Scheduling and Parameterized Complexity

Maher Mallem

Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France

21 January 2025

Outline

Outline

- [INFORMS 2012] Operations Research:

  "A branch of applied mathematics that deals with the development and application of analytical methods to improve decision-making."

Operations Research

- [INFORMS 2012] Operations Research:

  "A branch of applied mathematics that deals with the development and application of analytical methods to improve decision-making."

  Examples:

  - Lot Sizing
  - Vehicle Routing Problem

Operations Research

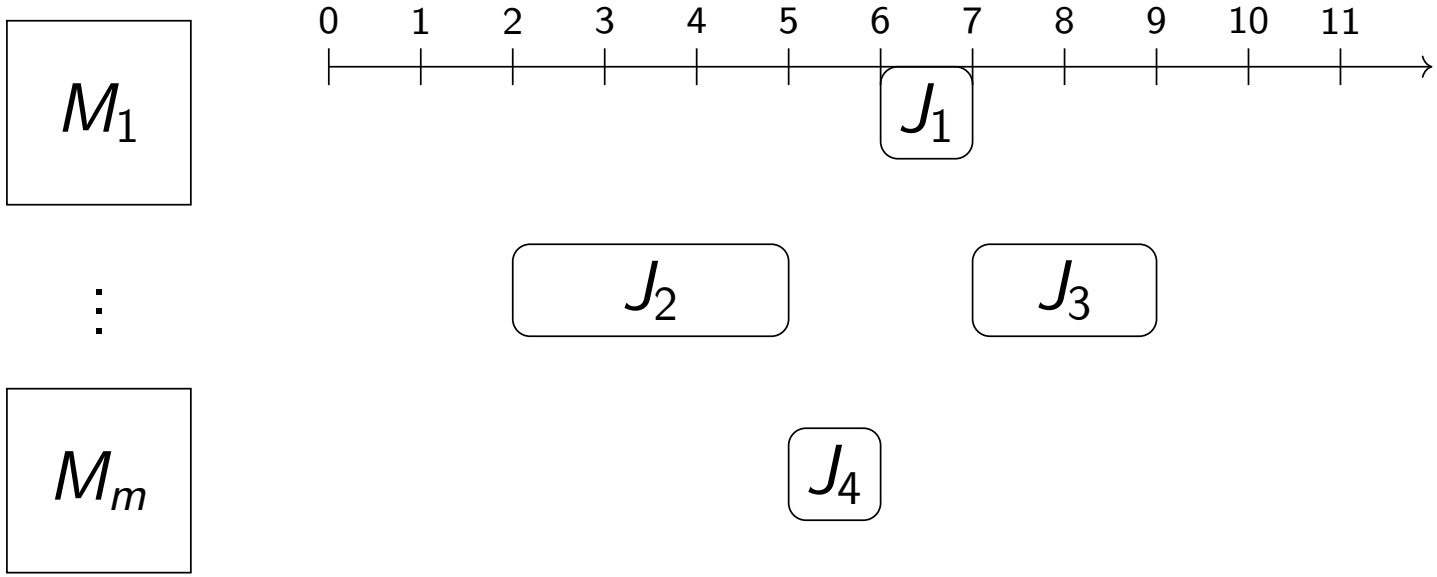
- [INFORMS 2012] Operations Research:

  "A branch of applied mathematics that deals with the development and application of analytical methods to improve decision-making."

  Examples:

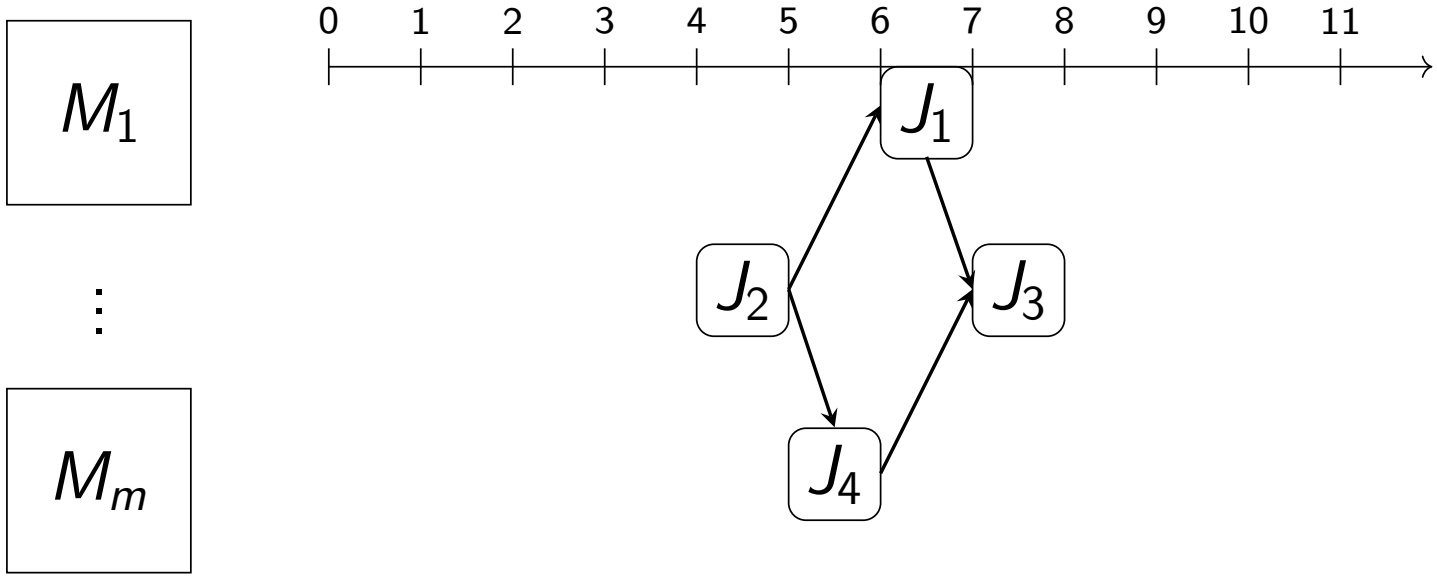
  - Lot Sizing
  - Vehicle Routing Problem
  - Scheduling

Scheduling



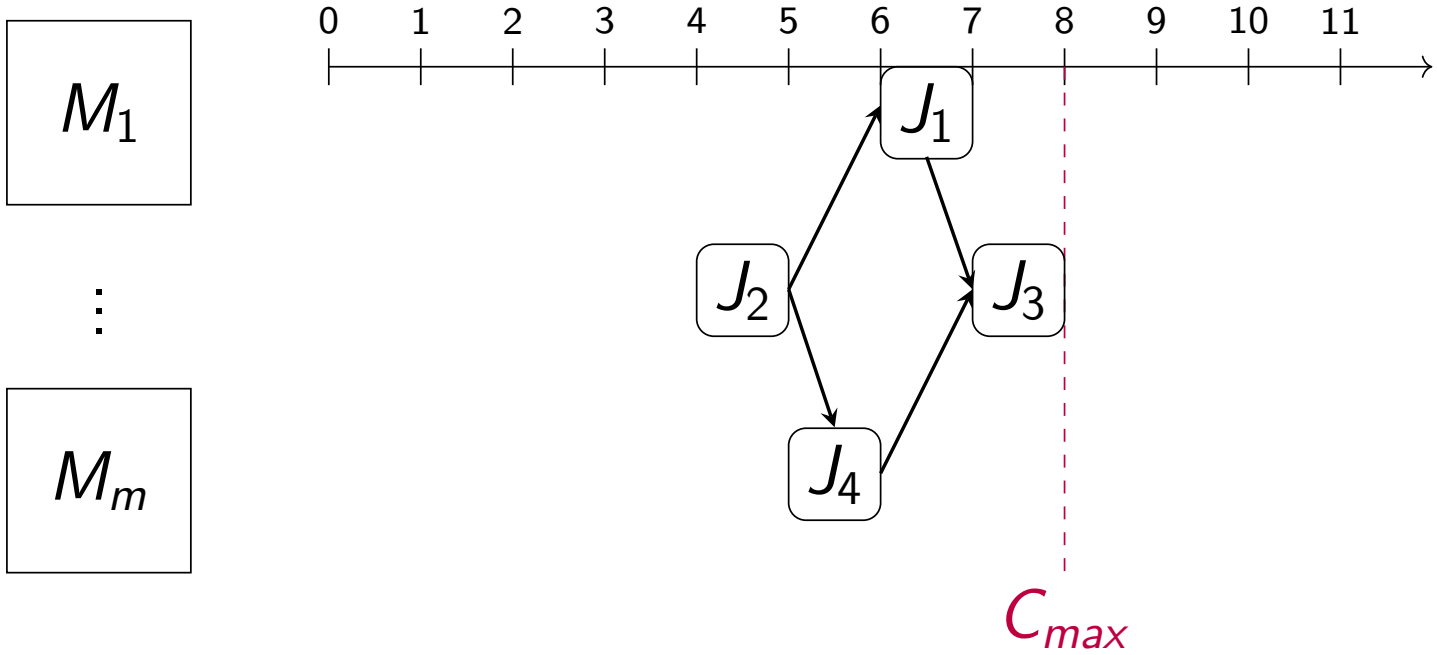
m machines M_i, n jobs J_j with processing time p_j

Scheduling



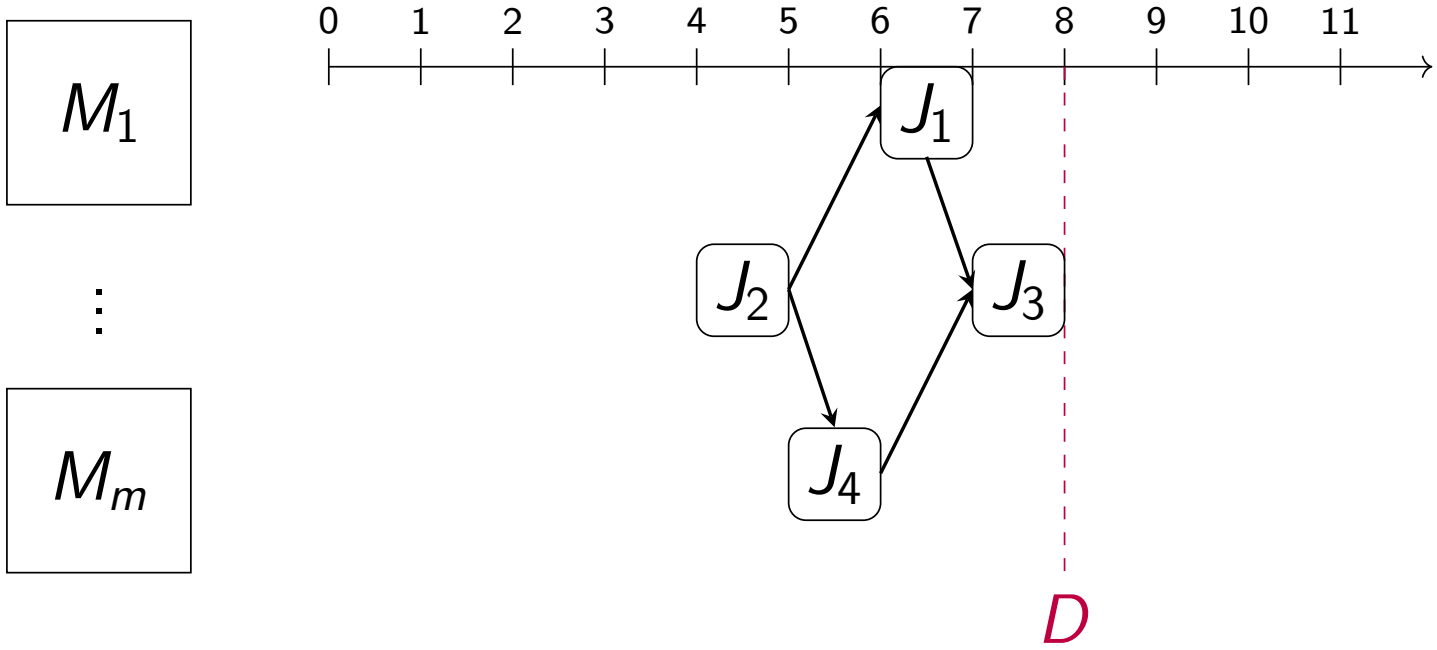
m machines M_i, n jobs J_j with processing time p_j

Scheduling



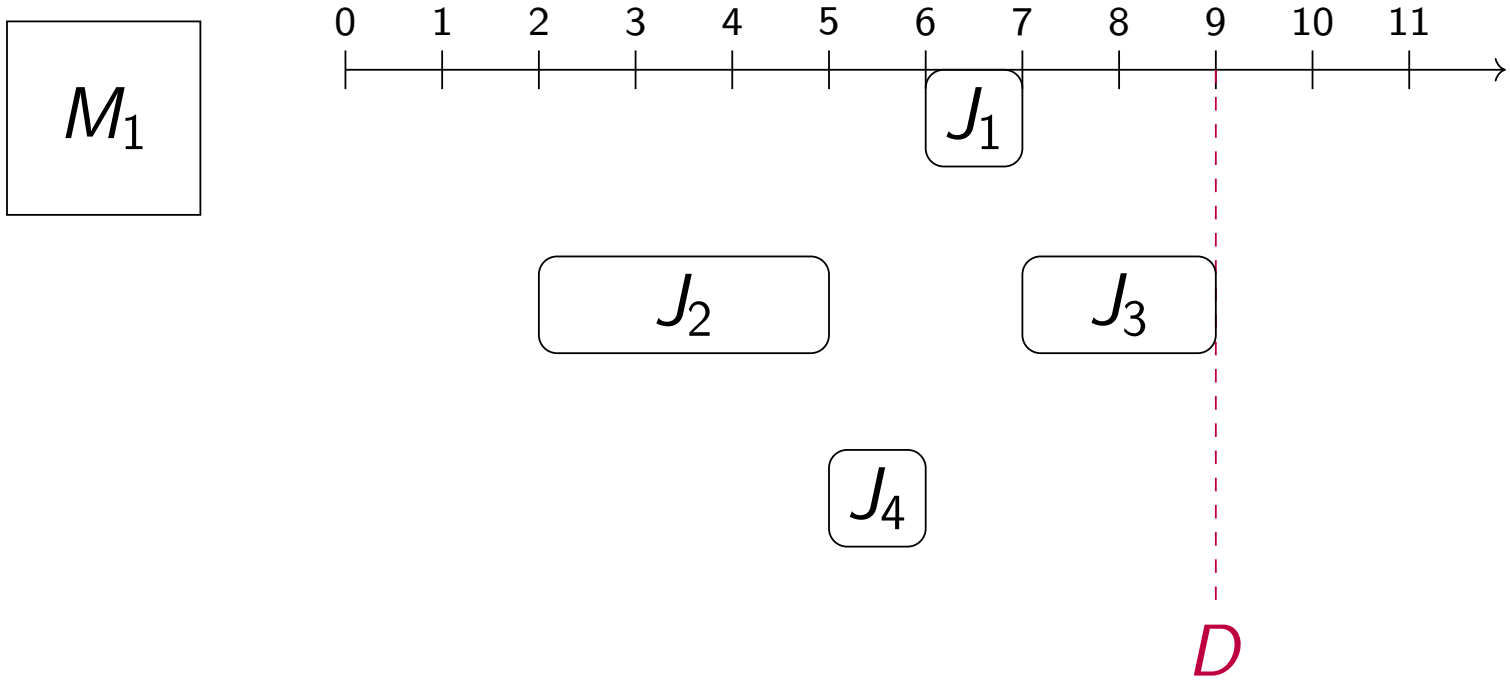
m machines M_i, n jobs J_j with processing time p_j

