Approximation Bounds for SLACK on Identical Parallel Machines

Louis-Claude Canon[1], *Anthony Dugois*[1], *Pierre-Cyrille Héam*[1], *and Ismaël Jecker*[1]

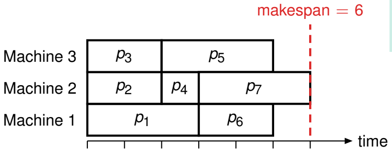
[1]Marie and Louis Pasteur University, CNRS, FEMTO-ST institute (France)

Table of Contents

Table of Contents

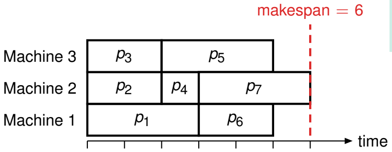
The $P||C_{\max}$ Problem

- **Input**: m identical machines and
  n tasks with processing times $p_1, p_2, \ldots, p_n$
- **Output**: an assignment for each task
- **Objective**: minimize the makespan $C_{\max}$



The $P||C_{\max}$ Problem

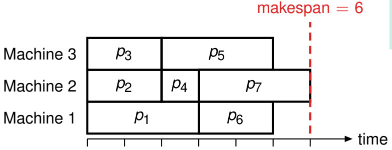
- **Input**: m identical machines and
  n tasks with processing times $p_1, p_2, \ldots, p_n$
- **Output**: an assignment for each task
- **Objective**: minimize the makespan $C_{\max}$



NP-hard

The $P||C_{\max}$ Problem

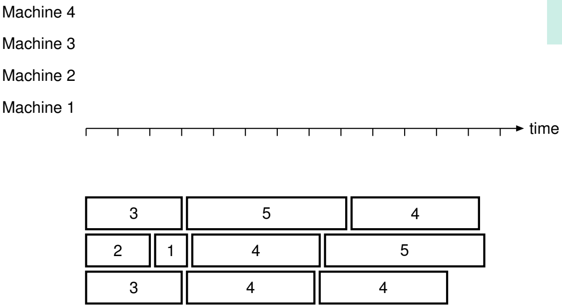
- **Input**: m identical machines and
  n tasks with processing times $p_1, p_2, \ldots, p_n$
- **Output**: an assignment for each task
- **Objective**: minimize the makespan $C_{\max}$



NP-hard $\implies$ **Approximation results**

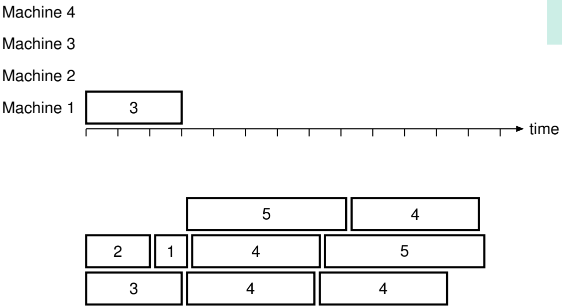
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



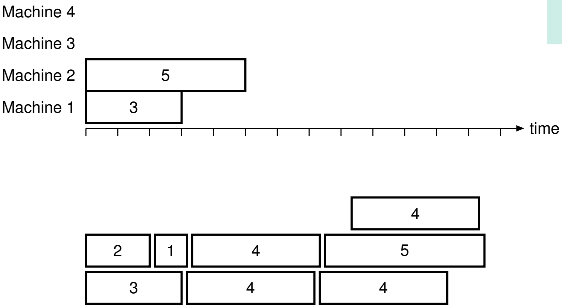
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



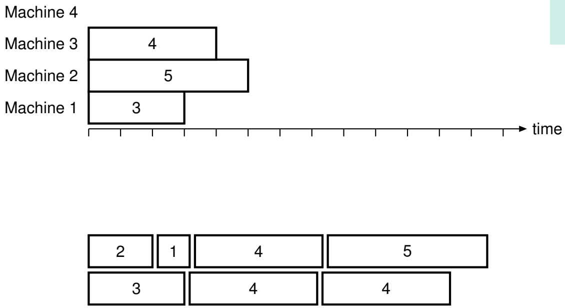
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



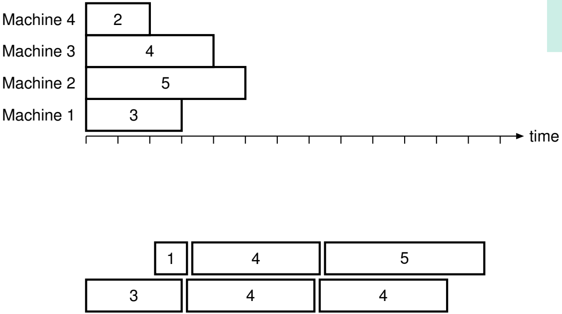
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



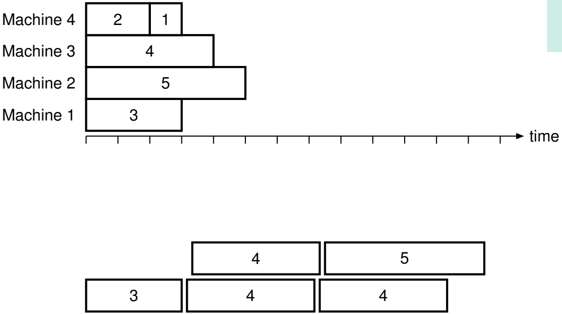
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



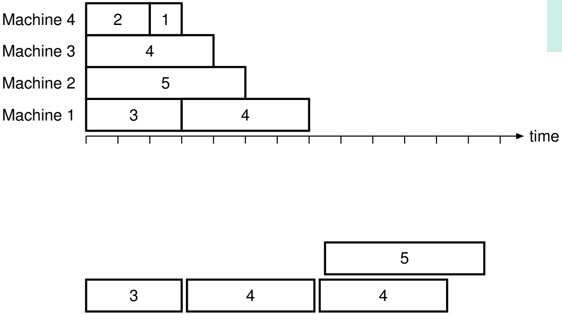
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



