

# 620-362

# Applied Operations Research

## Scheduling and Complexity

Department of Mathematics and Statistics  
The University of Melbourne

*This presentation has been made in accordance with the provisions of Part VB of the copyright act for the teaching purposes of the University.*

*For use of students of the University of Melbourne enrolled in the subject 620-362.  
Copyright©2008 by Heng-Soon Gan*

Some contents of this presentation are adapted from  
*Scheduling: Theory, Algorithms and Systems*, Michael L. Pinedo (3<sup>rd</sup> Edition, 2008)

# Outline

- Theory of Complexity (the difficulty of a problem)
  - Encoding scheme
  - Theory of NP-hardness
- Some Machine Scheduling Problems
  - Notations (data, environment, objectives)
  - Single Machine Models
  - Parallel Machine Models
  - Flow Shop

# Complexity

## A problem

A generic description of the problem, e.g. an LP, a scheduling problem, a facility location problem...

## An instance

A problem with given set of numerical data

## Size

Length of data string necessary to specify the instance.

Also called the “length of encoding”.

# Complexity

## Encoding scheme

e.g. two parallel machine scheduling problem with 5 jobs;  
encoding convention used: #machines, #jobs, processing  
times of all jobs

Decimal:

2,5,2,3,5,5,8 (Size: 7)

Binary:

10,101,10,11,101,101,1000 (Size: 19)

Unary:

11,11111,11,111,11111,11111,11111111 (Size: 30)

# Complexity

## Efficiency

Maximum (worst-case) number of computational steps needed to obtain an optimal solution as a function of the size of the instance.

## Computational step

Based on a standard *Turing* machine – a state machine.

Loosely speaking, this is the *maximum number of iterations* the algorithm has to go through, as a function of the size of the instance, e.g. the number of jobs.

E.g. if max. # iterations for an algorithm is  $1500 + 100n^2 + 5n^3$ , then this is a  $O(n^3)$  algorithm – this is a *polynomial algorithm*.

# Complexity

If the maximum number of iterations for an algorithm is  $O(n^k)$  for  $k > 0$ ,  
then this is a *polynomial-time algorithm*.

If the maximum number of iterations for an algorithm is  $O(k^n)$  for  $k > 1$ ,  
then this is a *exponential-time algorithm*.

# Complexity

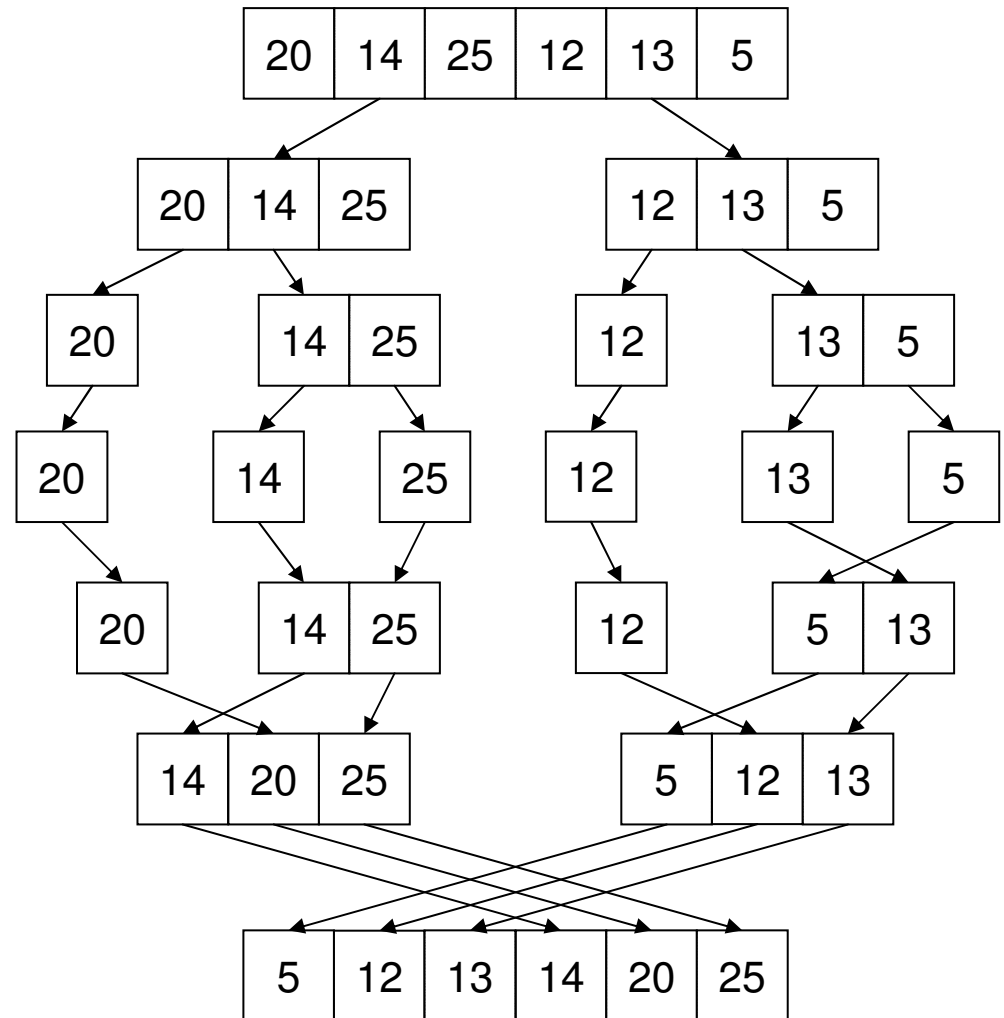
Some of the easiest scheduling problems can be solved through simple priority rule, i.e. by sorting jobs.

