Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

Sven Jäger Philipp Warode

Symposium on Simplicity of Algorithms | 08 January 2024 | Alexandria, Virginia

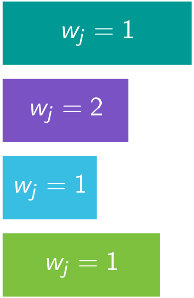
Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

Sven Jäger Philipp Warode

Symposium on Simplicity of Algorithms | 08 January 2024 | Alexandria, Virginia

Minimizing the Total Weighted Completion Time

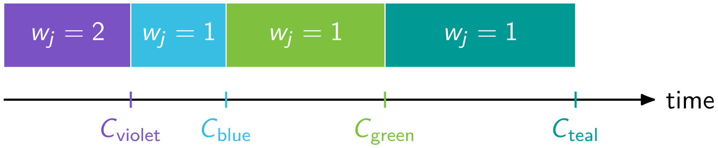
Given: set N of n jobs with processing times $p_j > 0$ and weights $w_j > 0$;



Minimizing the Total Weighted Completion Time

Given: set N of n jobs with processing times $p_j > 0$ and weights $w_j > 0$;

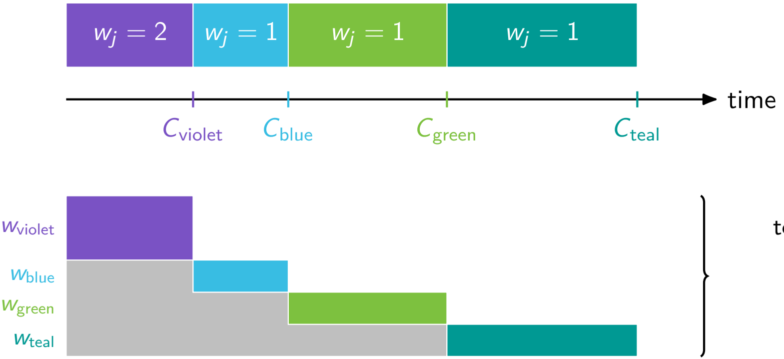
Task: schedule the jobs on a single machine so that the sum of weighted completion times $\sum_{j \in J} w_j C_j$ is minimized.



Minimizing the Total Weighted Completion Time

Given: set N of n jobs with processing times $p_j > 0$ and weights $w_j > 0$;

Task: schedule the jobs on a single machine so that the sum of weighted completion times $\sum_{j \in J} w_j C_j$ is minimized.

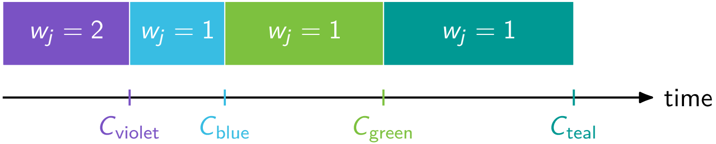


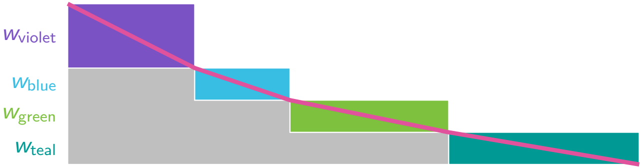
total area $= \sum_{j \in N} w_j C_j$

Minimizing the Total Weighted Completion Time

Given: set N of n jobs with processing times $p_j > 0$ and weights $w_j > 0$;

Task: schedule the jobs on a single machine so that the sum of weighted completion times $\sum_{j \in J} w_j C_j$ is minimized.





$$\text{total area} = \sum_{j \in N} w_j C_j$$

minimized if jobs are ordered by Smith ratio w_j / p_j

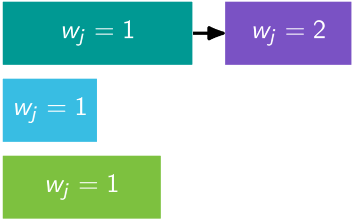
Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

Sven Jäger Philipp Warode

Symposium on Simplicity of Algorithms | 08 January 2024 | Alexandria, Virginia

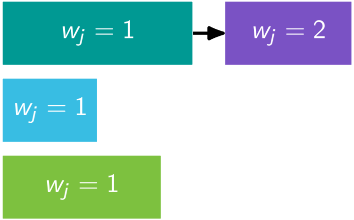
Precedence-Constrained Jobs

- Jobs are nodes of a directed acyclic graph $D = (N, A)$.



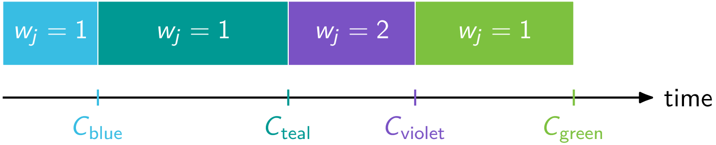
Precedence-Constrained Jobs

- Jobs are nodes of a directed acyclic graph $D = (N, A)$.
- A job can only be processed when all its predecessors have been completed.



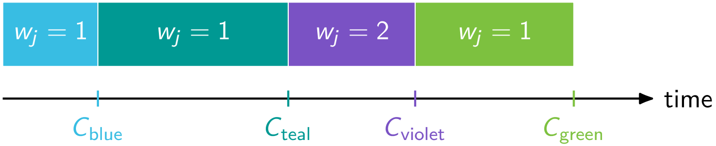
Precedence-Constrained Jobs

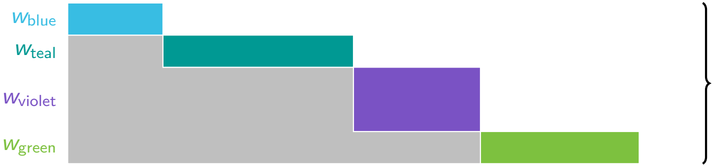
- Jobs are nodes of a directed acyclic graph $D = (N, A)$.
- A job can only be processed when all its predecessors have been completed.



Precedence-Constrained Jobs

- Jobs are nodes of a directed acyclic graph $D = (N, A)$.
- A job can only be processed when all its predecessors have been completed.

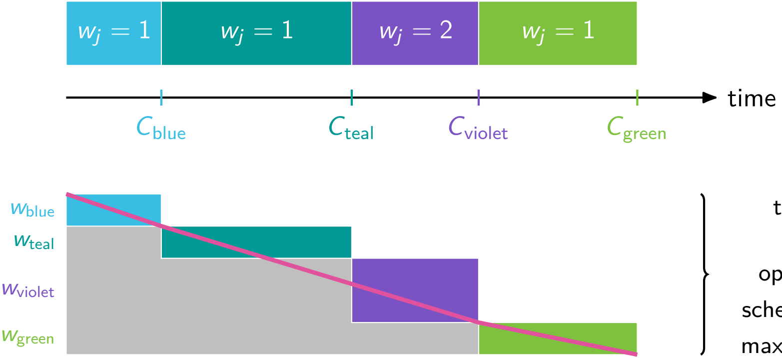




$$\text{total area} = \sum_{j \in N} w_j C_j$$

Precedence-Constrained Jobs

- Jobs are nodes of a directed acyclic graph $D = (N, A)$.
- A job can only be processed when all its predecessors have been completed.