Scheduling



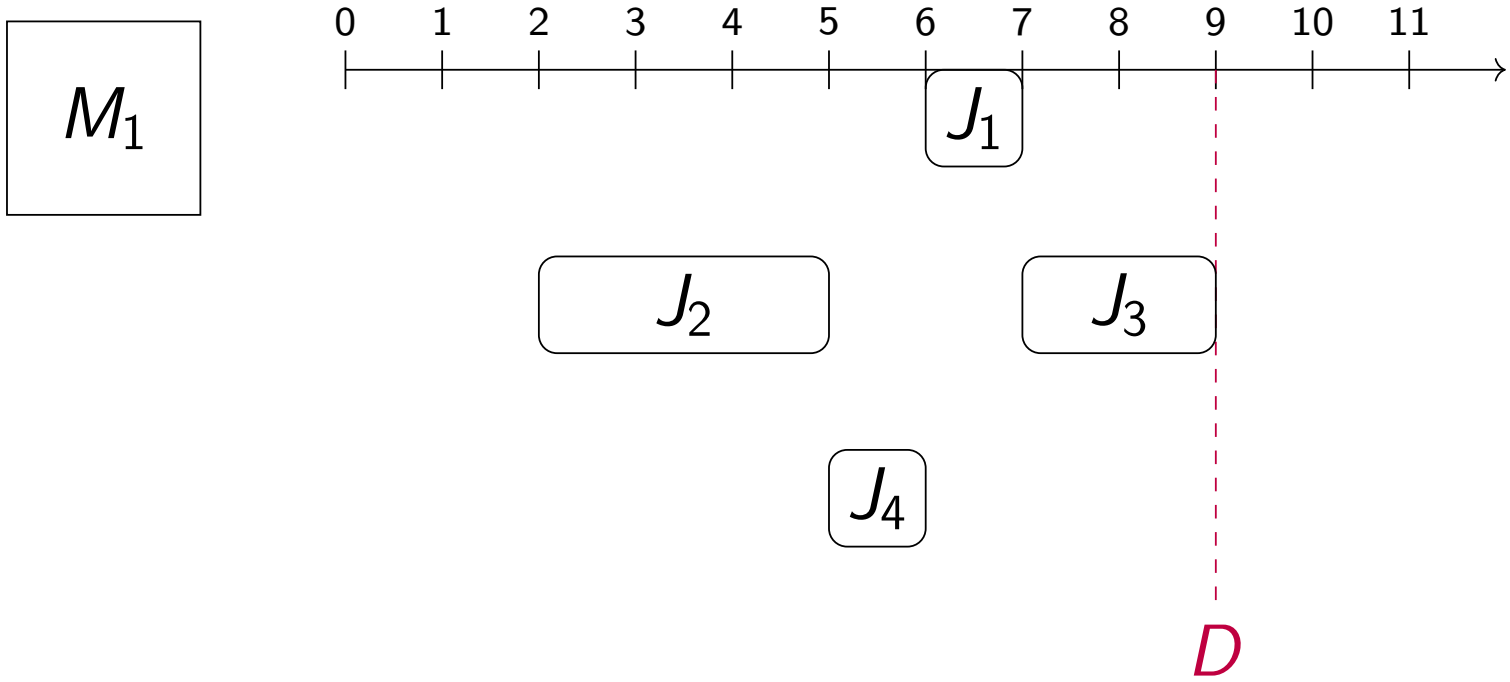
m machines M_i, n jobs J_j with processing time p_j

Classical complexity on scheduling



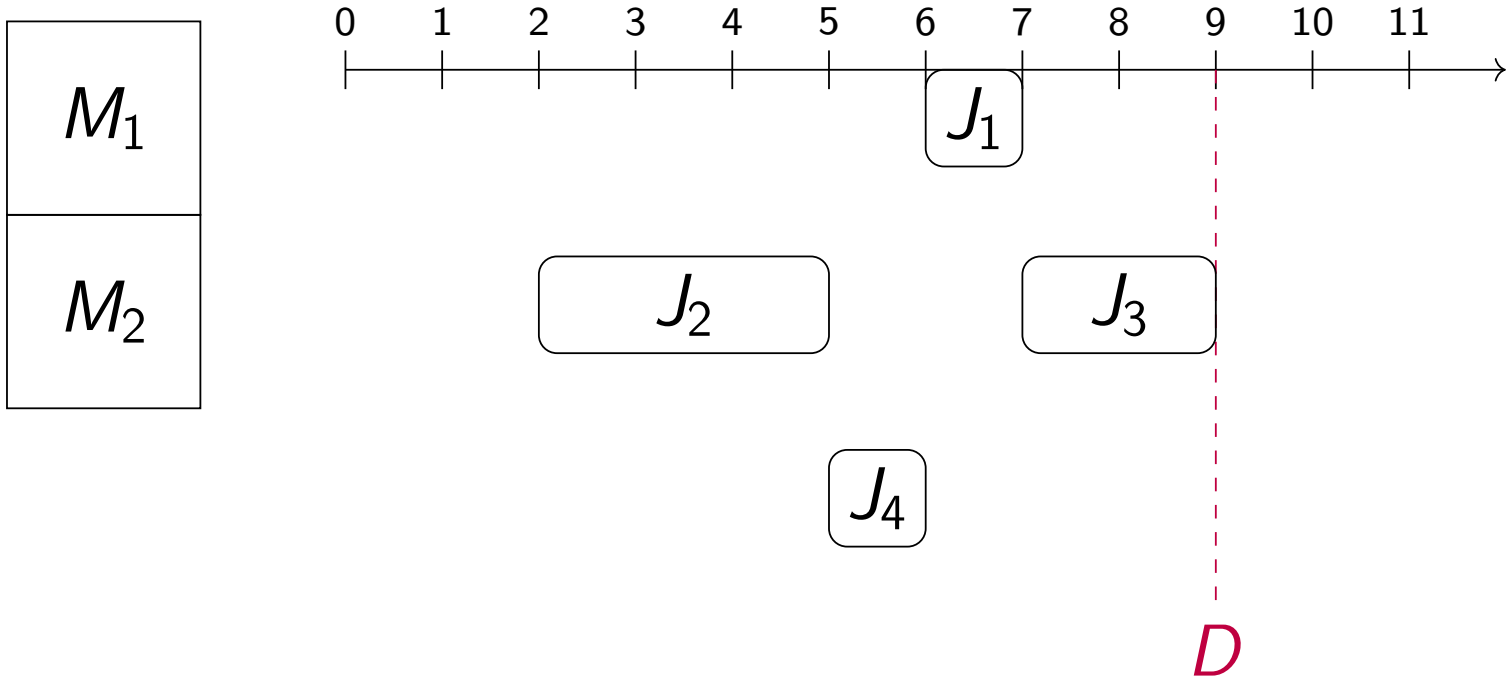
One machine:

Classical complexity on scheduling



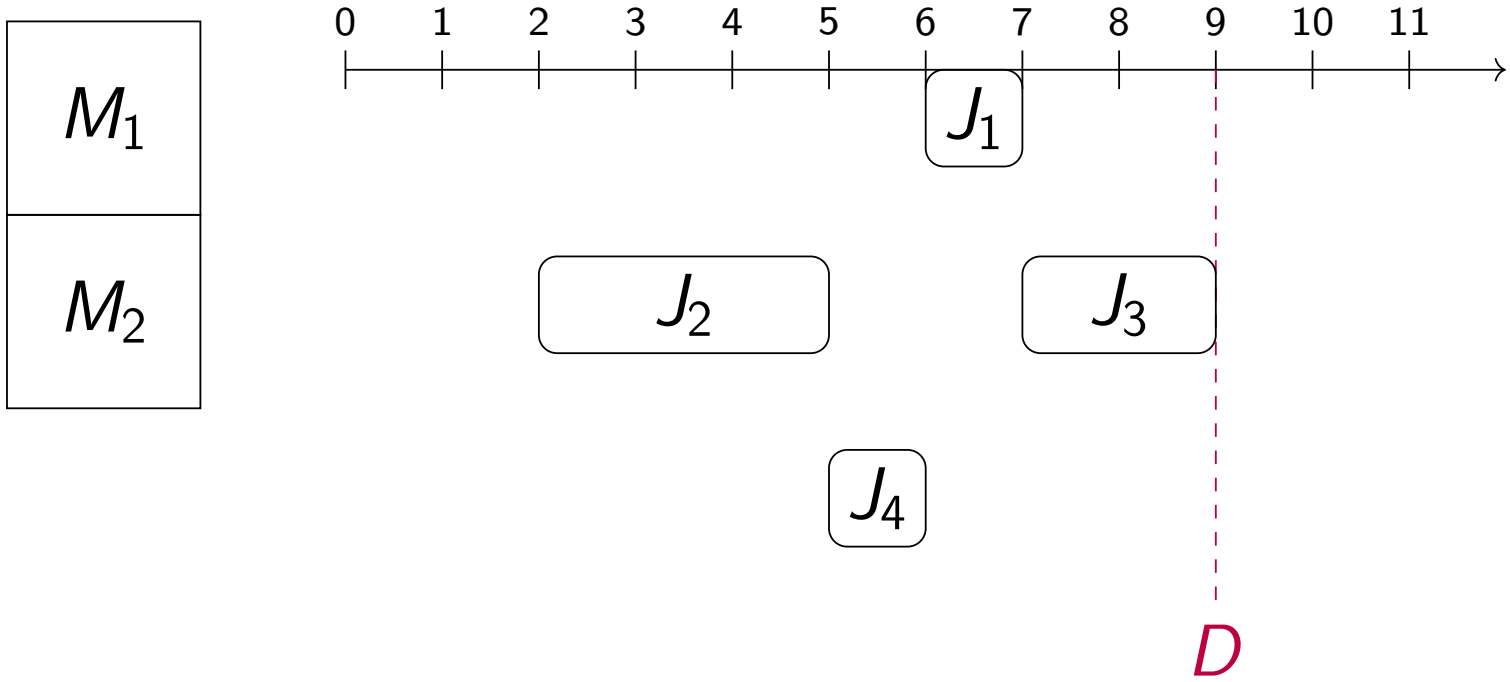
One machine: time $\mathcal{O}(n)$

Classical complexity on scheduling



Two identical machines:

Classical complexity on scheduling



Two identical machines: NP-hard

Polynomial reductions

Definition: Polynomial reduction

Decision problem P polynomially reduces to decision problem P' when for each instance I of P, one can build an instance I' of P' in time $poly(|I|)$ such that I is a YES instance if and only if I' is a YES instance.

Polynomial reductions

Definition: Polynomial reduction

Decision problem P polynomially reduces to decision problem P' when for each instance I of P, one can build an instance I' of P' in time $poly(|I|)$ such that I is a YES instance if and only if I' is a YES instance.

- Example:
  PARTITION $\longrightarrow$ Scheduling with two identical machines

Polynomial reductions

Definition: Polynomial reduction

Decision problem P polynomially reduces to decision problem P' when for each instance I of P, one can build an instance I' of P' in time $poly(|I|)$ such that I is a YES instance if and only if I' is a YES instance.

- Example:
  PARTITION $\longrightarrow$ Scheduling with two identical machines

  Instance I of PARTITION:
  - n elements $a_j \in \mathbb{N}^*$.
  - Partition length T.

Polynomial reductions

Definition: Polynomial reduction

Decision problem P polynomially reduces to decision problem P' when for each instance I of P, one can build an instance I' of P' in time $poly(|I|)$ such that I is a YES instance if and only if I' is a YES instance.

- Example:
  PARTITION $\longrightarrow$ Scheduling with two identical machines

  Instance I of PARTITION:
  - n elements $a_j \in \mathbb{N}^*$.
  - Partition length T.

  Build scheduling instance I':
  - For each a_j a job J_j with processing time $p_j = a_j$.
  - Deadline $D = T$.

Going beyond NP-hardness

- Problem $\mathcal{P}$, instance $\mathcal{I}$.
- P: deterministic time $poly(|\mathcal{I}|)$.
- NP: nondeterministic time $poly(|\mathcal{I}|)$.

Going beyond NP-hardness

- Problem $\mathcal{P}$, instance $\mathcal{I}$.
  - Problem $(\mathcal{P}, k)$, instance $\mathcal{I}$ with parameter value $\leq k$.
- P: deterministic time $poly(|\mathcal{I}|)$.
  - FPT: deterministic time $f(k) \cdot poly(|\mathcal{I}|)$.
- NP: nondeterministic time $poly(|\mathcal{I}|)$.
  - para-NP: nondeterministic time $f(k) \cdot poly(|\mathcal{I}|)$.

Going beyond NP-hardness (cont.)

Examples:

- SAT in time $\mathcal{O}(2^{\#variables} \cdot |\varphi|)$.
- k-COLORING is para-NP-hard with respect to the number of colors k.

Going beyond NP-hardness (cont.)

Examples:

- SAT in time $\mathcal{O}(2^{\#variables} \cdot |\varphi|)$.
- k-COLORING is para-NP-hard with respect to the number of colors k.

Remark

If parameter k is $|I|$, then:

problem P is FPT if and only if it is decidable.

Proposition [Flüm, Grohe 99]

- P = NP if and only if FPT = para-NP.
- Parameterized problem (P, k) is para-NP-complete if and only if problem P is NP-complete for some fixed value of k.