A simple sorting algorithm: **Mergesort**

– Run time of mergesort for list of size  $n$  is

$$T(n) = 2T(n/2) + (n-1)$$

– It can be shown that  $T(n)$  is  $O(n \log n)$ .



# Complexity

## Decision problems

These are yes-no problems.

Answer to the question raised is either a *yes* or a *no*.

## Optimization problems

With every optimization problems, one can associate a decision problem.

e.g. for a minimisation problem, in the associated decision problem the question is raised whether there exists a solution with objective value less than a given value  $z$ .

## Problem reduction

A problem  $Q$  reduces to problem  $R$  if for any instance of  $Q$ , an equivalent instance of  $R$  can be constructed.

More strictly, a problem  $Q$  *polynomially* reduces to problem  $R$  if a polynomial-time algorithm for  $R$  implies a polynomial-time algorithm for  $Q$ .

# Complexity – Problem classes

## Class P

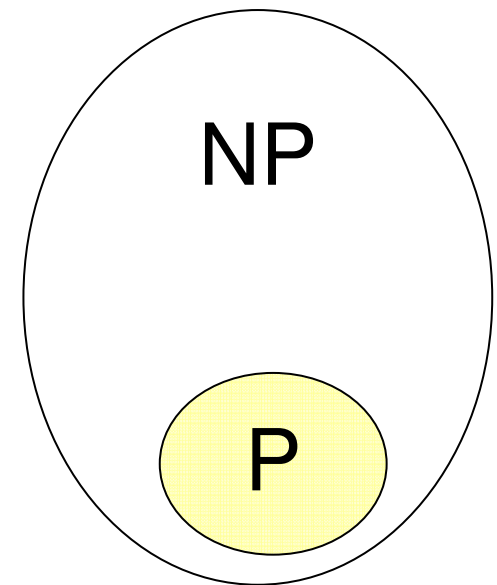
The class P contains all decision problems for which there exists a polynomial-time Turing machine algorithm that leads to the right yes-no answer.

- this is based on the time it takes a Turing machine to solve a decision problem.

## Class NP (Nondeterministic Polynomial)

The class NP contains all decision problems for which the correct answer, given a proper clue, can be verified by a Turing machine in polynomial-time.

We do not know if polynomial-time algorithm exists for this class.



# Complexity – Problem classes

## NP-complete

A decision problem  $Q$  is called NP-complete if the entire class of NP problems polynomially reduces to  $Q$ .

## NP-hard

A problem  $Q$ , either a decision problem or an optimisation problem, is called NP-hard if the entire class of NP problems polynomially reduces to  $Q$ .

