Scheduling Algorithms

(Part I)

Resources

- As far as the exam is concerned, it is enough studying the following slides.
- However, the following arguments (and much more with respect to what we are going to see here) about Scheduling can be found in the following book:

Peter Brucker. Scheduling Algorithms (Fifth edition). Springer, 2006.

Introduction

- Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries.
- It deals with the allocation of resources to tasks and its goal is to optimize one or more objectives.
- The resources and tasks in an organization can take many different forms.
- The resources may be machines in a workshop, runways at an airport, crews at a construction site, processing units in a computing environment, and so on.
- The tasks may be operations in a production process, take-offs and landings at an airport, stages in a construction project, executions of computer programs, and so on.
- The objectives can also take many different forms. One objective may be the minimization of the completion time of the last task and another may be the minimization of the sum of the completion times of all the resources.
- Scheduling, as a decision-making process, plays an important role in most manufacturing and production systems as well as in most information processing environments. It is also important in transportation and distribution settings and in other types of service industries.

Introduction (2)

- The practice of this field dates to the first time two humans contended for a shared resource and developed a plan to share it without bloodshed.
- The theory of the design of algorithms for scheduling is younger, but still has a significant history; the earliest papers in the field were published more than sixty years ago!

Introduction (3)

- **Example 1:** Consider a team of five astronauts preparing for the reentry of their space shuttle into the atmosphere. There is a set of tasks that must be accomplished by the team before reentry. Each task must be carried out by exactly one astronaut, and certain tasks can not be started until other tasks are completed. Which tasks should be performed by which astronaut, and in which order, to ensure that the entire set of tasks is accomplished as quickly as possible?

- **Example 2:** Consider a factory that produces different sorts of widgets. Each widget must first be processed by machine 1, then machine 2, and then machine 3, but different widgets require different amounts of processing time on different machines. The factory has orders for batches of widgets; each order has a date by which it must be completed. In what order should the machines work on different widgets in order to insure that the factory completes as many orders as possible on time?

Introduction (4)

Example 3: Consider an airline terminal at a major airport. There are dozens of gates and hundreds of planes arriving and departing each day. The gates are not all identical and neither are the planes. Some of the gates are in locations with a lot of space where large planes (wide bodies) can be accommodated easily. Other gates are in locations where it is difficult to bring in the planes; certain planes may actually have to be towed to their gates.

Planes arrive and depart according to a certain schedule. However, the schedule is subject to a certain amount of randomness, which may be weather related or caused by unforeseen events at other airports. During the time that a plane occupies a gate the arriving passengers have to be deplaned, the plane has to be serviced and the departing passengers have to be boarded. The scheduled departure time can be viewed as a due date and the airline's performance is measured accordingly. However, if it is known in advance that the plane cannot land at the next airport because of anticipated congestion at its scheduled arrival time, then the plane does not take off (such a policy is followed to conserve fuel).

If a plane is not allowed to take off, operating policies usually prescribe that passengers remain in the terminal rather than on the plane. If boarding is postponed, a plane may remain at a gate for an extended period of time, thus preventing other planes from using that gate.

The scheduler has to assign planes to gates in such a way that the assignment is physically feasible while optimizing a number of objectives. This implies that the scheduler has to assign planes to suitable gates that are available at the respective arrival times. The objectives include minimization of work for airline personnel and minimization of airplane delays. In this scenario the gates are the resources and the handling and servicing of the planes are the tasks.

Introduction (5)

Example 4: One of the functions of a multi-tasking computer operating system is to schedule the time that the CPU devotes to the different programs that have to be executed. Each task usually has a certain priority level (the operating system typically allows operators and users to specify the priority level or weight of each task). In such case, the objective is to minimize the sum of the weighted completion times of all tasks.

To avoid the situation where relatively short tasks remain in the system for a long time waiting for much longer tasks that have a higher priority, the operating system "slices" each task into little pieces. The operating system then rotates these slices on the CPU so that in any given time interval, the CPU spends some amount of time on each task. This way, if by chance the processing time of one of the tasks is very short, the task will be able to leave the system relatively quickly. An interruption of the processing of a task is often referred to as a preemption.