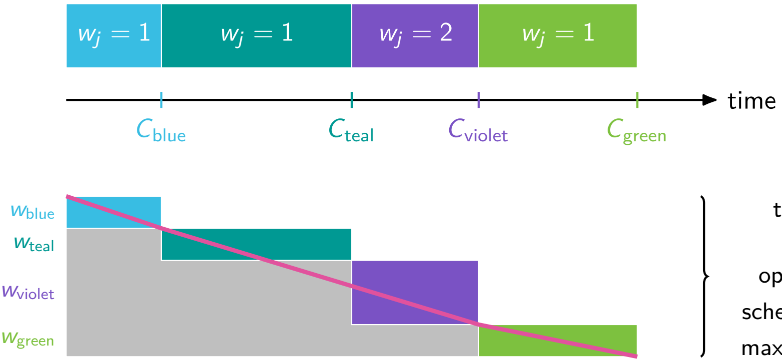


total area $= \sum_{j \in N} w_j C_j$

optimal solution always schedules initial set J with maximum ratio $w(J)/p(J)$

Precedence-Constrained Jobs

- Jobs are nodes of a directed acyclic graph $D = (N, A)$.
- A job can only be processed when all its predecessors have been completed.
- The problem is strongly NP-hard. (Lawler '78)



total area $= \sum_{j \in N} w_j C_j$

optimal solution always schedules initial set J with maximum ratio $w(J)/p(J)$

Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

Sven Jäger Philipp Warode

Symposium on Simplicity of Algorithms | 08 January 2024 | Alexandria, Virginia

Approximation Algorithms

- Pisaruk '92: 2-approximation algorithm for more general submodular ordering problem

Approximation Algorithms

- Pisaruk '92: 2-approximation algorithm for more general submodular ordering problem
- Hall et al. '97: 2-approximation algorithm based on LP relaxation with exponentially many efficiently separable constraints

Approximation Algorithms

- Pisaruk '92: 2-approximation algorithm for more general submodular ordering problem
- Hall et al. '97: 2-approximation algorithm based on LP relaxation with exponentially many efficiently separable constraints
- Chudak, Hochbaum '99: Two 2-approximation algorithms based on LP relaxation with only two variables in each constraint, which can be solved by min-cut computation

Approximation Algorithms

- Pisaruk '92: 2-approximation algorithm for more general submodular ordering problem
- Hall et al. '97: 2-approximation algorithm based on LP relaxation with exponentially many efficiently separable constraints
- Chudak, Hochbaum '99: Two 2-approximation algorithms based on LP relaxation with only two variables in each constraint, which can be solved by min-cut computation
- Chekuri, Motwani '99; Margot et al. '03; Pisaruk '03: 2-approximation algorithm that determines a Sidney decomposition and arbitrarily orders jobs in each block

Approximation Algorithms

- Pisaruk '92: 2-approximation algorithm for more general submodular ordering problem
- Hall et al. '97: 2-approximation algorithm based on LP relaxation with exponentially many efficiently separable constraints
- Chudak, Hochbaum '99: Two 2-approximation algorithms based on LP relaxation with only two variables in each constraint, which can be solved by min-cut computation
- Chekuri, Motwani '99; Margot et al. '03; Pisaruk '03: 2-approximation algorithm that determines a Sidney decomposition and arbitrarily orders jobs in each block

- Bansal, Khot '09: Under a variant of the Unique Games Conjecture, not better guarantee is possible.

Approximation Algorithms via Sidney Decompoistion

Algorithm (2-Approximation).

1. Let $U \leftarrow N$, $\mathsf{S} \leftarrow [\,]$.
2. **While** $U \neq \emptyset$,
3. determine initial set J in $D[U]$ with maximum ratio $w(J)/p(J)$;
4. append jobs from J to schedule S in arbitrary topological order.

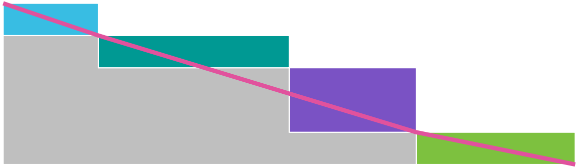
Approximation Algorithms via Sidney Decompoistion

Algorithm (2-Approximation).

1. Let $U \leftarrow N$, $\mathsf{S} \leftarrow []$.
2. **While** $U \neq \emptyset$,
3. determine initial set J in $D[U]$ with maximum ratio $w(J)/p(J)$;
4. append jobs from J to schedule S in arbitrary topological order.

Analysis:

- The algorithm computes a solution whose objective value is at most twice the optimum objective value.



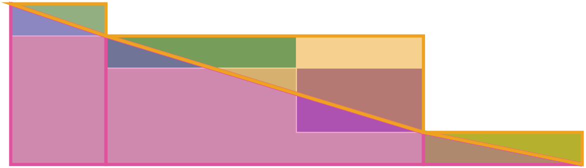
Approximation Algorithms via Sidney Decompoistion

Algorithm (2-Approximation).

1. Let $U \leftarrow N$, $\mathsf{S} \leftarrow [\,]$.
2. **While** $U \neq \emptyset$,
3. determine initial set J in $D[U]$ with maximum ratio $w(J)/p(J)$;
4. append jobs from J to schedule S in arbitrary topological order.

Analysis:

- The algorithm computes a solution whose objective value is at most twice the optimum objective value.



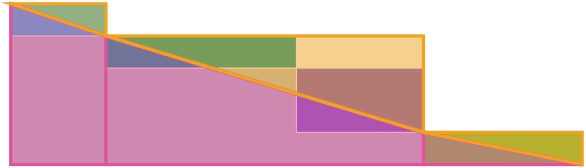
Approximation Algorithms via Sidney Decompoistion

Algorithm (2-Approximation).

1. Let $U \leftarrow N$, $S \leftarrow []$.
2. **While** $U \neq \emptyset$,
3. determine initial set J in $D[U]$ with maximum ratio $w(J)/p(J)$;
4. append jobs from J to schedule S in arbitrary topological order.

Analysis:

- The algorithm computes a solution whose objective value is at most twice the optimum objective value.
- The algorithm can be executed in polynomial time.



Approximation Algorithms via Sidney Decompoistion

Algorithm (2-Approximation).

1. Let $U \leftarrow N$, $\mathsf{S} \leftarrow []$.
2. **While** $U \neq \emptyset$,
3. determine initial set J in $D[U]$ with maximum ratio $w(J)/p(J)$;
4. append jobs from J to schedule S in arbitrary topological order.