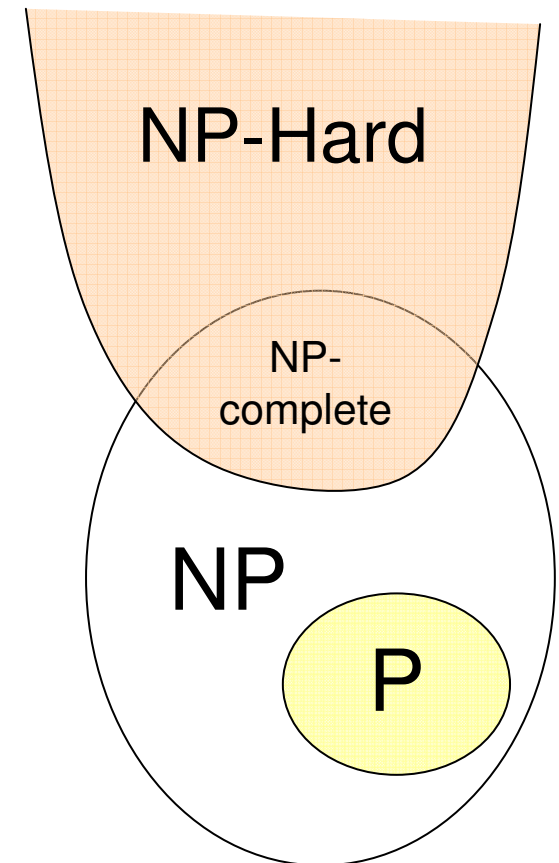
## NP-hard in the ordinary sense or NP-hard

A class of problems that can be solved in polynomial-time under unary encoding, but not binary encoding.

Algorithms for this class of problems are called *pseudo-polynomial* algorithms.

## Strongly NP-hard

There may not exist polynomial-time algorithms under either unary or binary encoding for this class of problems.



# Machine Scheduling

...is the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives.

# Machine Scheduling

Problem notation:  $\alpha|\beta|\gamma$

$\alpha$  - environment:

single machine (1), identical parallel machines ( $Pm$ ), parallel machines with different speeds ( $Qm$ ), unrelated machines in parallel ( $Rm$ ), flow shop ( $Fm$ ), flexible flow shop ( $FFc$ ), job shop ( $Jm$ ), flexible flow shop ( $FJc$ ), open shop ( $Om$ )

$\beta$  – processing restrictions and constraints:

release dates ( $r_j$ ), preemption ( $prmp$ ), precedence constraints ( $prec$ ), sequence dependent setup times ( $s_{jk}$ ), job families ( $fmls$ ) etc.

$\gamma$  – objective function:

makespan ( $C_{\max}$ ), maximum lateness ( $L_{\max}$ ), total weighted completion time or total weighted flowtime ( $\sum w_j C_j$ ) etc.

# 1 | $\sum C_j$

Given  $n$  non-preemptable jobs with processing time  $p_j$  for job  $j$ .  
We want to minimise  $\sum_j C_j$ , i.e. minimise the total flowtime.

**The Shortest Processing Time first (SPT) rule is optimal.**

Schedule job with shortest processing time first.

This rule only requires sorting of jobs in the list in non-decreasing order –  $O(n \log n)$ .

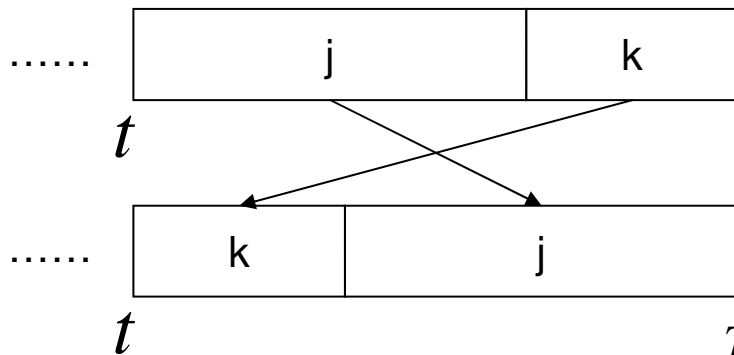
Proof by contradiction.

Given a schedule  $S$  which not in SPT order, and it is optimal.

There must be two adjacent jobs, say job  $j$  followed by job  $k$ , with  $p_j > p_k$ .

