LPT) = 15$

ÓPTIMO

$C_{max}(OPT) = 12$

Si no todas las máquinas hacen todas las tareas: $P \mid M_j \mid C_{max}$

- Sea el problema $P2 \mid M_j \mid C_{max}$ con 7 jobs

Jobs	1	2	3	4	5	6	7
p_j	10	10	9	8	5	5	3

Los conjuntos M_j son los siguientes:

$$M_1=\{1,2\},\ M_2=\{1,2\},\ M_3= M_4= M_5= M_7= \{2\},\ M_6=\{1\}$$

- Aplicar la regla **LPT** (que da prioridad al job más largo)
- Aplicar la regla **LFJ** (que da prioridad al job menos flexible)

Otras reglas de prioridad

- **SIRO**: (Service in random Order): Como el nombre indica, cuando se libera una máquina, se escoge un trabajo al azar de entre los disponibles
- **ERD**: (Earliest release date first): Escoge el trabajo con menor r_j primero
- **MS**: (Minimum Slack first): Variación de EDD. Si una máquina se libera en t, la holgura remanente de cada trabajo en ese tiempo es $\max\{d_j - p_j - t, 0\}$. Se secuencia el trabajo con el mínimo slack
- **SST**: (Shortest Setup Time first): Cuando queda liberada una máquina, el trabajo con menor tiempo de setup en esa máquina se secuencia

Heurísticas para el flow-shop

- Se trata de uno de los problemas más estudiados y es más realista que los problemas de máquinas paralelas y de una única máquina

- El algoritmo de Johnson (1954) es uno de los primeros en aparecer y proporciona una solución óptima para **$F2 || C_{max}$**

- La regla no es óptima para *m* máquinas. De hecho sólo es óptima para 2 máquinas y para un caso concreto de 3 máquinas

- La mayoría de los problemas de taller de flujo son NP-Difíciles

Flow-shop con dos máquinas: $F2| |C_{max}$

Resolver el problema $F2 \mid prmu \mid C_{max}$ con

Jobs	1	2	3	4
p_{1j}	8	6	4	12
p_{2j}	4	6	10	6

Flow-shop con dos máquinas: $F2||C_{max}$

- **Regla de Johnson (1954):**
  - Particionar los jobs en 2 conjuntos:
    - **Set I** : jobs con $p_{1j} < p_{2j}$
    - **Set II**: jobs con $p_{1j} > p_{2j}$

      (jobs con $p_{1j} = p_{2j}$ pueden ir a Set I ó Set II)
  - Secuenciar primero los jobs de Set I con **SPT** para los p_{1j}
  - Después los jobs de Set II con **LPT** para los p_{2j}

    (los empates se rompen arbitrariamente)

EJERCICIO: Aplicar la regla de Johnson al ejemplo anterior

Heurísticas para el flow-shop: NEH

- El algoritmo tiene tres pasos:
  1. Se calcula el tiempo total de proceso para cada trabajo:

$$P_j = \sum_{i=1}^{m} p_{ij}, \qquad j = (1,\ldots,n)$$

  2. Los trabajos se ordenan descendentemente de acuerdo con P_j y se almacenan en una lista L. Se extraen los dos primeros trabajos L_1 y L_2 y se prueban las dos posibles secuencias que los contienen (L_1,L_2) y (L_2,L_1), la mejor de las dos se mantiene

  3. Extraemos el trabajo k de L, k=3,…,n y se busca la mejor secuencia insertando el trabajo L_k en todas las posibles k posiciones de la secuencia parcial actual

