Fundamental Computer Science
Lecture 5: Approximation (1)

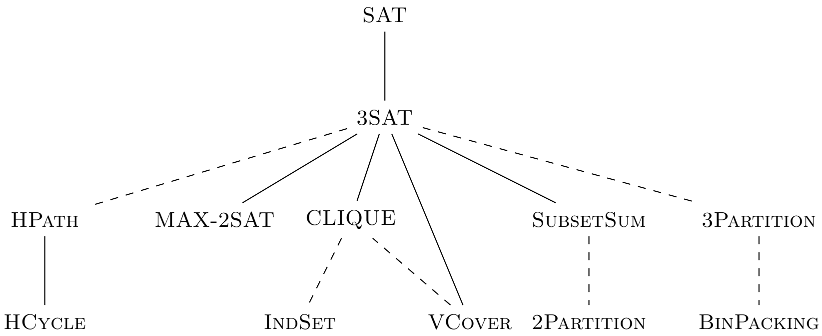
Denis Trystram
MoSIG1 and M1Info – University Grenoble-Alpes

March, 2021

Agenda

- Introduction to Approximation
- A new problem: scheduling
  - Complexity analysis
    Studying two variants
  - A path for discussing the various aspects of approximation

Some NP-complete problems



Dealing with NP-HARD (optimization) problems

There are multiple ways to *solve* a NP-hard problem...

- exact algorithms
  - exact optimal solution but non-polynomial complexity
  - efficient for small instances
  - methodology: dynamic programming, branch-and-bound, pseudo-polynomial algorithms
- study special cases
  - could be polynomially solvable
  - examples: 2-SAT
- heuristics
  - non-optimal solution in polynomial time
  - without guarantees but good performance in practice
- randomized algorithms
  - polynomial-time complexity
  - produce the optimal with high probability

Dealing with NP-HARD (optimization) problems

Another –trade-off– solution is the following:

- exact algorithms
- study special cases
- approximation algorithms
  - non-optimal solution
  - running in polynomial time
  - theoretical worst case guarantees:
    the solution of the algorithm is not too far from the optimal
- heuristics
- randomized algorithms

Approximation ratio

- consider a problem Π and an algorithm $\mathcal{A}$ for solving this problem
- OPT_I: the objective value of an optimal solution for the instance I of the problem Π
- SOL_I: the objective value of the solution of our algorithm $\mathcal{A}$ for the instance I of the problem Π

Approximation ratio

- consider a problem Π and an algorithm $\mathcal{A}$ for solving this problem
- OPT_I: the objective value of an optimal solution for the instance I of the problem Π
- SOL_I: the objective value of the solution of our algorithm $\mathcal{A}$ for the instance I of the problem Π

approximation ratio (for a minimization problem)

$$\rho = \max_{I \in \text{Instances}} \left\{ \frac{SOL_I}{OPT_I} \right\}$$

- for each instance I: $\quad OPT_I \leq SOL_I \leq \rho \cdot OPT_I$
- $\rho > 1$

Approximation ratio

approximation ratio (for a maximization problem)

$$\rho = \max_{I \in \text{Instances}} \left\{ \frac{OPT_I}{SOL_I} \right\}$$

- for each instance I: $\quad OPT_I \geq SOL_I \geq \frac{1}{\rho} \cdot OPT_I = \rho' \cdot OPT_I$
- $\rho > 1$, $\rho' < 1$

Case study on scheduling

Scheduling on parallel machines

Input: a set $\mathcal{J}$ of n jobs, a set $\mathcal{M}$ of m identical machines, a processing time $p_j \in \mathbb{N}^+$ for each job $J_j \in \mathcal{J}$, and a positive integer $C_{\max}$

Question: is there a schedule of all jobs on the machines such that no machine executes two jobs at the same time and all jobs are completed before time $C_{\max}$?

Scheduling on parallel machines

Input: a set $\mathcal{J}$ of n jobs, a set $\mathcal{M}$ of m identical machines, a processing time $p_j \in \mathbb{N}^+$ for each job $J_j \in \mathcal{J}$, and a positive integer $C_{\max}$

Question: is there a schedule of all jobs on the machines such that no machine executes two jobs at the same time and all jobs are completed before time $C_{\max}$?

- all jobs are available at time zero
- C_j: completion time of job J_j
- $C_{\max} = \max_{J_j \in \mathcal{J}} \{C_j\}$
- optimization version: minimize the maximum completion time over all jobs (makespan or schedule's length)

Scheduling on parallel machines

Input: a set $\mathcal{J}$ of n jobs, a set $\mathcal{M}$ of m identical machines, a processing time $p_j \in \mathbb{N}^+$ for each job $J_j \in \mathcal{J}$, and a positive integer $C_{\max}$

Question: is there a schedule of all jobs on the machines such that no machine executes two jobs at the same time and all jobs are completed before time $C_{\max}$?

- all jobs are available at time zero
- C_j: completion time of job J_j
- $C_{\max} = \max\limits_{J_j \in \mathcal{J}} \{C_j\}$
- optimization version: minimize the maximum completion time over all jobs (makespan or schedule's length)
- denoted in short by: $P \mid\mid C_{\max}$

Three-field notation for scheduling: $\alpha \mid \beta \mid \gamma$

α: machine environment

- 1: single machine
- P: identical parallel machines
- $P2$: two identical parallel machines
- Q: related parallel machines
- R: unrelated parallel machines

Three-field notation for scheduling: $\alpha \mid \beta \mid \gamma$

α: machine environment

- 1: single machine
- P: identical parallel machines
- $P2$: two identical parallel machines
- Q: related parallel machines
- R: unrelated parallel machines

β: jobs characteristics / constraints

- r_j: release date of J_j
- $pmtn$: preemptions and migrations are allowed, dup duplication
- $prec$: precedence constraints
- w_j: weight implying priority or p_j processing times of J_j

Three-field notation for scheduling: $\alpha \mid \beta \mid \gamma$

α: machine environment

- 1: single machine
- P: identical parallel machines
- $P2$: two identical parallel machines
- Q: related parallel machines
- R: unrelated parallel machines

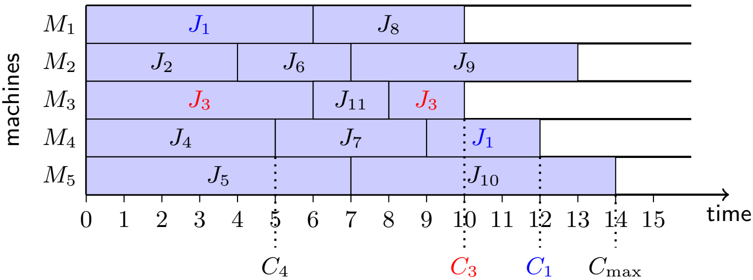
β: jobs characteristics / constraints

- r_j: release date of J_j
- $pmtn$: preemptions and migrations are allowed, dup duplication
- $prec$: precedence constraints
- w_j: weight implying priority or p_j processing times of J_j

γ: objective

- $C_{\max} = \max_{j \in \mathcal{J}} \{C_j\}$: schedule's length or makespan
- $\sum C_j$: average completion time
- $F_{\max} = \max_{j \in \mathcal{J}} \{C_j - r_j\}$: maximum flow-time
- $\sum F_j = \sum_{j \in \mathcal{J}} (C_j - r_j)$: average flow-time

Gantt chart



- J_2 is non-preemptively executed ($p_2 = 4$)
- J_3 is preempted ($p_3 = 8$)
- J_1 is preempted and migrated ($p_1 = 9$)
- makespan: $C_{\max} = C_{10} = 14$

Related questions

- $P \mid pmtn \mid C_{\max}$: polynomial or NP-COMPLETE?

