

# *JOB SCHEDULING*

---

Sociologia das Organizações

2016/17

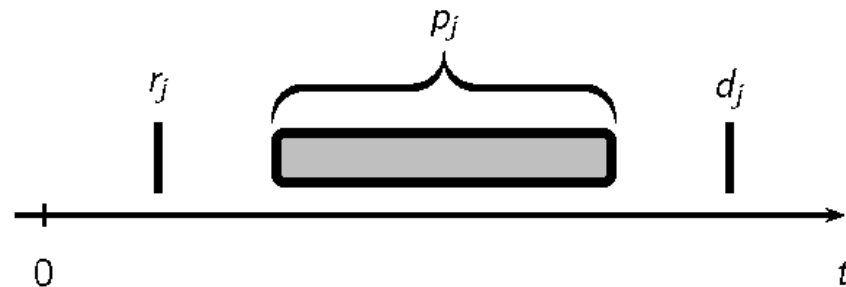
# Job scheduling

- ❑ O tema do *job scheduling* é um dos mais cruciais na gestão da produção
- ❑ A correcta utilização das máquinas para concluir diversas tarefas conduz a uma maior produtividade e eficiência
- ❑ Genericamente, considera-se que um certo conjunto de tarefas  $J$  tem que ser processado por um certo conjunto de máquinas
- ❑ É possível considerar um conjunto de critérios, que ajudam a definir o que será um “bom” escalonamento dessas tarefas

# Job scheduling

□ Para cada tarefa  $j$ , considerem-se as seguintes características:

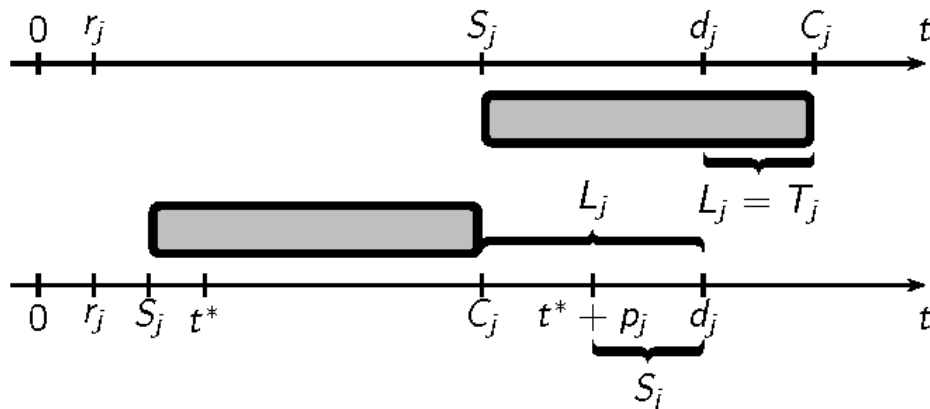
- Tempo de processamento ( $p_j$ )
- Data de disponibilização ( $r_j$ )
- Prazo pretendido ( $d_j$ )
- Peso/importância ( $w_j$ )



# Job scheduling

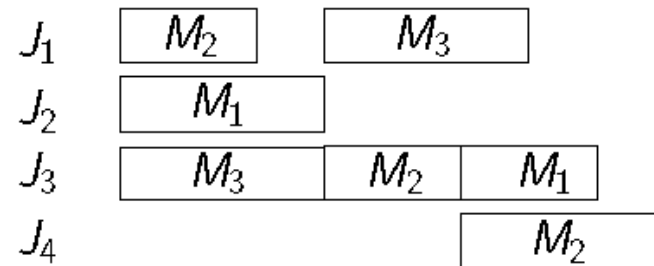
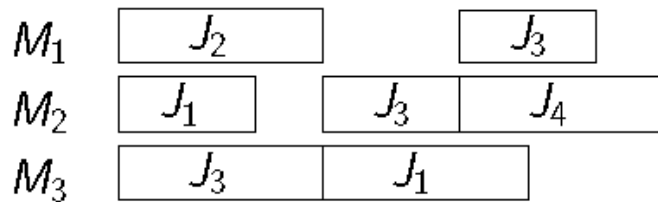
□ Com base nas características da tarefa  $j$ , é possível deduzir algumas métricas:

- Tempo de início do processamento ( $S_j$ )
- Data de conclusão ( $C_j$ )
- Atraso ( $L_j = C_j - d_j$ )
- Tardeza ( $T_j = \max(L_j; 0)$ )
- Folga no instante  $t$  ( $S_j(t) = \max(d_j - p_j - t; 0)$ )
- Tarefa em atraso ( $U_j = 1$ , se  $T_j > 0$ )



# Escalonamento

- ❑ De uma forma geral, é possível considerar que existem  $m$  máquinas e  $n$  tarefas a realizar
- ❑ Um escalonamento pode ser representado com recurso a um diagrama de Gantt



# Escalonamento – Definições gerais

- ❑ No quadro genérico do escalonamento ( $m$  máquinas,  $n$  tarefas), uma afectação entre uma tarefa  $j$  e uma máquina  $i$  é designada por operação e representada por  $(i,j)$
- ❑ O tempo de processamento da operação  $(i,j)$  é representado por  $p_{ij}$  (basta  $p_j$ , quando só existe uma máquina)

# Precedências

- ❑ Uma questão que pode ser importante no escalonamento das tarefas é a eventual existência de precedências
- ❑ Caso existam precedências, considera-se como definida uma rede de precedências, onde a tarefa  $j_1$  está ligada  $j_2$ , se a segunda só pode ser realizada depois da primeira estar concluída
- ❑ Esta rede de precedências terá que ser, necessariamente acíclica

# Escalonamento – Problemas

- ❑ Os problemas de escalonamento são usualmente representados através da notação  $\alpha|\beta|\gamma$ 
  - $\alpha$  fornece informação sobre as máquinas
  - $\beta$  descreve as características das tarefas
  - $\gamma$  dá indicação do critério a considerar no escalonamento
- ❑ Do ponto de vista das máquinas, consideram-se os seguintes problemas:
  - Uma só máquina ( $\alpha = 1$ )
  - Máquinas paralelas (idênticas) ( $\alpha = P$  ou  $Pm$ )
    - O tempo de processamento da tarefa  $j$  é sempre  $p_j$
  - Máquinas paralelas uniformes ( $\alpha = Q$  ou  $Qm$ )
    - As máquinas têm velocidades diferentes ( $s_1, \dots, s_m$ )
    - Os tempos de processamento são iguais a  $p_{ij} = p_j/s_i$