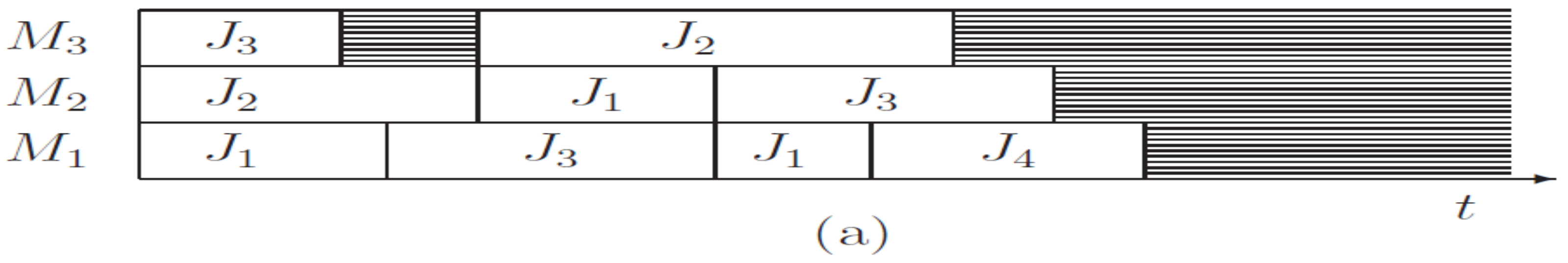
It may not be immediately clear what impact schedules may have on objectives of interest. Does it make sense to invest time and effort searching for a good schedule rather than just choosing a schedule at random? In practice, it often turns out that the choice of schedule does have a significant impact on the system's performance and that it does make sense to spend some time and effort searching for a suitable schedule.

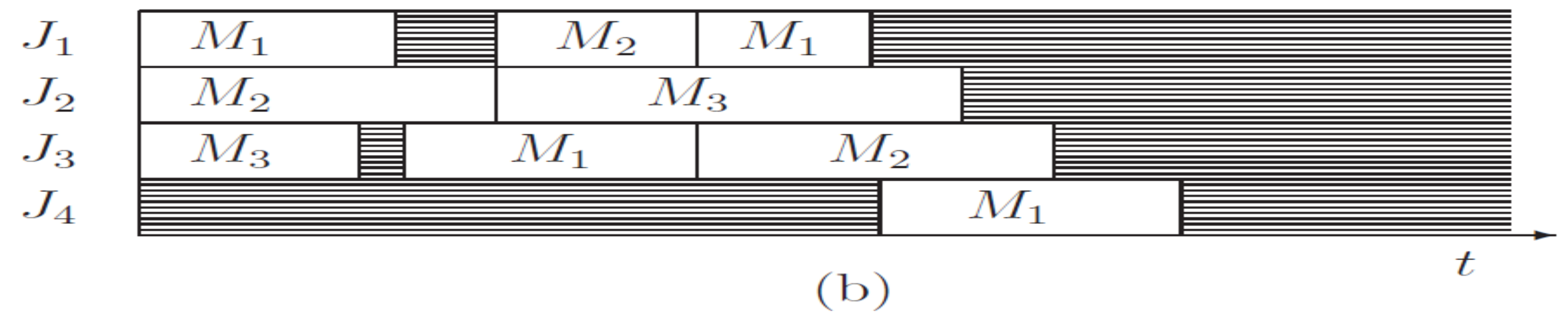
Motivations

- The assignment of tasks to resources (i.e. scheduling) is crucial for optimally exploiting resources.
- Clearly, scheduling problems arise in cloud computing. For instance, a cloud provider has to assign processes (or users’ requests) to resources in order to satisfy the users’ requests in the “best” possible way, that is by optimizing performances.

Preliminaries.

- Suppose that *m* machines M_h where ($h = 1, . . .,m$) have to process *n* jobs J_i where ($i = 1, . . . , n$). Typically we use the index *h* for machines (instead of M_h) and the index *i* for jobs (instead of J_i).
- A **schedule** is an allocation of jobs to machines satisfying certain restrictions and optimizing an objective function.
- Schedules (with preemption) are shown in Figure. Specifically we have machine-oriented representation (Figure a) or job-oriented representation (Figure b). We will mainly use machine-oriented representation.