Related questions

- $P \mid pmtn \mid C_{\max}$: polynomial or NP-complete?
- $P2 \mid\mid C_{\max}$: polynomial or NP-complete?
- $P2 \mid p_j = 1 \mid C_{\max}$: polynomial or NP-complete?
- $P2 \mid prec \mid C_{\max}$: polynomial or NP-complete?

Related questions

- $P \mid pmtn \mid C_{\max}$: polynomial or NP-complete?
- $P2 \mid\mid C_{\max}$: polynomial or NP-complete?
- $P2 \mid p_j = 1 \mid C_{\max}$: polynomial or NP-complete?
- $P2 \mid prec \mid C_{\max}$: polynomial or NP-complete?
- $P3 \mid\mid C_{\max}$: polynomial or NP-complete?
- $P3 \mid p_j = 1 \mid C_{\max}$: polynomial or NP-complete?

Related questions

- $P \mid pmtn \mid C_{\max}$: polynomial or NP-COMPLETE?
- $P2 \,||\, C_{\max}$: polynomial or NP-COMPLETE?
- $P2 \mid p_j = 1 \mid C_{\max}$: polynomial or NP-COMPLETE?
- $P2 \mid prec \mid C_{\max}$: polynomial or NP-COMPLETE?
- $P3 \,||\, C_{\max}$: polynomial or NP-COMPLETE?
- $P3 \mid p_j = 1 \mid C_{\max}$: polynomial or NP-COMPLETE?
- $P \,||\, C_{\max}$: polynomial or NP-COMPLETE?

Analysis of two scheduling problems

- $P \mid\mid C_{\max}$
- $P2 \mid prec \mid C_{\max}$

$P \mid\mid C_{\max}$: complexity analysis

- $P2 \mid\mid C_{\max}$ is weakly NP-complete by a straight forward reduction from $2\textsc{Partition}$
- Thus, $P \mid\mid C_{\max}$ is also NP-complete (in the weak sense) since $P2 \mid\mid C_{\max}$ is a particular sub-problem

$P \mid\mid C_{\max}$: complexity analysis

- $P2 \mid\mid C_{\max}$ is weakly NP-complete by a straight forward reduction from $2\textsc{Partition}$
- Thus, $P \mid\mid C_{\max}$ is also NP-complete (in the weak sense) since $P2 \mid\mid C_{\max}$ is a particular sub-problem
- Can we expect a more precise result?

$P \mid\mid C_{\max}$: complexity analysis

- $P2 \mid\mid C_{\max}$ is weakly NP-complete by a straight forward reduction from 2PARTITION
- Thus, $P \mid\mid C_{\max}$ is also NP-complete (in the weak sense) since $P2 \mid\mid C_{\max}$ is a particular sub-problem
- Can we expect a more precise result?

  Yes!
  It is NP-complete in the strong sense.
  We will show this result by a reduction from 3-PARTITION:
  3-PARTITION $\leq_P P \mid\mid C_{\max}$

Proof of the reduction

3-PARTITION

Input: A positive integer B and a set $\mathcal{J}$ of $3n$ integers denoted by p_j with values in the interval $[B/4, B/2]$ and $\Sigma_{j \in \mathcal{J}} p_j = n \cdot B$

Question: is there a partition into n multi-sets (each containing exactly 3 integers) such that the integers within each set sums up to B?

Transformation

Consider an instance of 3-PARTITION $< B, \mathcal{J} >$
we define an instance of $P \mid\mid C_{\max}$ as follows:

- $m = \mid \mathcal{J} \mid /3$
- For each item j of $\mathcal{J}$,
  define one task whose processing time is p_j
- $C_{max} = B$

Proof

We prove now that an instance of 3-PARTITION is positive iff the transformed instance for $P \,||\, C_{\max}$ is positive

($\Rightarrow$) Start by a positive instance of 3-PARTITION.
We assign each of these sets to one machine,
the makespan is B, thus, the instance is positive.

($\Leftarrow$) Assume now that the instance of $P \,||\, C_{\max}$ is positive.

- Since $\Sigma_{j \in \mathcal{J}} = n$, each of the m machines has a load of at least B in this schedule.
- Thus, partitioning the numbers into sets corresponding to the sets of tasks delivers a partition as required.
- It is a positive instance of 3-PARTITION as well.

$P \mid\mid C_{\max}$: a first approximation algorithm

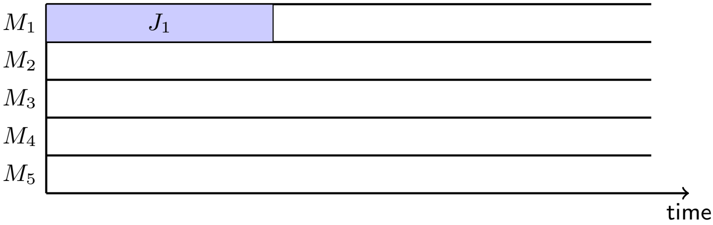
List Scheduling (LS) [Graham]

- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order

$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

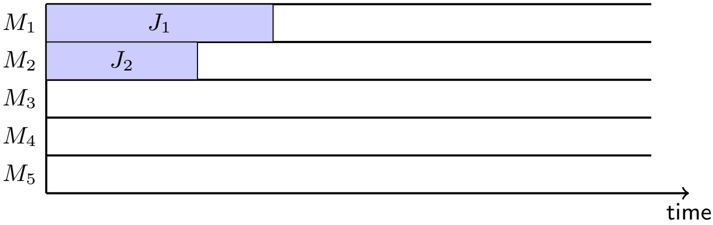
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

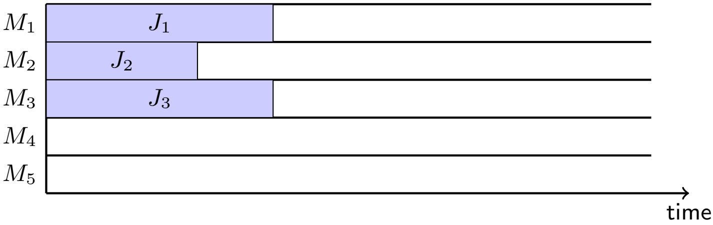
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