NEH

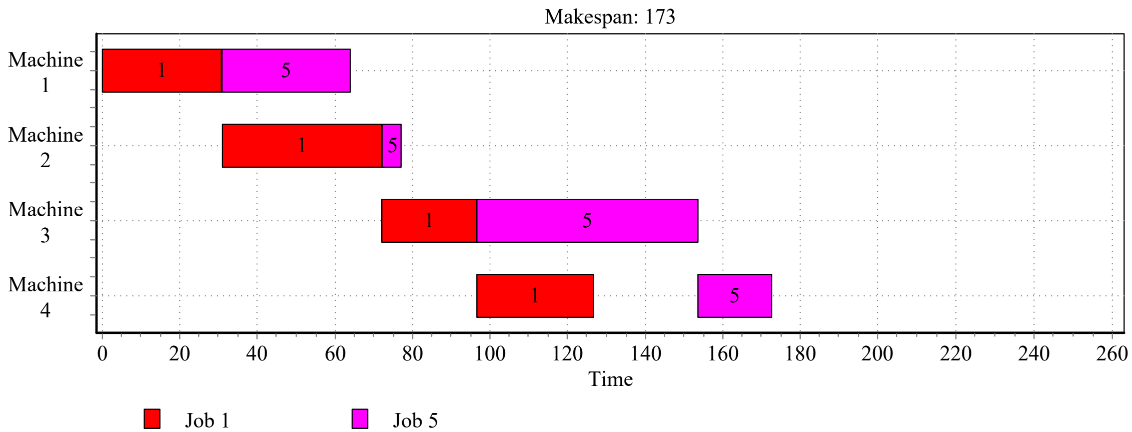
- Ejemplo de NEH:

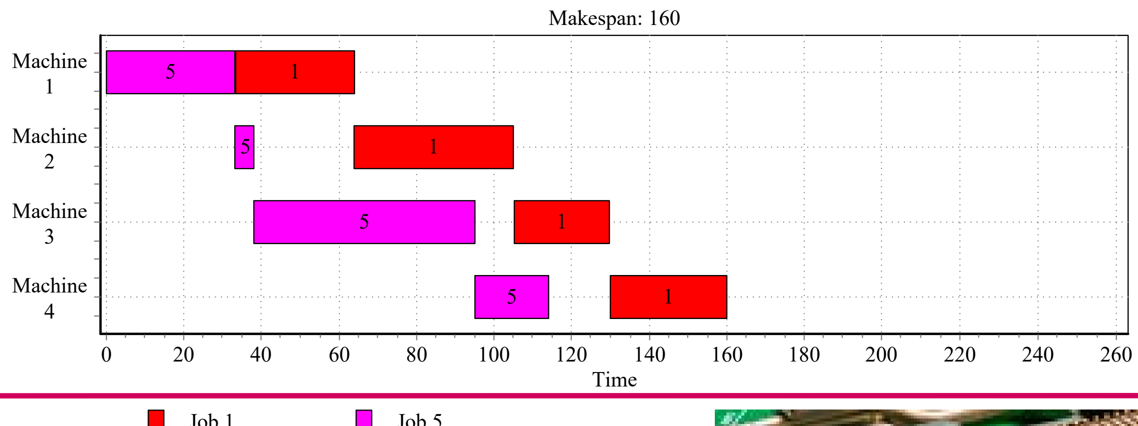
Máquinas (i)	Trabajos (j) 1	2	3	4	5
1	31	19	23	13	33
2	41	55	42	22	5
3	25	3	27	14	57
4	30	34	6	13	19