Perform adjacent pairwise interchange on jobs  $j$  and  $k$  – all other jobs remain in their original position. Call this schedule  $S'$ .

Total flowtime for  $S'$  is strictly smaller than  $S$  – a contradiction.



$$p_j > p_k$$

$$TF_S = TF_{<j} + (t + p_j) + (t + p_j + p_k) + TF_{>k}$$

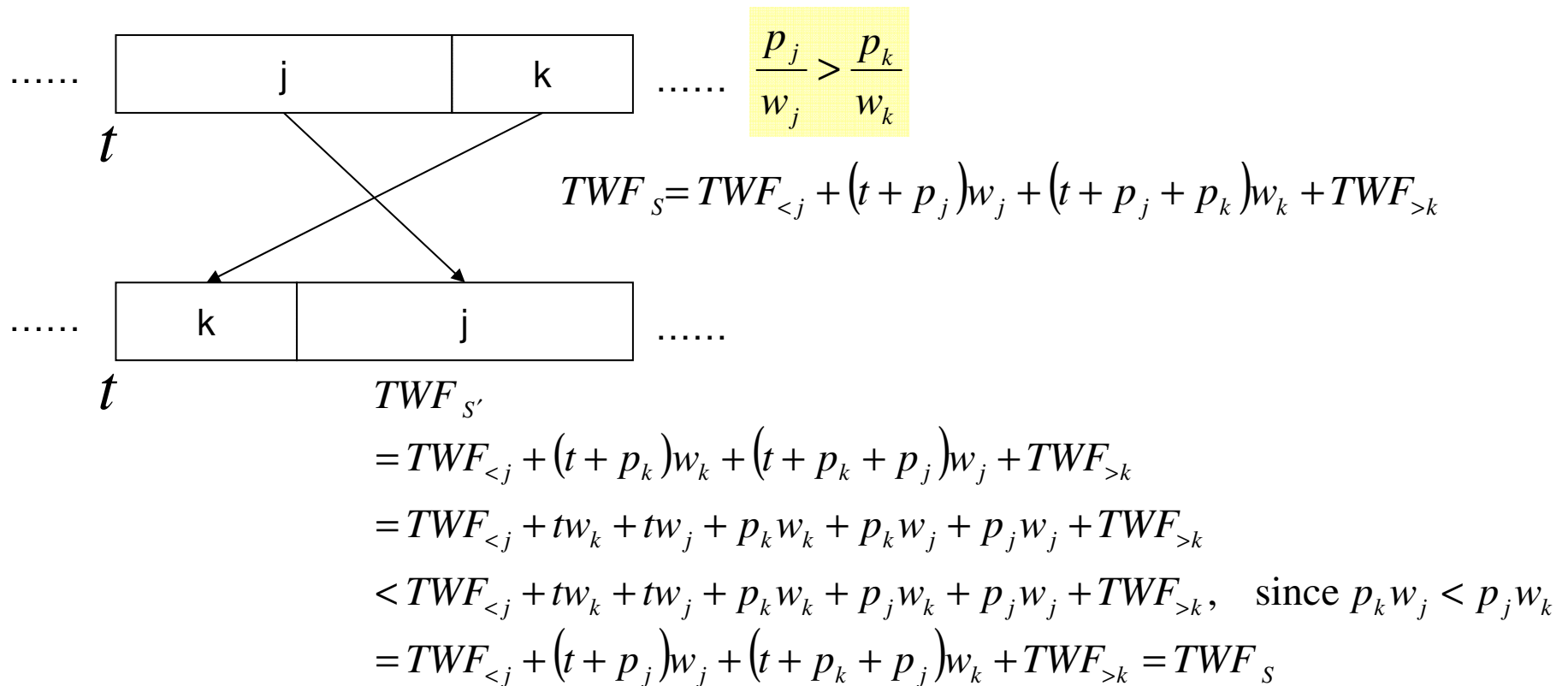
$$TF_{S'} = TF_{<j} + (t + p_k) + (t + p_k + p_j) + TF_{>k}$$

$$< TF_{<j} + (t + p_j) + (t + p_j + p_k) + TF_{>k} = TF_S \quad \text{since } p_k < p_j$$

# 1 | $|\Sigma w_j C_j$

For problem minimising total weighted completion times, the **Weighted Shortest Processing Time first (WSPT)** rule is optimal – this is a generalisation of the SPT rule.