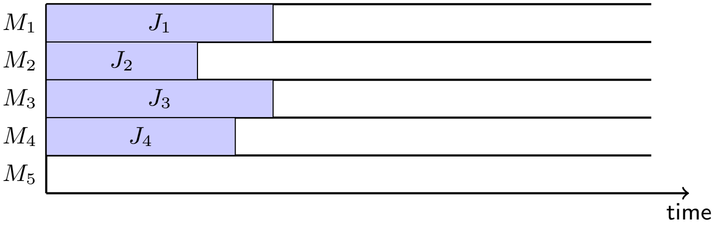
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

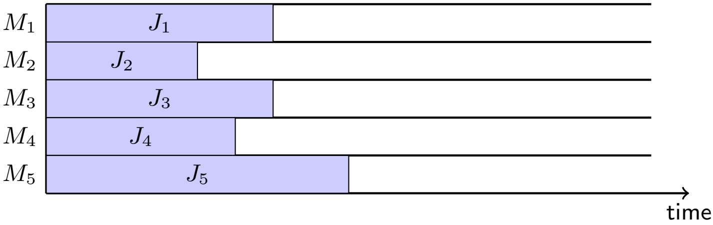
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

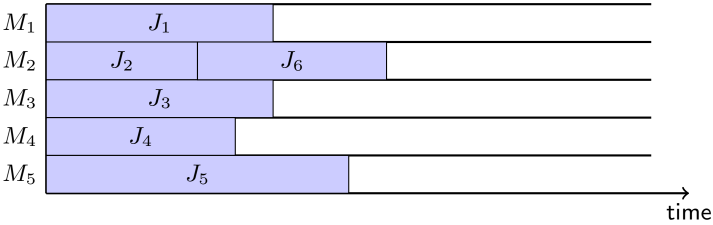
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

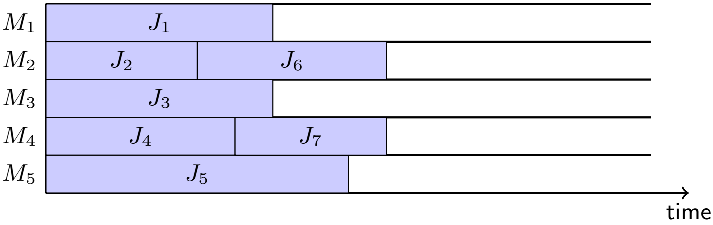
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

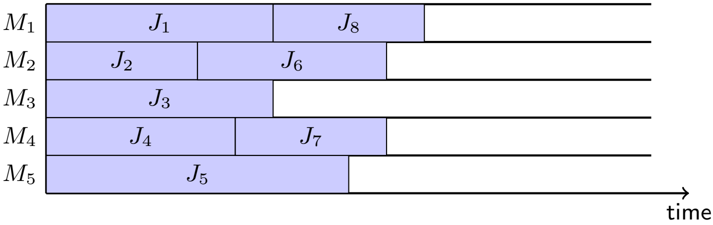
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

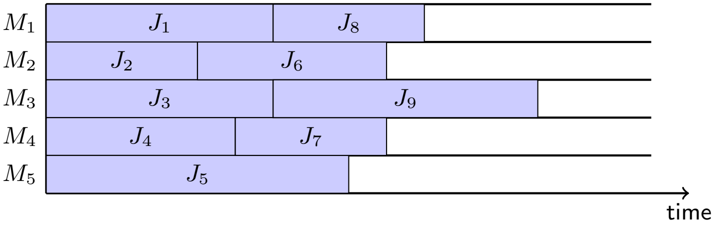
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

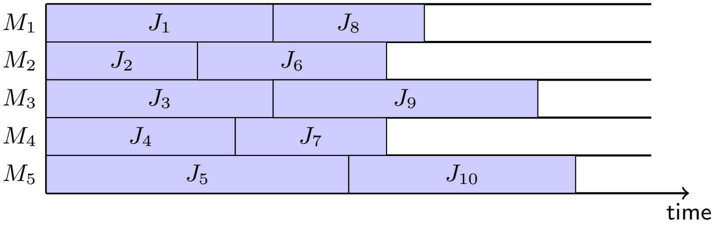
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

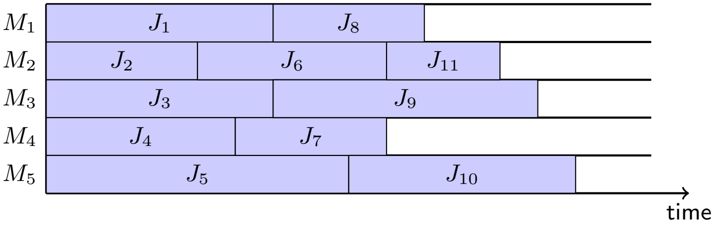
- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

List Scheduling (LS) [Graham]

- consider the jobs in an arbitrary order, $J_1, J_2, \ldots, J_n$
- each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order



$P \mid\mid C_{\max}$: a first approximation algorithm

- how to calculate the objective function of the optimal solution?
  - we cannot (because the problem is NP-COMPLETE...)

$P \mid\mid C_{\max}$: a first approximation algorithm

- how to calculate the objective function of the optimal solution?
  - we cannot (because the problem is NP-COMPLETE...)
- use estimations (lower bounds): $OPT \geq LB$

$$SOL \leq \rho \cdot LB$$

$P \mid\mid C_{\max}$: a first approximation algorithm

- how to calculate the objective function of the optimal solution?
  - we cannot (because the problem is NP-complete...)
- use estimations (lower bounds): $OPT \geq LB$

$$SOL \leq \rho \cdot LB \leq \rho \cdot OPT$$

$P \mid\mid C_{\max}$: a first approximation algorithm

- how to calculate the objective function of the optimal solution?
  - we cannot (because the problem is NP-COMPLETE...)
- use estimations (lower bounds): $OPT \geq LB$

$$SOL \leq \rho \cdot LB \leq \rho \cdot OPT$$

- what lower bounds can we use for $P \mid\mid C_{\max}$?