- P_1=31+41+25+30=127
  P_2=111
  P_3=98
  P_4=62
  P_5=114

NEH

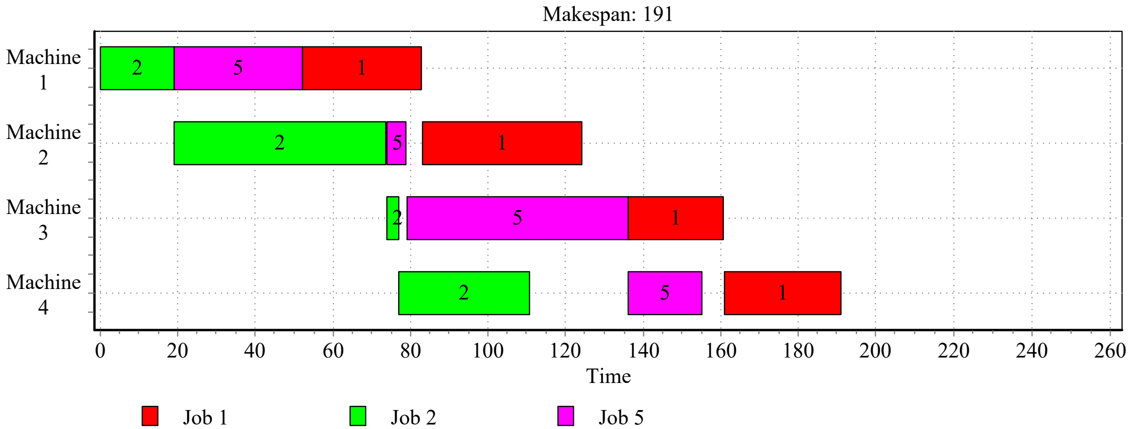
2. Ordenamos los trabajos por P_j y obtenemos L={1,5,2,3,4}. Extraemos L_1={1} y L_2={5} y probamos las dos secuencias {1,5} y {5,1}