How to prove parameterized hardness

Proposition [Flüm, Grohe 99]

- P = NP if and only if FPT = para-NP.
- Parameterized problem (P, k) is para-NP-complete if and only if problem P is NP-complete for some fixed value of k.

Examples:

- COLORING is NP-complete with $k = 3$ colors.

How to prove parameterized hardness

Proposition [Flüm, Grohe 99]

- P = NP if and only if FPT = para-NP.
- Parameterized problem (P, k) is para-NP-complete if and only if problem P is NP-complete for some fixed value of k.

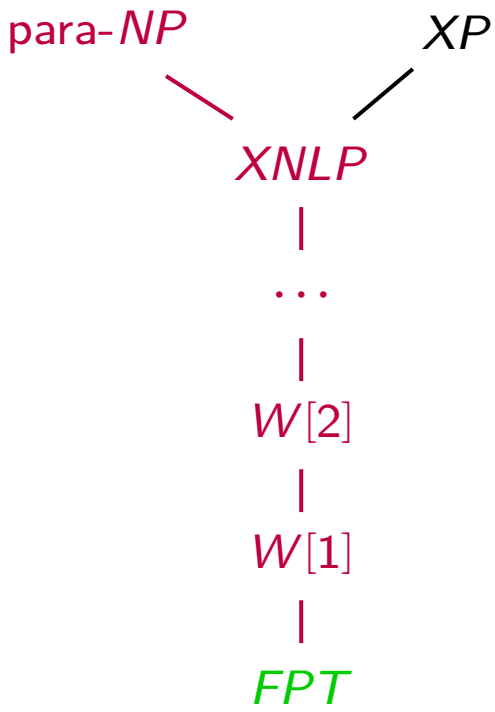
Examples:

- COLORING is NP-complete with $k = 3$ colors.
- Scheduling with m machines is NP-complete when $m = 2$.

Parameterized complexity classes

nondet. time $f(k) \cdot poly(|\mathcal{I}|)$

det. time $|\mathcal{I}|^{f(k)}$



det. time $f(k) \cdot poly(|\mathcal{I}|)$

FPT reductions

Definition: FPT reduction

Decision problem (P, k) fpt-reduces to decision problem (P', k') when there is a computable function g such that for each instance I of (P, k), one can build an instance I' of (P', k') in time $f(k) \cdot poly(|I|)$ such that:

- I is a YES instance if and only if I' is a YES instance,
- if k_I is the value of parameter k in I, then the value of parameter k' in I' is bounded by $g(k_I)$.

FPT reductions

Definition: FPT reduction

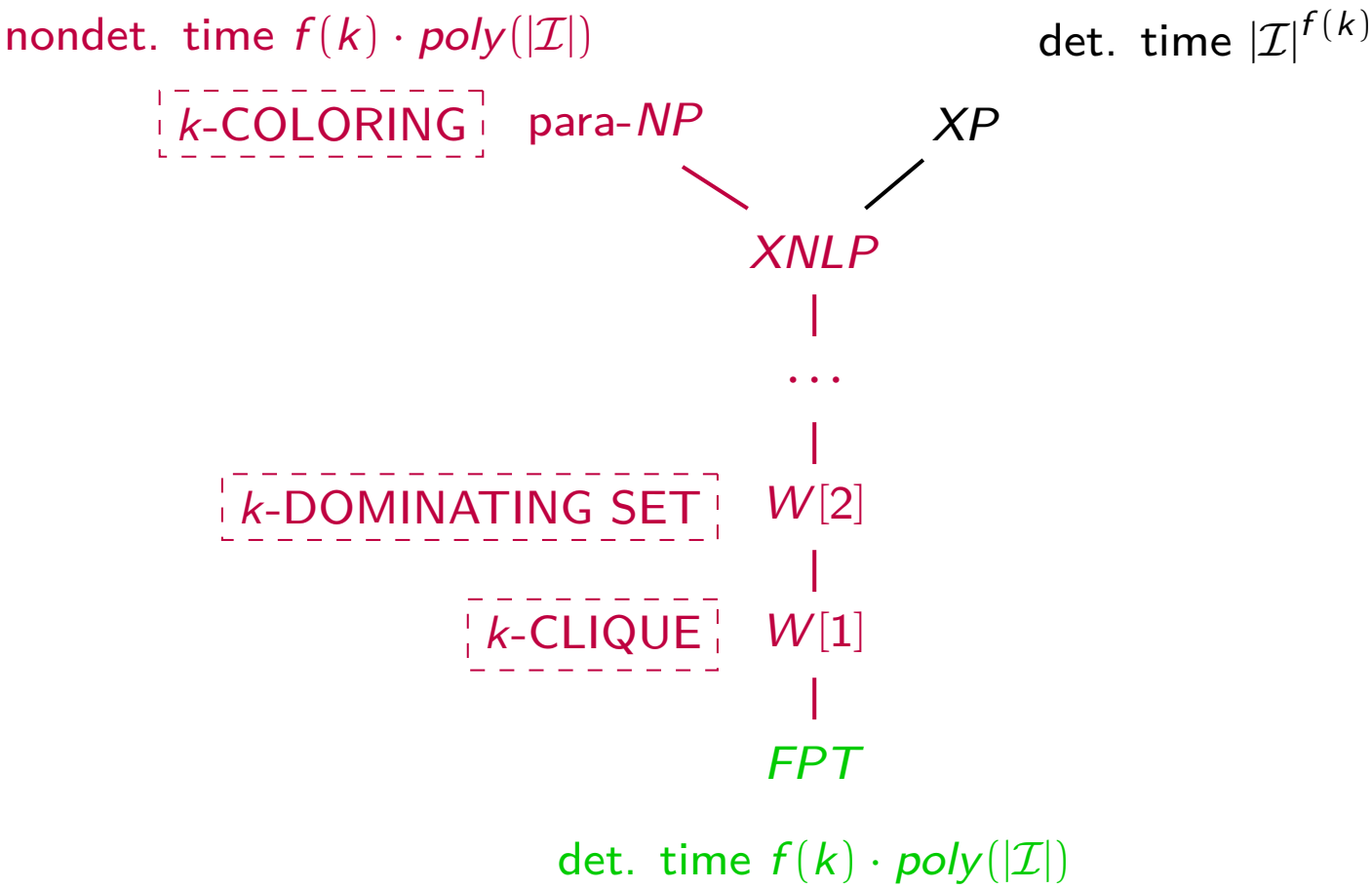
Decision problem (P, k) fpt-reduces to decision problem (P', k') when there is a computable function g such that for each instance I of (P, k), one can build an instance I' of (P', k') in time $f(k) \cdot poly(|I|)$ such that:

- I is a YES instance if and only if I' is a YES instance,
- if k_I is the value of parameter k in I, then the value of parameter k' in I' is bounded by $g(k_I)$.

Definition: Polynomial reduction

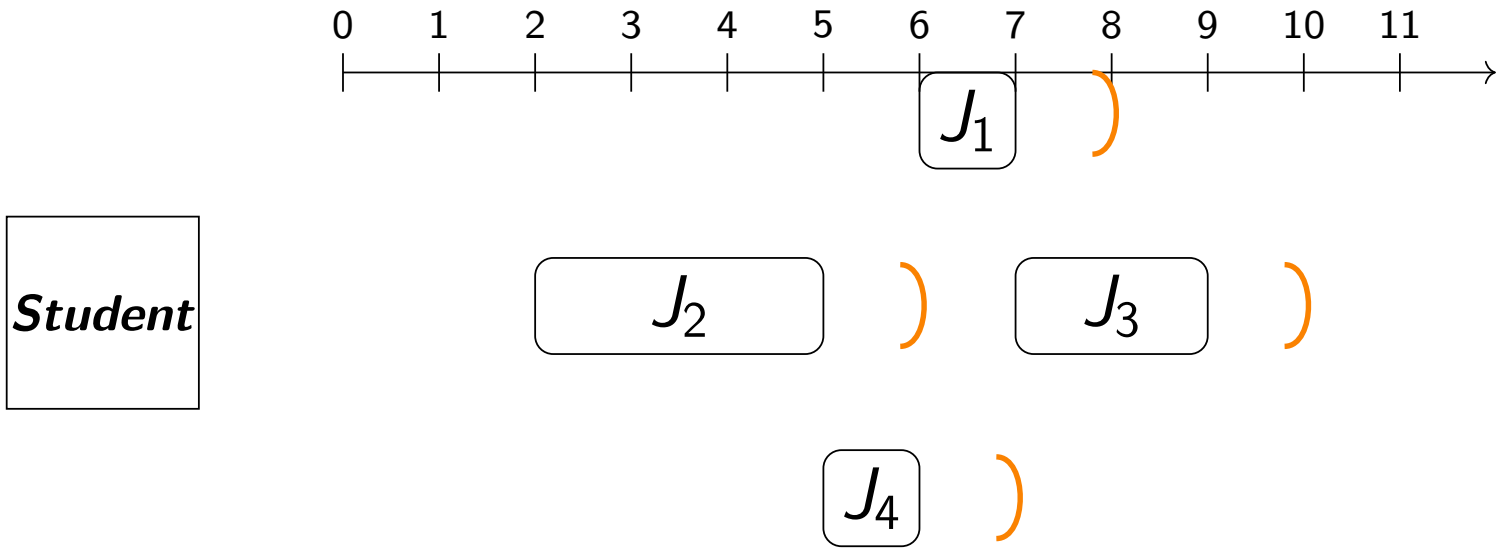
Decision problem P polynomially reduces to decision problem P' when for each instance I of P, one can build an instance I' of P' in time $poly(|I|)$ such that I is a YES instance if and only if I' is a YES instance.

