Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$\begin{array}{llrcl}
\min & & & & L \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\
& \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\
& \forall i, j & x_{j,i} & \in & \{0,1\}
\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$
\begin{array}{llrcl}
\min & & & L & \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \le & L \\
 & \forall \text{jobs } j & \sum_i x_{j,i} \ge 1 & & \\
 & \forall i, j & x_{j,i} & \in & \{0, 1\}
\end{array}
$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$\begin{array}{llrcl}
\min & & & L & \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\
& \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\
& \forall i, j & x_{j,i} & \in & \{0, 1\}
\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$\begin{array}{llrcl}
\min & & & L & \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\
 & \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\
 & \forall i, j & x_{j,i} & \in & \{0, 1\}
\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$\begin{array}{llrcl}
\min & & & L & \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\
& \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\
& \forall i, j & x_{j,i} & \in & \{0, 1\}
\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$.

Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

Hence, the length of the schedule is at most

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

Hence, the length of the schedule is at most

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

Hence, the length of the schedule is at most

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \neq \ell} p_j \ .$$

Hence, the length of the schedule is at most

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \le (2 - \frac{1}{m})C^*_{\max}$$

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \ne \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \le (2 - \frac{1}{m})C^*_{\max}$$

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \leq (2 - \frac{1}{m})C^*_{\max}$$

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

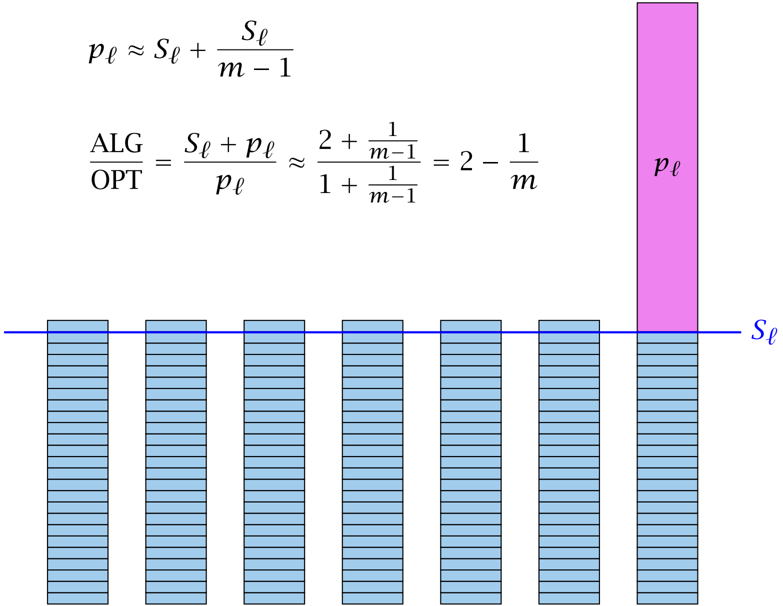
Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j \ .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \leq (2 - \frac{1}{m})C^*_{\max}$$

A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

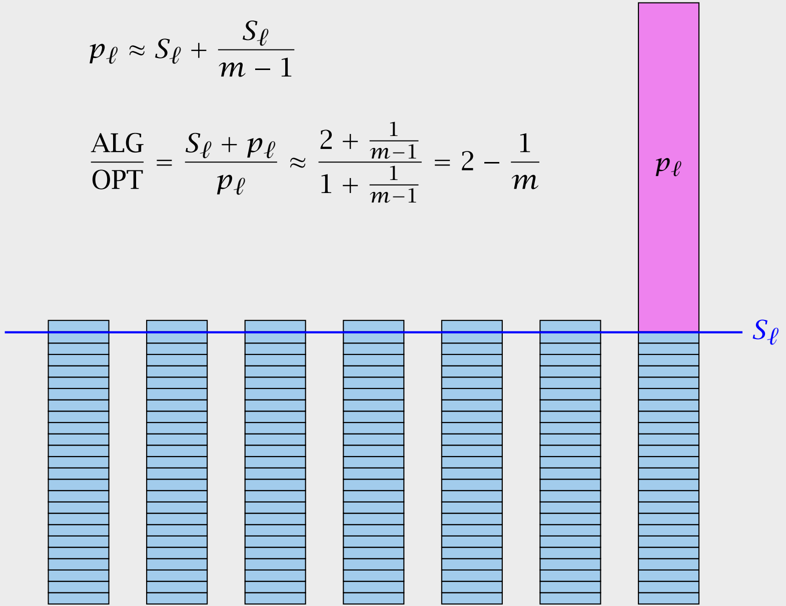
Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