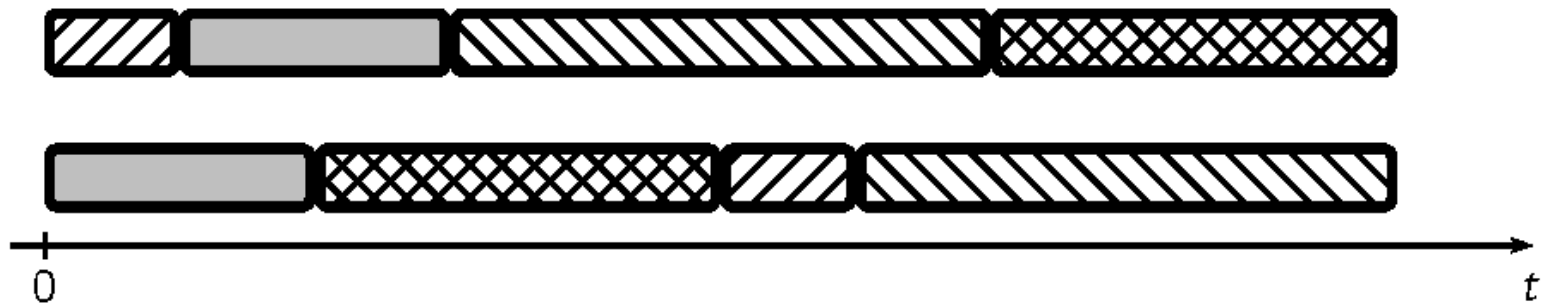
# Escalonamento – Problemas

- Máquinas paralelas não relacionadas ( $\alpha = R$  ou  $Rm$ )
  - As tarefas são processadas em velocidades diferentes pelas máquinas ( $s_{ij}$ )
  - Os tempos de processamento são iguais a  $p_{ij} = p_j/s_{ij}$
  - Cada tarefa tem que ser processada por uma máquina
- *Flow shop* ( $\alpha = F$  ou  $Fm$ )
  - $m$  máquinas em série
  - Cada tarefa tem que passar por cada uma das máquinas
- *Job shop* ( $\alpha = J$  ou  $Jm$ )
  - Cada tarefa tem o seu percurso pelas máquinas pré-definido
  - É possível que uma tarefa não tenha que passar por todas as máquinas
- *Open shop* ( $\alpha = O$  ou  $Om$ )
  - Cada tarefa tem que ser processada por uma máquina, sem ordem pré-determinada

# Escalonamento com uma só máquina

## $(1 \parallel C_{\max})$

- Caso exista apenas uma máquina para as tarefas a realizar, a data máxima de conclusão ( $C_{\max}$ ) será sempre igual
  - $C_{\max} = \max(C_1, C_2, \dots, C_n)$



- Isto significa que a minimização de  $C_{\max}$  é trivial e irrelevante

# Escalonamento com uma só máquina

## $(1 \parallel C_W)$

- ❑ Uma questão que pode ser relevante no escalonamento de tarefas com uma só máquina é a data de conclusão ponderada  $C_W$ 
  - $C_W = \sum w_j C_j$
- ❑ Um caso especial dá-se quando os pesos são todos unitários ( $w_j=1$ )
- ❑ Nesse caso,  $C_W$  é igual à soma de todas as datas de conclusão
- ❑ Para encontrar o escalonamento que minimize  $C_W$  nessas condições, basta aplicar a regra SPT (shortest processing time)

# Escalonamento com uma só máquina

## $(1 \parallel C_W)$

- Se os pesos atribuídos às tarefas forem distintos (caso sejam todos iguais, é possível reduzir a um caso unitário), é necessário aplicar a regra weighted shortest processing time, em que se escolhe primeiro as tarefas que apresentam menor rácio  $p_j/w_j$
- Exemplo:

Tarefas	1	2	3	4
$p_j$	10	20	40	30
$w_j$	2	5	8	1
$p_j / w_j$	5	4	5	30

# Escalonamento com uma só máquina

## $(1 \parallel L_{\text{MAX}})$

- Se o objectivo for minimizar o atraso máximo (considerando que existem prazos), a regra *Earliest Due Date* fornece a solução óptima, se as datas de disponibilização forem todas iguais (por exemplo, iguais a 0)

Tarefas	1	2	3	4	5
$p_j$	20	20	50	40	30
$d_j$	70	180	60	100	90

# Escalonamento com uma só máquina

## ( $1|prec| L_{MAX}$ )

- ❑ Caso existam precedências estabelecidas entre as tarefas, é possível ainda assim resolver o problema de minimização de  $L_{max}$
- ❑ Para tal, basta considerar ir considerando as tarefas que têm os seus sucessores já calendarizados, e escolher dessas a que tem o menor atraso

Tarefas	1	2	3	4	5
$p_j$	20	20	50	40	30
$d_j$	70	180	60	100	90
Prec.	-	4	1	1	2,3

# Escalonamento com uma só máquina

## $(1||r_j| C_{\max})$

- ❑ Caso existam datas de disponibilização das tarefas, o problema de minimizar a data máxima de conclusão pode ser convertido num problema de minimização do atraso máximo
- ❑ Para tal, considere-se uma constante  $K > \max\{r_j\}$  e definam-se prazos de conclusão  $d_j = K - r_j$
- ❑ Resolva-se o problema  $1||L_{\max}$ , considerando esses prazos de conclusão
- ❑ A solução óptima para o problema inicial é dada pela ordem inversa da solução obtida

