

Scheduling problems

Outline

- Why parallel machine scheduling?
- Overview of the family
- Scheduling on identical parallel machines
- Counting the solutions
- A first schedule by hand
- A combinatorial lower bound
- Relation with the bin packing problem
- Remarks
- Typical mistakes

Learning objectives

At the end of this chapter you will be able to:

- ▶ classify a scheduling problem with the three-field notation, and state the problem $P \parallel C_{\max}$;
- ▶ describe a schedule by the sets of jobs of the machines, and compute loads and makespan;
- ▶ count the assignments and the distinct schedules of an instance;
- ▶ build a first schedule by hand, assigning every job to the least loaded machine;
- ▶ compute a lower bound on the optimal makespan and use it to prove that a schedule is optimal.

Prerequisites

- ▶ From this module, the chapter «Combinatorial optimization problems»:
 - ▶ what a combinatorial optimization problem is;
 - ▶ the number of candidate solutions;
 - ▶ primal and dual bounds;
 - ▶ the partitions of a set (Bell numbers).
- ▶ From this module, the chapter «Packing problems»:
 - ▶ the bin packing problem.

Why parallel machine scheduling?

Introduction: Why parallel machine scheduling?

- ▶ We start from a concrete question: how to share jobs among identical machines, so that all the jobs are finished as early as possible
- ▶ The same question arises for printers and for the processors of a computer
- ▶ We set a challenge on a small instance: the chapter answers it
- ▶ We recall the famous place of this problem in the history of approximation algorithms

Seven print jobs, three printers

- ▶ Seven print jobs must be sent to three identical printers.
- ▶ Each job runs without interruption on one printer.
- ▶ How should the jobs be distributed, so that all of them are finished as early as possible?
- ▶ The same question arises for tasks on the processors of a computer.

A challenge

The reference instance of the chapter: $m = 3$ machines and $n = 7$ jobs, with processing times

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
t_j	5	5	4	4	3	3	3

- ▶ Before reading on, try to find an assignment of the jobs that finishes at time 9.
- ▶ Then ask yourself why no assignment can finish earlier.
- ▶ The chapter answers both questions.

Scheduling and approximation algorithms

- ▶ Scheduling with identical machines has a famous place in the history of approximation algorithms.
- ▶ In 1966 R. L. Graham proved that any list of jobs, processed greedily, gives a makespan at most $2 - 1/m$ times the optimal one.
- ▶ In 1969 he proved the better ratio $4/3 - 1/(3m)$ for the list sorted by non-increasing processing time (the rule called LPT, announced in the paper of 1966).
- ▶ These guarantees, and the algorithms behind them, are studied later in the course, with the heuristic and approximation algorithms; this chapter introduces the problem and its simplest bound.

Summary

The question:

- ▶ Distribute jobs among identical machines (printers, processors) so that all of them are finished as early as possible

The challenge:

- ▶ Finish the reference instance at time 9, and explain why no assignment finishes earlier

The history:

- ▶ Graham's ratios $2 - 1/m$ (1966) and $4/3 - 1/(3m)$ (1969, LPT): later in the course

Overview of the family

Introduction: Overview of the family

- ▶ In this section we describe what the scheduling problems of the chapter have in common: the data, the solutions and the objective
- ▶ The members of the family differ only in the machine environment
- ▶ The three-field notation is the standard way to name a scheduling problem: it gives the problem of the chapter its name, $P \parallel C_{\max}$

Data, solutions and objective

- ▶ The **scheduling problems** of this chapter share the same data: a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n jobs, m parallel machines and, for each job, a *processing time*.
- ▶ A solution assigns every job to a machine: the jobs J_h of machine h are processed one after the other, and the machine completes them at the time L_h , its **load**.
- ▶ The objective is to minimize the **makespan** $C_{\max} = \max_h L_h$, the completion time of the last job.

Parallel machine scheduling in short

A parallel machine scheduling problem partitions the jobs among the machines so that the last machine finishes as early as possible.