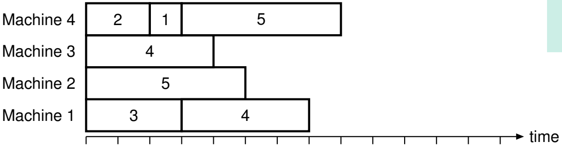
List Scheduling (Graham, 1969)

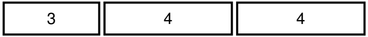
- Strategy: put each task on the least-loaded machine



List Scheduling (Graham, 1969)

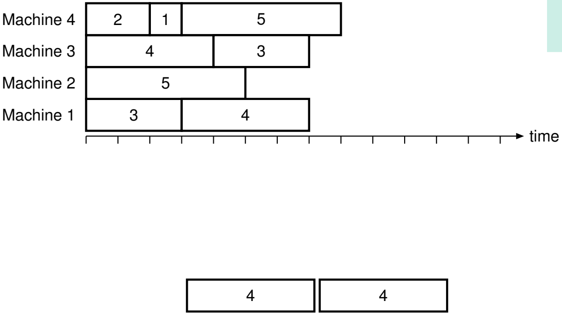
- Strategy: put each task on the least-loaded machine





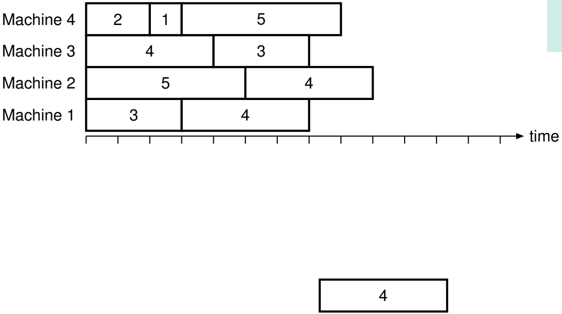
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



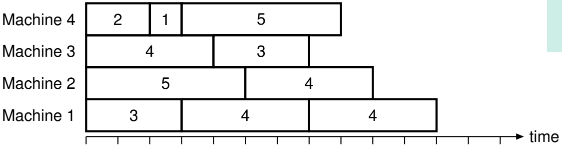
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



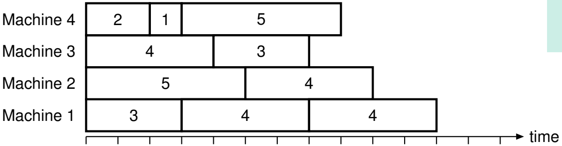
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine



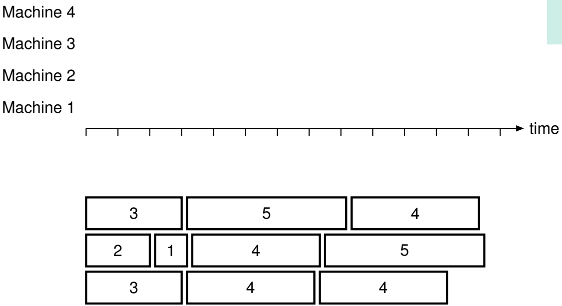
List Scheduling (Graham, 1969)

- Strategy: put each task on the least-loaded machine
- Time: $O(n \log m)$
- Approx. ratio: $2 - 1/m$



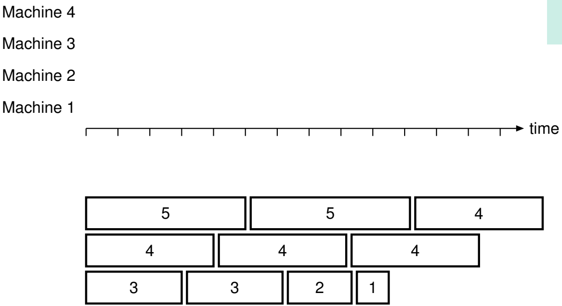
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



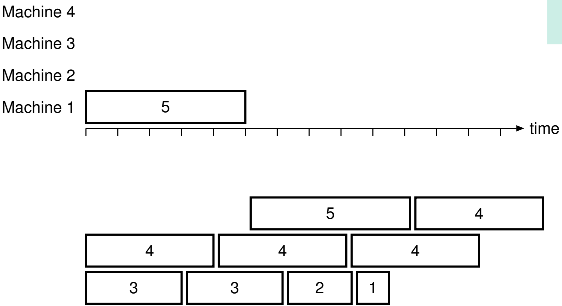
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



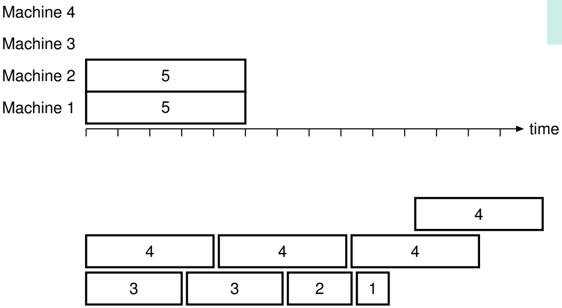
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



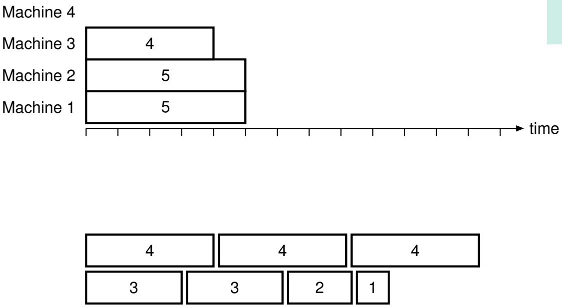
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



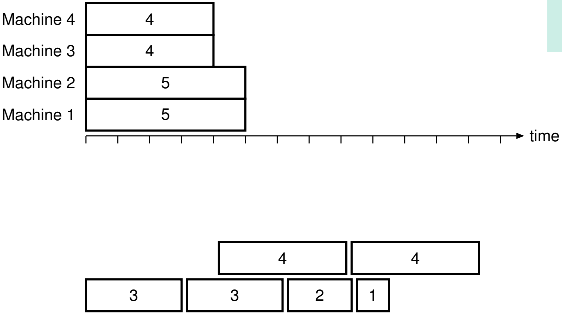
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