$P \mid\mid C_{\max}$: a first approximation algorithm

- how to calculate the objective function of the optimal solution?
  - we cannot (because the problem is NP-COMPLETE...)
- use estimations (lower bounds): $OPT \geq LB$

$$SOL \leq \rho \cdot LB \leq \rho \cdot OPT$$

- what lower bounds can we use for $P \mid\mid C_{\max}$?
  - total load:

$$Load = \frac{1}{m} \sum_{J_j \in \mathcal{J}} p_j \leq OPT$$

$P \mid\mid C_{\max}$: a first approximation algorithm

- how to calculate the objective function of the optimal solution?
  - we cannot (because the problem is NP-COMPLETE...)
- use estimations (lower bounds): $OPT \geq LB$

$$SOL \leq \rho \cdot LB \leq \rho \cdot OPT$$

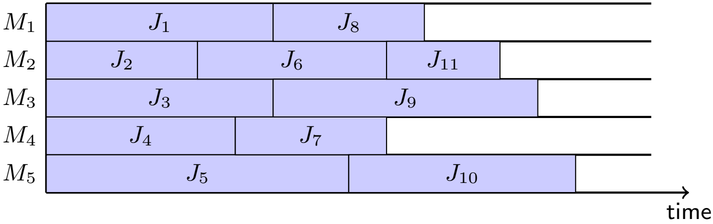
- what lower bounds can we use for $P \mid\mid C_{\max}$?
  - total load:

$$Load = \frac{1}{m} \sum_{J_j \in \mathcal{J}} p_j \leq OPT$$

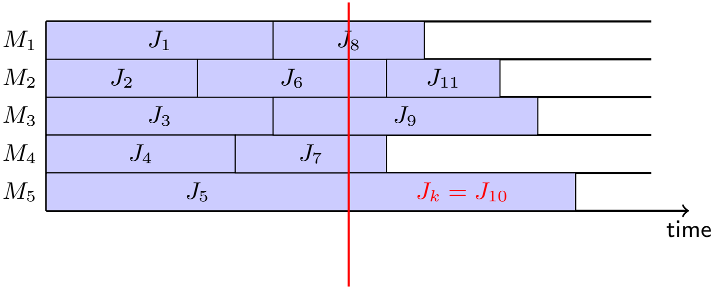
  - maximum processing time:

$$p_{\max} = \max\{p_j \mid J_j \in \mathcal{J}\} \leq OPT$$

$P \mid\mid C_{\max}$: a first approximation algorithm

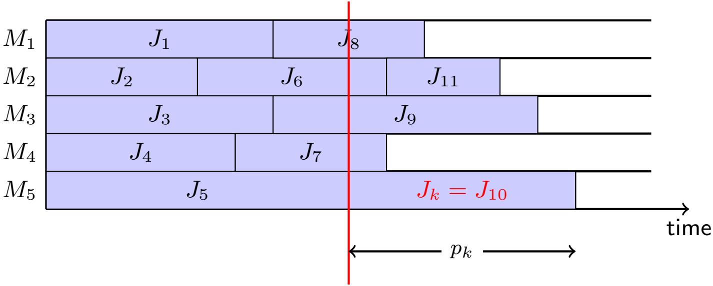


$P \mid\mid C_{\max}$: a first approximation algorithm



- J_k: the job that completes last

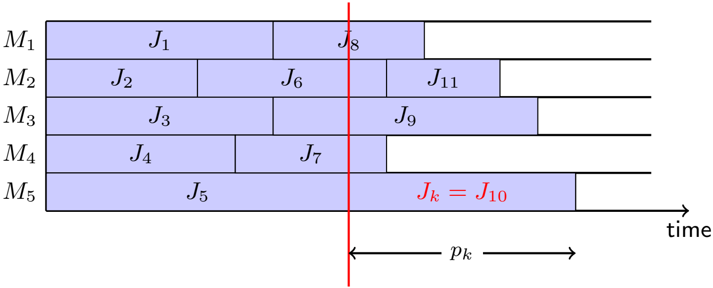
$P \mid\mid C_{\max}$: a first approximation algorithm



- J_k: the job that completes last

$$C_{\max} \quad \leq \quad \frac{1}{m} \sum_{J_j \neq J_k} p_j$$

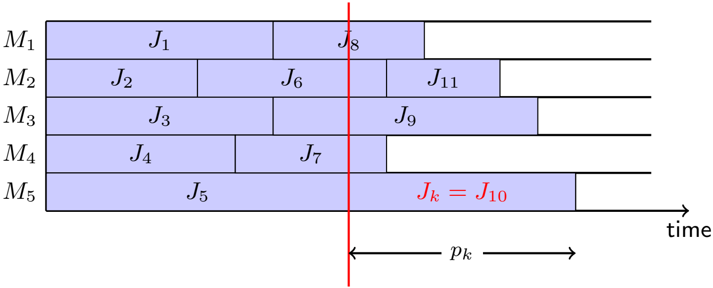
$P \mid\mid C_{\max}$: a first approximation algorithm



- J_k: the job that completes last

$$C_{\max} \quad \leq \quad \frac{1}{m} \sum_{J_j \neq J_k} p_j + p_k$$

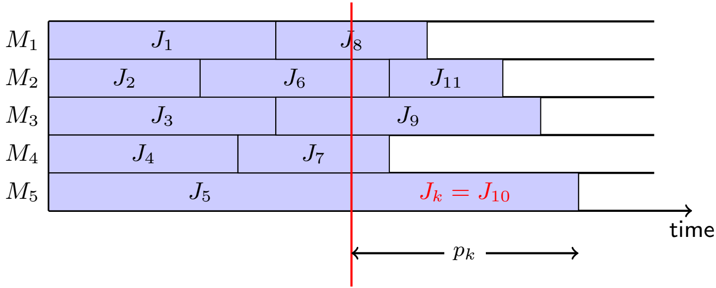
$P \,||\, C_{\max}$: a first approximation algorithm



- J_k: the job that completes last

$$C_{\max} \quad \leq \quad \frac{1}{m} \sum_{J_j \neq J_k} p_j + p_k \;=\; \frac{1}{m} \sum_{J_j} p_j + \frac{m-1}{m} p_k$$

$P \mid\mid C_{\max}$: a first approximation algorithm

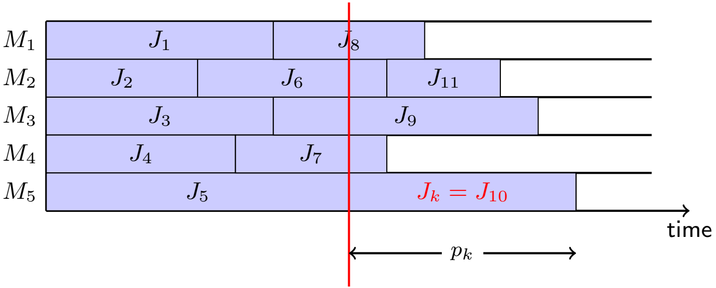


- J_k: the job that completes last

$$C_{\max} \leq \frac{1}{m} \sum_{J_j \neq J_k} p_j + p_k = \frac{1}{m} \sum_{J_j} p_j + \frac{m-1}{m} p_k$$

$$\leq Load + \frac{m-1}{m} p_{\max}$$

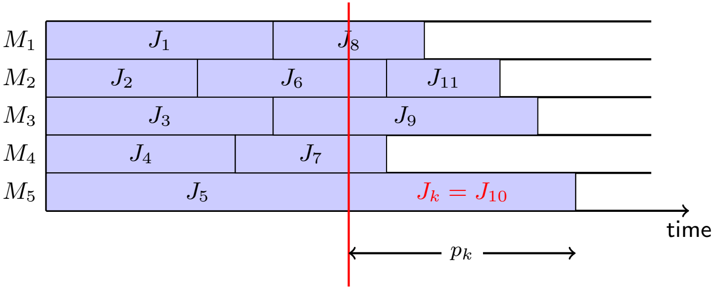
$P \mid\mid C_{\max}$: a first approximation algorithm