The members of the family differ in the machine environment, described by the three-field notation.

Members of the family

- ▶ The table lists the members of the family.

Problem	Notation	Machines
identical parallel machines	$P \parallel C_{\max}$	every job takes t_j on every machine
machines with different speeds	$Q \parallel C_{\max}$	machine h has its own speed
unrelated parallel machines	$R \parallel C_{\max}$	job j takes t_{jh} on machine h

- ▶ $P \parallel C_{\max}$ is defined in the section «Scheduling on identical parallel machines»; $Q \parallel C_{\max}$ and $R \parallel C_{\max}$ are described by the three-field notation.

The three-field notation

- ▶ Scheduling problems are classified with the **three-field notation** $\alpha | \beta | \gamma$ of Graham, Lawler, Lenstra and Rinnooy Kan, where:
 1. α describes the **machine environment**: 1 for a single machine, P for identical parallel machines, Q for parallel machines with different speeds, R for unrelated parallel machines;
 2. β describes the **job characteristics**, such as release dates r_j or precedence constraints $prec$; an empty field means that there are none;
 3. γ describes the **objective function**, such as the makespan $C_{\max}$ or the total completion time $\sum_j C_j$.
- ▶ In this notation, $P || C_{\max}$ is the problem of scheduling jobs without additional characteristics on identical parallel machines, minimizing the makespan.

Summary

Common structure:

- ▶ Data: n jobs, m parallel machines, a processing time for each job
- ▶ Solution: every job assigned to a machine; the load L_h is the completion time of machine h
- ▶ Objective: minimize the makespan $C_{\max} = \max_h L_h$

Three-field notation $\alpha | \beta | \gamma$:

- ▶ Machine environment $(1, P, Q, R)$, job characteristics, objective function
- ▶ $P || C_{\max}$: identical parallel machines, no additional characteristics, makespan

Scheduling on identical parallel machines



Introduction: Scheduling on identical parallel machines

- ▶ In this section we define the problem $P \parallel C_{\max}$
- ▶ We see how an instance and a solution are described: the assignment vector, the sets of jobs of the machines, the loads and the makespan
- ▶ We introduce the reference instance of the chapter, used in all the following sections
- ▶ A precise definition is what we need to count the solutions, build a schedule and compute a bound

Definition 1: Scheduling on identical parallel machines

Given:

1. *a set of m identical machines,*
2. *a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n jobs,*
3. *for each job $j \in \mathcal{J}$, a processing time $t_j \in \mathbb{Z}_{\geq 1}$,*

the problem $P \parallel C_{\max}$ asks to assign each job to a machine, so that the time needed to complete all the jobs, i.e., the largest total processing time assigned to a machine, is minimized.

The name and the instance

- ▶ The name $P \parallel C_{\max}$ comes from the three-field notation of scheduling problems.
- ▶ A $P \parallel C_{\max}$ **instance** is specified by:
 1. two integer numbers:
 - the number of machines $m \in \mathbb{Z}_{\geq 1}$,
 - the number of jobs $n \in \mathbb{Z}_{\geq 1}$,
 2. one vector of integer numbers of length n :
 - $\mathbf{t} \in \mathbb{Z}_{\geq 1}^n$, containing the processing times.

Feasible solutions

- ▶ A $P \parallel C_{\max}$ **feasible solution** is an assignment of the jobs to the machines, which we store in a vector $\text{mach} = (\text{mach}_1, \text{mach}_2, \dots, \text{mach}_n)$:

$$\text{mach}_j = h \iff \text{job } j \text{ is assigned to machine } h.$$

- ▶ The set of jobs assigned to machine h is

$$J_h = \{j \in \mathcal{J} : \text{mach}_j = h\},$$

and its **load** is

$$L_h = \sum_{j \in J_h} t_j.$$

Loads and makespan

- ▶ Each machine processes its jobs one after the other, without idle times, so machine h completes its jobs at time L_h .
- ▶ The **makespan** of the solution is the completion time of the last job,

$$C_{\max} = \max_{h \in \{1, 2, \dots, m\}} L_h.$$

- ▶ The order in which a machine processes its jobs does not change its load: a solution is fully described by the sets $J_1, J_2, \dots, J_m$, which form a partition of $\mathcal{J}$ (some of them may be empty).