Parameterized complexity classes

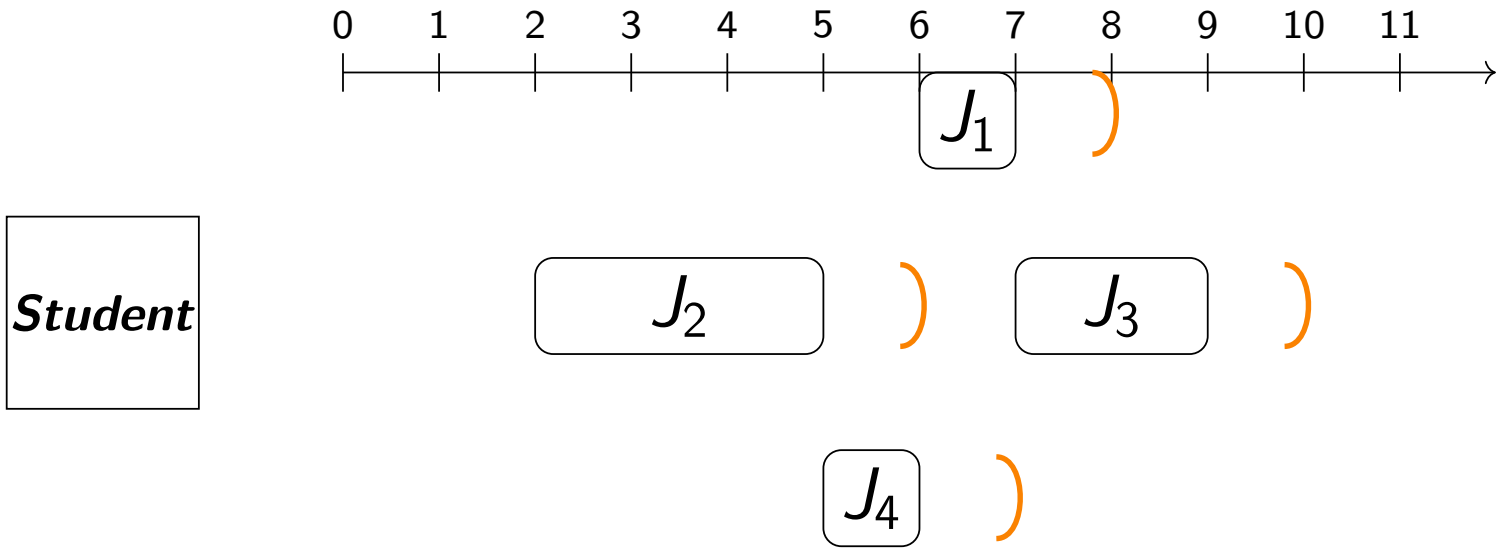


Outline

Adding job time windows $[r_j, d_j)$

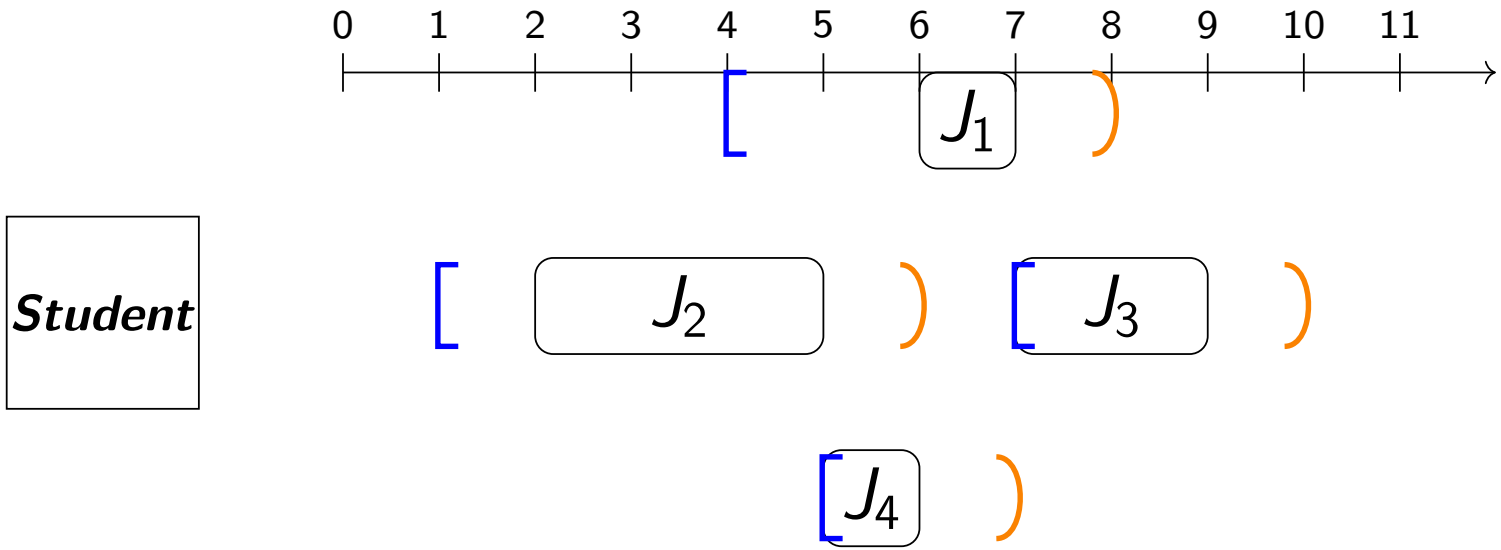


Adding job time windows $[r_j, d_j)$



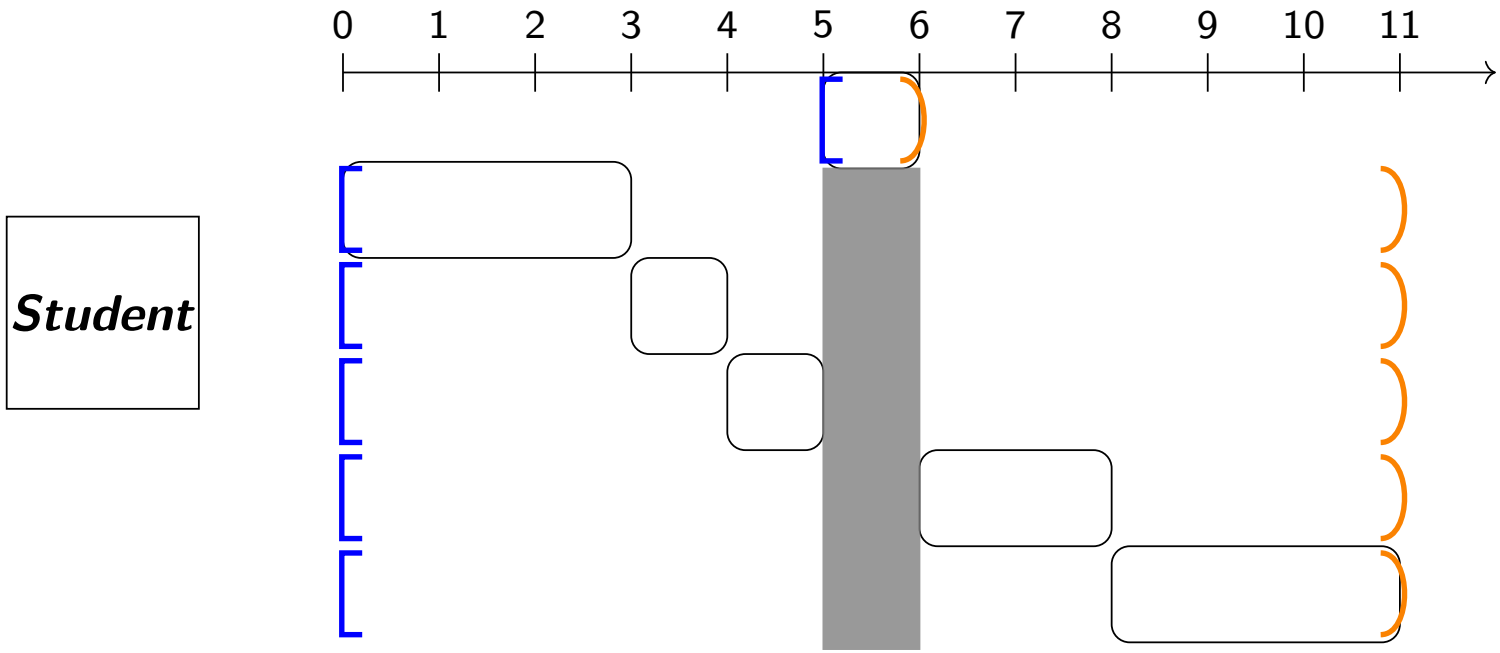
Time $\mathcal{O}(n \cdot log(n))$.

Adding job time windows $[r_j, d_j)$



Strongly NP-hard.

Adding job time windows $[r_j, d_j)$



Strongly NP-hard.

Quiz time!

Proposed parameters:

- Height μ = maximum number of time windows overlapping at any time.
- Slack $\sigma = \max_{J_j \in \mathcal{J}}(d_j - r_j - p_j)$.

Quiz time!

Proposed parameters:

- Height μ = maximum number of time windows overlapping at any time.
- Slack $\sigma = \max_{J_j \in \mathcal{J}}(d_j - r_j - p_j)$.

	Height μ	Slack σ
One machine with time windows		
Two machines with time windows		

Quiz time!

Proposed parameters:

- Height μ = maximum number of time windows overlapping at any time.
- Slack $\sigma = \max_{J_j \in \mathcal{J}}(d_j - r_j - p_j)$.

	Height μ	Slack σ
One machine with time windows	FPT	
Two machines with time windows		

Quiz time!

Proposed parameters:

- Height μ = maximum number of time windows overlapping at any time.
- Slack $\sigma = \max_{J_j \in \mathcal{J}}(d_j - r_j - p_j)$.

	Height μ	Slack σ
One machine with time windows	FPT	FPT
Two machines with time windows		

Quiz time!

Proposed parameters:

- Height μ = maximum number of time windows overlapping at any time.
- Slack $\sigma = \max_{J_j \in \mathcal{J}}(d_j - r_j - p_j)$.

	Height μ	Slack σ
One machine with time windows	FPT	FPT
Two machines with time windows	para-NP-hard	

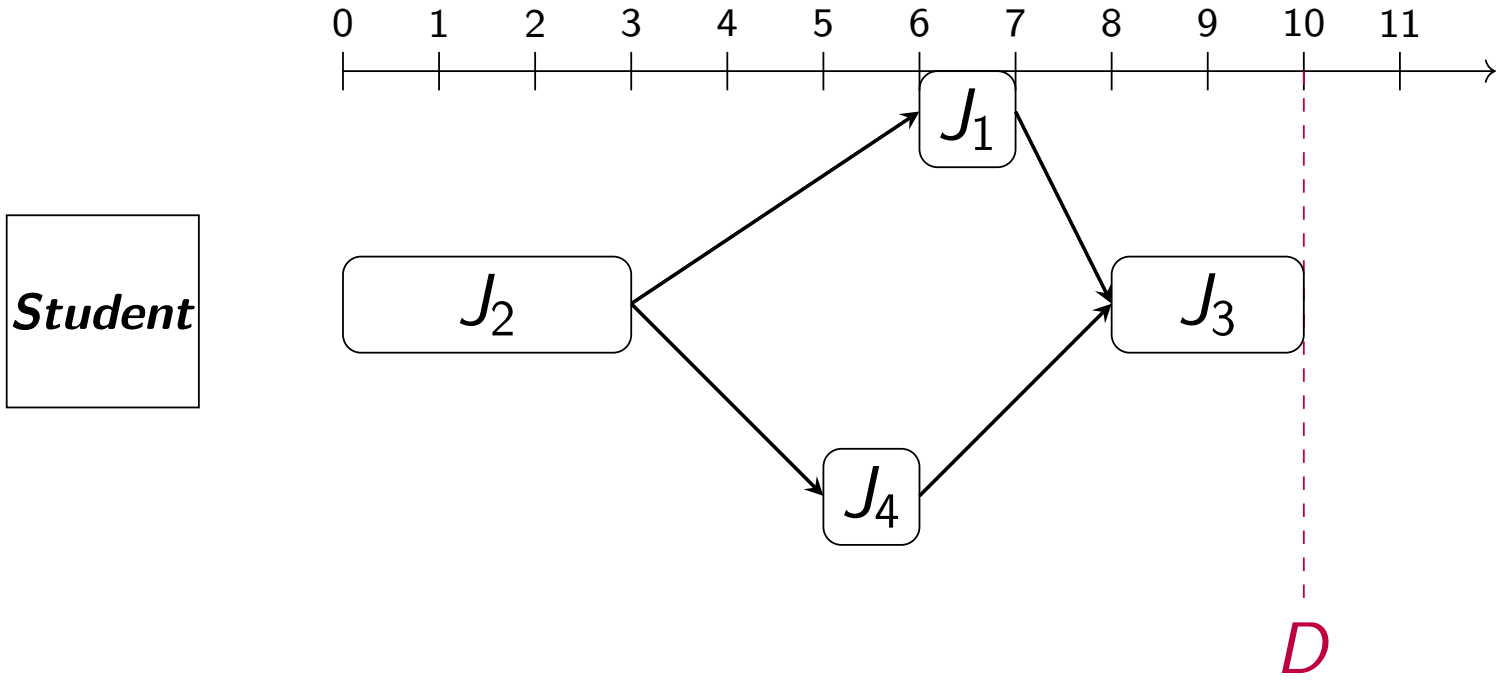
Quiz time!

Proposed parameters:

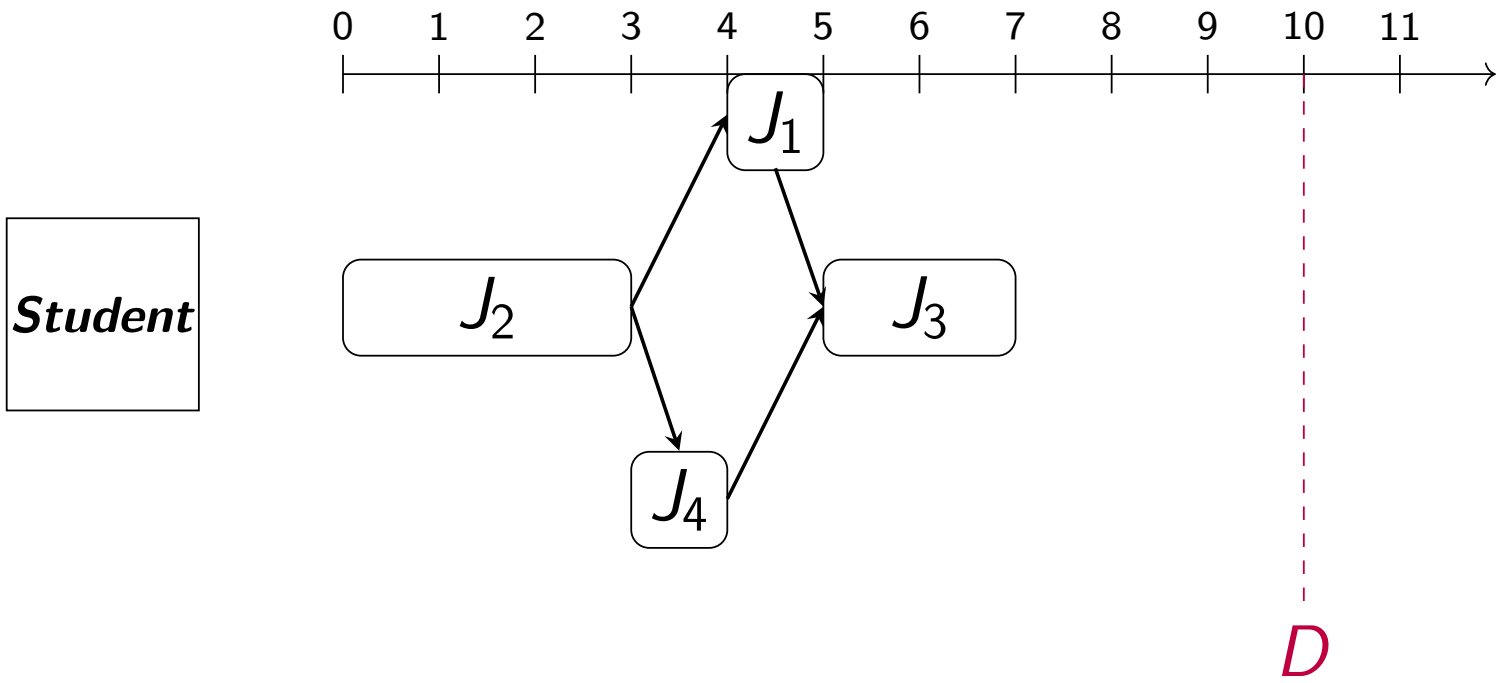
- Height μ = maximum number of time windows overlapping at any time.
- Slack $\sigma = \max_{J_j \in \mathcal{J}}(d_j - r_j - p_j)$.

	Height μ	Slack σ
One machine with time windows	FPT	FPT
Two machines with time windows	para-NP-hard	FPT

Adding precedence delays $\ell_{i,j}$

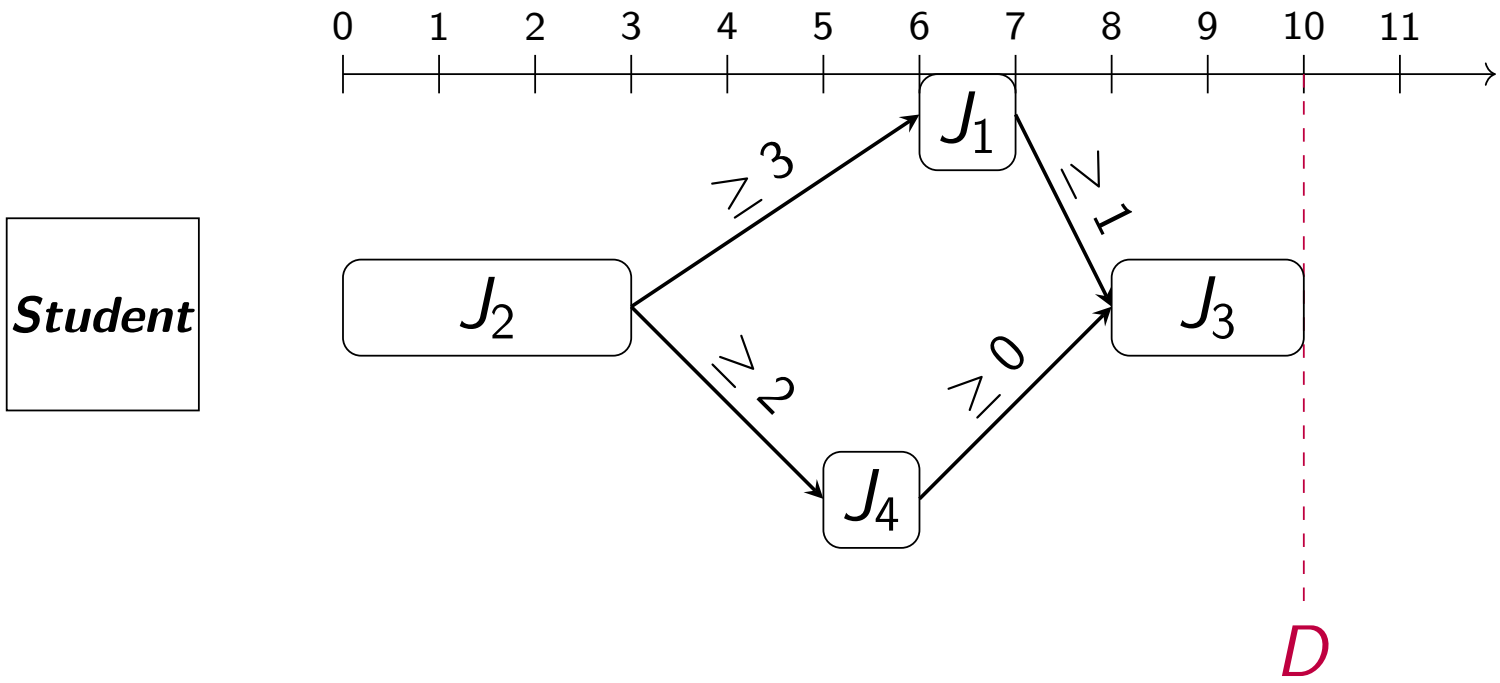


Adding precedence delays $\ell_{i,j}$



Time $\mathcal{O}(n)$.