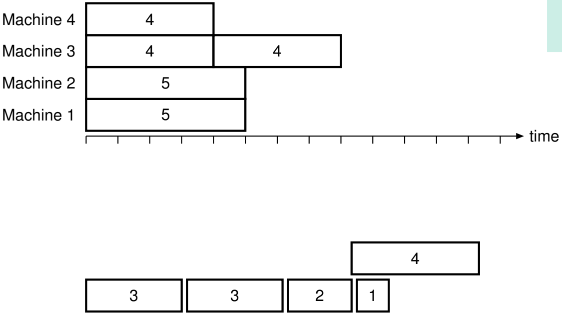
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



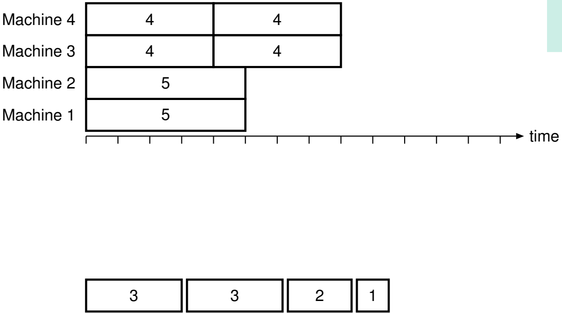
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



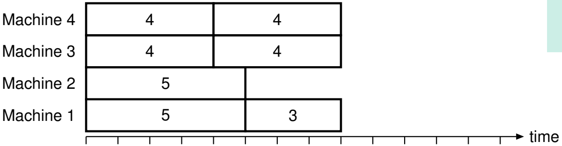
Longest Processing Time (Graham, 1969)

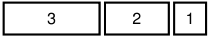
- Strategy: schedule tasks in non-increasing order of processing time



Longest Processing Time (Graham, 1969)

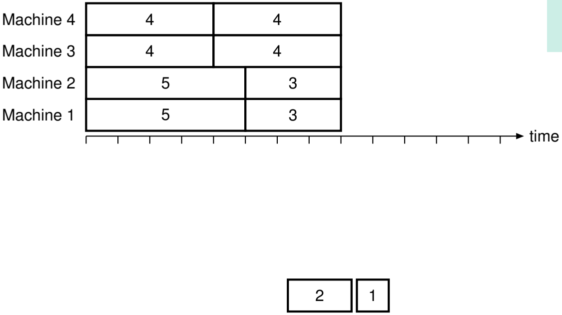
- Strategy: schedule tasks in non-increasing order of processing time





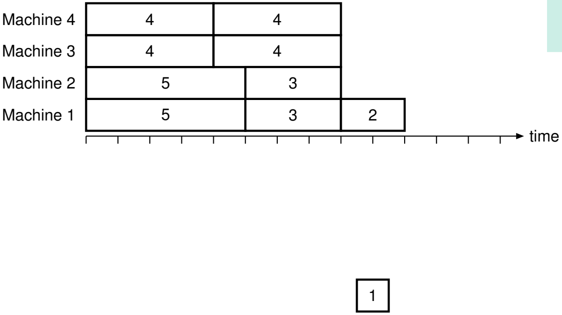
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



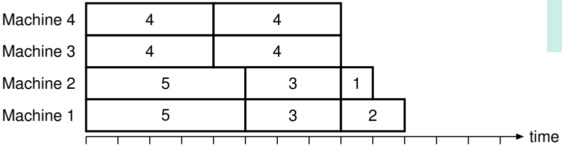
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



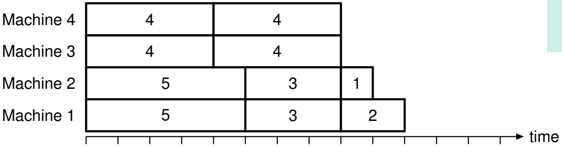
Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time



Longest Processing Time (Graham, 1969)

- Strategy: schedule tasks in non-increasing order of processing time
- Time: $O(n \log n)$
- Approx. ratio: $4/3 - 1/3m$



… and so on

Strategy	Time	Approx. ratio	Ref.
LS	$O(n \log m)$	$2 - 1/m$	Graham, 1969
LPT	$O(n \log n)$	$4/3 - 1/3m$	Graham, 1969
Multi-Fit	$O(n \log n + knm)$	$13/11 + 1/2^k$	Coffman, 1978; Yue, 1990
Combine	same as MF	same as MF	Lee and Massey, 1988
List-Fit	same as MF	same as MF	Gupta and Ruiz-Torres, 2001
PTAS	$n^{O((1/\varepsilon)^2 \log(1/\varepsilon))}$	$1 + \varepsilon$	Hochbaum and Shmoys, 1987
EPTAS	$2^{O(1/\varepsilon \log^4(1/\varepsilon))} + O(n \log n)$	$1 + \varepsilon$	Jansen et al., 2020

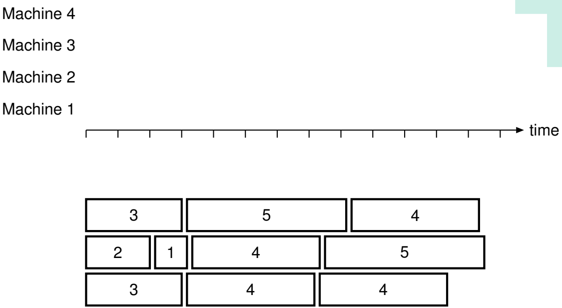
. . . and so on

Strategy	Time	Approx. ratio	Ref.
LS	$O(n \log m)$	$2 - 1/m$	Graham, 1969
LPT	$\boldsymbol{O(n \log n)}$	$\boldsymbol{4/3 - 1/3m}$	**Graham, 1969**
Multi-Fit	$O(n \log n + knm)$	$13/11 + 1/2^k$	Coffman, 1978; Yue, 1990
Combine	same as MF	same as MF	Lee and Massey, 1988
List-Fit	same as MF	same as MF	Gupta and Ruiz-Torres, 2001
PTAS	$n^{O((1/\varepsilon)^2 \log(1/\varepsilon))}$	$1 + \varepsilon$	Hochbaum and Shmoys, 1987
EPTAS	$2^{O(1/\varepsilon \log^4(1/\varepsilon))} + O(n \log n)$	$1 + \varepsilon$	Jansen et al., 2020