The problem in formulas

Given a $P \parallel C_{\max}$ instance, the problem can be formally stated as:

$$\min \left\{ \max_{h \in \{1, \dots, m\}} \sum_{j \in J_h} t_j : J_1, J_2, \dots, J_m \text{ is a partition of } \mathcal{J} \right\}.$$

- We denote by $z(I)$ the optimal makespan of the instance I .

Assumptions

► We make the following assumption:

1. There are more jobs than machines (otherwise each job gets a machine of its own, and the optimal makespan is $\max_j t_j$).

Formally:

$$n > m$$

From the definition to an instance

- ▶ We now fix a small instance, with 3 machines and 7 jobs.
- ▶ It is the **reference instance** of the chapter: on it we will count the schedules, build a first schedule by hand and prove that a schedule is optimal.

Example: the reference instance (1/2)

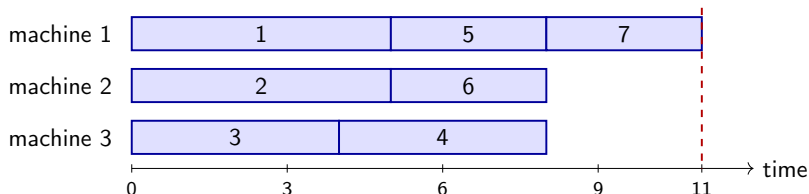
We consider $m = 3$ machines and a set $\mathcal{J} = \{1, 2, \dots, 7\}$ of 7 jobs ($n = 7$), with the following processing times:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
t_j	5	5	4	4	3	3	3

- We call this instance the **reference instance** of the chapter.

Example: the reference instance (2/2)

The assignment $\text{mach} = (1, 2, 3, 3, 1, 2, 1)$ gives $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, with loads $L_1 = 11$, $L_2 = 8$, $L_3 = 8$ and makespan 11:



Summary

The problem $P \parallel C_{\max}$:

- ▶ Assign each job to one of m identical machines, minimizing the largest load
- ▶ Instance: m , n and the vector $\mathbf{t}$ of processing times; assumption $n > m$

Solutions:

- ▶ The vector mach , or the partition $J_1, J_2, \dots, J_m$ of the jobs
- ▶ Loads $L_h = \sum_{j \in J_h} t_j$, makespan $C_{\max} = \max_h L_h$; $z(I)$ is the optimal makespan

Reference instance:

- ▶ $m = 3$, $\mathbf{t} = (5, 5, 4, 4, 3, 3, 3)$; the assignment $(1, 2, 3, 3, 1, 2, 1)$ has makespan 11

Counting the solutions

Introduction: Counting the solutions

- ▶ In this section we count the solutions of a $P \parallel C_{\max}$ instance
- ▶ We distinguish the assignments from the distinct schedules: with identical machines, many assignments describe the same schedule
- ▶ The count shows why listing all the solutions is hopeless already for small instances

Assignments and distinct schedules

- ▶ Every job can be assigned to any of the m machines, independently of the others: there are m^n assignments mach.
- ▶ The machines are identical, so two assignments that differ only by the names of the machines describe the same schedule.
- ▶ The distinct schedules are the partitions of $\mathcal{J}$ into at most m non-empty sets, and their number is

$$\sum_{k=1}^m S(n, k),$$

where $S(n, k)$ is the Stirling number of the second kind, the number of partitions of n elements into exactly k non-empty sets.