- J_k: the job that completes last

$$
\begin{aligned}
C_{\max} \quad &\leq \quad \frac{1}{m} \sum_{J_j \neq J_k} p_j + p_k \;=\; \frac{1}{m} \sum_{J_j} p_j + \frac{m-1}{m} p_k \\
&\leq \quad Load + \frac{m-1}{m} p_{\max} \;\leq\; OPT + \frac{m-1}{m} OPT
\end{aligned}
$$

$P \mid\mid C_{\max}$: a first approximation algorithm



- J_k: the job that completes last

$$
\begin{aligned}
C_{\max} &\le \frac{1}{m}\sum_{J_j \neq J_k} p_j + p_k = \frac{1}{m}\sum_{J_j} p_j + \frac{m-1}{m} p_k \\
&\le Load + \frac{m-1}{m} p_{\max} \le OPT + \frac{m-1}{m} OPT = \left(2 - \frac{1}{m}\right) OPT
\end{aligned}
$$

$P \mid\mid C_{\max}$

Theorem

List Scheduling achieves an approximation ratio of $2 - \frac{1}{m}$.

$P \mid\mid C_{\max}$

Theorem

List Scheduling achieves an approximation ratio of $2 - \frac{1}{m}$.

Questions

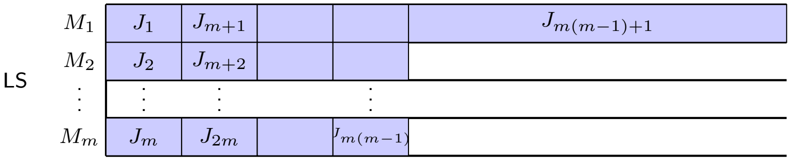
- can we improve the analysis?
- is there a better approximation algorithm?

$P \mid\mid C_{\max}$: can we improve the analysis of LS?

- No!

$P \mid\mid C_{\max}$: can we improve the analysis of LS?

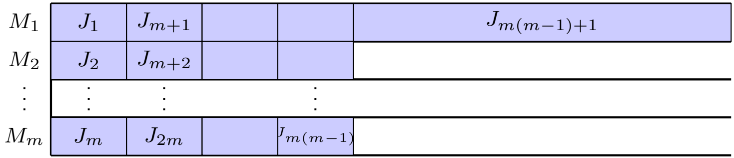
- No!
- consider the following instance
  - $n = m(m-1) + 1$ jobs
  - $p_1 = p_2 = \ldots = p_{m(m-1)} = 1$ and $p_{m(m-1)+1} = m$



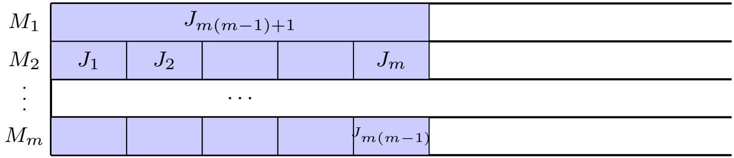
$P \mid\mid C_{\max}$: can we improve the analysis of LS?

- No!
- consider the following instance
  - $n = m(m-1) + 1$ jobs
  - $p_1 = p_2 = \ldots = p_{m(m-1)} = 1$ and $p_{m(m-1)+1} = m$

LS

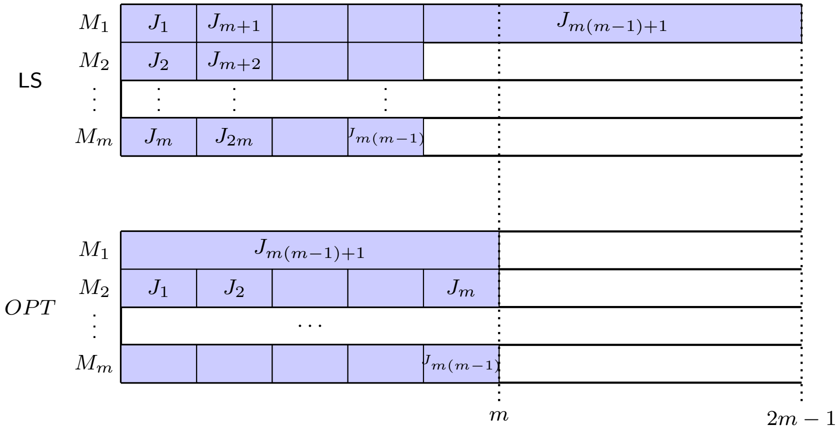


OPT



$P \mid\mid C_{\max}$: can we improve the analysis of LS?

- No!
- consider the following instance
  - $n = m(m-1) + 1$ jobs
  - $p_1 = p_2 = \ldots = p_{m(m-1)} = 1$ and $p_{m(m-1)+1} = m$



$P \mid\mid C_{\max}$: a refined algorithm

Longest Processing Time (LPT)

1: consider the jobs in non-increasing order of their processing times, i.e., $p_1 \geq p_2 \geq \ldots \geq p_n$

2: each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order

$P \mid\mid C_{\max}$: a refined algorithm

Longest Processing Time (LPT)

1: consider the jobs in non-increasing order of their processing times, i.e., $p_1 \geq p_2 \geq \ldots \geq p_n$

2: each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order

Analysis

- J_k: the job that completes last which is assigned to M_i

$P \mid\mid C_{\max}$: a refined algorithm

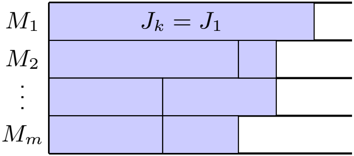
Longest Processing Time (LPT)

1: consider the jobs in non-increasing order of their processing times, i.e., $p_1 \geq p_2 \geq \ldots \geq p_n$

2: each time a machine becomes idle, schedule on it the first non-scheduled job according to the above order

Analysis

- J_k: the job that completes last which is assigned to M_i
- if J_k is the only job on its machine
  - $J_k = J_1$
  - $M_i = M_1$
  - LPT creates the optimal schedule with $C_{\max} = p_1 = p_{\max}$



$P \mid\mid C_{\max}$: a refined algorithm

- if there are at least two jobs in M_i
  - $k \geq m+1$ and $p_k \leq p_{m+1}$ (due to LPT rule)
  - there are at least $m+1$ jobs
  - in the optimal solution: there is a machine to which are assigned at least two jobs in $\{J_1, J_2, \ldots, J_{m+1}\}$
  - $OPT \geq 2p_{m+1}$
  - as in LS:

$$C_{\max} \quad \leq \quad \frac{1}{m} \sum_{J_j \neq J_k} p_j + p_k \ \leq \ \frac{1}{m} \sum_{J_j} p_j + \frac{m-1}{m} p_k$$