Analysis:

- The algorithm computes a solution whose objective value is at most twice the optimum objective value.
- The algorithm can be executed in polynomial time.
- Chekuri, Motwani '99; Pisaruk '03: Solve multiple max-flow flow problems.
- Margot et al. '03: Solve a parametric max-flow problem, using the algorithm of Gallo et al. $\rightsquigarrow O(n^3)$

Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

Sven Jäger Philipp Warode

Symposium on Simplicity of Algorithms | 08 January 2024 | Alexandria, Virginia

Simple Approximation Algorithm Overview

Algorithm (Simple 2-Approximation).

1. Compute "virtual" fractional/preemptive schedule S.
2. Perform list scheduling in order of C_j^S.

Simple Approximation Algorithm Overview

Algorithm (Simple 2-Approximation).

1. Compute "virtual" fractional/preemptive schedule S.
2. Perform list scheduling in order of C_j^{S}.

Analysis:

1. $$\sum_{j \in N} w_j C_j^{\mathrm{S}} \leq 2 \cdot \sum_{j \in N} w_j C_j^{\mathrm{OPT}}$$

Simple Approximation Algorithm Overview

Algorithm (Simple 2-Approximation).

1. Compute "virtual" fractional/preemptive schedule S.
2. Perform list scheduling in order of C_j^S.

Analysis:

1. $$\sum_{j \in N} w_j C_j^S \leq 2 \cdot \sum_{j \in N} w_j C_j^{OPT}$$
2. $$\sum_{j \in N} w_j C_j^{ALG} \leq \sum_{j \in N} w_j C_j^S$$

Simple Approximation Algorithm Virtual Fractional Schedule

- A fractional schedule S assigns to each available job j a **processing rate** $R_j^S(t) \in [0,1]$ at any time $t \geq 0$ so that the sum of all processing rates never exceeds 1.

Simple Approximation Algorithm Virtual Fractional Schedule

- A fractional schedule S assigns to each available job j a **processing rate** $R_j^{\mathsf{S}}(t) \in [0,1]$ at any time $t \geq 0$ so that the sum of all processing rates never exceeds 1.
- The processing time of j before time t is $Y_j^{\mathsf{S}}(t) := \int_0^t R_j^{\mathsf{S}}(s)\,\mathrm{d}s$.

Simple Approximation Algorithm Virtual Fractional Schedule

- A fractional schedule S assigns to each available job j a **processing rate** $R_j^S(t) \in [0,1]$ at any time $t \geq 0$ so that the sum of all processing rates never exceeds 1.
- The processing time of j before time t is $Y_j^S(t) := \int_0^t R_j^S(s)\,\mathrm{d}s$.

Algorithm (Fractional Schedule).

At the beginning and whenever a job completes, **do**

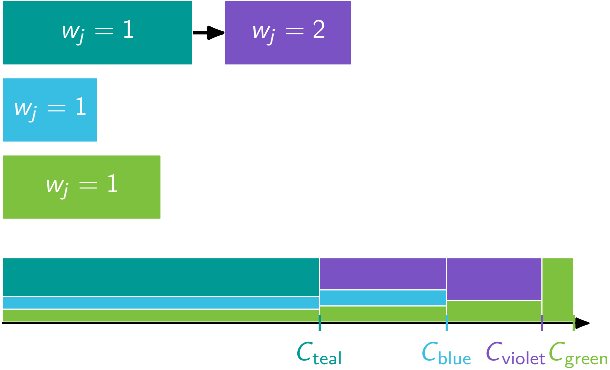
1 let U be the set of unfinished jobs, and let $F \subseteq U$ be the jobs without predecessor;

2 **for** $i \in F$

3 let $T(i)$ be the successors of i in U;

4 set $U \leftarrow U \setminus T(i)$;

5 process each job $i \in F$ at rate $R_i(t) \leftarrow \frac{\sum_{j \in T(i)} w_j}{\sum_{j \in U} w_j}$.



Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^{\mathrm{S}} \leq 2 \cdot \sum_{j \in N} w_j C_j^{\mathrm{OPT}}.$$

Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^{\mathrm{S}} \leq 2 \cdot \sum_{j \in N} w_j C_j^{\mathrm{OPT}}.$$

Proof. Induction on $n := |N|$:

Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j\in N} w_j C_j^{\mathrm{S}} \leq 2 \cdot \sum_{j\in N} w_j C_j^{\mathrm{OPT}}.$$

Proof. Induction on $n := |N|$:

- For a single job, the algorithm computes the optimal schedule.

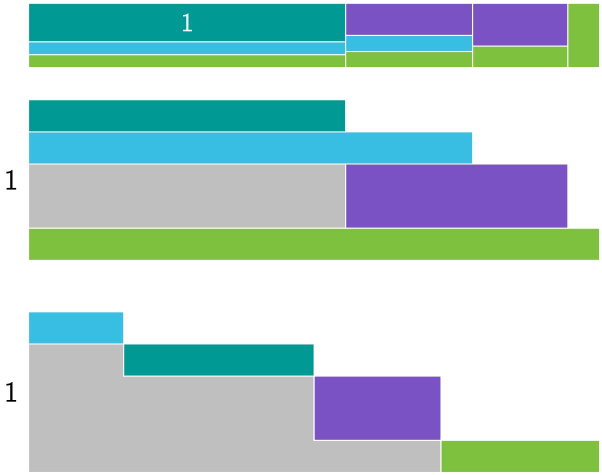
Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^S \leq 2 \cdot \sum_{j \in N} w_j C_j^{OPT}.$$

Proof. Induction on $n := |N|$:

- For a single job, the algorithm computes the optimal schedule.
- Let $n > 1$, and assume w.l.o.g. that job 1 finishes first in S and that $\sum_{j \in N} w_j = 1$.



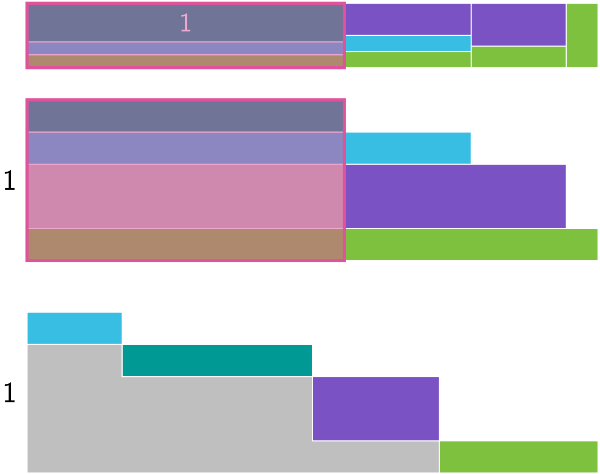
Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j\in N} w_j C_j^{S} \leq 2 \cdot \sum_{j\in N} w_j C_j^{OPT}.$$

Proof. Induction on $n := |N|$:

- For a single job, the algorithm computes the optimal schedule.
- Let $n > 1$, and assume w.l.o.g. that job 1 finishes first in S and that $\sum_{j\in N} w_j = 1$.
- Consider instance I' remaining at C_1^S.