Table of Contents

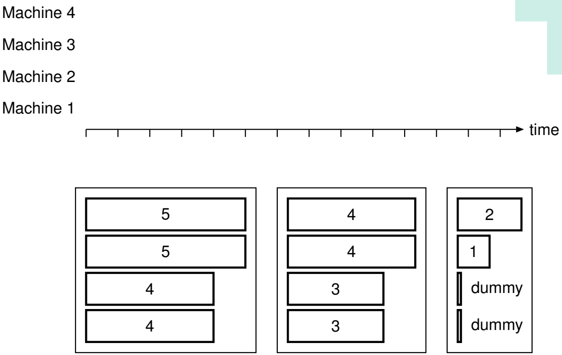
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



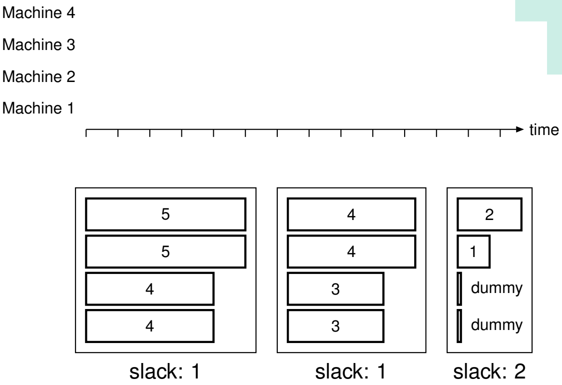
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



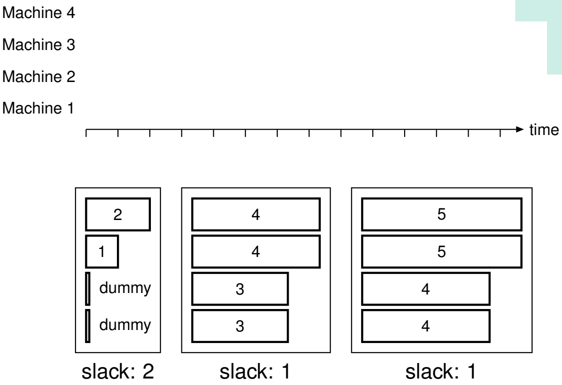
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



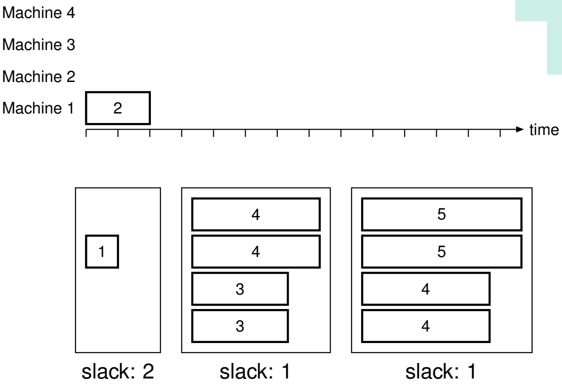
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



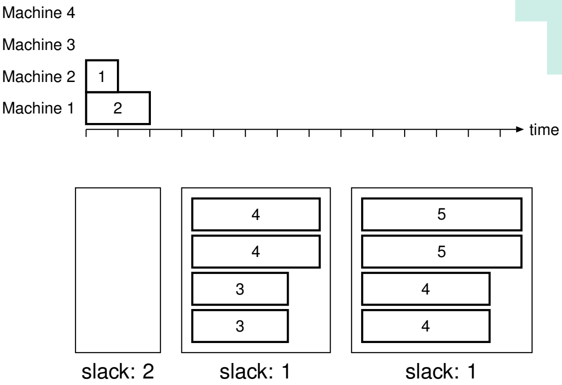
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



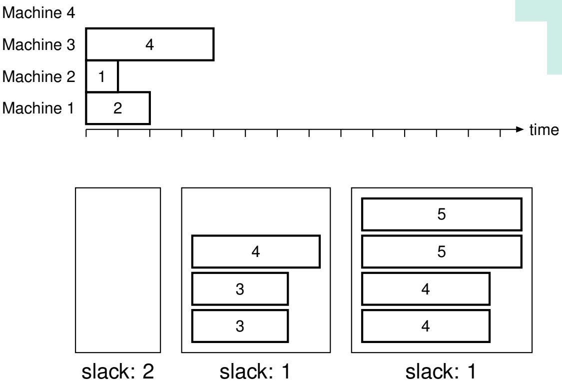
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size m
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



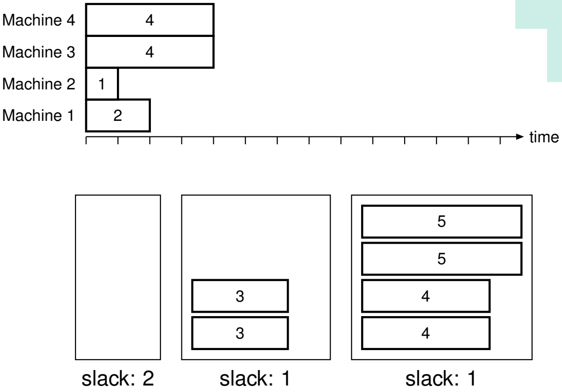
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size m
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



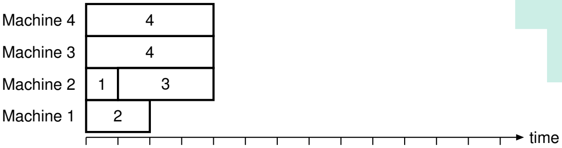
SLACK (Della Croce and Scatamacchia, 2020)

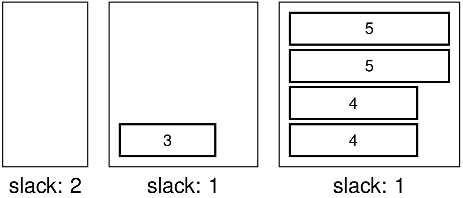
- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