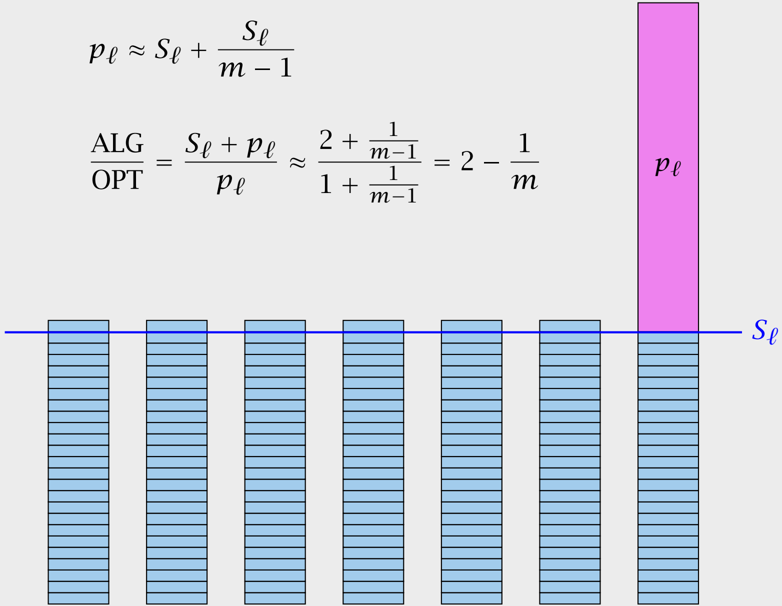
Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\text{ALG}}{\text{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

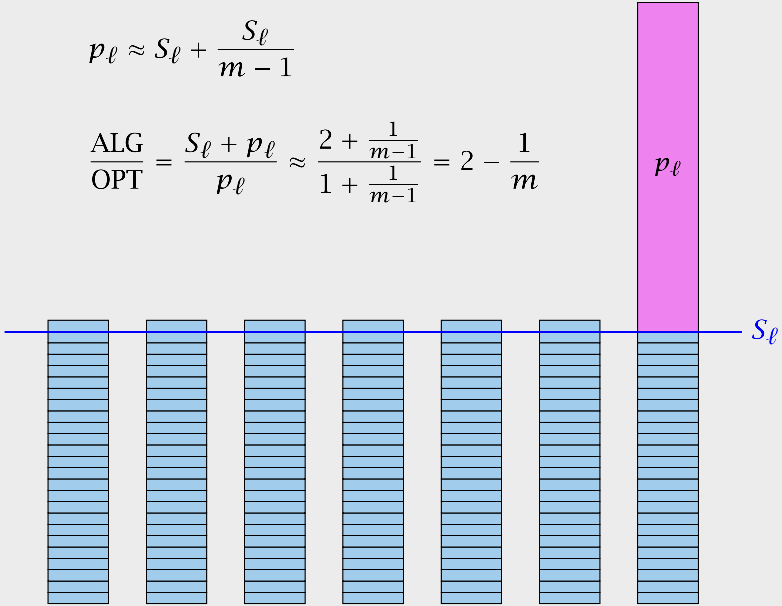
Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