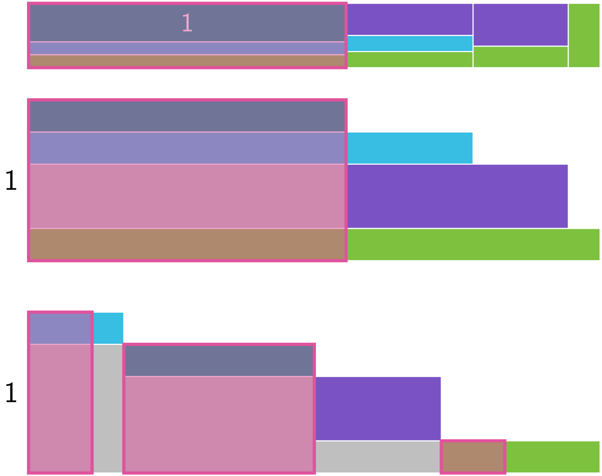
Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^{S} \leq 2 \cdot \sum_{j \in N} w_j C_j^{OPT}.$$

Proof. Induction on $n := |N|$:

- For a single job, the algorithm computes the optimal schedule.
- Let $n > 1$, and assume w.l.o.g. that job 1 finishes first in S and that $\sum_{j \in N} w_j = 1$.
- Consider instance I' remaining at C_1^S.
- Removing the processed parts from OPT yields feasible schedule OPT′ for I'.



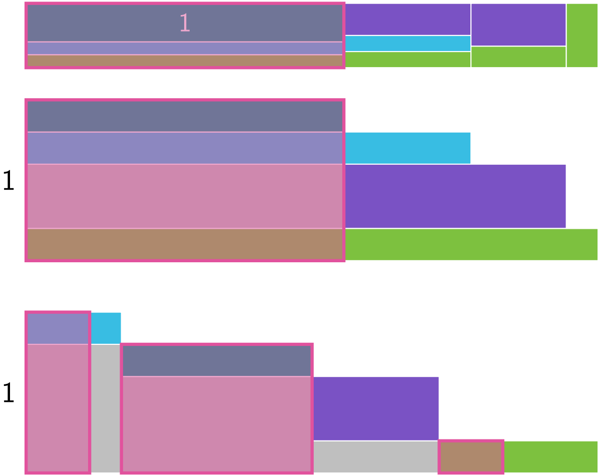
Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^{S} \leq 2 \cdot \sum_{j \in N} w_j C_j^{OPT}.$$

Proof. Induction on $n := |N|$:

- For a single job, the algorithm computes the optimal schedule.
- Let $n > 1$, and assume w.l.o.g. that job 1 finishes first in S and that $\sum_{j \in N} w_j = 1$.
- Consider instance I' remaining at C_1^S.
- Removing the processed parts from OPT yields feasible schedule OPT$'$ for I'.
- By induction, remaining part of S costs at most twice as much as OPT$'$.



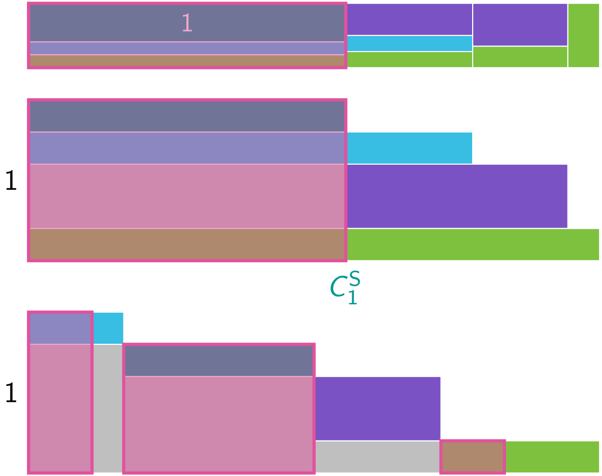
Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^{S} \leq 2 \cdot \sum_{j \in N} w_j C_j^{OPT}.$$

Proof. Induction on $n := |N|$:

- The upper red area is $C_1^{S} \cdot 1$.



Performance Guarantee for Virtual Fractional Schedule

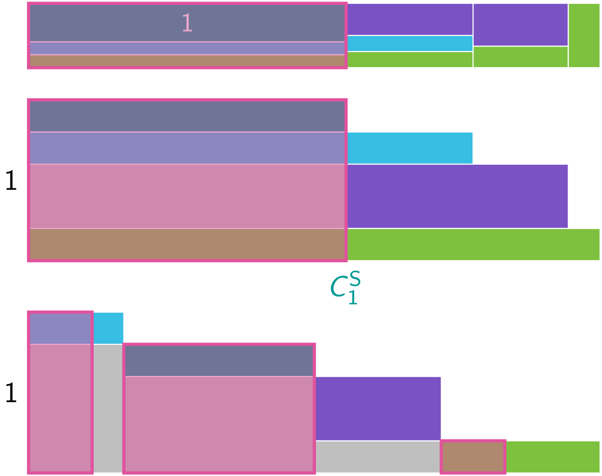
Theorem 3.1.

$$\sum_{j\in N} w_j C_j^{\mathrm{S}} \leq 2 \cdot \sum_{j\in N} w_j C_j^{\mathrm{OPT}}.$$

Proof. Induction on $n := |N|$:

- The upper red area is $C_1^{\mathrm{S}} \cdot 1$.
- The lower red area is

$$\sum_{j\in N} Y_j^{\mathrm{S}}(C_1^{\mathrm{S}}) \cdot \sum_{k:C_k^{\mathrm{OPT}} \geq C_j^{\mathrm{OPT}}} w_k.$$



Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

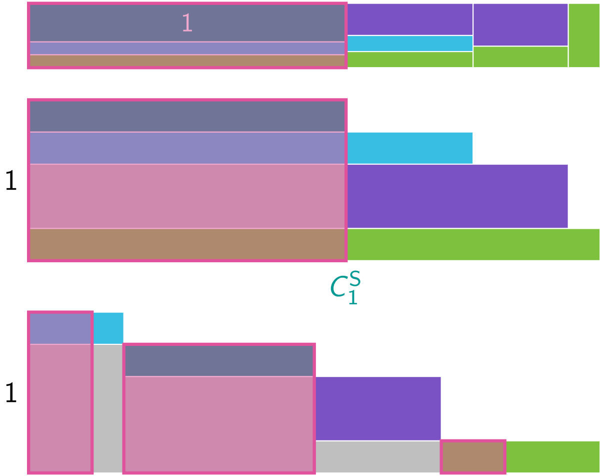
$$\sum_{j \in N} w_j C_j^{\mathrm{S}} \leq 2 \cdot \sum_{j \in N} w_j C_j^{\mathrm{OPT}}.$$