SLACK (Della Croce and Scatamacchia, 2020)

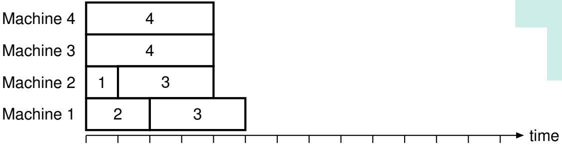
- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily

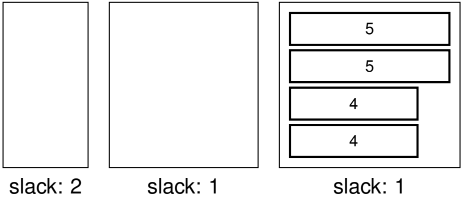




SLACK (Della Croce and Scatamacchia, 2020)

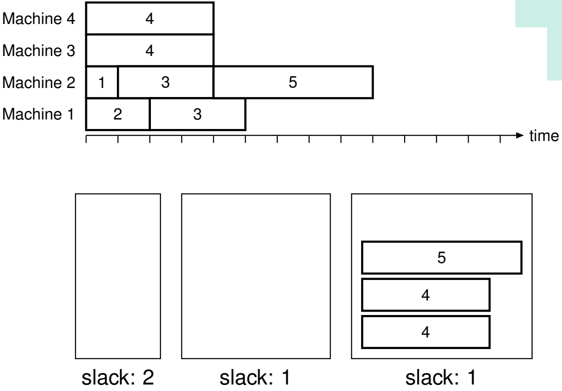
- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily





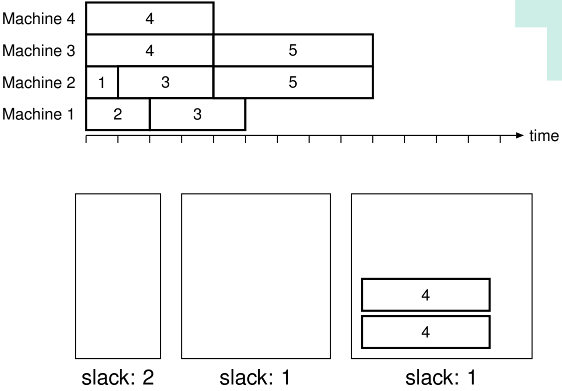
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



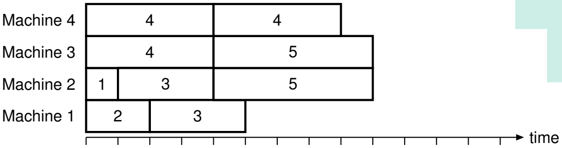
SLACK (Della Croce and Scatamacchia, 2020)

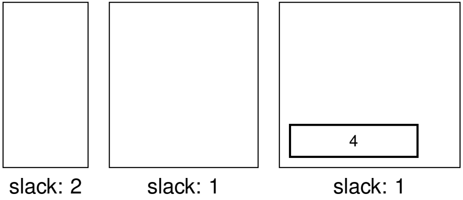
- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



SLACK (Della Croce and Scatamacchia, 2020)

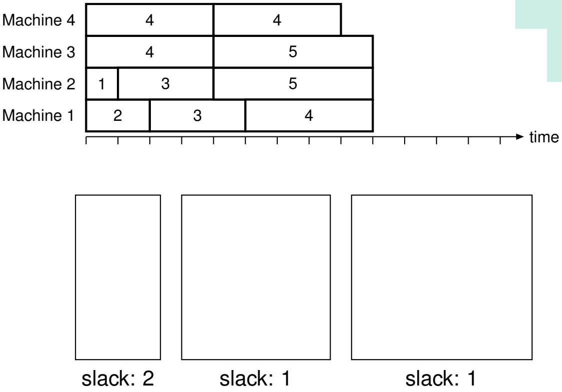
- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size m
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily





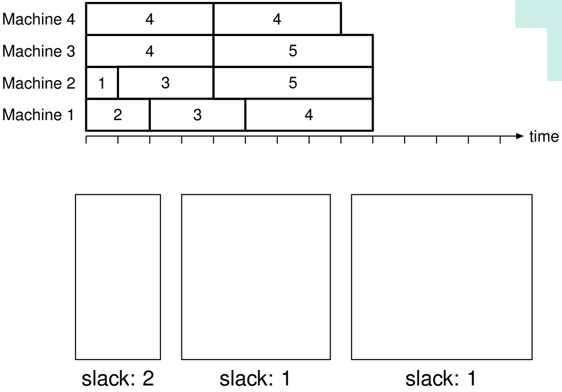
SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size *m*
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily



SLACK (Della Croce and Scatamacchia, 2020)

- Strategy:
  1. Sort tasks in non-increasing order of processing time and group them in packs of size m
  2. Compute slack of each pack (largest task minus smallest task)
  3. Sort packs in non-increasing order of slack
  4. Schedule tasks greedily
- Time: $O(n \log n)$
- **Approx. ratio: ???**



SLACK vs. LPT: makespan

Range of p_j	m	Nb. of instances	SLACK wins	draws	LPT wins
1-100	5	50	24%	74%	2%
	10	40	35%	50%	15%
	25	40	25%	72.5%	2.5%
1-1000	5	50	64%	30%	6%
	10	40	67.5%	12.5%	20%
	25	40	60%	30%	10%
1-10000	5	50	72%	24%	4%
	10	40	92.5%	0%	7.5%
	25	40	55%	27.5%	17.5%

Performance comparison on uniform instances ($n = 10, 50, 100, 500, 1000$).