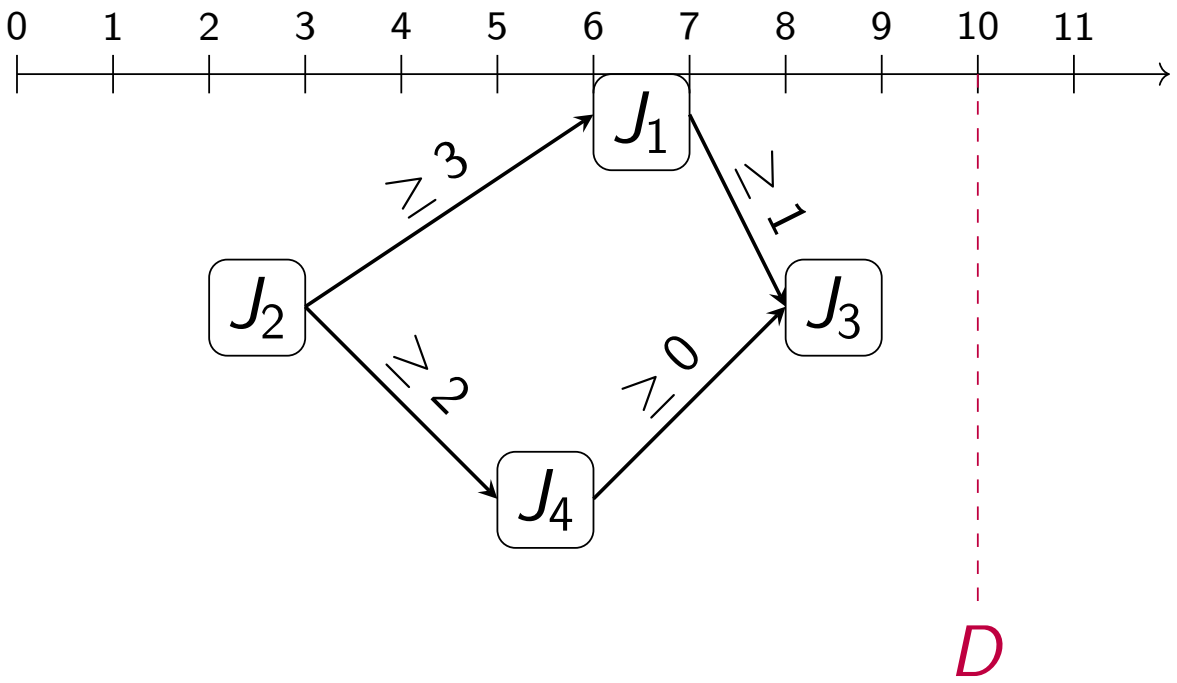
Adding precedence delays $\ell_{i,j}$



Strongly NP-hard.

Adding precedence delays $\ell_{i,j}$

Student



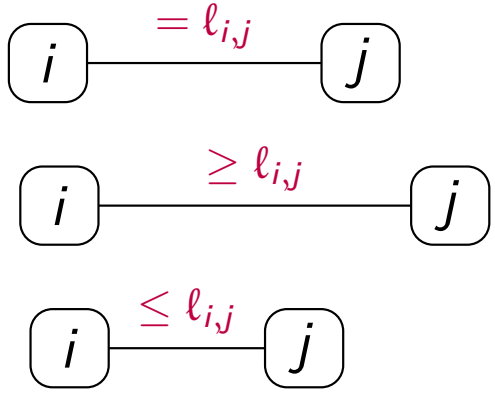
Strongly NP-hard.

Precedence delay types

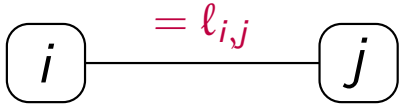
- Three precedence delay types studied: exact, minimum and maximum.

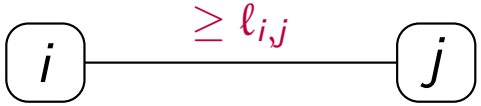
Precedence delay types

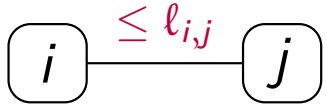
- Three precedence delay types studied: exact, minimum and maximum.
- If τ is a feasible schedule and $(i \xrightarrow{\ell_{i,j}} j)$:
  - Exact delay:
  - Minimum delay:
  - Maximum delay:



Precedence delay types

- Three precedence delay types studied: exact, minimum and maximum.
- If τ is a feasible schedule and $(i \xrightarrow{\ell_{i,j}} j)$:
  - Exact delay: 

  - Minimum delay: 

  - Maximum delay: 

Remark

- One machine with precedence chains and exact delays: UNARY BIN PACKING.
- One machine with maximum precedence delays: variant of BANDWIDTH.

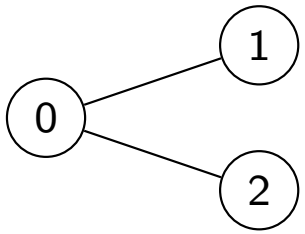
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:

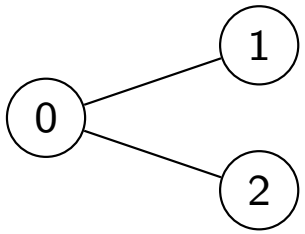


$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.

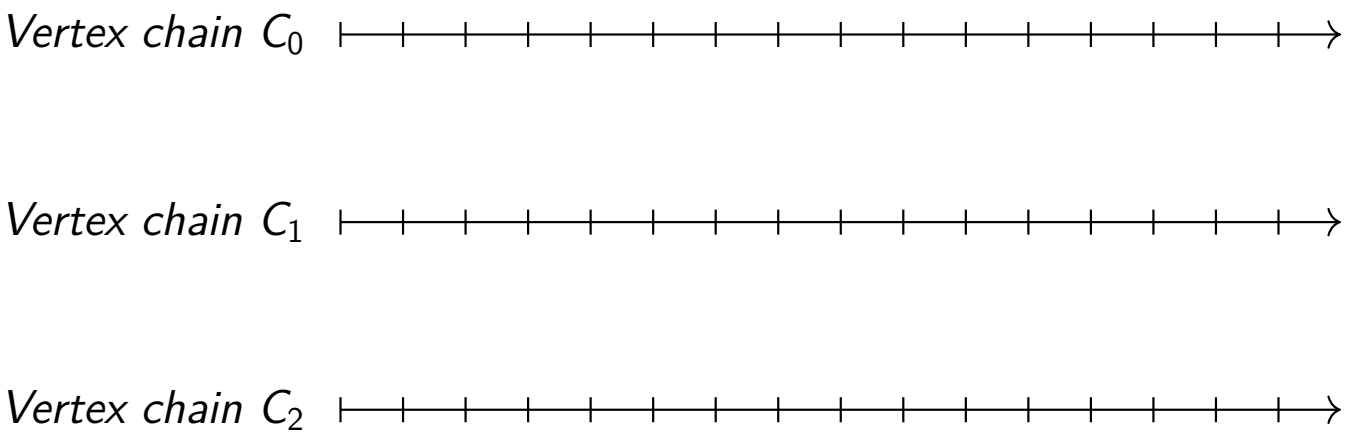
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



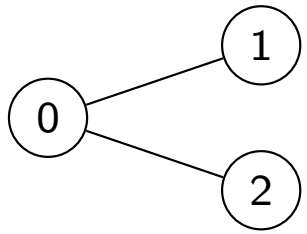
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



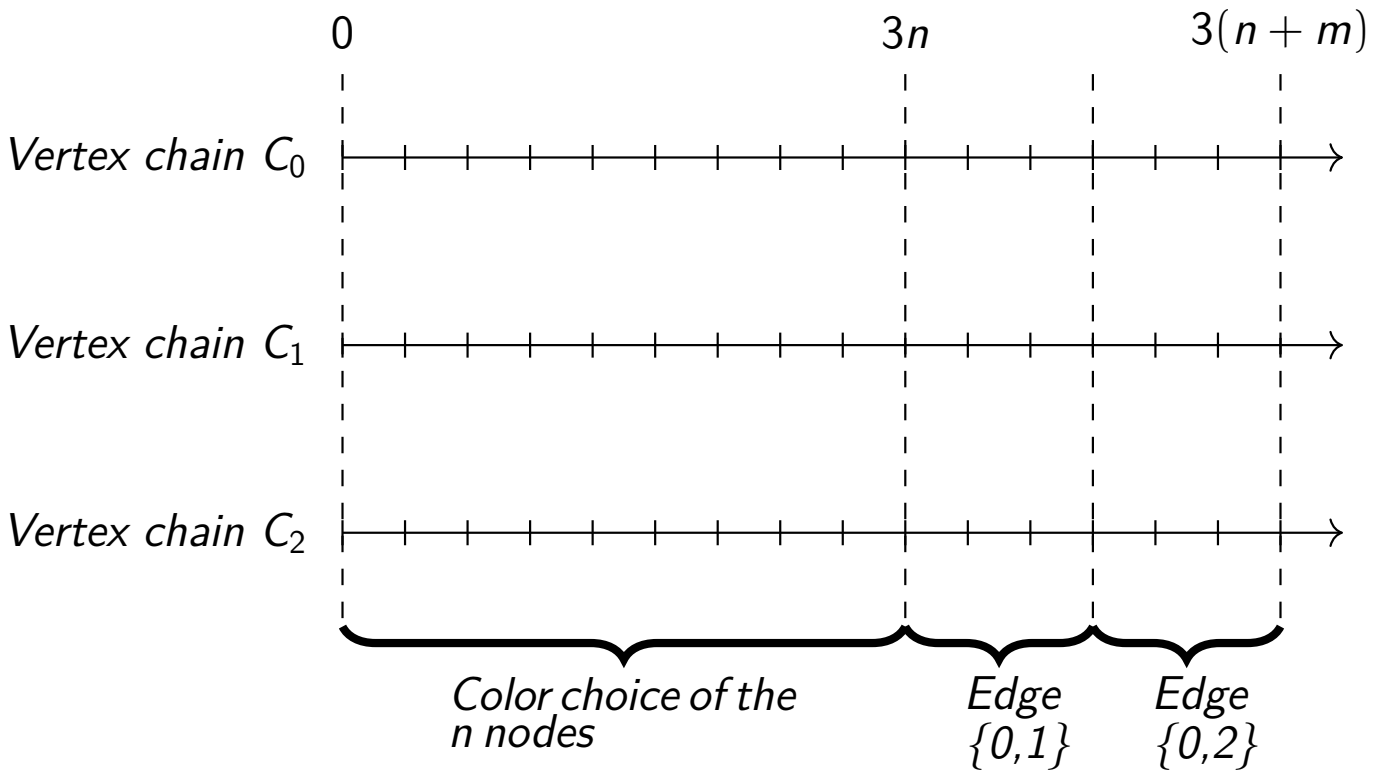
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



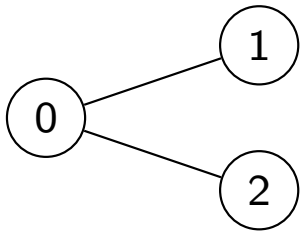
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



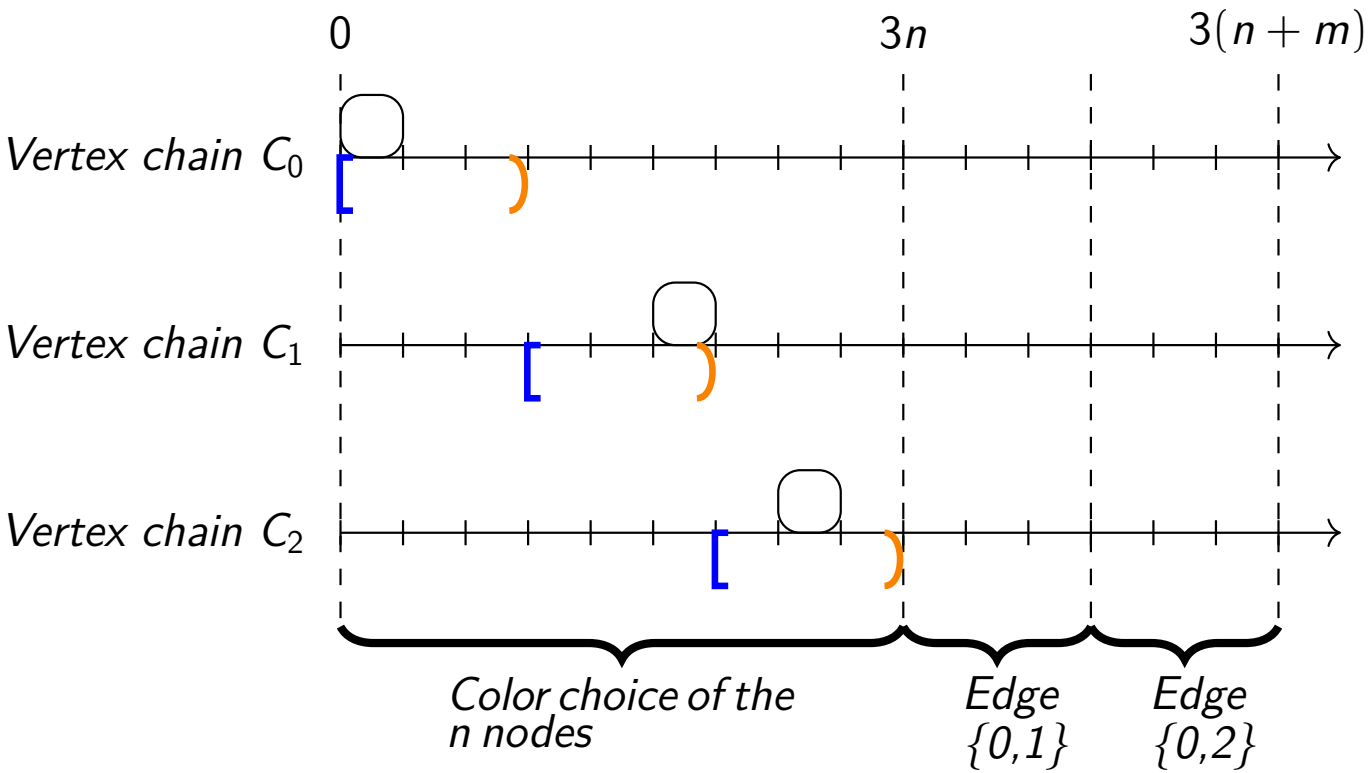
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