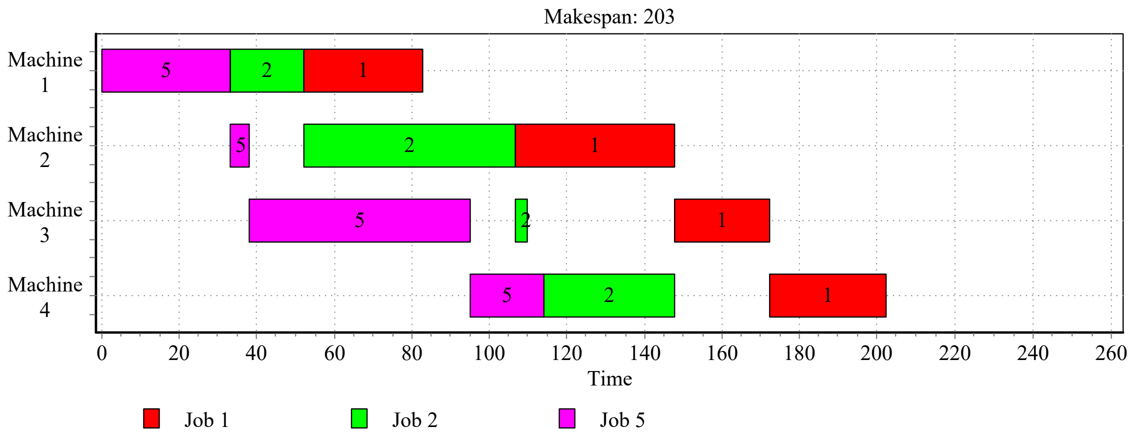


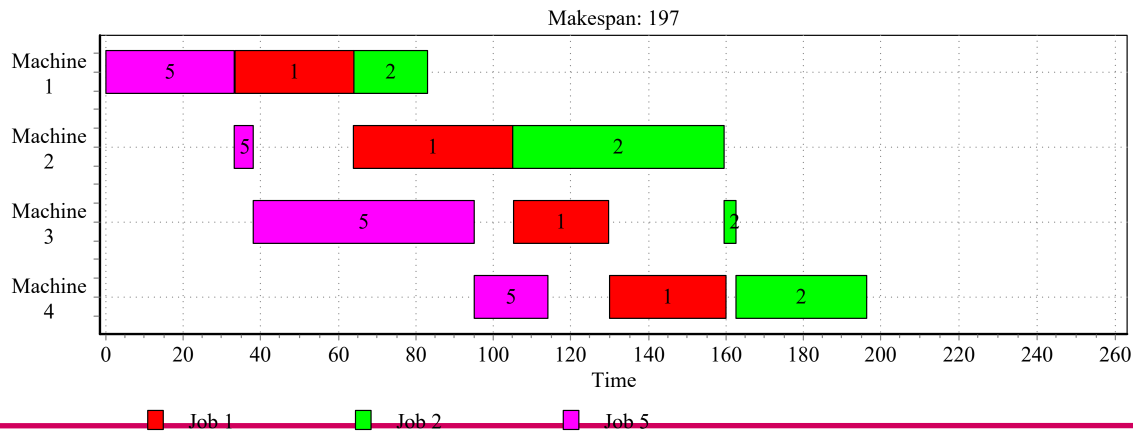
NEH

3. Extraemos el trabajo k de L, k=3,…,n y se busca la mejor secuencia insertando el trabajo L_k en todas las posibles k posiciones de la secuencia parcial actual. Empezamos por k=3, L_3={2}. Hay que probar las secuencias {2,5,1}, {5,2,1} y {5,1,2}



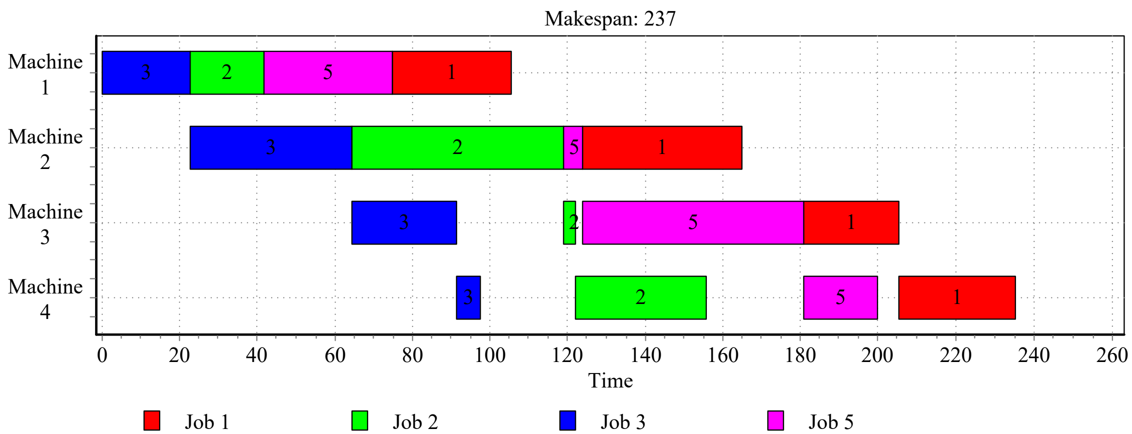
NEH

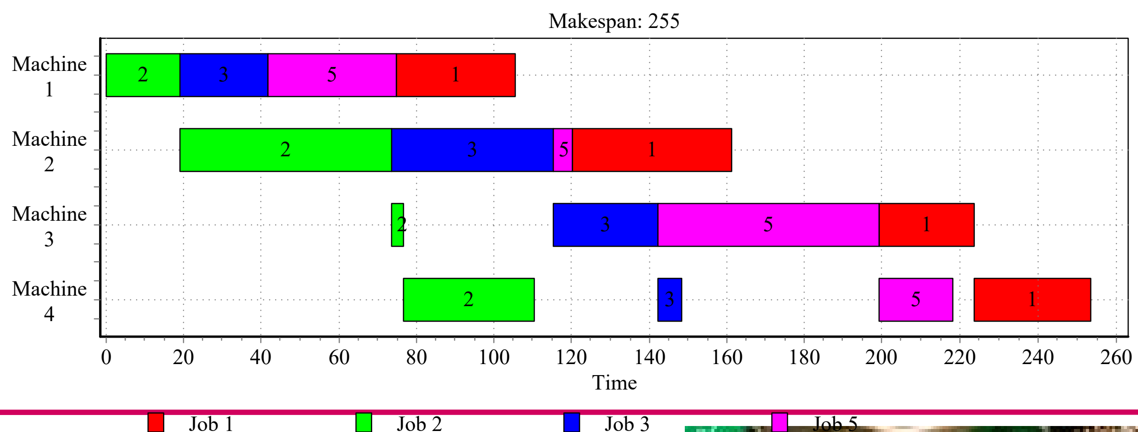

Makespan: 203
Machine 1
Machine 2
Machine 3
Machine 4
5
2
1
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Time
Job 1
Job 2
Job 5