Proof. Induction on $n := |N|$:

- The upper red area is $C_1^{\mathrm{S}} \cdot 1$.
- The lower red area is

$$\sum_{j \in N} Y_j^{\mathrm{S}}(C_1^{\mathrm{S}}) \cdot \sum_{k: C_k^{\mathrm{OPT}} \geq C_j^{\mathrm{OPT}}} w_k.$$

$$= \sum_{j \in F} C_1^{\mathrm{S}} \cdot w(T(j)) \cdot \sum_{k: C_k^{\mathrm{OPT}} \geq C_j^{\mathrm{OPT}}} w_k$$



Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j\in N} w_j C_j^{\mathrm{S}} \le 2 \cdot \sum_{j \in N} w_j C_j^{\mathrm{OPT}}.$$

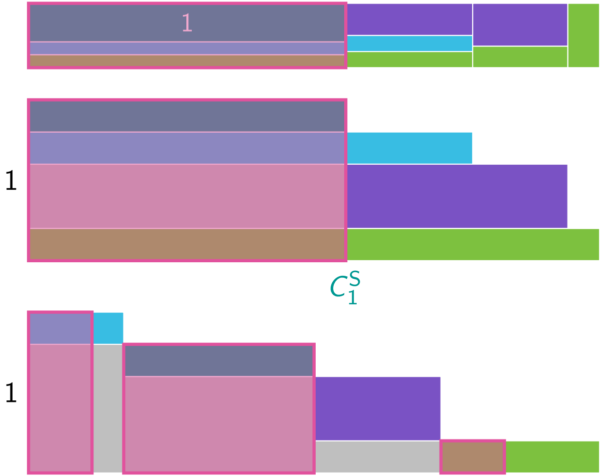
Proof. Induction on $n := |N|$:

- The upper red area is $C_1^{\mathrm{S}} \cdot 1$.
- The lower red area is

$$\sum_{j\in N} Y_j^{\mathrm{S}}(C_1^{\mathrm{S}}) \cdot \sum_{k: C_k^{\mathrm{OPT}} \ge C_j^{\mathrm{OPT}}} w_k.$$

$$= \sum_{j\in F} C_1^{\mathrm{S}} \cdot w(T(j)) \cdot \sum_{k: C_k^{\mathrm{OPT}} \ge C_j^{\mathrm{OPT}}} w_k$$

$$\ge C_1^{\mathrm{S}} \cdot \sum_{j\in F} w(T(j)) \cdot \sum_{i\in F: C_i^{\mathrm{OPT}} \ge C_j^{\mathrm{OPT}}} w(T(i))$$



Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j \in N} w_j C_j^S \leq 2 \cdot \sum_{j \in N} w_j C_j^{OPT}.$$

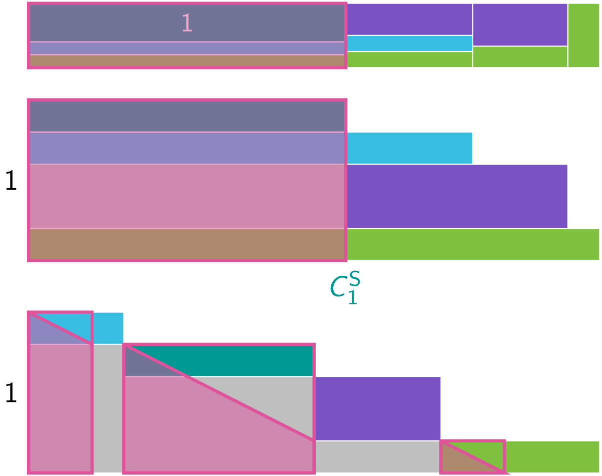
Proof. Induction on $n := |N|$:

- The upper red area is $C_1^S \cdot 1$.
- The lower red area is

$$\sum_{j \in N} Y_j^S(C_1^S) \cdot \sum_{k: C_k^{OPT} \geq C_j^{OPT}} w_k.$$

$$= \sum_{j \in F} C_1^S \cdot w(T(j)) \cdot \sum_{k: C_k^{OPT} \geq C_j^{OPT}} w_k$$

$$\geq C_1^S \cdot \sum_{j \in F} w(T(j)) \cdot \sum_{i \in F: C_i^{OPT} \geq C_j^{OPT}} w(T(i)) \geq C_1^S \cdot \frac{1}{2} \left(\sum_{i \in F} w(T(i)) \right)^2$$



Performance Guarantee for Virtual Fractional Schedule

Theorem 3.1.

$$\sum_{j\in N} w_j C_j^S \leq 2 \cdot \sum_{j\in N} w_j C_j^{OPT}.$$

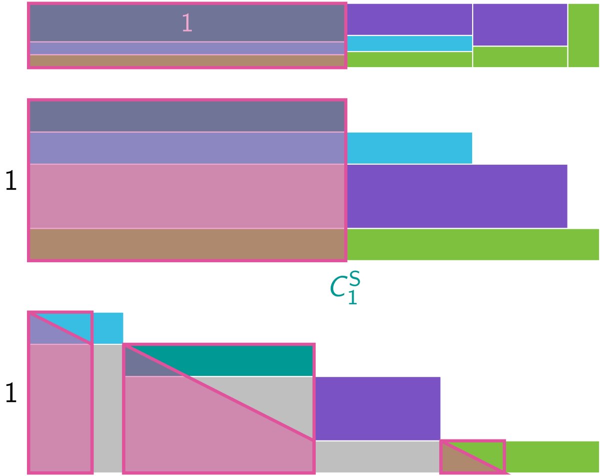
Proof. Induction on $n := |N|$:

- The upper red area is $C_1^S \cdot 1$.
- The lower red area is

$$\sum_{j\in N} Y_j^S(C_1^S) \cdot \sum_{k:C_k^{OPT}\geq C_j^{OPT}} w_k.$$

$$= \sum_{j\in F} C_1^S \cdot w(T(j)) \cdot \sum_{k:C_k^{OPT}\geq C_j^{OPT}} w_k$$

$$\geq C_1^S \cdot \sum_{j\in F} w(T(j)) \cdot \sum_{i\in F:C_i^{OPT}\geq C_j^{OPT}} w(T(i)) \geq C_1^S \cdot \frac{1}{2}\left(\sum_{i\in F} w(T(i))\right)^2 = \frac{C_1^S}{2}. \quad \square$$



Performance Guarantee for List Schedule

Lemma.

$$\sum_{j \in N} w_j C_j^{\mathsf{ALG}} \leq \sum_{j \in N} w_j C_j^{\mathsf{S}}.$$

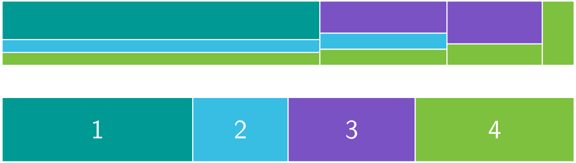
Performance Guarantee for List Schedule

Lemma.

$$\sum_{j\in N} w_j C_j^{\mathrm{ALG}} \leq \sum_{j\in N} w_j C_j^{\mathrm{S}}.$$

Proof.