Dark grey segments are idle times

Preliminaries. (2)

- Let $p_{ih} > 0$ be the *processing time* for job *i* over machine *h.* Machines can have different characteristics and therefore a job can need different processing times for different machines.
- A job may have a *release date* $r_i > 0$, representing the time by which *i* becomes available for processing. Otherwise, job *i* is available from the beginning (time 0).
- A job may have a *due date* $d_i > r_i$, representing the time by which *i* has to be processed. Otherwise there is no constraint on the time by which *i* has to be processed.
- A job may have a weight $w_i > 0$, representing the "importance" or the priority of *i*.
- Jobs may have *precedence relations*, like a job i_1 can be processed only when some different job i_2 has been already processed.
- Associated with each job *i* is a set of machines $u_i \subseteq \{M_1, \dots, M_m\}$. It means that job *i* can be only processed on any of the machines in u_i.
- Note that u_i can be any (non-empty) subset of machines. In particular if all u_i are one element (i.e. each job can be processed by exactly one machine) we have *dedicated machines.* If all u_i are the set of all the machines we have *parallel machines*.
- The remaining case (neither dedicated nor parallel) covers problems in flexible manufacturing where machines are equipped with different tools. This means that an operation can be processed on any machine equipped with the appropriate tool. We call scheduling problems of this type problems with *multi-purpose machines.*

Definitions: processing times.

- **Unrelated machines (general case):**

p_{ih} is the *processing time* for job *i* over machine *h.*

- **Related machines (particular case of Unrelated machines):**

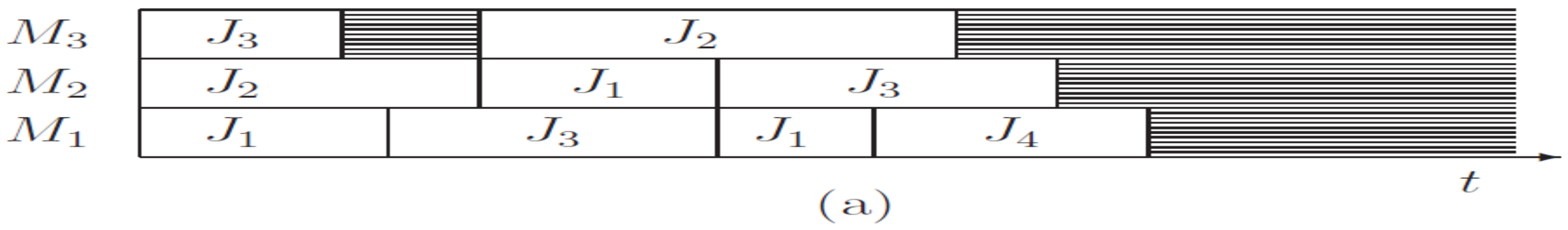
each job *i* has a load $l_i > 0$, each machine *h* has a speed $s_h > 0$, and thus the processing time of job *i* on machine *h* is given by $p_{ih} = l_i / s_h$.

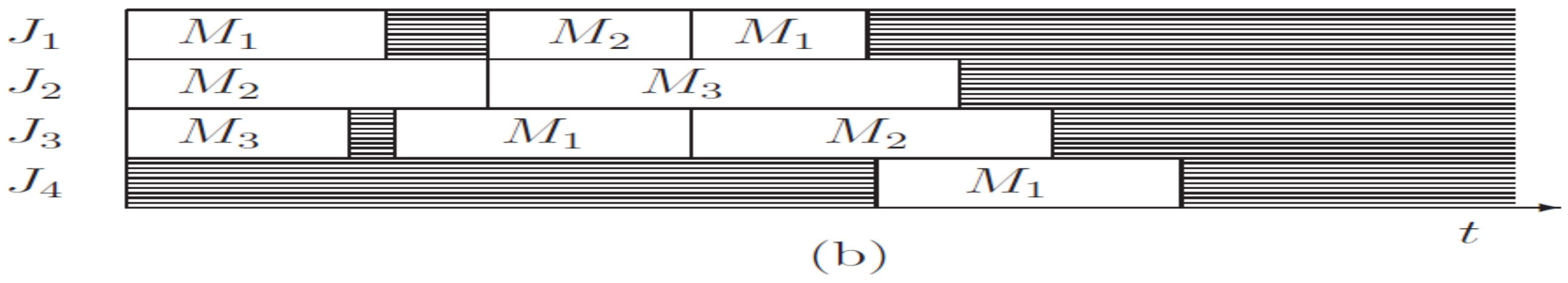
- **Identical machines (specific case of Related machines):**

It is a specific setting of Related machines in which the speed of each machine is *1* (i.e. $s_h = 1$ for each $h = 1, . . ., m$) and therefore $p_{ih} = l_i$.