Makespan: 197
Machine 1
Machine 2
Machine 3
Machine 4
5
1
2
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Time
Job 1
Job 2
Job 5

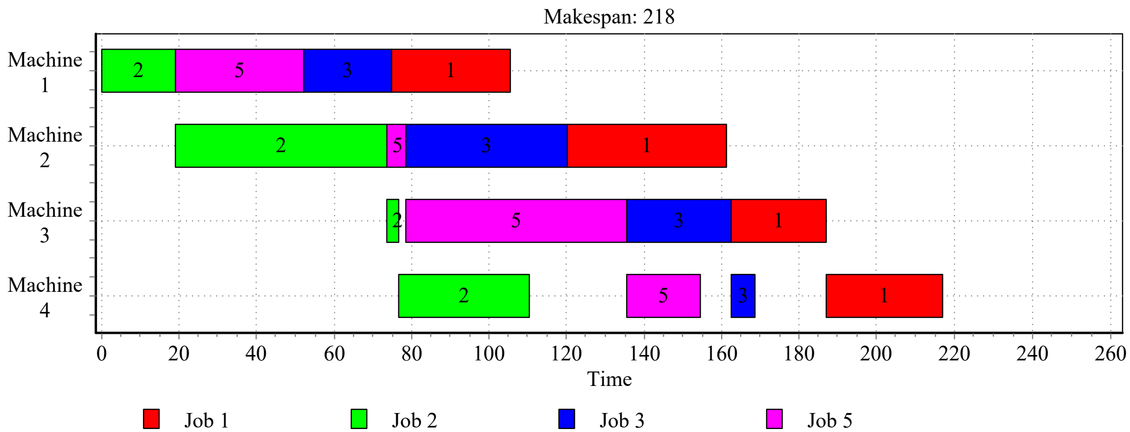
NEH

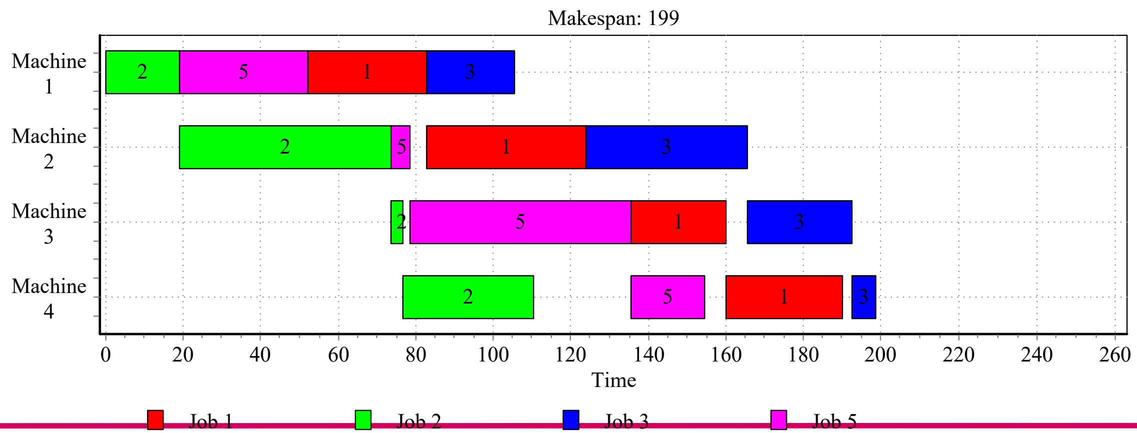
3. La mejor secuencia es {2,5,1}, luego pasamos a k=4, L_4={3}. Hay que probar las secuencias {3,2,5,1}, {2,3,5,1}, {2,5,3,1} y {2,5,1,3}