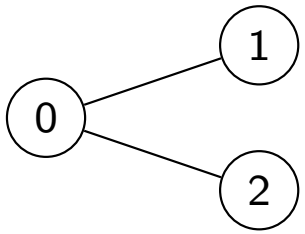
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



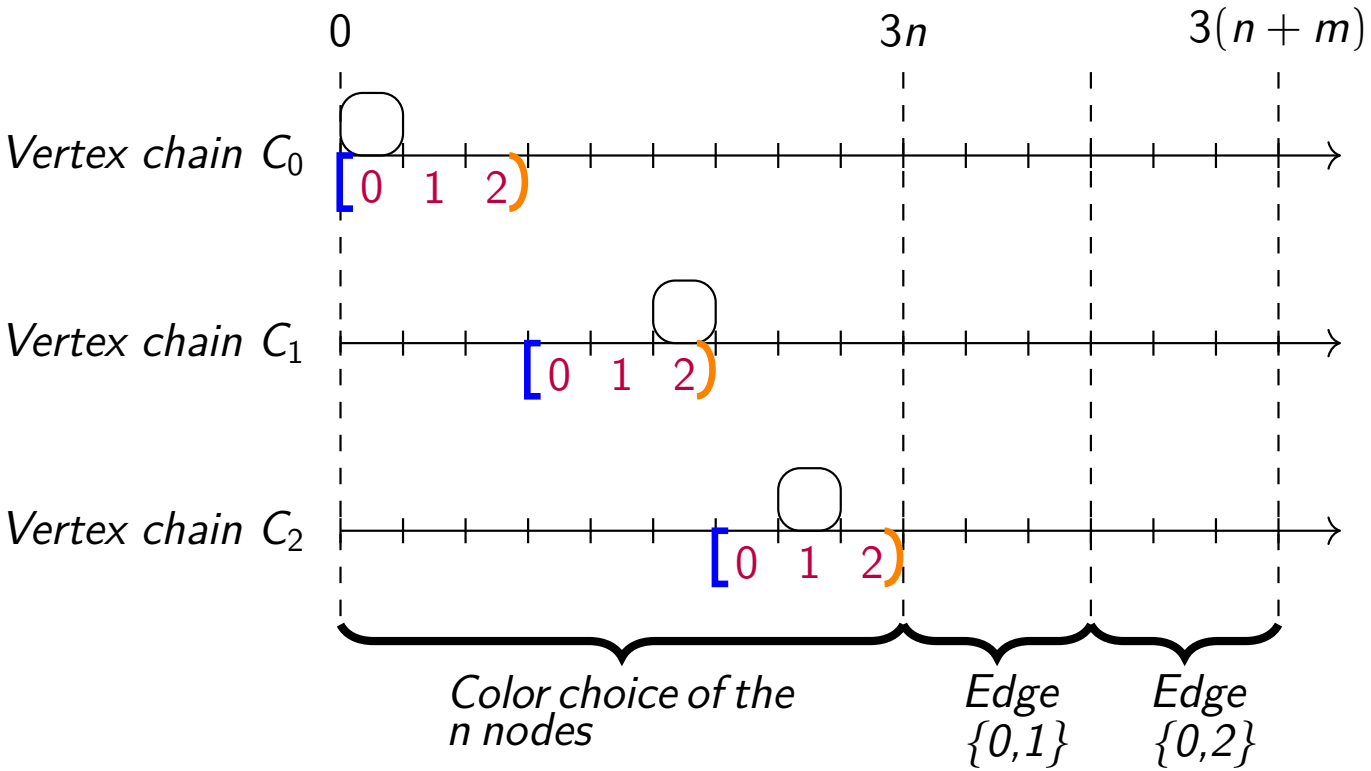
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



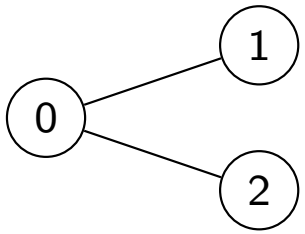
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



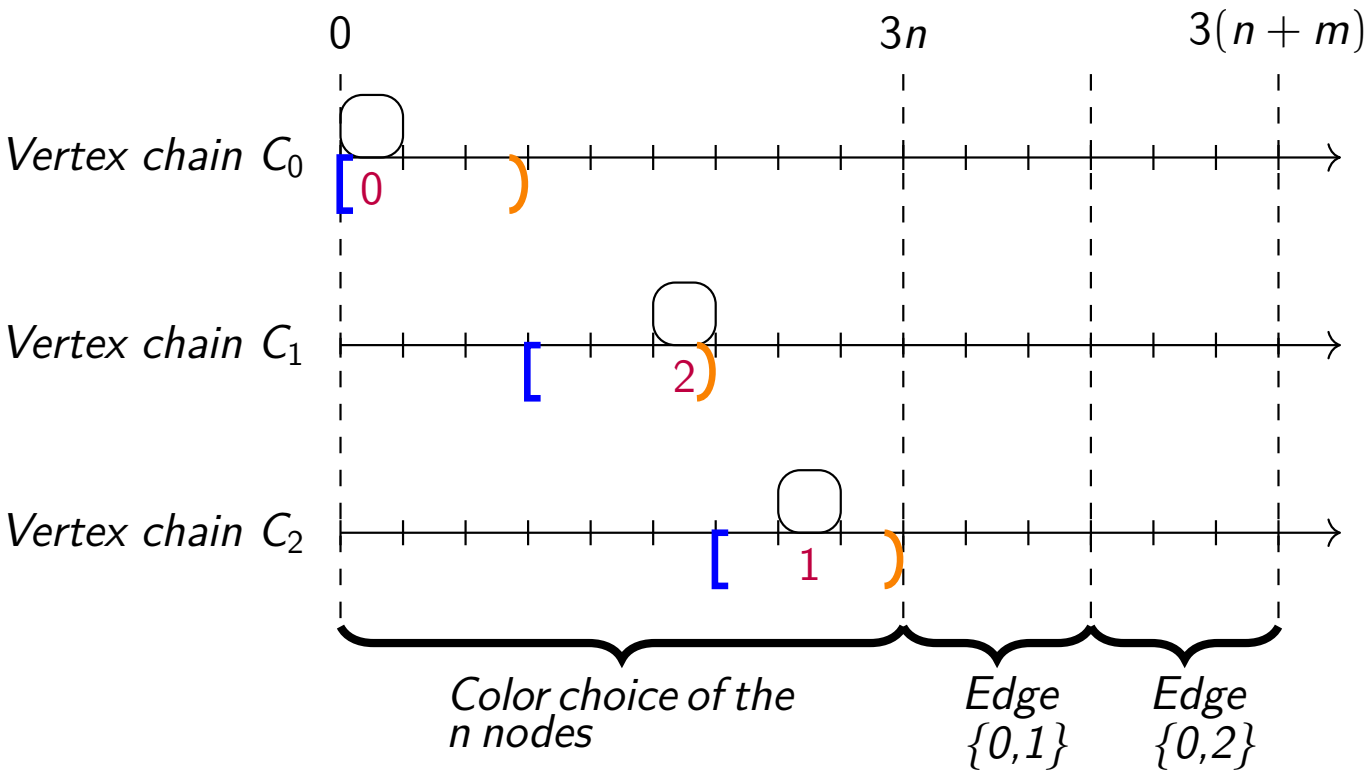
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



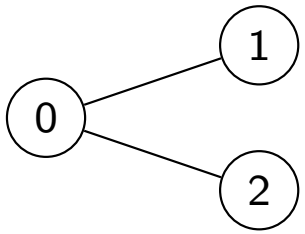
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



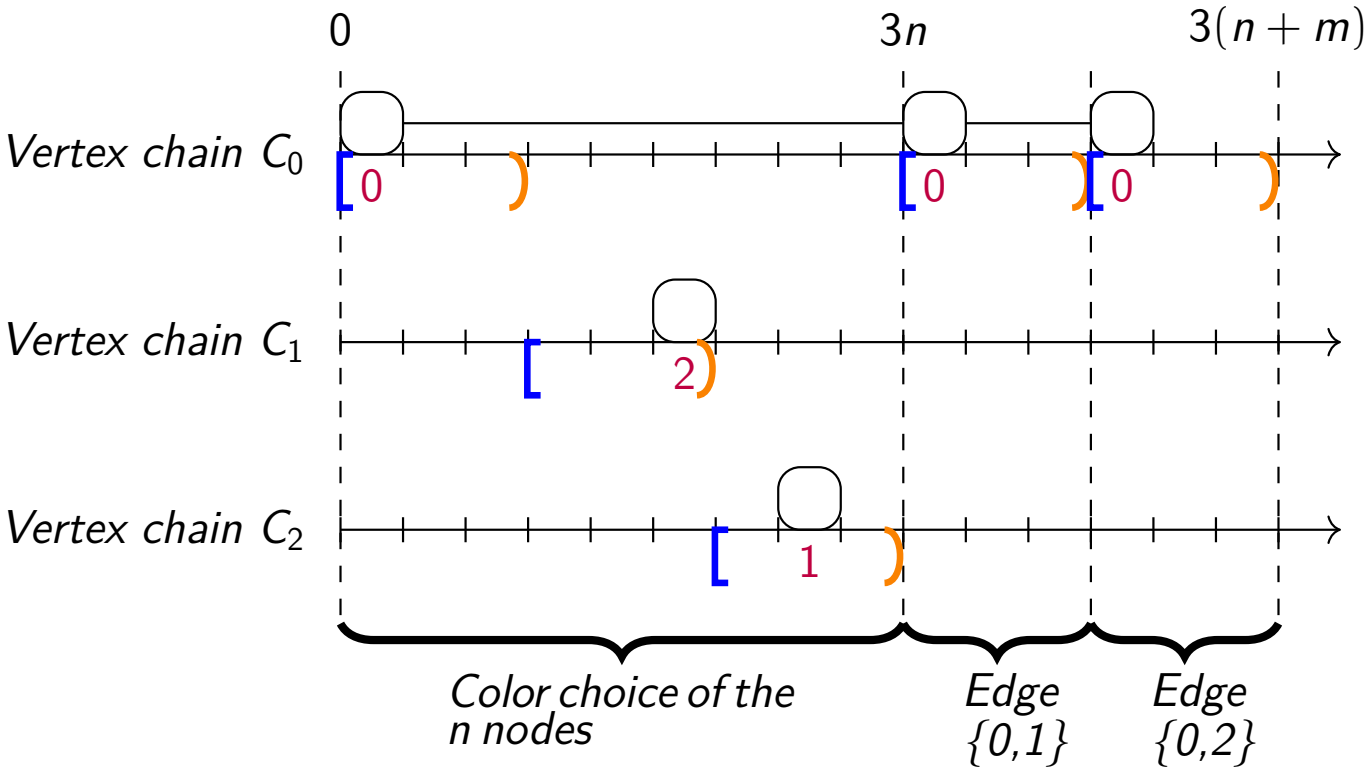
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



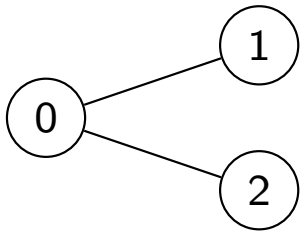
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



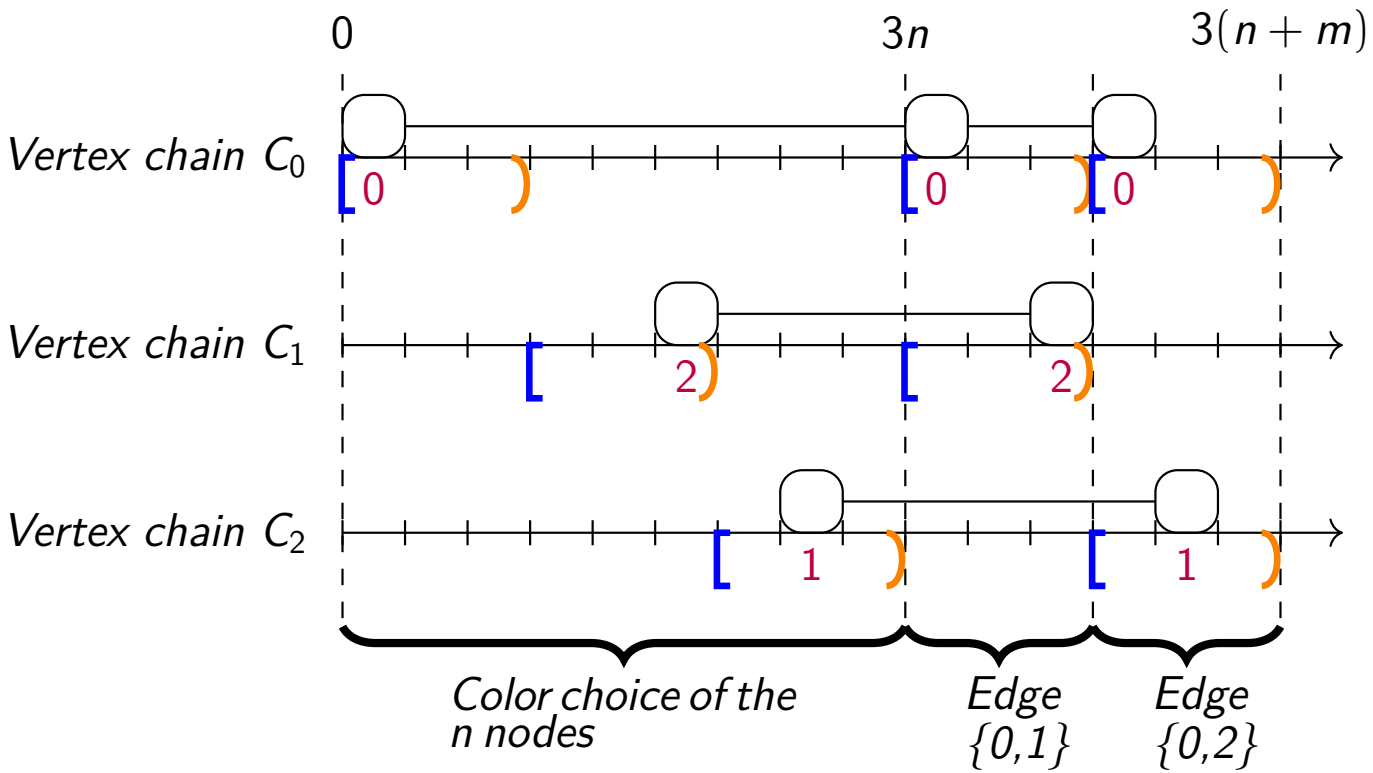
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