- Assume that $C_1^{\mathrm{S}} \leq \cdots \leq C_n^{\mathrm{S}}$.



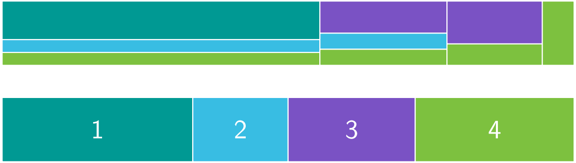
Performance Guarantee for List Schedule

Lemma.

$$\sum_{j \in N} w_j C_j^{\mathrm{ALG}} \leq \sum_{j \in N} w_j C_j^{\mathrm{S}}.$$

Proof.

- Assume that $C_1^{\mathrm{S}} \leq \cdots \leq C_n^{\mathrm{S}}$.
- In the list schedule, we have $C_j^{\mathrm{ALG}} = \sum_{k=1}^{j} p_k$ for all $j \in N$.



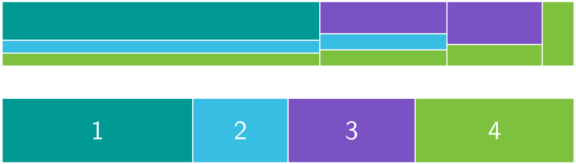
Performance Guarantee for List Schedule

Lemma.

$$\sum_{j \in N} w_j C_j^{\mathrm{ALG}} \leq \sum_{j \in N} w_j C_j^{\mathrm{S}}.$$

Proof.

- Assume that $C_1^{\mathrm{S}} \leq \cdots \leq C_n^{\mathrm{S}}$.
- In the list schedule, we have $C_j^{\mathrm{ALG}} = \sum_{k=1}^{j} p_k$ for all $j \in N$.
- In the fractional schedule, we have $\sum_{k=1}^{j} p_k \leq C_j^{\mathrm{S}}$ for all $j \in N$. □



Comments on the Algorithm

- The algorithm runs in time $O(n^2)$.

Comments on the Algorithm

- The algorithm runs in time $O(n^2)$.
- The computation of the preemptive schedule S is **non-clairvoyant**.

Comments on the Algorithm

- The algorithm runs in time $O(n^2)$.
- The computation of the preemptive schedule S is **non-clairvoyant**.

Corollary.
There is a 2-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on a single machine.

Comments on the Algorithm

- The algorithm runs in time $O(n^2)$.
- The computation of the preemptive schedule S is **non-clairvoyant**.

Corollary.
There is a 2-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on a single machine.

- No better non-clairvoyant algorithm is possible. (Motwani et al. '94)

Comments on the Algorithm

- The algorithm runs in time $O(n^2)$.
- The computation of the preemptive schedule S is **non-clairvoyant**.

Corollary.
There is a 2-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on a single machine.

- No better non-clairvoyant algorithm is possible. (Motwani et al. '94)
- The best previously known competitive ratios were
  - 4 for out-forest precedence constraints (Lassota et al. '23) and
  - 8 for general precedence constraints (on identical machines with release dates) (Jäger '21).

Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

Sven Jäger Philipp Warode

Symposium on Simplicity of Algorithms | 08 January 2024 | Alexandria, Virginia

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),
  - $1 + \sqrt{2}$ for unit processing time jobs (Li '20).

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),
  - $1 + \sqrt{2}$ for unit processing time jobs (Li '20).
- The best previously known competitive ratios of non-clairvoyant algorithms were

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),
  - $1 + \sqrt{2}$ for unit processing time jobs (Li '20).
- The best previously known competitive ratios of non-clairvoyant algorithms were
  - 6 for out-forest precedence constraints (Lassota et al. '23),

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),
  - $1 + \sqrt{2}$ for unit processing time jobs (Li '20).
- The best previously known competitive ratios of non-clairvoyant algorithms were
  - 6 for out-forest precedence constraints (Lassota et al. '23),
  - 8 for general precedence constraints (with release dates) (Jäger '21).

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),
  - $1 + \sqrt{2}$ for unit processing time jobs (Li '20).
- The best previously known competitive ratios of non-clairvoyant algorithms were
  - 6 for out-forest precedence constraints (Lassota et al. '23),
  - 8 for general precedence constraints (with release dates) (Jäger '21).
- No lower bound above 2 is known.

Identical Parallel Machines

Theorem.
There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines.

- The algorithm is based on a parametric max-flow computation.
- The performance guarantee of the best known (clairvoyant) approximation algorithms are
  - $3 - 1/m$ for arbitrary jobs (Hall et al. '97),
  - $1 + \sqrt{2}$ for unit processing time jobs (Li '20).
- The best previously known competitive ratios of non-clairvoyant algorithms were
  - 6 for out-forest precedence constraints (Lassota et al. '23),
  - 8 for general precedence constraints (with release dates) (Jäger '21).
- No lower bound above 2 is known.
- Our algorithm cannot be made non-preemptive without impairing the performance guarantee.

Summary

1. There is a simple 2-competitive non-clairvoyant round-robin type algorithm for scheduling precedence-constrained jobs on a single machine. This matches the lower bound for non-clairvoyant scheduling.

Summary

1. There is a simple 2-competitive non-clairvoyant round-robin type algorithm for scheduling precedence-constrained jobs on a single machine. This matches the lower bound for non-clairvoyant scheduling.

2. There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines. This is based on a parametric flow computation.

Summary

1. There is a simple 2-competitive non-clairvoyant round-robin type algorithm for scheduling precedence-constrained jobs on a single machine. This matches the lower bound for non-clairvoyant scheduling.
2. There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines. This is based on a parametric flow computation.
3. Both algorithms attain the best known constant performance guarantee of any clairvoyant approximation algorithm for the problem.

Summary

1. There is a simple 2-competitive non-clairvoyant round-robin type algorithm for scheduling precedence-constrained jobs on a single machine. This matches the lower bound for non-clairvoyant scheduling.
2. There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines. This is based on a parametric flow computation.
3. Both algorithms attain the best known constant performance guarantee of any clairvoyant approximation algorithm for the problem.
4. Their running times improve upon the running times of previously known approximation algorithms.

Summary

1. There is a simple 2-competitive non-clairvoyant round-robin type algorithm for scheduling precedence-constrained jobs on a single machine. This matches the lower bound for non-clairvoyant scheduling.

2. There is a 3-competitive non-clairvoyant algorithm for preemptive precedence-constrained scheduling on identical parallel machines. This is based on a parametric flow computation.