Definitions: schedule with preemption.

- Preemptions imply that it is not necessary to keep a job on a machine, once started, until its completion (i.e. for the whole processing time).
- The scheduler is allowed to interrupt the processing of a job (preempt) at any point in time and put a different job on the machine instead.
- The amount of processing a preempted job already has received is not lost. When a preempted job is afterwards put back on the machine (or on another machine), it only needs the machine for its remaining processing time.

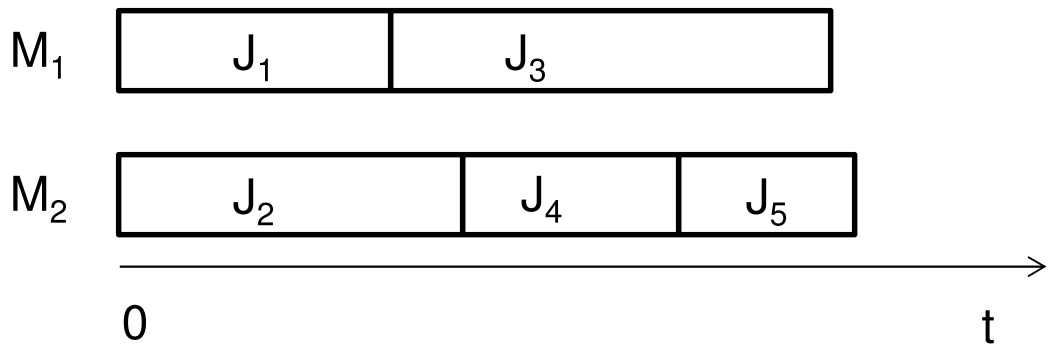




Dark grey segments are idle times

Definitions: schedule without preemption.

- In this case, it is necessary to keep a job on a machine for the whole processing time. In other words, if a job *i* is assigned (or allocated) to a machine *h*, then *i* is entirely processed on *h*. Notice that in this case, for each machine, we assume there is no idle time between the processing of any two jobs.
- Suppose we have 2 machines and 5 jobs



Assumptions.

- From now on we only consider scheduling without preemption.
- We suppose that jobs do not have neither *release date* r_i nor *due date* d_i. It means that all the jobs are available from the beginning (time 0) and there is no constraint on the time by which job *i* has to be processed.
- We suppose that each job can be processed on any machine, that is we only consider parallel machines.
- We suppose there is no *precedence relation* among jobs.

- Most of the time (if not differently specified) we will suppose that the *weight* $w_i = 1$ for each job $i = 1, \dots, n$.

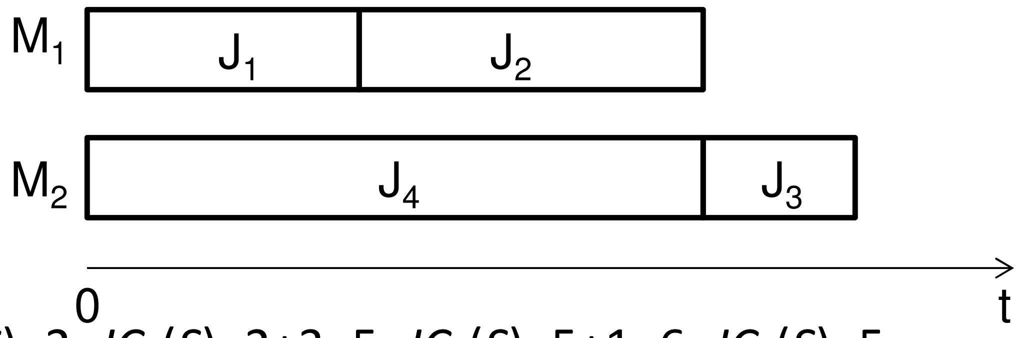
Definitions: schedule and machine completion time (MC).

- A *schedule* $S=(S_1, . . . ,S_m)$ is an assignment of jobs to machines, where S_h (for $h = 1, . . .,m$) is the set of jobs assigned to machine h.
- In particular a schedule has the following properties:
  - *Each job is assigned to exactly one machine; (no preemption)*
  - *All the jobs are assigned;*
  - *Some machines could get no job;*
  - $S_h =(S_{h,1},\dots,S_{h,k_h})$ *where* $k_h =| S_h |$, and $S_{h,1}$ denotes the first job processed by h.