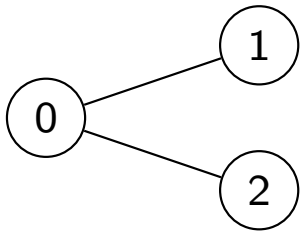
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



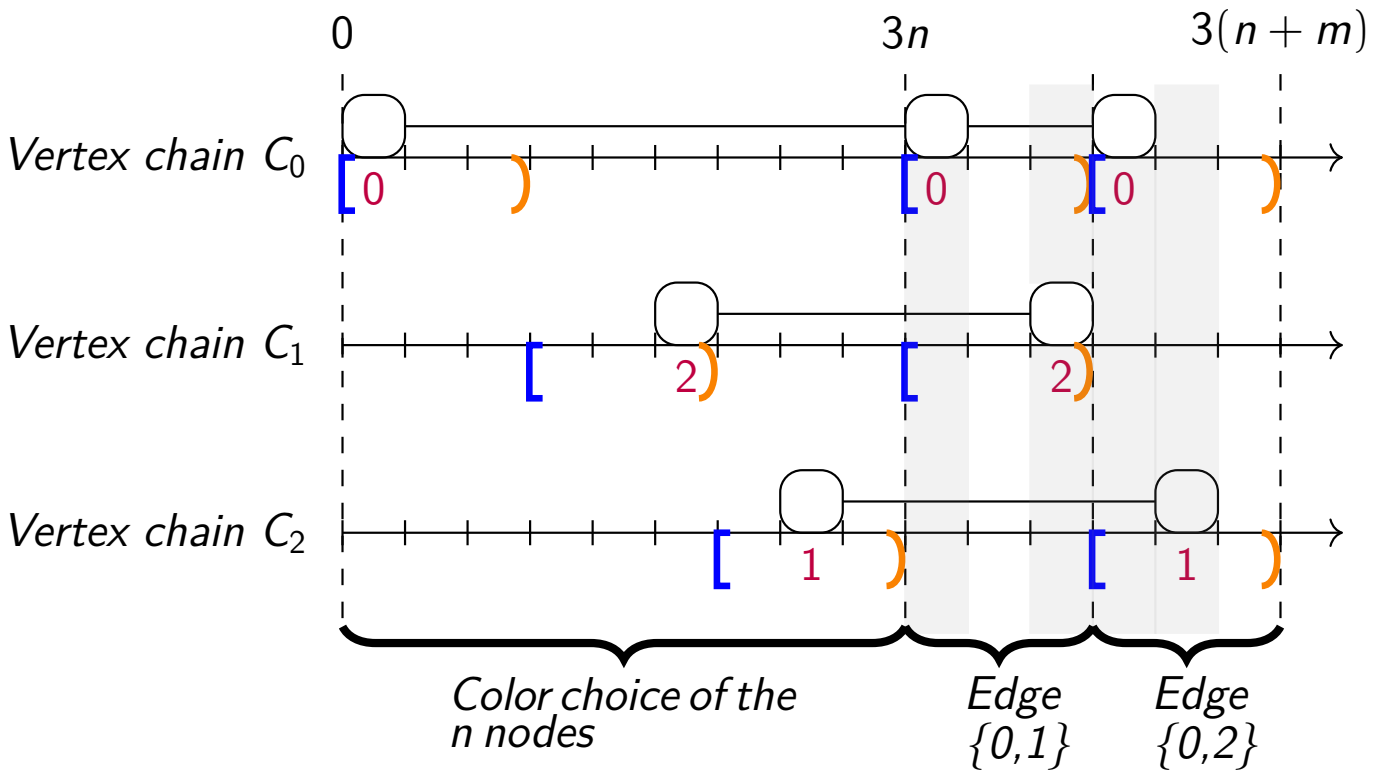
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:



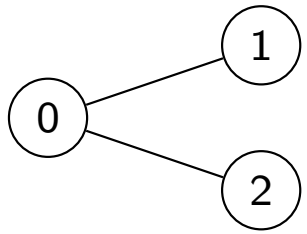
$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.



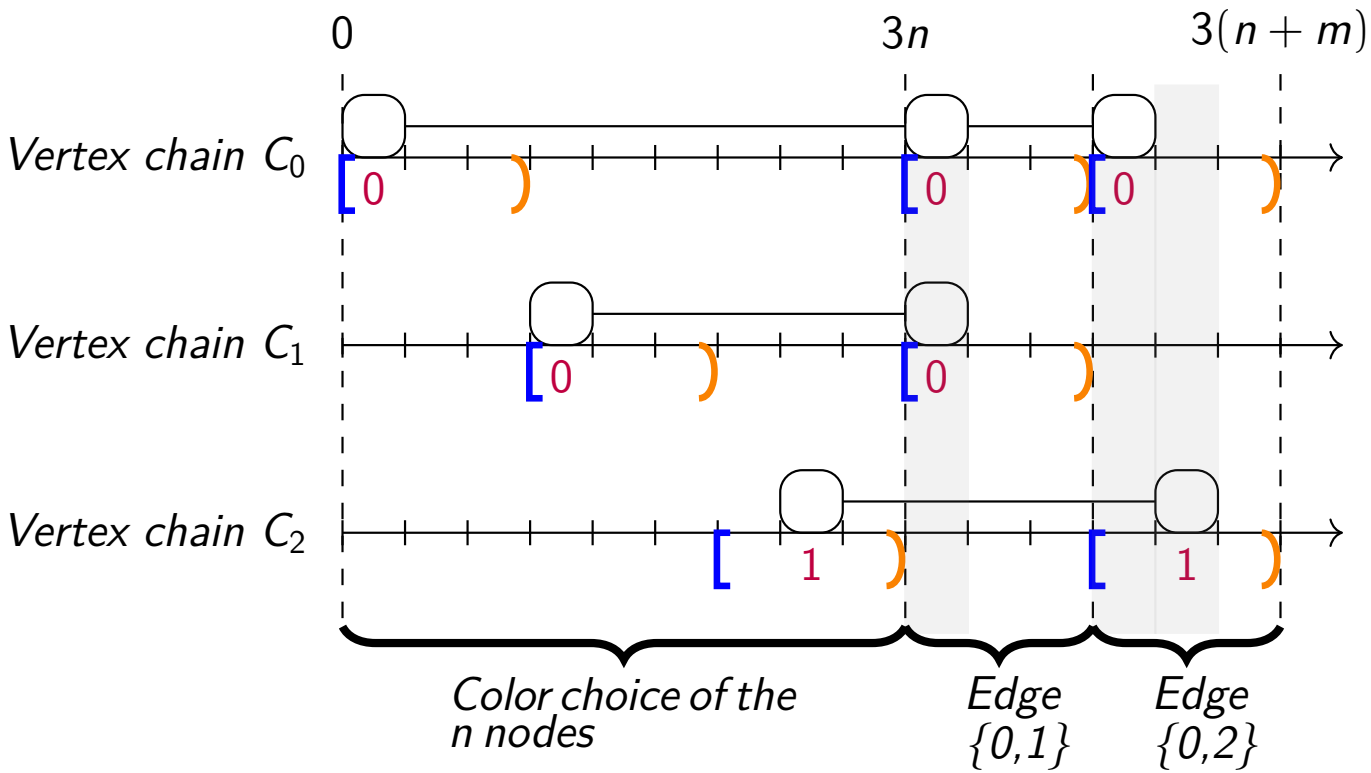
One machine with precedence delays is NP-hard with fixed height μ and slack σ.

- Reduction from 3-COLORING: $G = (V, E), n = |V|, m = |E|$

Example:

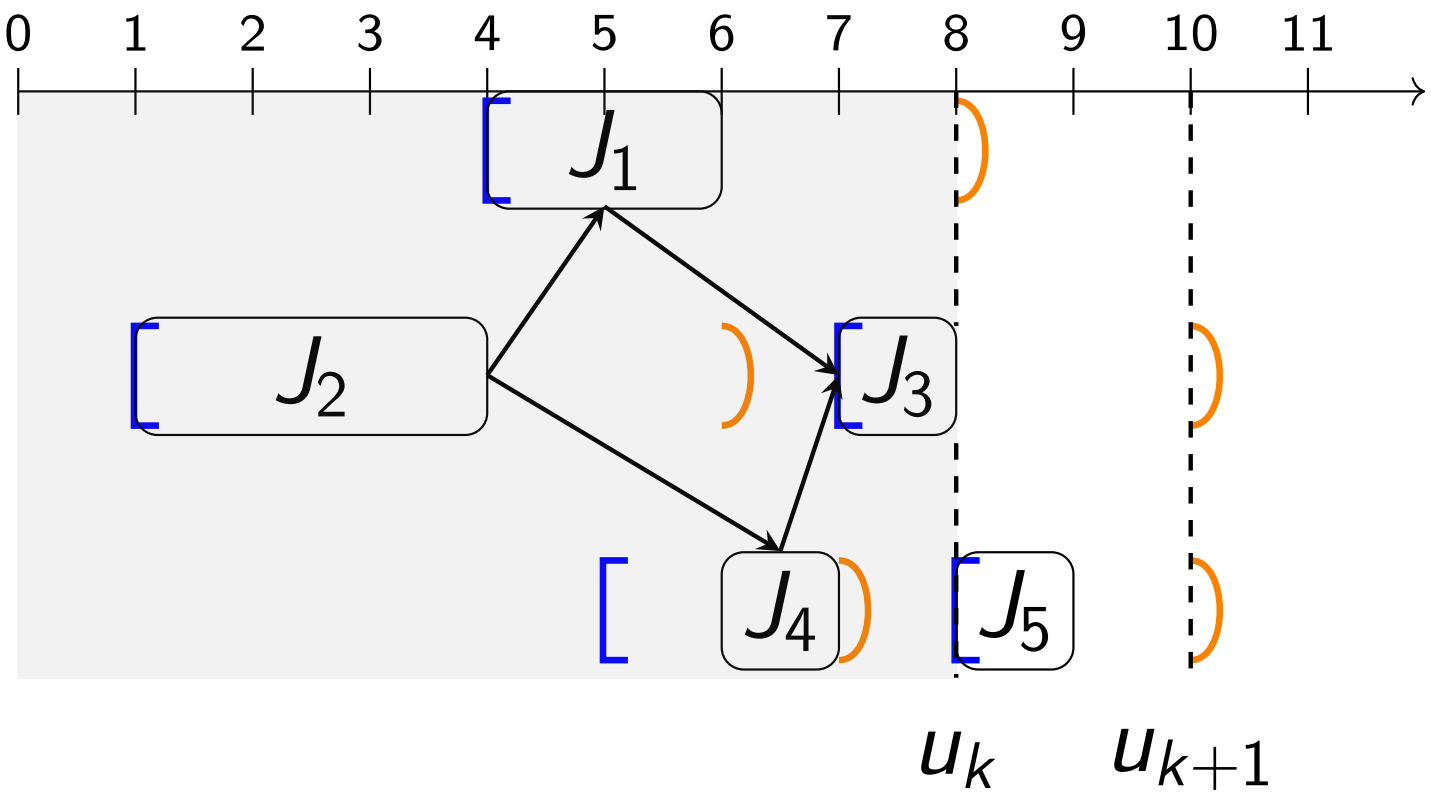


$\longrightarrow$ Scheduling instance with $3n + 2m$ jobs and exact delays.