$P \,||\, C_{\max}$: a refined algorithm

- if there are at least two jobs in M_i
  - $k \geq m+1$ and $p_k \leq p_{m+1}$ (due to LPT rule)
  - there are at least $m+1$ jobs
  - in the optimal solution: there is a machine to which are assigned at least two jobs in $\{J_1, J_2, \ldots, J_{m+1}\}$
  - $OPT \geq 2p_{m+1}$
  - as in LS:

$$\begin{aligned} C_{\max} &\leq \frac{1}{m}\sum_{J_j \neq J_k} p_j + p_k \leq \frac{1}{m}\sum_{J_j} p_j + \frac{m-1}{m} p_k \\ &\leq Load + \frac{m-1}{m} p_{m+1} \end{aligned}$$

$P \mid\mid C_{\max}$: a refined algorithm

- if there are at least two jobs in M_i
  - $k \geq m+1$ and $p_k \leq p_{m+1}$ (due to LPT rule)
  - there are at least $m+1$ jobs
  - in the optimal solution: there is a machine to which are assigned at least two jobs in $\{J_1, J_2, \ldots, J_{m+1}\}$
  - $OPT \geq 2p_{m+1}$
  - as in LS:

$$
\begin{aligned}
C_{\max} &\leq \frac{1}{m}\sum_{J_j \neq J_k} p_j + p_k \leq \frac{1}{m}\sum_{J_j} p_j + \frac{m-1}{m} p_k \\
&\leq Load + \frac{m-1}{m} p_{m+1} \leq OPT + \frac{m-1}{m} \cdot \frac{OPT}{2}
\end{aligned}
$$

$P \mid\mid C_{\max}$: a refined algorithm

- if there are at least two jobs in M_i
  - $k \geq m + 1$ and $p_k \leq p_{m+1}$ (due to LPT rule)
  - there are at least $m + 1$ jobs
  - in the optimal solution: there is a machine to which are assigned at least two jobs in $\{J_1, J_2, \ldots, J_{m+1}\}$
  - $OPT \geq 2p_{m+1}$
  - as in LS:

$$
\begin{aligned}
C_{\max} &\leq \frac{1}{m}\sum_{J_j \neq J_k} p_j + p_k \;\leq\; \frac{1}{m}\sum_{J_j} p_j + \frac{m-1}{m} p_k \\
&\leq Load + \frac{m-1}{m} p_{m+1} \;\leq\; OPT + \frac{m-1}{m} \cdot \frac{OPT}{2} \\
&\leq \left(\frac{3}{2} - \frac{1}{2m}\right) OPT
\end{aligned}
$$

$P \mid\mid C_{\max}$: a refined algorithm

- if there are at least two jobs in M_i
  - $k \geq m+1$ and $p_k \leq p_{m+1}$ (due to LPT rule)
  - there are at least $m+1$ jobs
  - in the optimal solution: there is a machine to which are assigned at least two jobs in $\{J_1, J_2, \ldots, J_{m+1}\}$
  - $OPT \geq 2p_{m+1}$
  - as in LS:

$$
\begin{aligned}
C_{\max} &\leq \frac{1}{m}\sum_{J_j \neq J_k} p_j + p_k \leq \frac{1}{m}\sum_{J_j} p_j + \frac{m-1}{m} p_k \\
&\leq Load + \frac{m-1}{m} p_{m+1} \leq OPT + \frac{m-1}{m} \cdot \frac{OPT}{2} \\
&\leq \left(\frac{3}{2} - \frac{1}{2m}\right) OPT
\end{aligned}
$$

- Can we provide a better analysis of LPT? See the Lab. session of today.

$P2 \mid prec \mid C_{\max}$

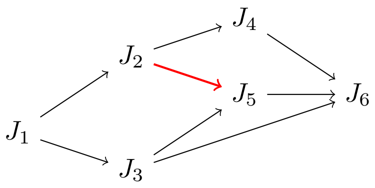
This is an optimization problem.
Let consider the decision version.

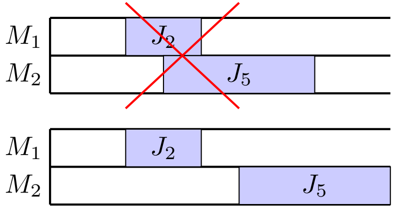
- Definition
- Complexity
- Approximation

$P2 \mid prec \mid C_{\max}$

Input: a set $\mathcal{J}$ of n jobs, 2 identical machines, a processing time $p_j \in \mathbb{N}^+$ for each job $J_j \in \mathcal{J}$, a directed graph $G = (V, E)$ describing precedence relations between jobs, and a positive integer $C_{\max}$

Question: is there a schedule of all jobs on the two machines s.t.
(i) no machine executes two jobs at the same time,
(ii) for each two jobs J_j and $J_{j'}$, if there is an arc $(J_j, J_{j'})$, then $J_{j'}$ cannot start its execution before the completion of J_j, and
(iii) all jobs are completed before time $C_{\max}$?





Compexity of $P2 \mid prec \mid C_{\max}$

- $P2 \mid prec \mid C_{\max}$ is $\text{NP-}\textsc{complete}$ as generalization of $P2 \mid\mid C_{\max}$
  - however, in the weak sense

Compexity of $P2 \mid prec \mid C_{\max}$

- $P2 \mid prec \mid C_{\max}$ is NP-COMPLETE as generalization of $P2 \mid\mid C_{\max}$
  - however, in the weak sense
- We will prove that it is also strongly NP-COMPLETE

Compexity of $P2 \mid prec \mid C_{\max}$

- $P2 \mid prec \mid C_{\max}$ is NP-COMPLETE as generalization of $P2 \mid\mid C_{\max}$
  - however, in the weak sense
- We will prove that it is also strongly NP-COMPLETE
- $P \mid prec \mid C_{\max}$ is strongly NP-COMPLETE
  - as generalization of $P2 \mid prec \mid C_{\max}$

Complexity of $P2 \mid prec \mid C_{\max}$

Let prove that the problem is strongly NP-COMPLETE.
What can be a reference problem?

Complexity of $P2 \mid prec \mid C_{\max}$

Let prove that the problem is strongly NP-COMPLETE.
What can be a reference problem?

Reduction from 3-PARTITION

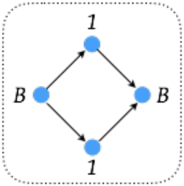
The idea is:

- to construct an adequate graph
- to relate its execution to successive equal-sized intervals

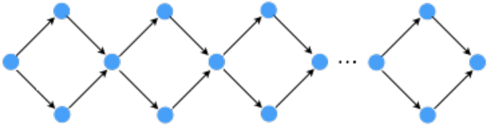
Exercise: write the detailed proof.

Idea of the reduction

Let consider the following gadget:

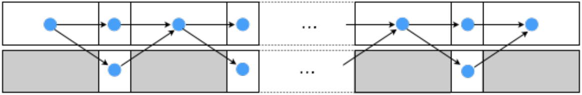


We concatenate $n-1$ such gadgets:



Its execution creates n idle intervals of length B where $3n$ remaining tasks will be scheduled according to an instance of 3-Partition.