Example: counting the schedules (1/2)

- ▶ For the reference instance there are $3^7 = 2187$ assignments.
- ▶ The Stirling numbers are $S(7, 1) = 1$, $S(7, 2) = 63$ and $S(7, 3) = 301$, so the distinct schedules are

$$1 + 63 + 301 = 365.$$

Example: counting the schedules (2/2)

- By enumeration, the schedules with the smallest makespan, 9, are exactly two:

$$\{1, 3\}, \{2, 4\}, \{5, 6, 7\} \quad \text{and} \quad \{1, 4\}, \{2, 3\}, \{5, 6, 7\},$$

which give $3! \cdot 2 = 12$ of the 2187 assignments.

- The optimal schedule is therefore not unique.

Exponential growth

- ▶ The growth is exponential in n : with 3 machines, 20 jobs have $3^{20} \approx 3.5 \cdot 10^9$ assignments, and 30 jobs have $3^{30} \approx 2.1 \cdot 10^{14}$.

machines m	jobs n	assignments m^n
3	7	$3^7 = 2187$
3	20	$3^{20} \approx 3.5 \cdot 10^9$
3	30	$3^{30} \approx 2.1 \cdot 10^{14}$

Summary

Assignments:

- ▶ m^n vectors mach: every job can go on any machine

Distinct schedules:

- ▶ Partitions of the jobs into at most m non-empty sets:
$$\sum_{k=1}^m S(n, k)$$
- ▶ Reference instance: 2187 assignments, 365 schedules, two optimal schedules with makespan 9

Growth:

- ▶ Exponential in n : listing all the solutions is not a practical method

A first schedule by hand

Introduction: A first schedule by hand

- ▶ In this section we build a schedule by hand, with a simple rule: each job goes to the least loaded machine
- ▶ The rule is fast and gives a feasible schedule, whose makespan is an upper bound on the optimal one
- ▶ We apply it to the reference instance, and we ask how far the result is from the optimum

The least loaded machine

A simple way to build a schedule:

- ▶ take the jobs one by one, in the order $1, 2, \dots, n$;
- ▶ assign each job to the machine with the smallest load at that moment;
- ▶ in case of a tie, take the machine with the smallest index.

Example: the least loaded machine

On the reference instance the loads of the three machines evolve as follows:

job j (t_j)	machine chosen	loads (L_1, L_2, L_3)
1 (5)	1	(5, 0, 0)
2 (5)	2	(5, 5, 0)
3 (4)	3	(5, 5, 4)
4 (4)	3	(5, 5, 8)
5 (3)	1	(8, 5, 8)
6 (3)	2	(8, 8, 8)
7 (3)	1	(11, 8, 8)

We obtain the assignment $\text{mach} = (1, 2, 3, 3, 1, 2, 1)$ of the reference instance, with makespan 11.

How good is this schedule?

- ▶ When the last job arrives all the machines have load 8: whichever machine receives it, the largest load becomes $8 + 3 = 11$.
- ▶ We do not know yet whether 11 is far from the optimal makespan: we need a lower bound.

Summary

The rule:

- ▶ Jobs in the order $1, 2, \dots, n$; each one to the least loaded machine; ties to the smallest index

On the reference instance:

- ▶ Assignment $(1, 2, 3, 3, 1, 2, 1)$, makespan 11
- ▶ The makespan of a schedule is an upper bound: to judge it we need a lower bound

A combinatorial lower bound

Introduction: A combinatorial lower bound

- ▶ In this section we derive a lower bound on the optimal makespan from two simple facts: the longest job and the average load
- ▶ The makespan of a schedule is an upper bound on the optimal one; the lower bound says that no schedule can finish earlier
- ▶ When the two values coincide, the schedule is optimal: we use the bound to solve the reference instance

Observation 1

For every $P \parallel C_{\max}$ instance, the optimal makespan $z(I)$ satisfies

$$z(I) \geq \max \left\{ \max_{j \in \mathcal{J}} t_j, \left\lceil \frac{\sum_{j \in \mathcal{J}} t_j}{m} \right\rceil \right\}.$$

Why the bound holds

- ▶ The machine processing the longest job has load at least $\max_j t_j$.
- ▶ The m loads sum up to $\sum_j t_j$, so the largest one is at least their average

$$\frac{1}{m} \sum_j t_j$$

and, being an integer, at least its ceiling.