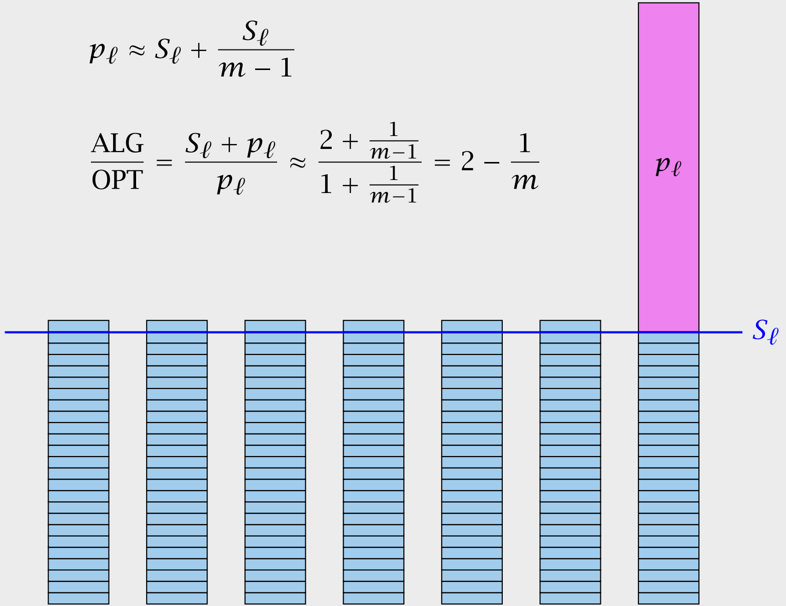
Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



A Greedy Strategy

Lemma 2
If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

A Greedy Strategy

Lemma 2
If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} .$$

A Greedy Strategy

Lemma 2
If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3}C^*_{\max} \enspace .$$

Hence, $p_n > C^*_{\max}/3$.

A Greedy Strategy

Lemma 2
If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3}C^*_{\max} \enspace .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

A Greedy Strategy

Lemma 2
If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3}C^*_{\max} \enspace .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

A Greedy Strategy

Lemma 2

If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \ .$$

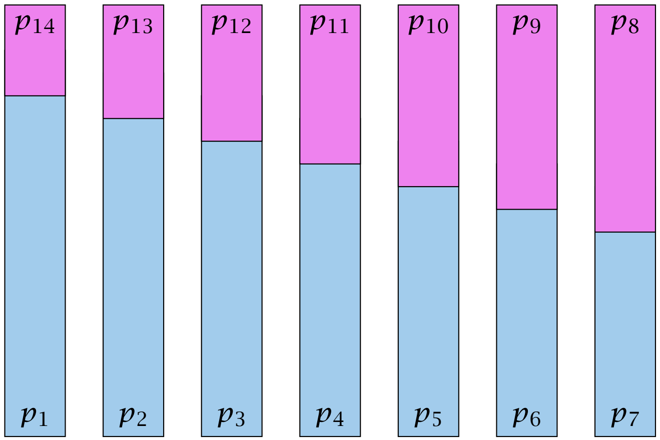
  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

A Greedy Strategy

Lemma 2

If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



Proof:

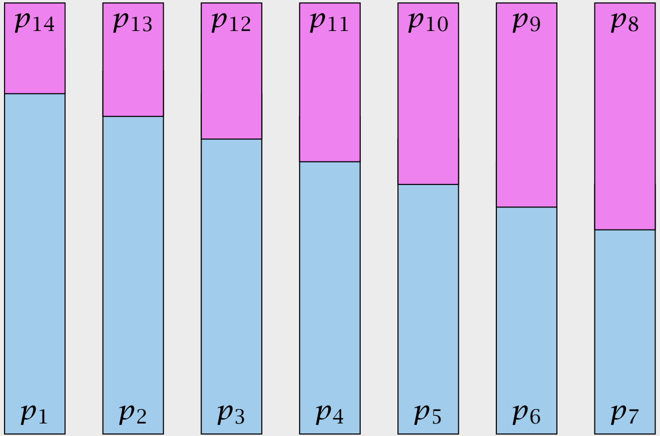
- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

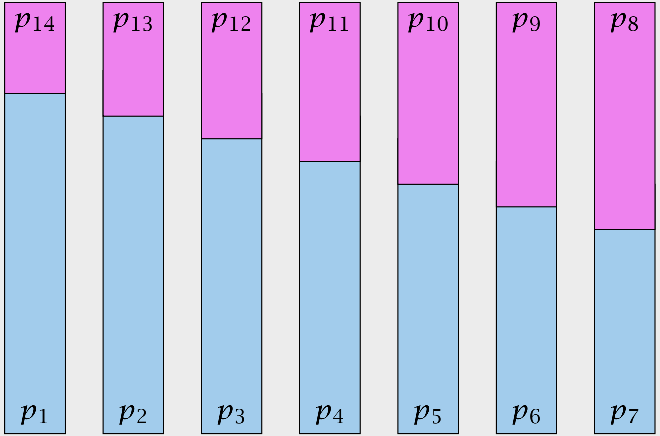
- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