Reduction



3-PARTITION

Input: A positive integer B and a set $\mathcal{J}$ of $3n$ integers denoted by p_j with values in the interval $[B/4, B/2]$ and $\Sigma_{j \in \mathcal{J}} p_j = n \cdot B$

Question: is there a partition into n multi-sets (each containing exactly 3 integers) such that the integers within each set sums up to B?

Transformation

The $3n$ integers of 3-PARTITION remain the same, they correspond to independent tasks.

The transformed instance adds the previous precedence graph.

$C_{\max} = n \cdot B + n - 1$

Proof

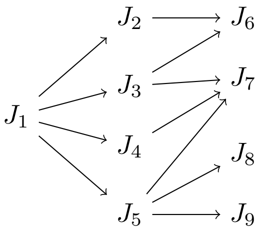
3-$\textsc{Partition} \leq_P P2 \mid prec \mid C_{\max}$

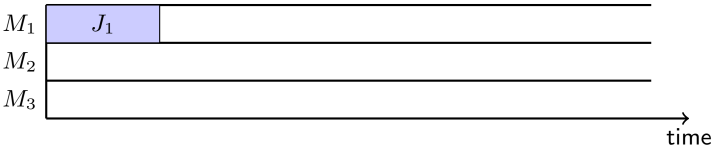
- $(\Rightarrow)$
  This is the easy part since the solution of 3-$\textsc{Partition}$ fits perfectly into the n intervals.
  Thus, the schedule is valid and optimal.
- $(\Leftarrow)$
  - The makespan of a solution of $P2 \mid prec \mid C_{\max}$ is $n \cdot B + n - 1$ and there is no other solution than the schedule with the previous shape.
  - Thus, the $3n$ independent tasks should be scheduled into the n intervals of length B that is a solution of 3-$\textsc{Partition}$

$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

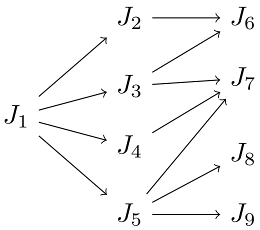


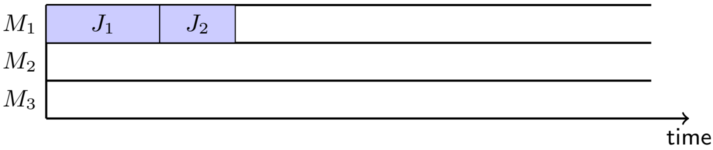


$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

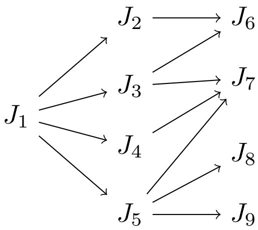


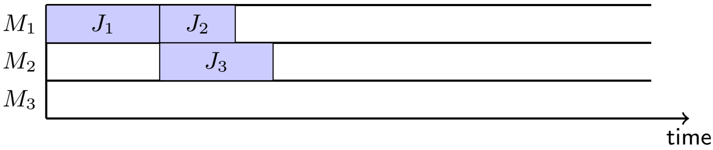


$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job,
i.e. a job whose predecessors are already completed

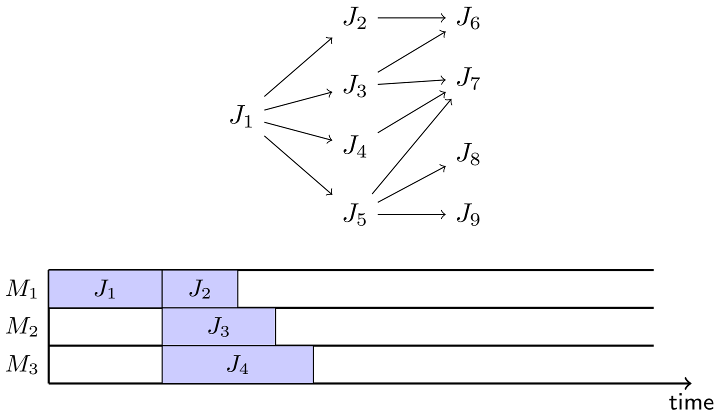




$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

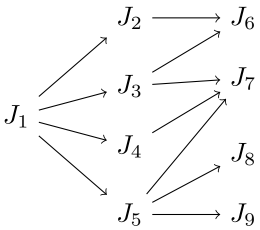
each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

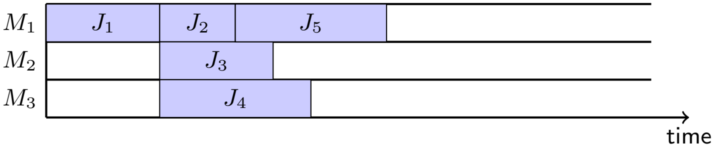


$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

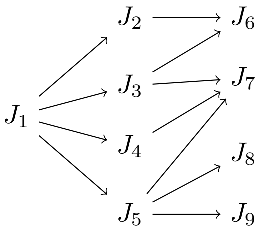


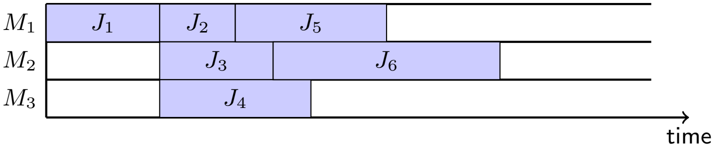


$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

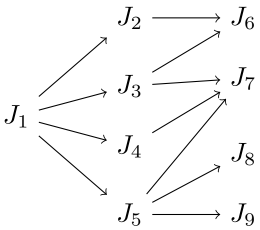


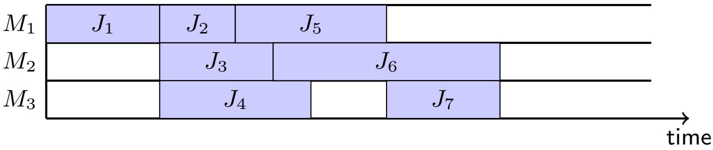


$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

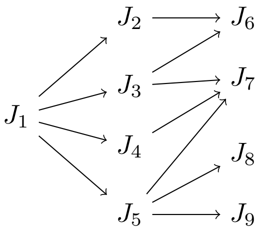


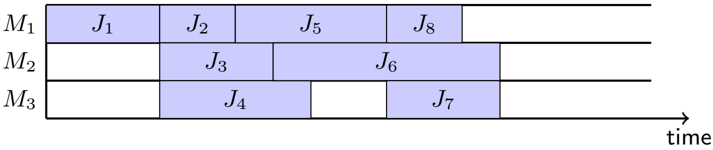


$P \mid prec \mid C_{\max}$: an approximation algorithm

List Scheduling (LS)