Jobs are ordered in non-decreasing order of  $p_j/w_j$  – this is often known as **Smith's rule**.



WSPT (and SPT) belongs to a class of **List Scheduling Rules**.

# $1|r_j|\Sigma C_j$

We are given  $n$  non-preemptable jobs with processing times and release dates.

Release dates: job  $j$  can only start on and after it is released at time  $r_j$ .

$1|r_j|\Sigma C_j$  is strongly NP-hard!!

But  $1|r_j, \text{prmp}|\Sigma C_j$  is easy.

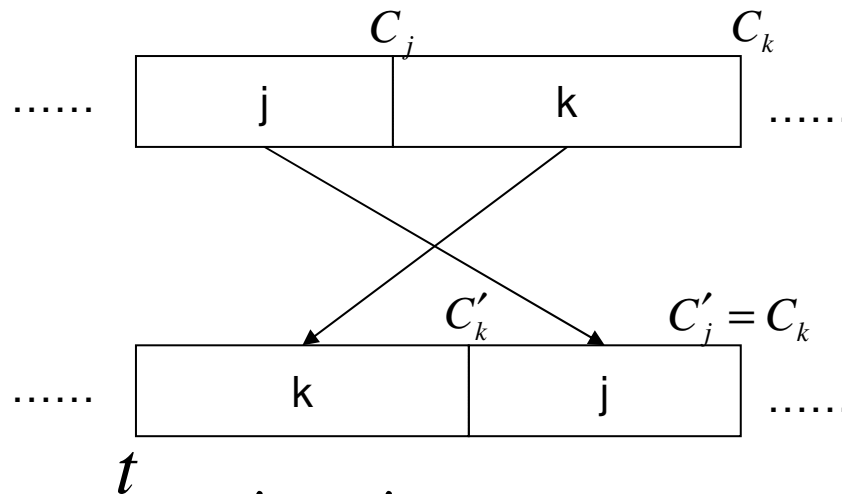
Optimal algorithm is an extension of Smith's rule.  
(Baker, 1974)

...But  $1|r_j, \text{prmp}|\Sigma w_j C_j$  is strongly NP-hard.

# 1 | $L_{\max}$

Given  $n$  non-preemptable jobs with due dates, we want to minimise maximum lateness.

**The Earliest Due Date first (EDD)** is optimal. EDD is an  $O(n \log n)$  algorithm that yields a schedule that orders the job in non-decreasing order of their due dates.



$$d_j > d_k$$

$$L_j = C_j - d_j$$

$$L_k = C_k - d_k$$

$$L_i = C_i - d_i, \quad \forall i \neq j, k$$

$$L'_k = C'_k - d_k < C_k - d_k = L_k$$

$$L'_j = C'_j - d_j = C_k - d_j < C_k - d_k = L_k, \quad \text{since } d_j > d_k$$

$$L'_i = C'_i - d_i = C_i - d_i, \quad \forall i \neq j, k$$

$$L_{\max} = \max\left(L_j, L_k, \max_{i \neq j, k}\{L_i\}\right) \geq \max\left(L'_j, L'_k, \max_{i \neq j, k}\{L'_i\}\right) = L'_{\max}$$

# 1 | $L_{\max}$

But  $1 | r_j | L_{\max}$  is strongly NP-hard.

Reduction of 3-PARTITION problem to  $1 | d_j, r_j | L_{\max}$ .

**The 3-PARTITION problem:**

Give positive integers  $a_1, \dots, a_{3t}, b$  with

$$b/4 < a_j < b/2 \text{ for } j = 1, \dots, 3t,$$

and

$$\sum a_j = tb$$

does there exist  $t$  pairwise disjoint three element subsets  $S_i \subset \{1, \dots, 3t\}$  such that

$$\sum_{j \in S_i} a_j = b$$

for  $i = 1, \dots, t$ ?

Refer to

But  $1 | r_j, prmp | L_{\max}$  can be solved to optimality using EDD.

Preemption: jobs can be stopped and started again from where it last stopped.

# $Pm \mid C_{\max}$

Given  $n$  non-preemptable jobs, and  $m$  parallel machines, minimising maximum completion time (makespan), i.e. minimise  $C_{\max} = \max_i(C_i)$ .