- Given a schedule $S=(S_1, . . . ,S_m)$, let $MC_h(S)=\sum_{i\in S_h} p_{ih}$ be the *completion time* of machine *h in S.*
- Notice that $MC_h(S)$ is only affected by the set of jobs assigned to h (i.e., S_h) and does not depend on the order they are processed by h.
- In other words, given a schedule S, $MC_h(S)$ is the total time the machine h needs to process all the jobs in S_h (recall we do not have idle times).
- Notice that, if a machine h gets no job (i.e. $S_h =\emptyset$), we have that $MC_h(S) = 0$.

Definitions: job completion time (JC).

- Given a schedule *S***,** the completion time of job *i* (denoted by JC_i(S)) is the time by which *i* is completely processed (we always suppose machines start to process jobs at time 0). Notice that the job completion time is affected by the order in which jobs are processed on the allocated machine.
- **Example**: suppose we have *2* identical machines and *4* jobs with processing times p_1=2; p_2=3; p_3=1; p_4=5 (for identical machines we use p_i instead of l_i);
- Consider the following scheduling *S*:



$JC_1(S)=2$; $JC_2(S)=2+3=5$; $JC_3(S)=5+1=6$; $JC_4(S)=5$.

$MC_1(S)=5$; $MC_2(S)=6$.

Definitions: objective functions considering machine completion times.

Typically objective functions considering the machine completion times are:

- $\sum_{h=1}^{m} MC_h(S)$ that we call Machine SUM;

- $Max_{h=1,\ldots,m}\{MC_h(S)\}$ that we call Machine MAKESPAN.

- The Machine SUM problem (respectively Machine MAKESPAN problem) asks to find a schedule that minimizes the Machine SUM (respectively the Machine MAKESPAN).

Definitions: objective functions considering job completion times.

Typically objective functions considering the job completion times are:

- $\sum_{i=1}^{n} w_i * JC_i(S)$ that we call Job SUM;
- $Max_{i=1,\ldots,n}\{JC_i(S)\}$ that we call Job MAKESPAN.
- Notice that Job MAKESPAN and Machine MAKESPAN are equal. That is, given a schedule *S,* we have that:

$$Max_{h=1,\ldots,m}\{MC_h(S)\} = Max_{i=1,\ldots,n}\{JC_i(S)\}$$

- The <u>Job SUM problem</u> (respectively <u>Job MAKESPAN problem</u>) asks to find a schedule that minimizes the Job SUM (respectively the Job MAKESPAN).

Definitions: Scheduling Machine SUM (minimization) problem.

- INPUT: *m* machines ($h = 1, \ldots, m$), *n* jobs ($i = 1, \ldots, n$), $p_{ih} > 0$.

- OUTPUT: a schedule $\boldsymbol{S}=(S_1, \ldots, S_m)$.

- GOAL*:* Minimizing the Machine SUM.

Definitions: Scheduling Job SUM (minimization) problem.

- INPUT: *m* machines (*h = 1, . . .,m*), *n* jobs (*i = 1, . . . , n*), $p_{ih} > 0, \mathrm{w}_i > 0.$
- OUTPUT: a schedule ***S***=*(S₁, . . . ,S_m)*.
- GOAL*:* Minimizing the Job SUM.

Definitions: Scheduling Machine MAKESPAN (minimization) problem.

- INPUT: m machines ($h = 1, . . .,m$), n jobs ($i = 1, . . . , n$), $p_{ih} > 0$.

- OUTPUT: a schedule $\boldsymbol{S}=(S_1, . . . ,S_m)$.

- GOAL*:* Minimizing the Machine MAKESPAN.

- Remind that the Scheduling Job MAKESPAN (minimization) problem is equivalent to the Scheduling Machine MAKESPAN (minimization) problem.

Goal.

- Practical experience shows that some computational problems are easier to solve than others. Complexity theory provides a mathematical framework in which computational problems are studied so that they can be classified as “easy” (P) or “hard” (NP-Hard).

- One of the main issues of complexity theory is to measure the performance of algorithms with respect to computational time, that roughly speaking is the number of steps that an algorithm takes in order to return a solution.