# Escalonamento com uma só máquina

$(1|r_j| L_{\max})$

- ❑ Curiosamente, este problema tem elevada complexidade (*NP-hard*)
- ❑ Para resolver este tipo de problemas, é necessário recorrer a processos mais sofisticados, embora o tempo de computação para a obtenção da solução óptima possa ser, em muitos casos, demasiadamente elevado
- ❑ Um método de resolução está relacionado com outro problema associado



# Escalonamento com uma só máquina – *Preemption* (interrupção)

- ❑ Em alguns contextos de escalonamento de tarefas, considera-se possível a interrupção da execução de tarefas, para que se possa avançar com outras por alguma motivo (prioridade)
- ❑ Essa situação é designada habitualmente por *preemption*
- ❑ Quando se admite esta situação, a regra *Earliest Due Date*, devidamente adaptada, resolve bem o problema da minimização do atraso máximo

# Escalonamento com uma só máquina

$(1 | p_{\text{mtn}}, r_j | L_{\text{max}})$

❑ Exemplo:

Tarefas	1	2	3	4
$p_j$	4	2	6	5
$r_j$	0	1	3	5
$d_j$	8	12	11	10

- ❑ Ordenam-se as tarefas por ordem crescente do prazo de conclusão
- ❑ Aplica-se a regra *Earliest Due Date* e, sempre que uma tarefa passar a estar disponível, interromper se adequado a que está a ser executada

# Escalonamento com uma só máquina

$(1||r_j| L_{\max})$

- ❑ Quando não é permitido interromper tarefas, é necessário recorrer a métodos mais complexos
- ❑ Um desses métodos é do tipo branch-and-bound
- ❑ No passo inicial, considera-se que  $t=0$  e que a primeira decisão consiste em decidir qual a primeira tarefa a executar
- ❑ Seja  $S$  o conjunto de tarefas já escalonadas, num certo ponto da árvore de pesquisa
- ❑ Só deverão ser consideradas as tarefas para pesquisa que verifiquem a seguinte condição:
  - $r_k < \min_{j \notin S} \{\max\{t, r_j\} + p_j\}$

# Escalonamento com uma só máquina

$(1|r_j| L_{\max})$

- ❑ Em cada ponto da árvore de pesquisa é possível usar como limite inferior a resolução do problema com as tarefas ainda não escalonados de acordo com  $1|pmt_n, r_j| L_{\max}$
- ❑ Um limite superior geral é dado por uma solução admissível que tenha sido encontrada
- ❑ O instante  $t$  associado a um ponto de pesquisa corresponde sempre ao instante anterior somado com o tempo de processamento da tarefa escalonada

# Escalonamento com uma só máquina

$(1|r_j|L_{\max})$

❑ Exemplo:

Tarefas	1	2	3	4
$p_j$	4	2	6	5
$r_j$	0	1	3	5
$d_j$	8	12	11	10

# Escalonamento com uma só máquina

## $(1 || \sum U_j)$

### ❑ Estrutura de uma solução óptima:

- Conjunto  $S_1$  de tarefas que cumprem o prazo de conclusão
- Conjunto  $S_2$  de tarefas em atraso
- As tarefas de  $S_1$  são escalonadas antes das tarefas de  $S_2$
- As tarefas de  $S_1$  estão escalonadas de acordo com a Earliest Due Date
- As tarefas de  $S_2$  estão escalonadas arbitrariamente

### ❑ Algoritmo:

- Ordenar as tarefas por ordem crescente do prazo de conclusão
- Escalonar as tarefas sucessivamente e, se uma ficar em atraso, remover a tarefa já escalonada com maior tempo de processamento
- As tarefas removidas ficam em atraso

# Escalonamento com uma só máquina

## $(1 || \sum U_j)$

❑ Exemplo:

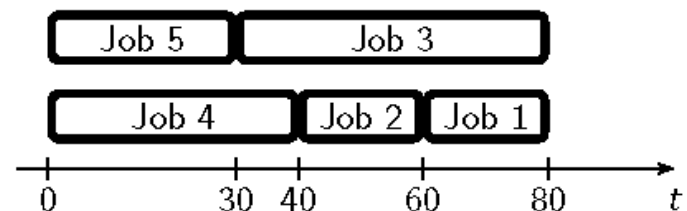
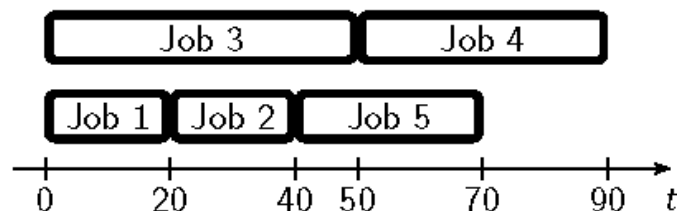
Tarefas	1	2	3	4	5
$p_j$	7	8	4	6	6
$d_j$	9	17	18	19	21

- ❑ O problema  $(1 || \sum w_j U_j)$  é complexo (*NP-hard*)
- ❑ Sugere-se a utilização de uma regra heurística para gerar soluções sub-optimais
- ❑ Um desses métodos pode ser a aplicação da regra *Weighted Shortest Processing Time* (tarefas ordenadas por ordem crescente de  $p_j/w_j$ )
- ❑ Esta complexidade verifica-se, mesmo que os prazos de conclusão sejam todos iguais

# Máquinas paralelas $Pm||C_{\max}$

- ❑ Caso só existisse uma máquina, a questão de minimizar  $C_{\max}$  é irrelevante
- ❑ Quando existem máquinas paralelas, o problema passa a ser complexo (NP-hard)
- ❑ O problema corresponde a balancear correctamente a ocupação das máquinas
- ❑ Exemplo:

Tarefas	1	2	3	4	5
$p_j$	20	20	50	40	30





# Máquinas paralelas $P_m || C_{\max}$

- ❑ Se as tarefas estiverem todas disponíveis no momento inicial,  $r_j = 0$ , a regra *Longest Processing Time* (LPT) fornece habitualmente bons resultados
- ❑ As soluções obtidas com esta regra nunca estarão mais distantes, em termos do  $C_{\max}$  gerado, de 33% do valor óptimo
- ❑ Na realidade, o rácio de entre o valor dado pela solução assim gerada (LPT) e o valor óptimo não ultrapassa  $4/3 - 1/(3m)$

# Máquinas paralelas $Pm||C_{\max}$

- ❑ Para observar um “pior caso”, considere-se a seguinte instância, para 4 máquinas paralelas:

Tarefas	1	2	3	4	5	6	7	8	9
$p_j$	7	7	6	6	5	5	4	4	4

- ❑ A regra LPT dá um  $C_{\max}$  igual a 15, mas é possível obter 12!

# Máquinas paralelas $Pm|| \sum C_j$

- ❑ Caso se pretende minimizar o tempo total de conclusão das tarefas (equivalente a minimizar o tempo médio de conclusão), a regra *Shortest Processing Time* (SPT) fornece sempre a solução ótima
- ❑ Porém, caso o problema seja  $Pm|| \sum w_j C_j$ , o problema passa a ser complexo
- ❑ A regra WSPT (considerando os rácios  $p_j/w_j$ ) pode fornecer boas soluções, mas só garante estar a 22% do ótimo

# Flow shop $Fm||C_{\max}$

- ❑ Nos problemas de *flow shop*, é necessário processar  $n$  tarefas que têm que atravessar  $m$  máquinas em série, pela ordem pré-especificada
- ❑ Embora possa parecer que basta determinar a permutação ideal de tarefas e fazê-las passar sequencialmente pelas máquinas, é possível que uma tarefa “ultrapasse” outra na espera por uma máquina
- ❑ Caso se dêem essas “ultrapassagens”, significa que não terá que se verificar, necessariamente, uma política *First Come First Served*