$Pm \mid C_{\max}$  is strongly NP-hard.

$P2 \mid C_{\max}$  is NP-hard in the ordinary sense.

- can be solved in pseudo-polynomial time via a dynamic programming algorithm.

What is the performance of List Scheduling Rules

$Pm \parallel C_{\max}$ ?

- list scheduling: assign the next job on top of the list to the next available machine

# Pm | C<sub>max</sub>

Let  $C_{\max}^*$  be the optimal solution to Pm | C<sub>max</sub>.

(Graham's, 1966) List scheduling (arbitrary list),  $O(n \log m)$ :

$$C_{\max}(LS) / C_{\max}^* \leq 2 - \frac{1}{m}$$

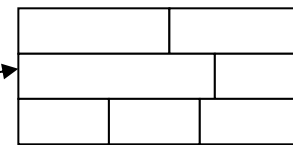
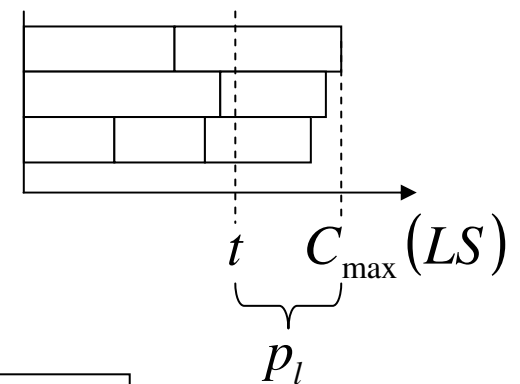
**Proof:**

Let  $p_l$  be the processing time of the last job in a list schedule, and note that no machine can be idle before time  $t = C_{\max}(LS) - p_l$ . Also, we know that

$$\sum_{j \neq l} p_j \geq mt$$

Therefore,

$$\begin{aligned} C_{\max}(LS) &= t + p_l \\ &\leq \frac{1}{m} \sum_{j \neq l} p_j + p_l \\ &= \frac{1}{m} \left( \sum_{j \neq l} p_j + p_l - p_l \right) + p_l = \frac{1}{m} \sum_j p_j + \frac{m-1}{m} p_l \\ &\leq C_{\max}^* + \frac{m-1}{m} C_{\max}^*, \quad \text{since } C_{\max}^* \geq \frac{1}{m} \sum_j p_j, \quad C_{\max}^* \geq p_l \\ &= \left( 2 - \frac{1}{m} \right) C_{\max}^* \end{aligned}$$



## $Pm | C_{\max}$

Longest processing time order (LPT) [Graham, 1969],  $O(n \log n + n \log m)$ :

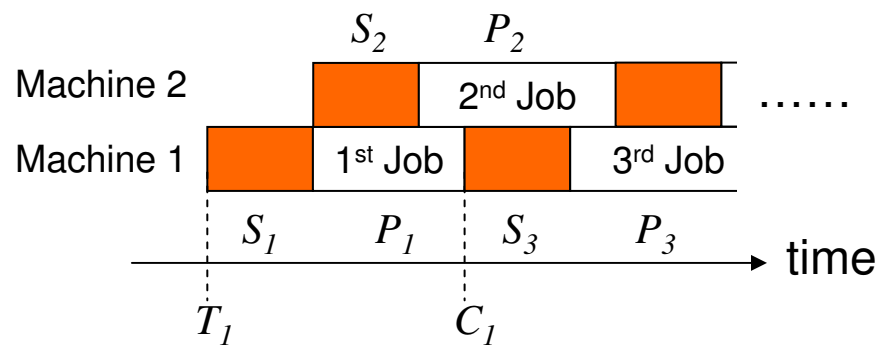
$$C_{\max}(LPT) / C_{\max}^* \leq \frac{4}{3} - \frac{1}{3m}$$