From the bound to a proof of optimality

- ▶ A schedule gives an upper bound on $z(I)$: its makespan.
- ▶ The observation gives a lower bound on $z(I)$.
- ▶ If a schedule has makespan equal to the lower bound, no schedule can do better: it is optimal.

Example: the bound on the reference instance (1/2)

- ▶ On the reference instance the bound is

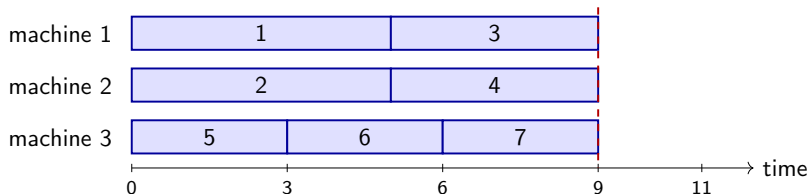
$$\max\{5, \lceil 27/3 \rceil\} = 9.$$

- ▶ Together with the schedule built with the least loaded machine,

$$9 \leq z(I) \leq 11.$$

Example: the bound on the reference instance (2/2)

The assignment $\text{mach} = (1, 2, 1, 2, 3, 3, 3)$ gives $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$, all with load 9, and makespan 9:



It attains the lower bound, so it is optimal: $z(I) = 9$. The three machines finish together, and no idle time is left before the makespan.

The bound is not always attained

- ▶ Consider $m = 2$ machines and three jobs of processing times $(4, 3, 3)$.
- ▶ The bound gives $\max\{4, \lceil 10/2 \rceil\} = 5$.
- ▶ The best schedule is $\{1\}, \{2, 3\}$, with makespan 6.

Summary

The lower bound:

- ▶ $z(I) \geq \max \left\{ \max_j t_j, \left\lceil \frac{1}{m} \sum_j t_j \right\rceil \right\}$: longest job and rounded average load

Optimality:

- ▶ Reference instance: $9 \leq z(I) \leq 11$, then the schedule $\{1, 3\}, \{2, 4\}, \{5, 6, 7\}$ attains 9, so $z(I) = 9$
- ▶ The bound is not always attained: $(4, 3, 3)$ on two machines gives 5, the optimum is 6

Relation with the bin packing problem

Introduction: Relation with the bin packing problem

- ▶ In this section we compare $P \parallel C_{\max}$ with the bin packing problem (BPP)
- ▶ The two problems ask the same question from opposite sides: machines are bins, loads are total weights
- ▶ This is why ideas and algorithms for one of them often have a counterpart for the other

Schedules and packings

- ▶ Given a value T , a schedule with makespan at most T exists if and only if the jobs, seen as items of weight t_j , can be packed into m bins of capacity T .
- ▶ The machines are the bins and the loads are the total weights of the packing patterns.

Opposite quantities

- ▶ The two problems fix and minimize opposite quantities:

	number of bins / machines	capacity / makespan
BPP	minimized	fixed (c)
$P \parallel C_{\max}$	fixed (m)	minimized

- ▶ For this reason the two problems share many ideas: several heuristic algorithms for one of them have a counterpart for the other.

Summary

Scheduling as packing:

- ▶ Makespan at most $T \iff$ the jobs fit into m bins of capacity T

Mirror problems:

- ▶ BPP: fixed capacity, minimize the number of bins
- ▶ $P \parallel C_{\max}$: fixed number of machines, minimize the makespan
- ▶ Heuristic algorithms for one of them often have a counterpart for the other

Remarks

Introduction: Remarks

- ▶ We close the chapter with two classical results of R. L. Graham on greedy list scheduling
- ▶ They tell how far a greedy schedule can be from the optimal one, in the worst case
- ▶ The algorithms behind them are studied later in the course



Graham's results

- ▶ In 1966 R. L. Graham proved that any list of jobs, processed greedily, gives a makespan at most $2 - 1/m$ times the optimal one.
- ▶ In 1969 he proved the better ratio $4/3 - 1/(3m)$ for the list sorted by non-increasing processing time (the rule called LPT).