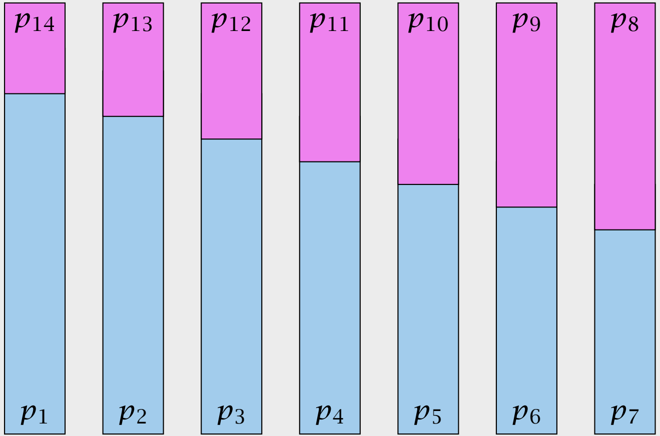
- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



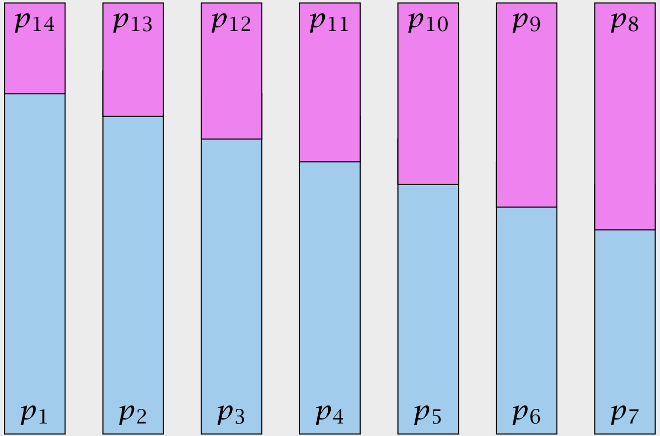
- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



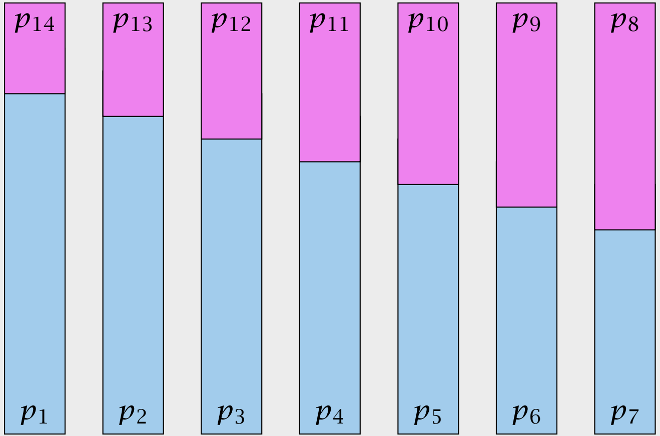
- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



Tight Example

- $2m + 1$ jobs

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m + 1$ jobs
- 2 jobs with length $2m, 2m - 1, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- 3 jobs of length m