- I suppose that students have knowledge about complexity theory (remind that Algorithms and Data Structures with Lab. Exam is a Course of the Bachelor Degree).

- We are interested in designing fast algorithms that solve scheduling problems.
  - “Solve” means that for any input, the algorithm finds optimal or almost optimal (approximated) schedules (or solutions).
  - “Fast” means that the complexity time is polynomial in the input size. (For instance, if the input size is n, then n^2 or n^6 are polynomials, while 2^n is not polynomial but it is exponential).

Scheduling Unrelated Machine SUM (minimization) problem.

- INPUT: *m* Unrelated machines ($h = 1, . . ., m$), *n* jobs ($i = 1, . . . , n$), $p_{ih} > 0$.
- OUTPUT: a schedule ***S***$=(S_1, . . . , S_m)$.
- GOAL: Minimizing the Machine SUM, that is minimizing $\sum_{h=1}^{m} MC_h(S)$

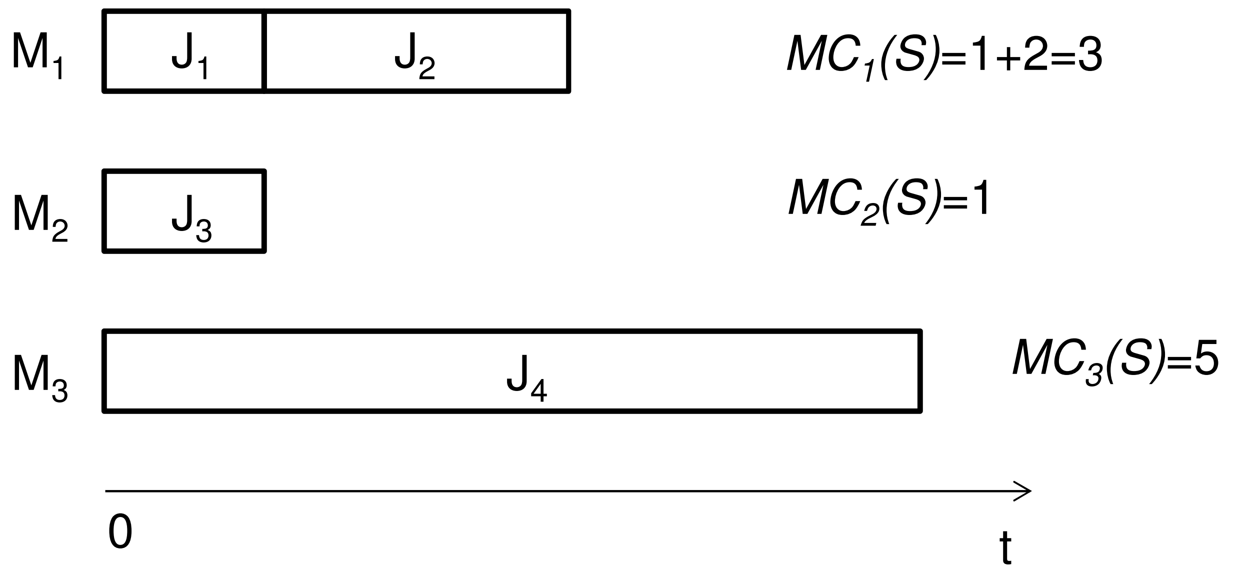
Does such problem admit a polynomial time algorithm that finds a schedule that minimizes the Machine SUM?

Scheduling Unrelated Machine SUM (minimization) problem: an example.

- INPUT: *3* Unrelated machines, 4 jobs.