Extracted from *Longest Processing Time rule for identical parallel machines revisited (Della Croce and Scatamacchia, 2020).*

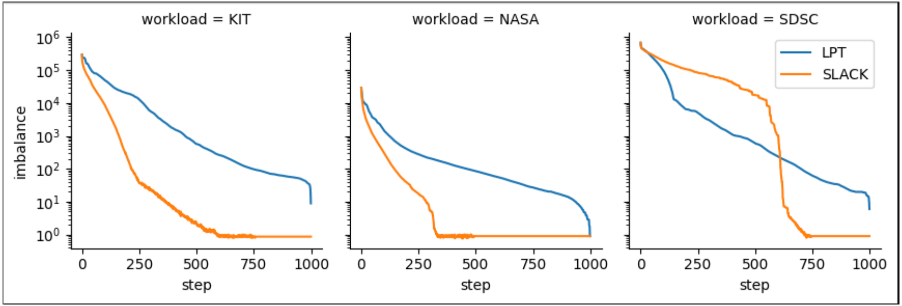
SLACK vs. LPT: makespan

Range of p_j	m	Nb. of instances	SLACK wins	draws	LPT wins
1-100	5	50	62%	32%	6%
	10	40	80%	20%	0%
	25	40	57.5%	42.5%	0%
1-1000	5	50	78%	20%	2%
	10	40	100%	0%	0%
	25	40	67.5%	30%	2.5%
1-10000	5	50	78%	20%	2%
	10	40	100%	0%	0%
	25	40	70%	25%	5%

Performance comparison on non-uniform instances ($n = 10, 50, 100, 500, 1000$).

Extracted from *Longest Processing Time rule for identical parallel machines revisited (Della Croce and Scatamacchia, 2020).*

SLACK vs. LPT: load imbalance



Evolution of load imbalance on real traces (lower is better).

Table of Contents

Approximation Ratio of SLACK

Theorem (Main result)

The approximation ratio of SLACK is $4/3$.

Approximation Ratio of SLACK

Theorem (Main result)

The approximation ratio of SLACK is $4/3$.

Lemma (Upper bound)

For any instance $\mathcal{I}$, *we have* $\frac{\mathit{SLACK}(\mathcal{I})}{\mathit{OPT}(\mathcal{I})} \leq 4/3$.

Approximation Ratio of SLACK

Theorem (Main result)

The approximation ratio of SLACK is $4/3$.

Lemma (Upper bound)

For any instance $\mathcal{I}$, *we have* $\frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} \leq 4/3$.

Lemma (Tightness)

There is a family of instances $\mathcal{F}$ *such that* $\sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$.

Proof Sketch of Upper Bound ($\forall \mathcal{I}, \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} \leq 4/3$)

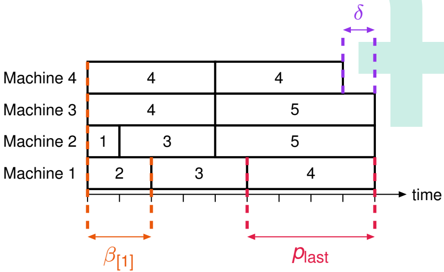
Lemma (Graham's bound)

For any instance $\mathcal{I}$, we have $C_{\max} \leq OPT(\mathcal{I}) + \delta$ in any list schedule.

Lemma

The imbalance δ of SLACK is lower than:

- *the slack $\beta_{[1]}$ of the first scheduled pack, and*
- *the processing time p_{last} of the last task.*



Proof Sketch of Upper Bound ($\forall \mathcal{I}, \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} \leq 4/3$)

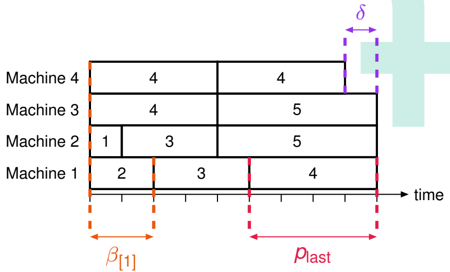
Lemma (Graham's bound)

For any instance $\mathcal{I}$, we have $C_{\max} \leq OPT(\mathcal{I}) + \delta$ in any list schedule.

Lemma

The imbalance δ of SLACK is lower than:

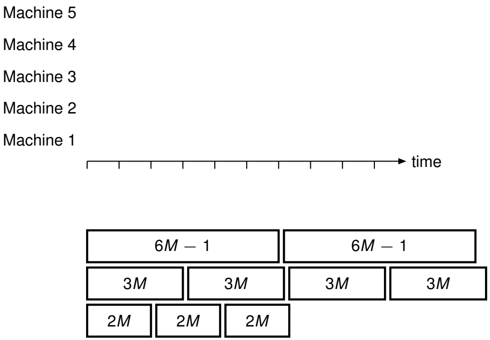
- *the slack $\beta_{[1]}$ of the first scheduled pack, and*
- *the processing time p_{last} of the last task.*



$\Longrightarrow$ Thus, if either $\beta_{[1]} \leq \frac{\text{OPT}(\mathcal{I})}{3}$ or $p_{last} \leq \frac{\text{OPT}(\mathcal{I})}{3}$, the result immediately follows.

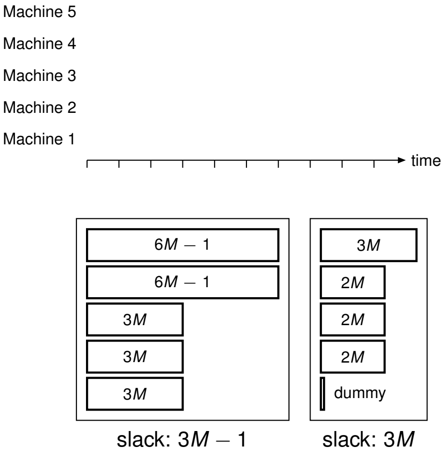
Rest of the proof deals with instances where $\beta_{[1]} > \frac{\text{OPT}(\mathcal{I})}{3}$ and $p_{last} > \frac{\text{OPT}(\mathcal{I})}{3}$.