3. Both algorithms attain the best known constant performance guarantee of any clairvoyant approximation algorithm for the problem.

4. Their running times improve upon the running times of previously known approximation algorithms.

Thank you!

References I

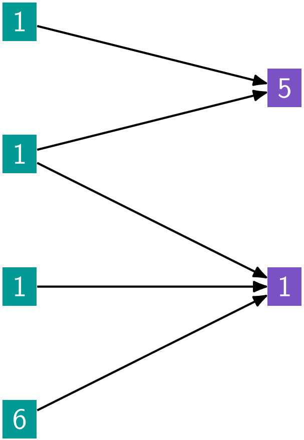
- Bansal, N., Khot, S. (2009). "Optimal Long Code Test with One Free Bit". *50th Annu. IEEE Symp. Found. Comput. Sci.* (FOCS), pp. 453–462.
- Chekuri, C., Motwani, R. (1999). "Precedence constrained scheduling to minimize sum of weighted completion times on a single machine". *Discrete Appl. Math.* 98(1-2), pp. 29–38.
- Chudak, F. A., Hochbaum, D. S. (1999). "A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine". *Oper. Res. Lett.* 25(5), pp. 199–204.
- Gallo, G., Grigoriadis, M. D., Tarjan, R. E. (1989). "A Fast Parametric Maximum Flow Algorithm and Applications". *SIAM J. Comput.* 18(1), pp. 30–55.
- Hall, L. A., Schulz, A. S., Shmoys, D. B., Wein, J. (1997). "Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms". *Math. Oper. Res.* 22(3), pp. 513–544.
- Jäger, S. J. (2021). "Approximation in deterministic and stochastic machine scheduling". PhD thesis. Technische Universität Berlin.
- Lassota, A. A., Lindermayr, A., Megow, N., Schlöter, J. (2023–2023). "Minimalistic Predictions to Schedule Jobs with Online Precedence Constraints". *Proc. 40th Int. Conf. Mach. Learn.* (ICML), pp. 18563–18583.

References II

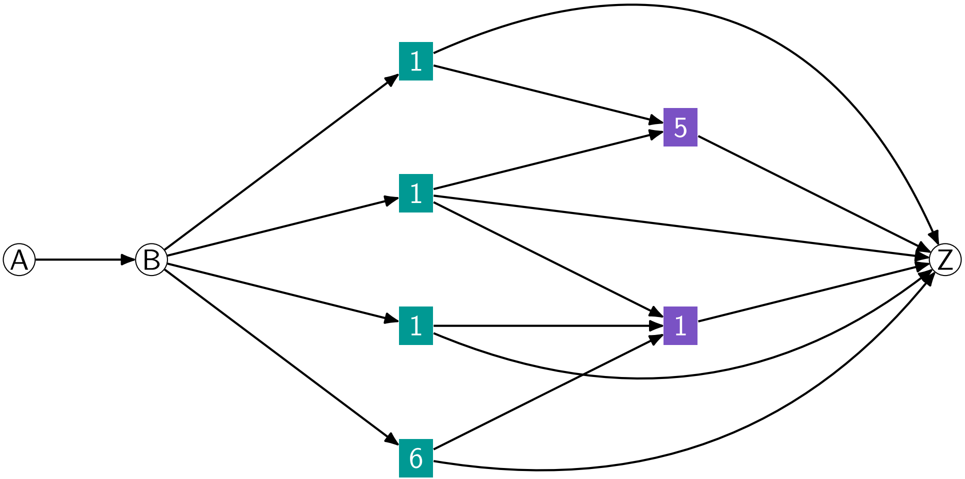
- Lawler, E. L. (1978). "Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints". In: *Algorithmic Aspects of Combinatorics*. Ed. by B. Alspach, P. Hell, D. J. Miller. Annals of Discrete Mathematics 2. Elsevier, pp. 75–90.
- Li, S. (2020). "Scheduling to Minimize Total Weighted Completion Time via Time-Indexed Linear Programming Relaxations". *SIAM J. Comput.* 49(4), FOCS17-409–FOCS17-440.
- Margot, F., Queyranne, M., Wang, Y. (2003). "Decompositions, Network Flows, and a Precedence Constrained Single-Machine Scheduling Problem". *Oper. Res.* 51(6), pp. 981–992.
- Motwani, R., Phillips, S., Torng, E. (1994). "Nonclairvoyant scheduling". *Theor. Comput. Sci.* 130(1), pp. 17–47.
- Pisaruk, N. N. (1992). "The boundaries of submodular functions". *Comput. Math. Math. Phys.* 32(12), pp. 1769–1783.
- Pisaruk, N. N. (2003). "A fully combinatorial 2-approximation algorithm for precedence-constrained scheduling a single machine to minimize average weighted completion time". *Discrete Appl. Math.* 131(3), pp. 655–663.

Appendix

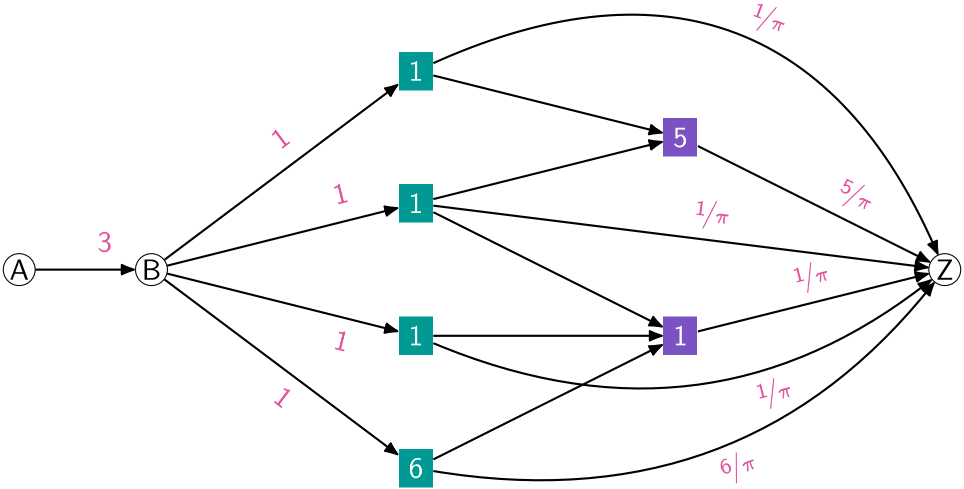
Extended Precedence Graph



Extended Precedence Graph



Extended Precedence Graph



Rate Distribution for Identical Parallel Machines

Algorithm (Rate Distribution).