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

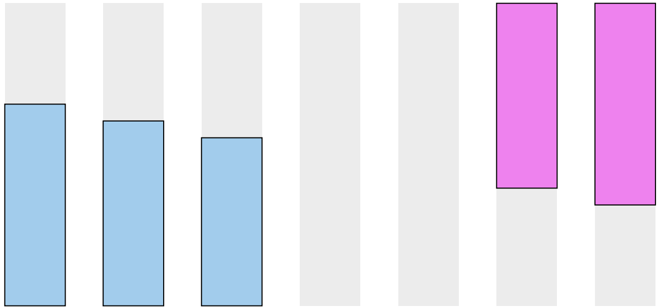
Tight Example

- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m

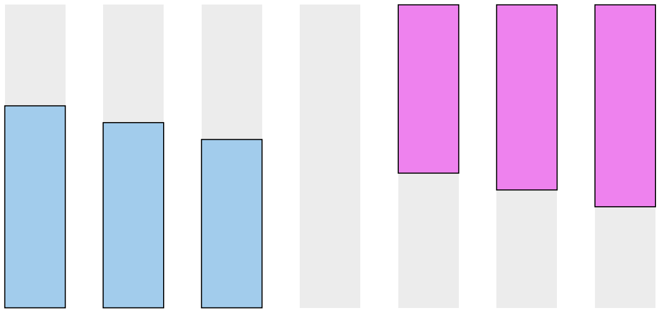


- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

EADS II
Harald Räcke
14.2 Greedy

Tight Example

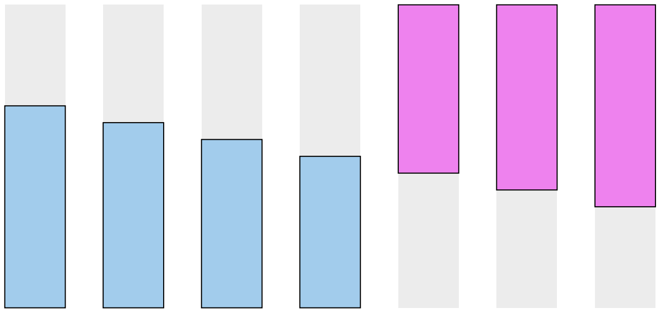
- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

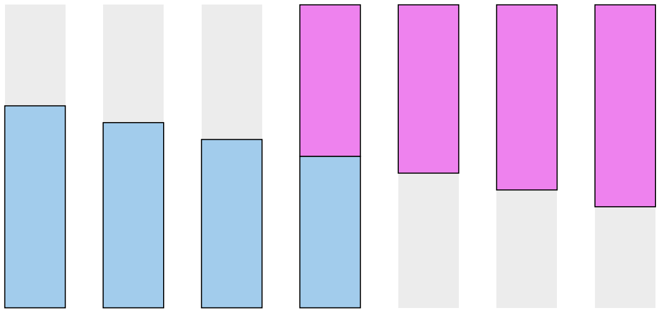
- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

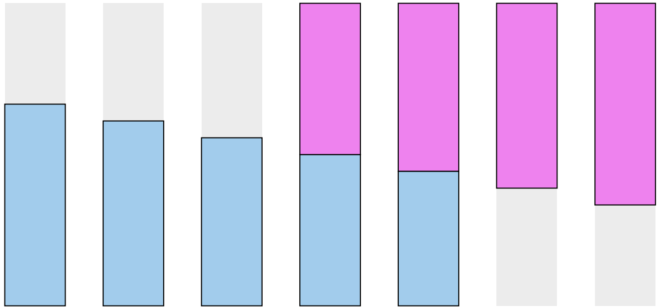
- $2m + 1$ jobs
- 2 jobs with length $2m, 2m - 1, 2m - 2, \ldots, m + 1$ ($2m - 2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

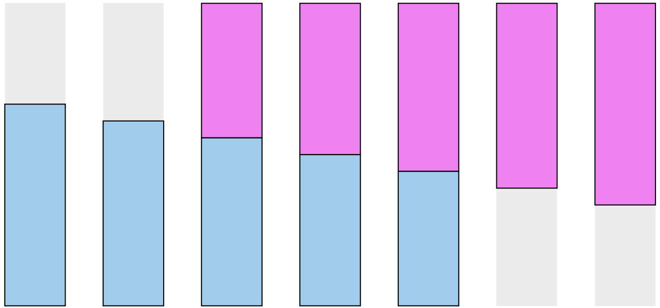
- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

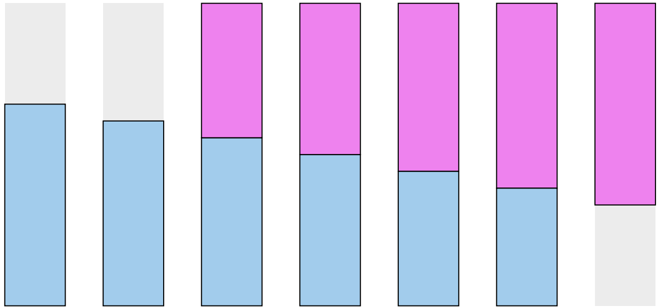
- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