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



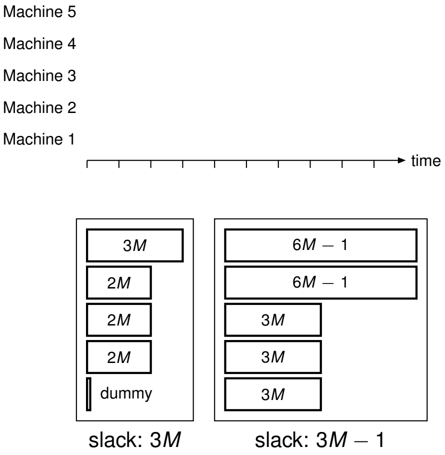
- M is an arbitrary constant

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



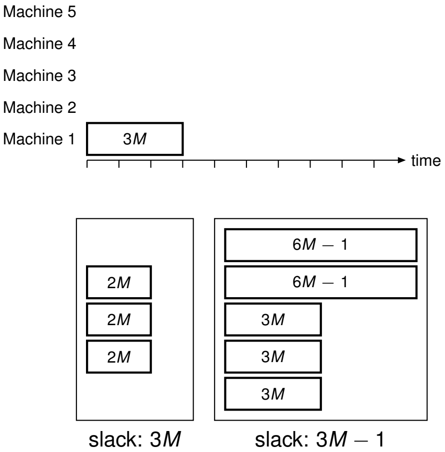
- M is an arbitrary constant

Proof Sketch of Tightness ($\exists\mathcal{F}, \sup_{\mathcal{I}\in\mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



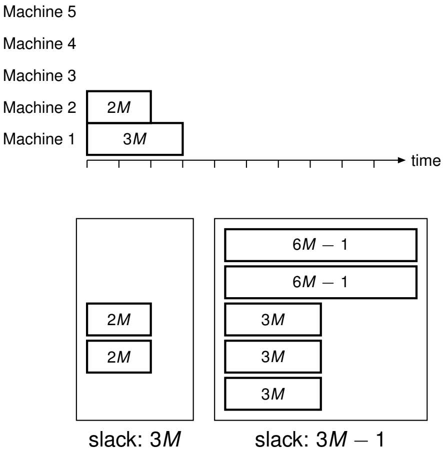
- M is an arbitrary constant

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



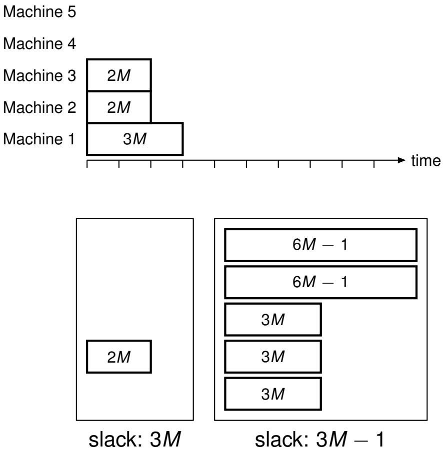
- M is an arbitrary constant

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



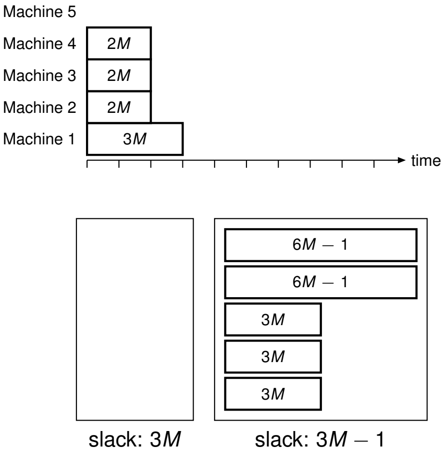
- M is an arbitrary constant

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



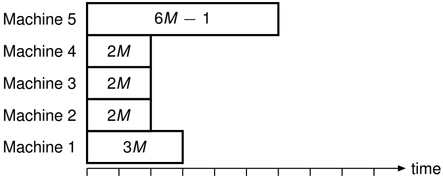
- M is an arbitrary constant

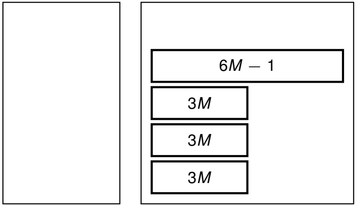
Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I}\in\mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



- M is an arbitrary constant

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)

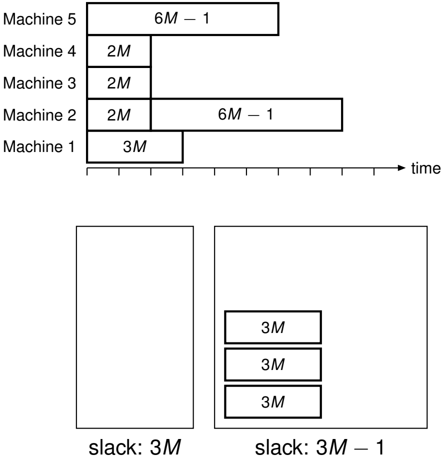




slack: $3M$ slack: $3M - 1$

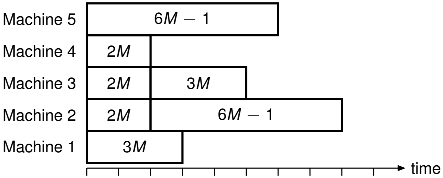
- M is an arbitrary constant

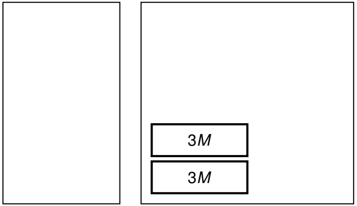
Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



- M is an arbitrary constant

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)

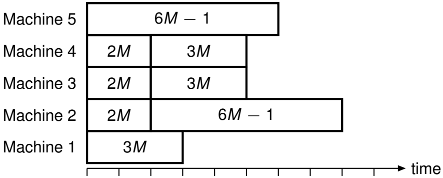


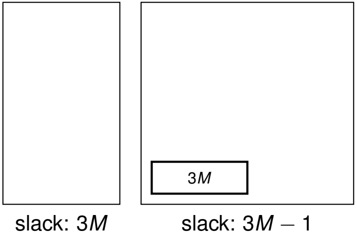


slack: $3M$ slack: $3M-1$

- M is an arbitrary constant

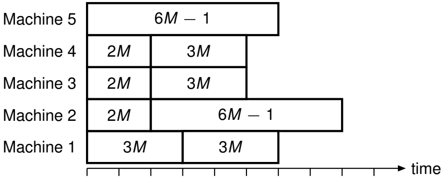
Proof Sketch of Tightness $\left(\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3\right)$