1. Let F be the unfinished jobs without unfinished predecessor.
2. **If** $|F| \leq m$,
3. set $R_j(t) \leftarrow 1$ for all $j \in F$;

Rate Distribution for Identical Parallel Machines

Algorithm (Rate Distribution).

```
Let F be the unfinished jobs without unfinished predecessor.
If |F| ≤ m,
    set R_j(t) ← 1 for all j ∈ F;
else
    compute π ← max{π > 0 | ({A}, V_t \ {A}) is a minimum-capacity A-Z-cut},
    and let x be a corresponding maximum A-Z-flow;
    set R_j(t) ← x_(B,j) for all j ∈ F.
```

Let F be the unfinished jobs without unfinished predecessor.

If $|F| \leq m$,

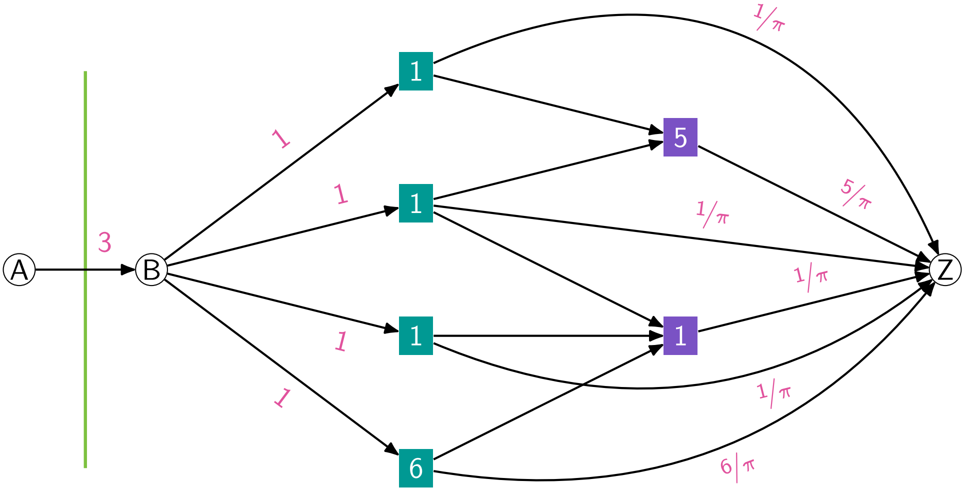
  set $R_j(t) \leftarrow 1$ for all $j \in F$;

else

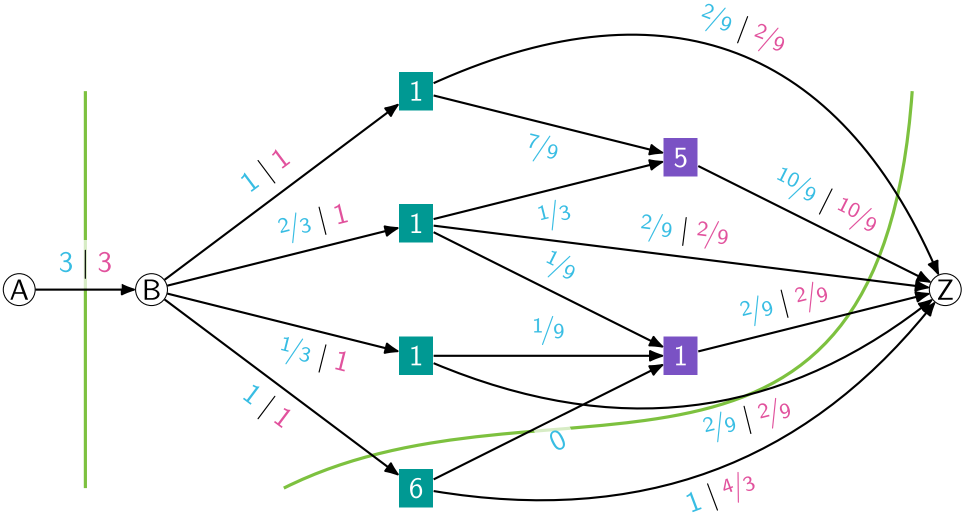
  compute $\pi \leftarrow \max\{\pi > 0 \mid (\{\mathrm{A}\}, \mathcal{V}_t \setminus \{\mathrm{A}\})$ is a minimum-capacity A-Z-cut$\}$, and let x be a corresponding maximum A-Z-flow;

  set $R_j(t) \leftarrow x_{(\mathrm{B},j)}$ for all $j \in F$.

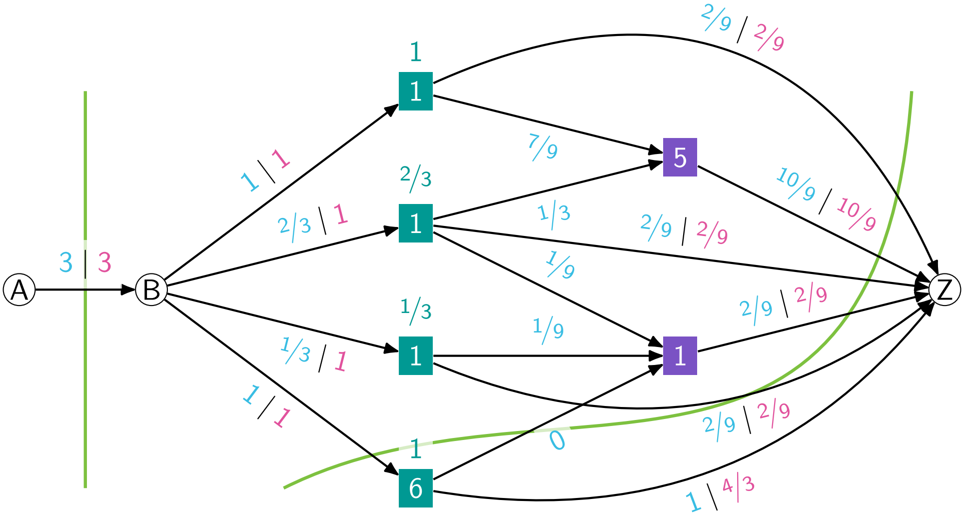
Rate Distribution for Identical Parallel Machines



Rate Distribution for Identical Parallel Machines



Rate Distribution for Identical Parallel Machines



Submodular Ordering Problem

For a permutation $\pi\colon [n] \to [n]$ and $i \in [n]$ let $\pi[i] := \{\pi(1), \dots, \pi(i)\}$.

Submodular Ordering Problem

Given: non-increasing submodular function $f\colon 2^{[n]} \to \mathbb{R}$ and non-decreasing submodular function $g\colon 2^{[n]} \to \mathbb{R}$;

Task: find a permutation $\pi\colon [n] \to [n]$ such that $\sum_{i=1}^{n} f(\pi[i]) \cdot \big(g(\pi[i]) - g(\pi[i-1])\big)$ is minimized.

Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs

- $f(J) := \sum_{j \in N \setminus J} w_j$
- $g(J) := \sum_{j \in \mathrm{pred}(J)} p_j$

An optimal permutation is consistent with the precedence constraints.