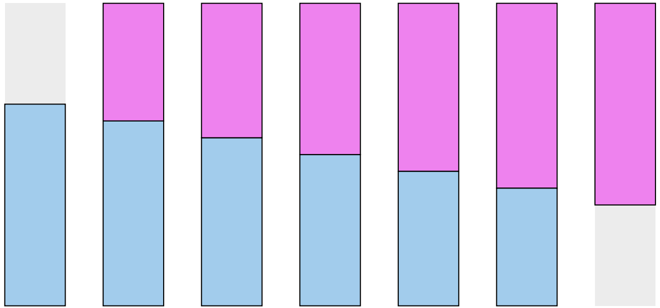
- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

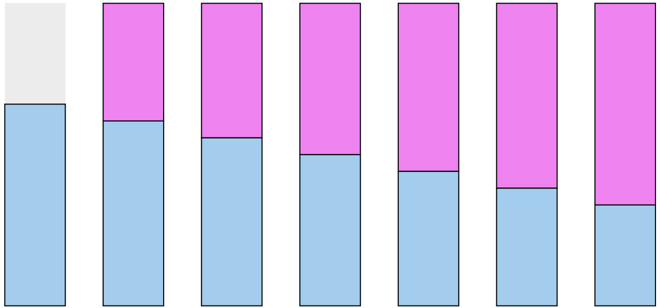
- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m

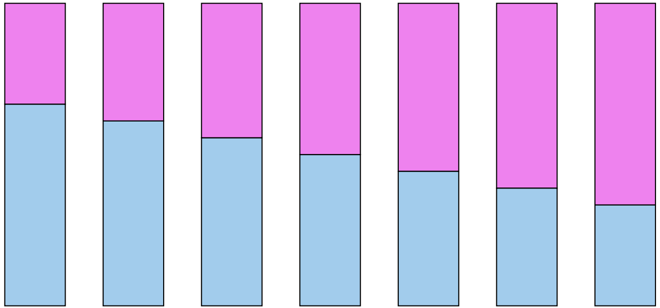


- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

EADS II
Harald Räcke
14.2 Greedy

Tight Example

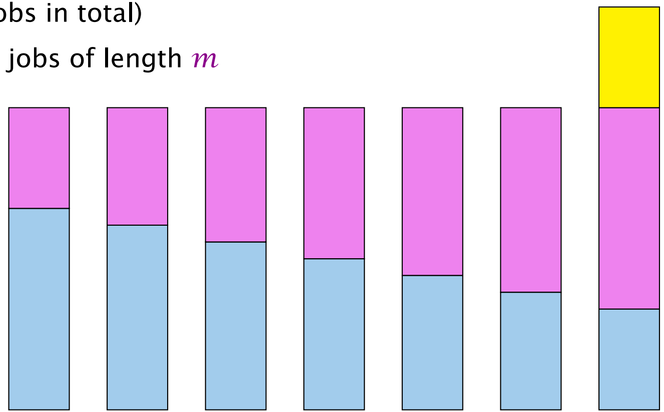
- $2m+1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

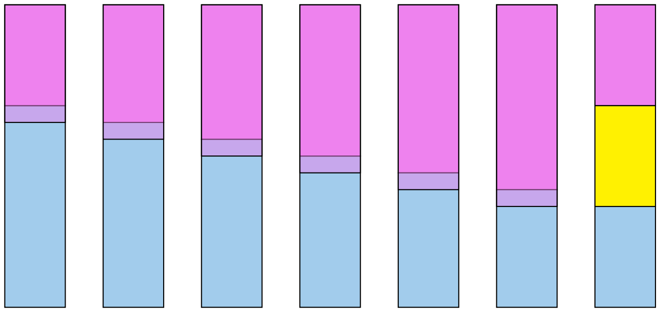
- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \ldots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

Tight Example

- $2m + 1$ jobs
- 2 jobs with length $2m, 2m-1, 2m-2, \dots, m+1$ ($2m-2$ jobs in total)
- 3 jobs of length m



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.