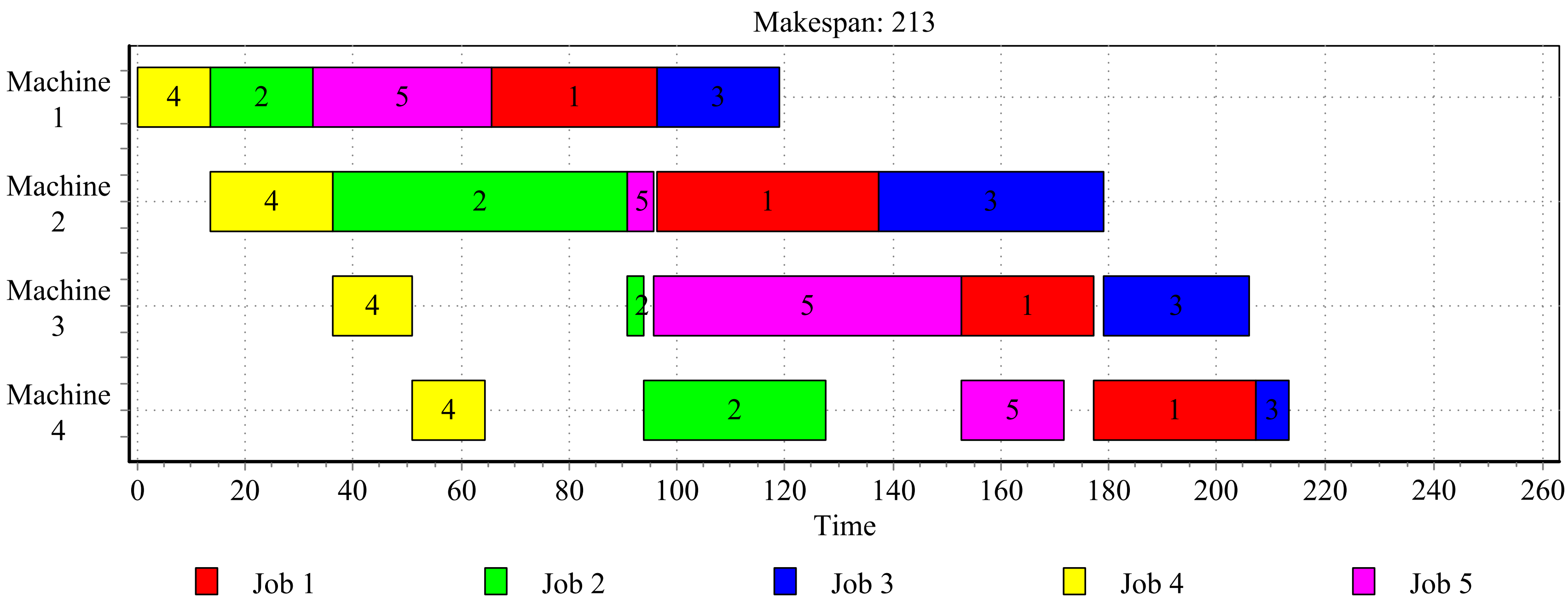
NEH


Makespan: 218
Machine 1
Machine 2
Machine 3
Machine 4
2
5
3
1
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Time
Job 1
Job 2
Job 3
Job 5


Makespan: 199
Machine 1
Machine 2
Machine 3
Machine 4
2
5
1
3
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Time
Job 1
Job 2
Job 3
Job 5

NEH

3. La mejor secuencia es {2,5,1,3}, luego pasamos a k=5, L_5={4}. Hay que probar las secuencias {4,2,5,1,3}, {2,4,5,1,3}, {2,5,4,1,3}, {2,5,1,4,3} y {2,5,1,3,4}. La mejor es {4,2,5,1,3} con C_{max}=213. Esta solución es óptima.



Heurísticas para el flow-shop

- Lectura artículo original NEH, Nawaz, Enscore y Ham (1983)
- Lectura Ruiz y Maroto (2005)
- Lectura Fernandez-Viagas, Ruiz y Framinan (2017)