each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed

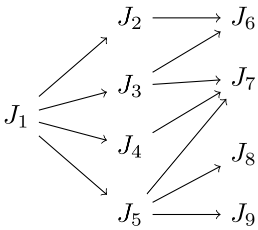


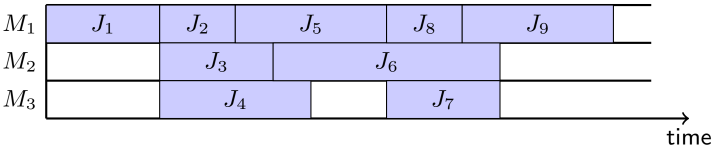


$P \mid prec \mid C_{\max}$: an approximation algorithm

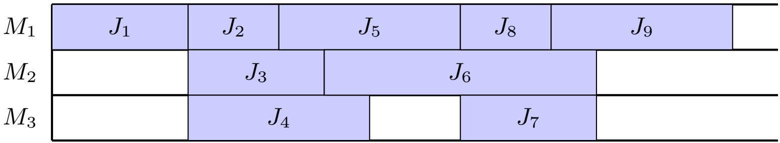
List Scheduling (LS)

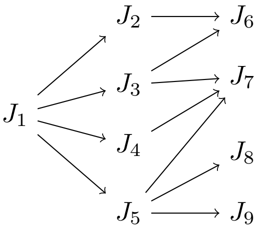
each time a machine becomes idle, schedule on it any ready job, i.e. a job whose predecessors are already completed



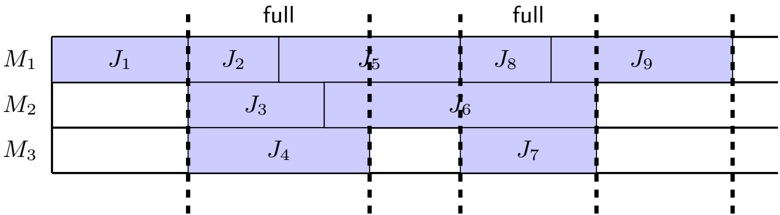


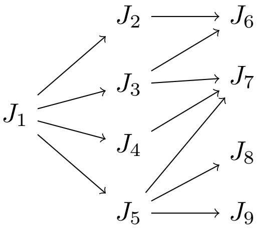
$P \mid prec \mid C_{\max}$: an approximation algorithm





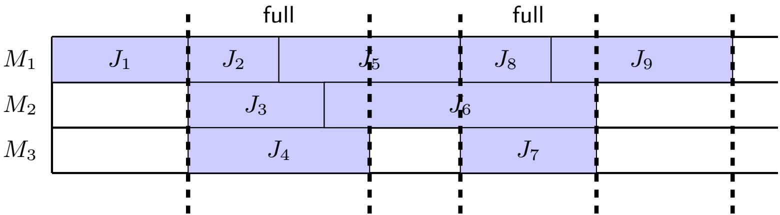
$P \mid prec \mid C_{\max}$: an approximation algorithm

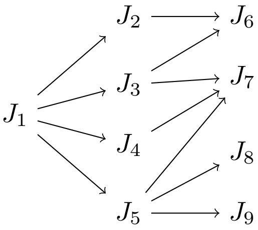




- full intervals bounded by total load

$P \mid prec \mid C_{\max}$: an approximation algorithm





- full intervals bounded by total load
- non-full intervals: there is a path in the precedence graph covering them ($J_1 \to J_5 \to J_9$)
- $OPT \geq$ maximum path (known as *critical path*)

$P \mid prec \mid C_{\max}$: an approximation algorithm

Analysis:

$$C_{\max} \;\leq\; \textsf{Load} + \textsf{Critical Path} \;\leq\; 2 \cdot OPT$$

$P \mid prec \mid C_{\max}$: an approximation algorithm

Analysis:

$$C_{\max} \;\leq\; \text{Load} + \text{Critical Path} \;\leq\; 2 \cdot OPT$$

- this ratio is tight (we cannot improve the analysis)

$P \mid prec \mid C_{\max}$: an approximation algorithm

Analysis:

$$C_{\max} \;\leq\; \text{Load} + \text{Critical Path} \;\leq\; 2 \cdot OPT$$

- this ratio is tight (we cannot improve the analysis)
- there is no algorithm for $P \mid prec \mid C_{\max}$ with approximation ratio smaller than 2 [Svensson 2007]

$P \mid prec \mid C_{\max}$: an approximation algorithm

Analysis:

$$C_{\max} \;\leq\; \text{Load} + \text{Critical Path} \;\leq\; 2 \cdot OPT$$

- this ratio is tight (we cannot improve the analysis)
- there is no algorithm for $P \mid prec \mid C_{\max}$ with approximation ratio smaller than 2 [Svensson 2007]
- how to show in-approximability results?

Gap reductions

- Π_1: a decision problem
- Π_2: a minimization problem
- f, α: two functions

A gap-introducing reduction transforms an instance I_1 of Π_1 to an instance I_2 of Π_2 such that

- if I_1 has a solution, then $OPT(I_2) \leq f(I_2)$
- if I_1 has no solution, then $OPT(I_2) > \alpha(|I_2|) \cdot f(I_2)$

Gap reductions

- Π_1: a decision problem
- Π_2: a minimization problem
- f, α: two functions

A gap-introducing reduction transforms an instance I_1 of Π_1 to an instance I_2 of Π_2 such that

- if I_1 has a solution, then $OPT(I_2) \leq f(I_2)$
- if I_1 has no solution, then $OPT(I_2) > \alpha(|I_2|) \cdot f(I_2)$

- usage
  - Π_1: an NP-COMPLETE problem
  - Π_2: our problem
  - α: the gap
- meaning: based on the value of the solution of our problem we can decide Π_1 which is NP-COMPLETE (contradiction)

Application to BinPacking

Bin-Packing

Input: a set of items A, a size $s(a)$ for each $a \in A$, a positive integer capacity C, and a positive integer k

Question: is there a partition of A into disjoint sets $A_1, A_2, \ldots, A_k$ such that the total size of the elements in each set A_j does not exceed the capacity C, i.e., $\sum_{a \in A_j} s(a) \leq C$?

Let us first prove that BinPacking is in NP-complete.
This is easy by a simple reduction from 2Partition.

Application to BinPacking

BinPacking can not be approximated by a factor better than $3/2$

Application to BinPacking

BinPacking can not be approximated by a factor better than $3/2$

The proof is by contradiction:

- assume by contradiction that it can be approximated by $\rho < 3/2$
- apply the gap reduction to a positive instance of $< A, C, 2 >$

Application to BinPacking

BinPacking can not be approximated by a factor better than $3/2$

The proof is by contradiction:

- assume by contradiction that it can be approximated by $\rho < 3/2$
- apply the gap reduction to a positive instance of $< A, C, 2 >$
- As the number of bins is an integer, the approximation also leads to an integer value < 3
- Thus, solving this problem corresponds to solve 2Partition in polynomial time, unless $\mathcal{P} = \mathcal{NP}$