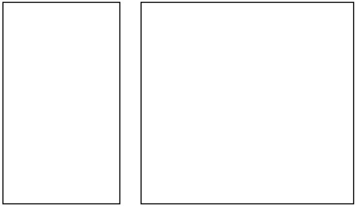




- M is an arbitrary constant

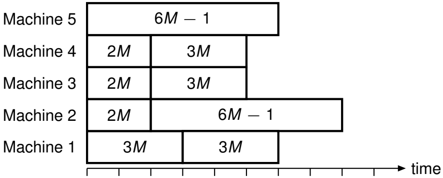
Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)

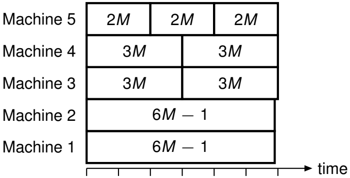


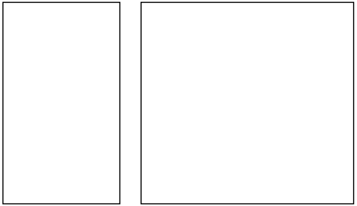


- M is an arbitrary constant
- $\text{SLACK}(\mathcal{I}) = 8M - 1$

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)



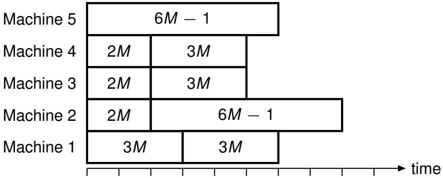


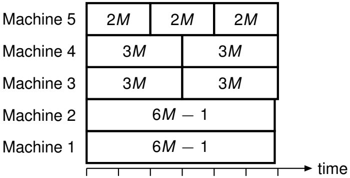


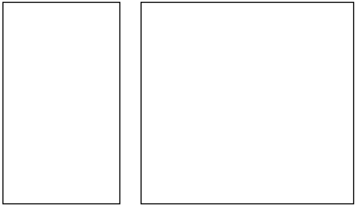
slack: $3M$ slack: $3M - 1$

- M is an arbitrary constant
- $\text{SLACK}(\mathcal{I}) = 8M - 1$
- $\text{OPT}(\mathcal{I}) = 6M$

Proof Sketch of Tightness ($\exists \mathcal{F}, \sup_{\mathcal{I} \in \mathcal{F}} \frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} = 4/3$)







slack: $3M$ slack: $3M - 1$

- M is an arbitrary constant
- $\text{SLACK}(\mathcal{I}) = 8M - 1$
- $\text{OPT}(\mathcal{I}) = 6M$
- $\frac{\text{SLACK}(\mathcal{I})}{\text{OPT}(\mathcal{I})} \to 4/3$ as $M \to \infty$

Refined Approximation Ratio of SLACK

Consider the set $\mathcal{S}_k$ of instances $\mathcal{I}$ where all tasks are “small”, i.e., $p_j \leq \frac{\mathrm{OPT}(\mathcal{I})}{k}$ for a fixed integer $k \geq 2$.

Refined Approximation Ratio of SLACK

Consider the set $\mathcal{S}_k$ of instances $\mathcal{I}$ where all tasks are "small", i.e., $p_j \leq \frac{\mathrm{OPT}(\mathcal{I})}{k}$ for a fixed integer $k \geq 2$.

Theorem (Upper bound)

For any instance $\mathcal{I} \in \mathcal{S}_k$, we have $\frac{SLACK(\mathcal{I})}{OPT(\mathcal{I})} \leq 1 + \frac{m-1}{m(k+1)}$.

Refined Approximation Ratio of SLACK

Consider the set $\mathcal{S}_k$ of instances $\mathcal{I}$ where all tasks are "small", i.e., $p_j \leq \frac{\mathrm{OPT}(\mathcal{I})}{k}$ for a fixed integer $k \geq 2$.

Theorem (Upper bound)

For any instance $\mathcal{I} \in \mathcal{S}_k$, we have $\frac{SLACK(\mathcal{I})}{OPT(\mathcal{I})} \leq 1 + \frac{m-1}{m(k+1)}$.

Theorem (Lower bound)

If $m \geq k$, there is an instance $\mathcal{I} \in \mathcal{S}_k$ such that $\frac{SLACK(\mathcal{I})}{OPT(\mathcal{I})} \geq 1 + \frac{k-1}{k(k+1)}$.

Refined Approximation Ratio of SLACK

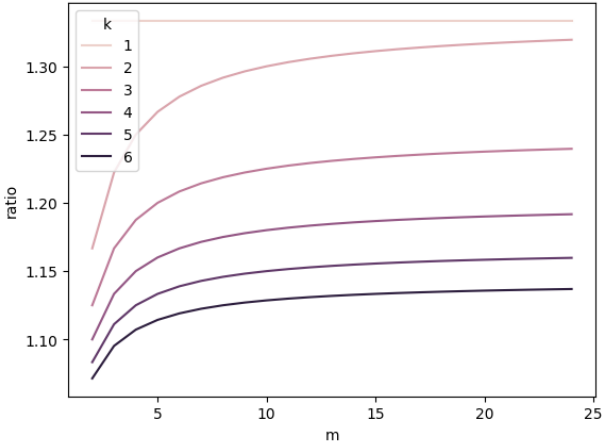


Table of Contents

Conclusion

Takeaways

- The SLACK heuristic was known to often outperform existing approximation algorithms such as LPT and COMBINE.
- We proved that SLACK is a $4/3$-approximation algorithm.
- In a worst-case analysis, SLACK is comparable to LPT even for “small” tasks.

Conclusion

Takeaways

- The SLACK heuristic was known to often outperform existing approximation algorithms such as LPT and COMBINE.
- We proved that SLACK is a $4/3$-approximation algorithm.
- In a worst-case analysis, SLACK is comparable to LPT even for “small” tasks.

Perspectives

- Reduce the gap between the upper and lower bounds of SLACK in the restricted case with “small” tasks only.
- Apply a different analysis framework to understand why SLACK often outperforms other algorithms.