# Flow shop $Fm||C_{\max}$

- ❑ Um resultado importante em problemas de flow shop, é que uma solução óptima nunca têm “ultrapassagens” entre as duas primeiras máquinas e entre as duas últimas
- ❑ Logo, as soluções óptimas dos problemas  $F2||C_{\max}$  e  $F3||C_{\max}$  nunca têm “ultrapassagens”
- ❑ Já nos problemas  $F4||C_{\max}$  podem existir “ultrapassagens” da segunda para a terceira máquina
- ❑ A possibilidade de ultrapassagens é algo que traz muita complexidade ao problema

# Flow shop $Fm||C_{\max}$

- Quando não são permitidas “ultrapassagens” (permutation flow shop), é possível determinar recursivamente o tempo de conclusão de cada tarefas em cada máquina
- Dada uma permutação  $j_1, \dots, j_n$  das tarefas, as datas de conclusão em cada máquina podem ser calculadas do seguinte modo:

$$C_{i,j_1} = \sum_{l=1}^i p_{l,j_1}, i = 1, \dots, m$$

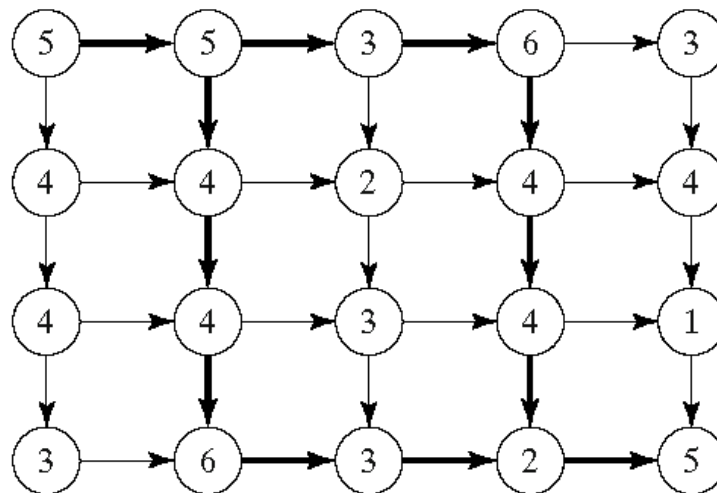
$$C_{1,j_k} = \sum_{l=1}^k p_{1,j_l}, k = 1, \dots, n$$

$$C_{i,j_k} = \max(C_{i-1,j_k}, C_{i,j_{k-1}}) + p_{i,j_k}, i = 2, \dots, m, k = 2, \dots, n$$

# Flow shop $Fm||C_{\max}$

□ Exemplo:

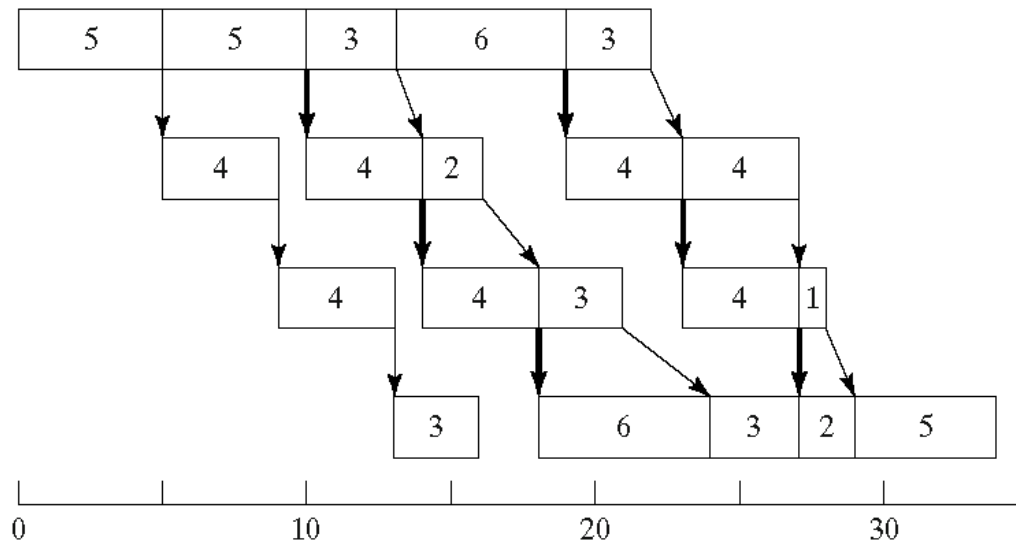
Tarefas	1	2	3	4	5
	5	5	3	6	3
	4	4	2	4	4
	4	4	3	4	1
	3	6	3	2	5