Multifit ( $MF_k$ ) [Coffman, Garey and Johnson, 1978],  $O(n \log n + kn \log m)$  :

binary search and bin-packing using first-fit-decreasing algorithm

$$C_{\max}(MF_k) / C_{\max}^* \leq 1.22 + 2^{-k}$$

# $P2, S1 | s_j, p_j | C_{max}$



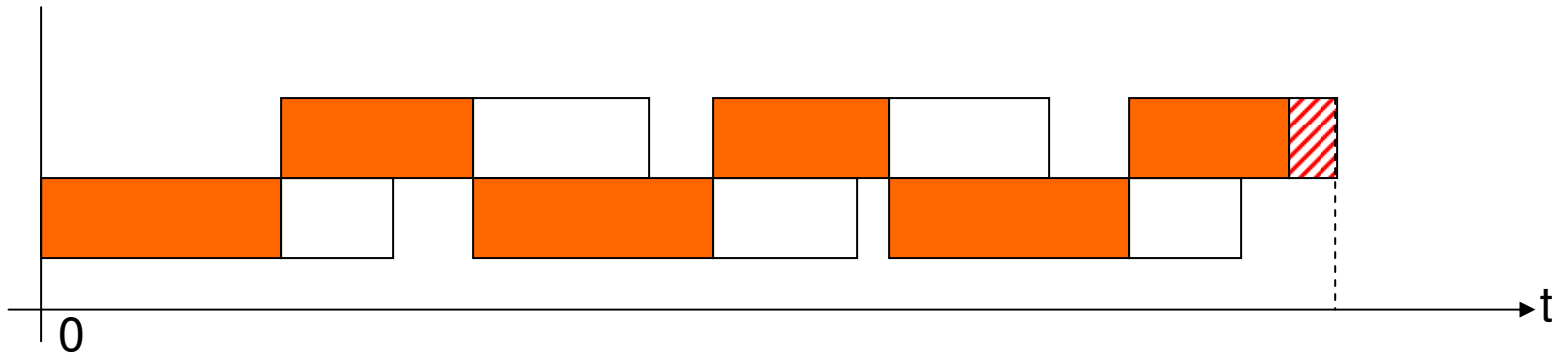
A Gantt Chart representation of a schedule

# $P2, S1 | s_j, p_j | C_{max}$

$P2, S1   s_j = 1   \sum_j C_j$	Polynomial Solvable Hall et al.[HPS00]
$P2, S1   s_j = 1   C_{max}$	Binary NP-hard Brucker et al.[BDFK <sup>+</sup> 02]
$P2, S1   s_j = s, p_i \leq s_j + p_j   C_{max}$	Binary NP-hard Abdekhodae, Wirth & Gan[AWG04]
$P2, S1   s_j = s   C_{max}$	Unary NP-hard Hall et al.[HPS00]
$P2, S1   s_j   C_{max}$	Unary NP-hard Abdekhodae and Wirth[AW02]
$P, S1   s_j = 1   C_{max}$	Unary NP-hard Brucker et al.[BDFK <sup>+</sup> 02]
$P2, S1   p_j = p   C_{max}$	Binary NP-hard Brucker et al.[BDFK <sup>+</sup> 02]
$P2, S1   p_i \leq s_j   C_{max}$	Polynomial Solvable Abdekhodae and Wirth[AW02]
$P2, S1   s_j + p_j = a   C_{max}$	Polynomial Solvable Abdekhodae and Wirth[AW02]

# $P2, S1 | p_j \leq s_j | C_{max}$

Job with smallest processing time scheduled last is optimal,  $O(n)$ .





## F2 | $C_{\max}$

There are  $n$  jobs to be scheduled on 2 machines. Each job  $J_i$  consists of a chain of two operations,  $(O_{i1}, O_{i2})$  where  $O_{ij}$  is to be processed on machine  $j$ . Operation  $O_{i2}$  must not start before the completion of operation  $O_{i1}$ .

**Johnson's algorithm (1954)** is optimal ( $O(n \log n)$ ):

1. Categorise the  $n$  jobs into two sets  $X$  and  $Y$ . Set  $X$  contains jobs for which  $p_{i1} \leq p_{i2}$  and Set  $Y$  contains jobs for which  $p_{i1} > p_{i2}$ .
2. Sequence jobs in Set  $X$  in order of non-decreasing processing times of operations on machine 1.
3. Sequence jobs in Set  $Y$  in order of non-increasing processing times of operations on machine 2.

# Machine scheduling models

- Stochastic scheduling
  - expected value
- Robust scheduling
  - Typically hedge against worst-case scenario
  - Robust initial schedule
  - Rescheduling
    - stability
    - speed
    - optimality