p_{ih}=

M\J	J_1	J_2	J_3	J_4
M_1	1	2	3	4
M_2	3	5	1	7
M_3	2	2	4	5



$MC_1(S)$=1+2=3

$MC_2(S)$=1

$MC_3(S)$=5

$$\sum_{h=1}^{3} MC_h(S) = 9$$

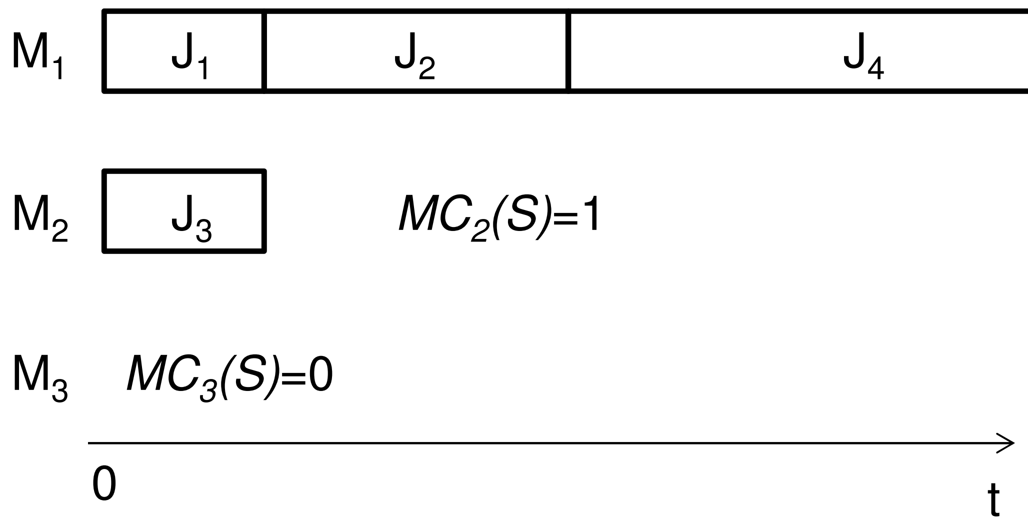
Is it optimal?

Scheduling Unrelated Machine SUM (minimization) problem: an example. (2)

- INPUT: *3* Unrelated machines, 4 jobs.

p_{ih}=

M\J	J_1	J_2	J_3	J_4
M_1	1	2	3	4
M_2	3	5	1	7
M_3	2	2	4	5



$MC_1(S)$=1+2+4=7

$MC_2(S)$=1

$MC_3(S)$=0

$$\sum_{h=1}^{3} MC_h(S) = 8$$

Is it optimal?

Scheduling Unrelated Machine SUM (minimization) problem: a simple polynomial time algorithm

- INPUT: *m* Unrelated machines ($h = 1, . . ., m$), *n* jobs ($i = 1, . . . , n$), $p_{ih} > 0$.
- OUTPUT: a schedule $\boldsymbol{S}=(S_1, . . . , S_m)$.
- GOAL: Minimizing the Machine SUM, that is minimizing $\sum_{h=1}^{m} MC_h(S)$

ALGORITHM MSUM:

1. For each job *i*, find the machine able to process *i* with the minimum processing time and assign *i* to such a machine.
2. Return the obtained schedule *S*.

COMPLEXITY: O(n*m)

Theorem 1: *Algorithm MSUM finds an optimal solution for the Scheduling Unrelated Machine SUM (minimization) problem.*

Proof of Theorem 1.

Proof:

$$\sum_{h=1}^{m} MC_h(S) = \sum_{h=1}^{m} \sum_{i \in S_h} p_{ih}$$

- By contradiction, let us consider a schedule *S'* (different than S) where a job *i* is <u>not</u> assigned to the machine able to process *i* with the minimum processing time. That is, in *S'* the job *i* is assigned to the machine *h'* and there exists a machine *h* such that $p_{ih'} > p_{ih}$

- It is easy to see that by assigning job *i* to machine *h* (instead of *h'*) the Machine SUM strictly decreases, and therefore *S'* is not an optimal solution.
- In fact in the above sums (for *S'*) if we replace $p_{ih'}$ with p_{ih} we get a strictly better solution.
- It follows that a schedule is optimal if and only if each job is assigned to the machine able to process the job with the minimum processing time.
- Thus, the schedule returned by Algorithm MSUM is optimal.

□

Scheduling Unrelated Job SUM (minimization) problem: a special case with m=1.

- INPUT: *one* machine, *n* jobs (*i = 1, . . . , n*), $p_i > 0$. (we use p_i since when m=1 clearly there is no difference between unrelated and identical machines)

- OUTPUT: a schedule ***S****=(S*$_1$*) where S*$_1$ *=(S*$_{1,1}$*,…,S*$_{1,n}$*).*

- GOAL*:* Minimizing the Job SUM, that is minimizing $\sum_{i=1}^{n} JC_i(S)$

Does such problem admit a polynomial time algorithm that finds a schedule that minimizes the Job SUM?