# Flow shop $Fm||C_{\max}$

Exemplo:

Tarefas	1	2	3	4	5
	5	5	3	6	3
	4	4	2	4	4
	4	4	3	4	1
	3	6	3	2	5





# Flow shop $Fm||C_{\max}$

- Um problema de flow shop dual corresponde a construir um outro problema com  $n$  tarefas e  $m$  máquinas, tal que o tempo de processamento da  $i$ -ésima tarefa, num problema, é igual ao da  $(m+1-i)$ -ésima, no outro

Tarefas	1	2	3	4	5
	3	6	3	5	5
	4	4	2	4	4
	1	4	3	4	4
	5	2	3	6	3

# Flow shop $Fm||C_{\max}$

- ❑ A permutação  $(j_1, j_2, j_3, j_4)$  no primeiro problema e a permutação  $(j_4, j_3, j_2, j_1)$  no problema dual apresentam o mesmo  $C_{\max}$
- ❑ Para determinar a melhor permutação no caso  $F2||C_{\max}$ , começa-se por dividir as tarefas em dois conjuntos
- ❑ No conjunto I colocam-se as tarefas que têm um menor tempo de processamento na máquina 1 ( $p_{1j} < p_{2j}$ ), e as restantes colocam-se no conjunto II
- ❑ Uma permutação SPT(I)-LPT(II) é óptima para o problema  $F2||C_{\max}$

# Job shop $Jm||C_{\max}$

- ❑ No job shop, cada tarefa tem a sua sequência de processamento pré-definida
- ❑ O problema consiste em determinar qual a forma de colocar as tarefas nas máquinas
- ❑ Exemplo:
  - $j_1$ : M1->M2->M3
  - $j_2$ : M2->M1->M4->M3
  - $j_3$ : M1->M2->M4

Tarefas\Máquinas	M1	M2	M3	M4
$j_1$	6	7	2	-
$j_2$	1	5	5	4
$j_3$	4	3	-	1

# Job shop $J2||C_{\max}$

- ❑ Quando existem apenas duas máquinas, é possível resolver o problema em tempo polinomial
- ❑ Dividam-se as tarefas em dois conjuntos
  - $J_{1,2}$  – conjunto das tarefas que têm que ser processadas em primeiro lugar pela máquina M1
  - $J_{2,1}$  – conjunto das tarefas que têm que ser processadas em primeiro lugar pela máquina M2
- ❑ Resolva-se o problema das tarefas em  $J_{1,2}$  como se tratasse de um  $F2||C_{\max}$ , ou seja, utilizando o método SPT(I)-LPT(II)
- ❑ Faça-se o mesmo para as tarefas em  $J_{2,1}$

# Job shop $Jm|| C_{\max}$

- ❑ Quando o número de máquinas é superior a 2, o problema torna-se muito complexo
- ❑ Uma alternativa à formulação em programação inteira é a utilização da heurística Shifting bottleneck\*

Tarefas\Máquinas	M1	M2	M3	M4
$j_1$	6	7	2	-
$j_2$	1	5	5	4
$j_3$	4	3	-	1

\* Não estudada neste contexto