Greedy list scheduling:

- ▶ Any list: makespan at most $2 - 1/m$ times the optimal one (1966)
- ▶ List sorted by non-increasing processing time (LPT): at most $4/3 - 1/(3m)$ times the optimal one (1969)

Typical mistakes

Introduction: Typical mistakes

- ▶ In this section we collect the most common mistakes on $P \parallel C_{\max}$
- ▶ Each mistake hides a key idea of the chapter: identical machines, makespan, integrality, dual bounds and multiple optima
- ▶ Recognizing them helps to read and solve the exercises correctly

Typical mistakes (1/2)

Observation 2: Warning

- ▶ **Counting m^n different schedules.** *The machines are identical: renaming them gives the same schedule, so the distinct schedules are the partitions of the jobs into at most m sets.*
- ▶ **Confusing makespan and total load.** *The makespan is the largest load of a machine, not the sum of the processing times and not the average load.*
- ▶ **Forgetting the ceiling.** *The processing times are integer, so is the makespan: the average load $27/3$ is rounded up when it is not an integer.*

Typical mistakes (2/2)

Observation 3: Warning

- ▶ ***Assuming that the bound is attained.*** The combinatorial lower bound is a dual bound: a schedule with that makespan may not exist, as the instance $(4, 3, 3)$ on two machines shows.
- ▶ ***Believing that the optimal schedule is unique.*** The reference instance has two optimal schedules, and twelve optimal assignments.

Summary

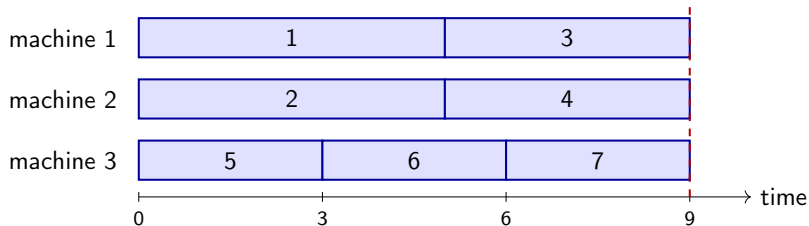
Schedules and makespan:

- ▶ Distinct schedules are partitions, not assignments
- ▶ The makespan is the largest load, an integer: the average load is rounded up

Bounds and optima:

- ▶ A dual bound may not be attained
- ▶ An optimal schedule need not be unique

The challenge, solved



- ▶ $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$: all the jobs are finished at time 9.
- ▶ No assignment finishes earlier: 9 is the lower bound $\max\{5, \lceil 27/3 \rceil\}$.

Summary of the chapter (1/3)

- ▶ The problem $P \parallel C_{\max}$ asks to assign jobs to identical machines minimizing the largest load, the makespan; a solution is a partition of the jobs into at most m sets.
- ▶ There are m^n assignments, but only $\sum_{k=1}^m S(n, k)$ distinct schedules.

Summary of the chapter (2/3)

- ▶ Giving each job to the least loaded machine builds a schedule quickly; the maximum between the longest job and the rounded average load is a lower bound on the optimal makespan.
- ▶ When a schedule attains the lower bound it is optimal; otherwise the gap remains open.
- ▶ The problem is the mirror of bin packing: here the number of machines is fixed and the capacity is minimized.

Summary of the chapter (3/3)

Dashboard of the $P \parallel C_{\max}$ instance (minimization: the lower bound is a dual bound, the upper bound is the makespan of a schedule):

Chapter	Lower bound	Upper bound	From
This chapter	9	11	longest job and average load; least loaded machine
This chapter, improved schedule	9	9	$J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$