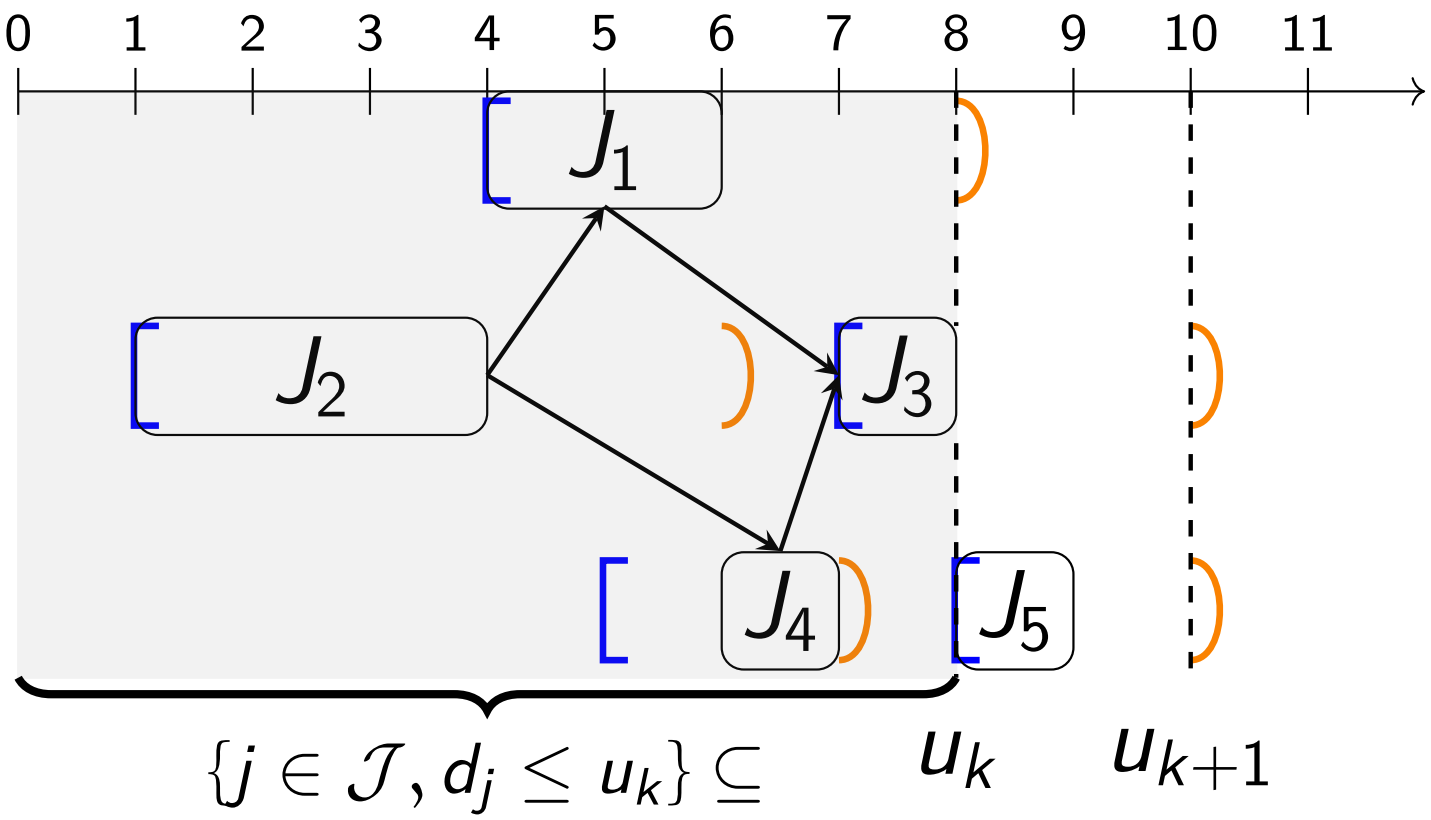


Outline

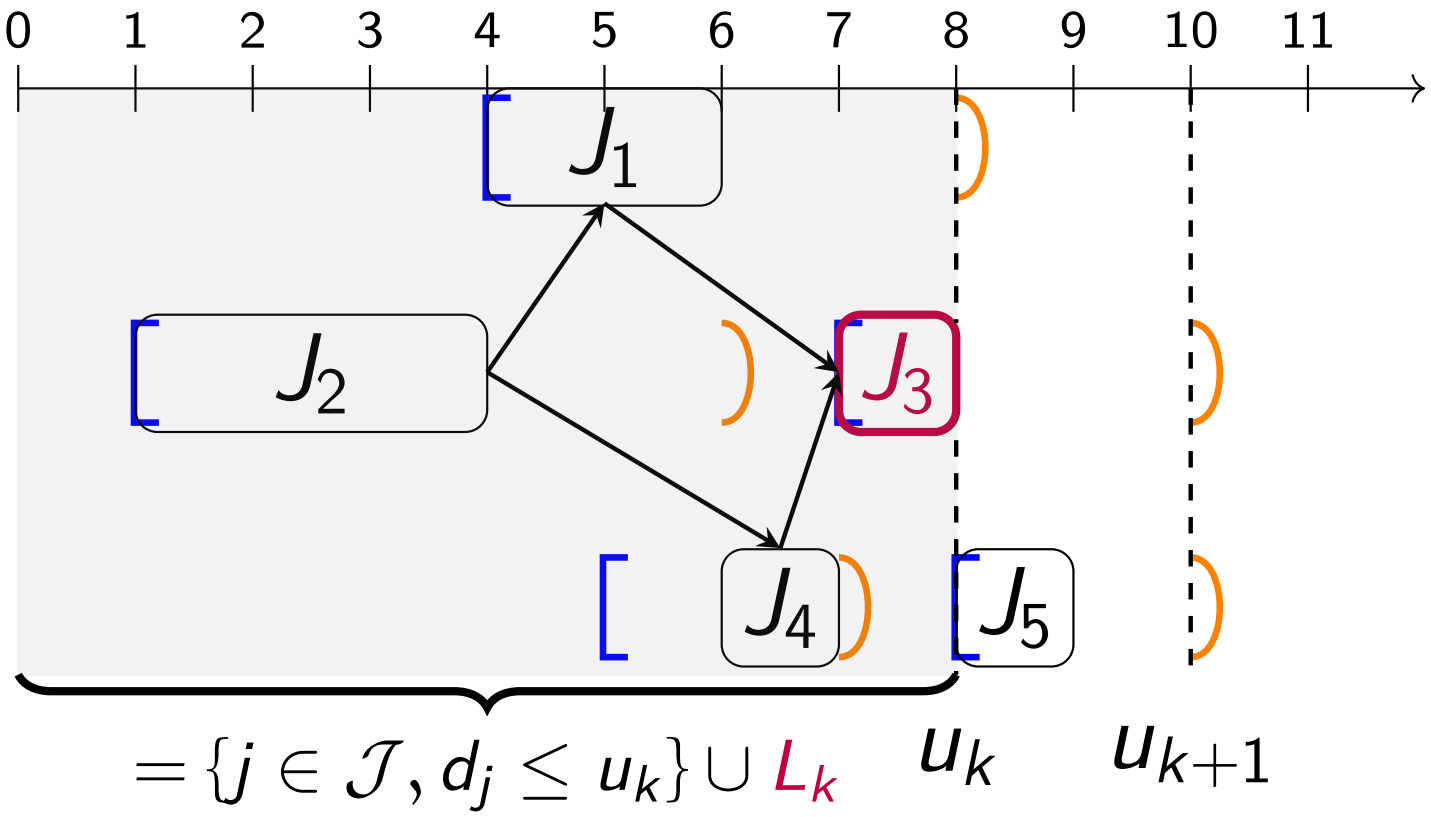
Dynamic programming with height μ



Dynamic programming with height μ

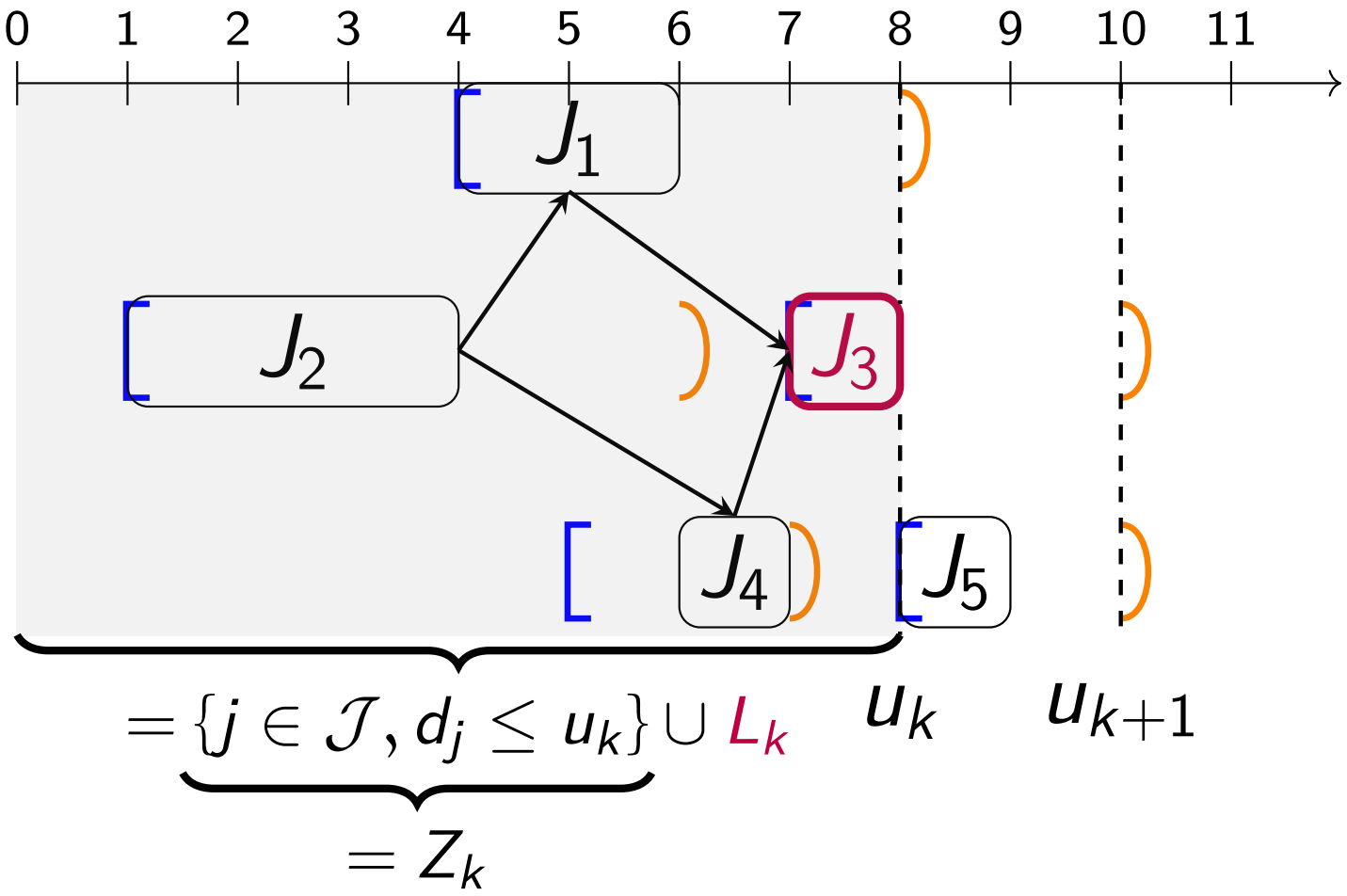


Dynamic programming with height μ



- $J_j \in L_k \implies [u_k, u_{k+1}) \subseteq [r_j, d_j)$: at most μ such jobs J_j.

Dynamic programming with height μ



- $J_j \in L_k \implies [u_k, u_{k+1}) \subseteq [r_j, d_j)$: at most μ such jobs J_j.

Dynamic programming with height μ (cont.)

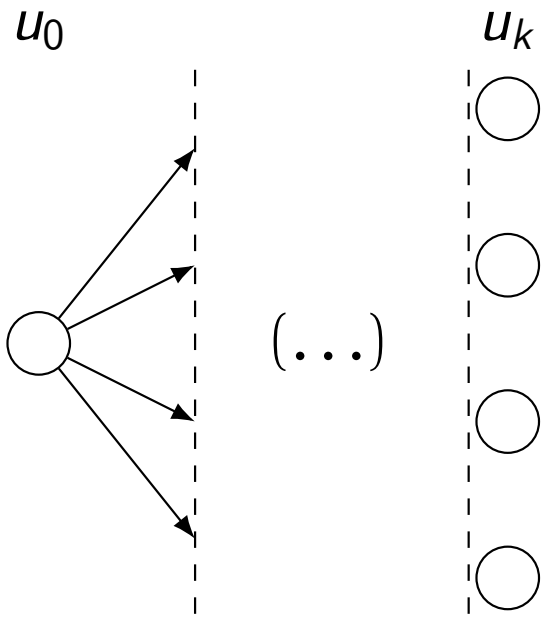
Dynamic programming state (k, L_k)

- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.

Dynamic programming with height μ (cont.)

Dynamic programming state (k, L_k)

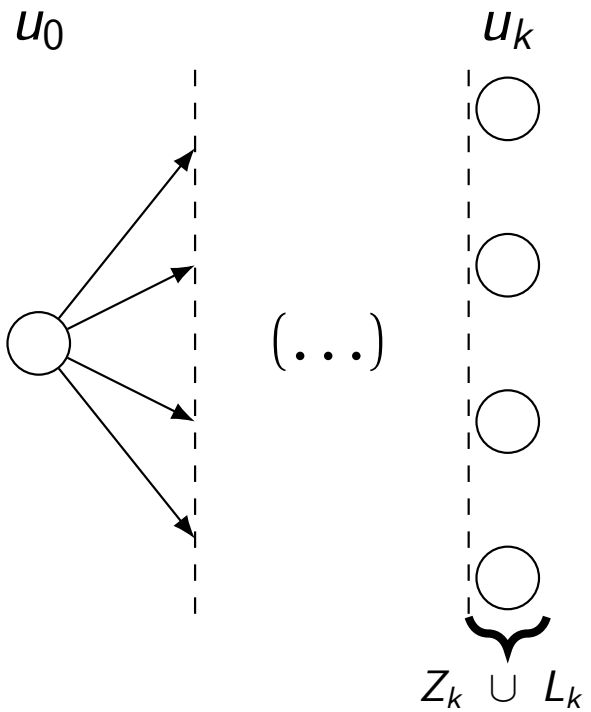
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with height μ (cont.)

Dynamic programming state (k, L_k)

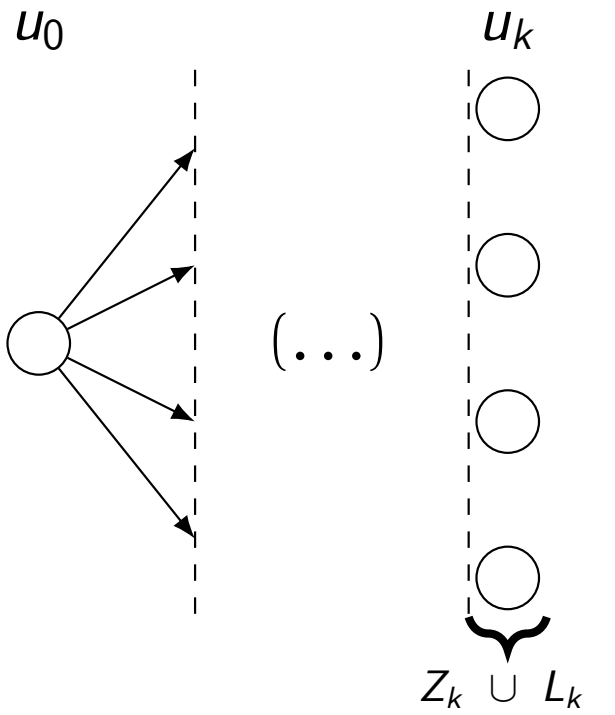
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with height μ (cont.)

Dynamic programming state (k, L_k)

- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.

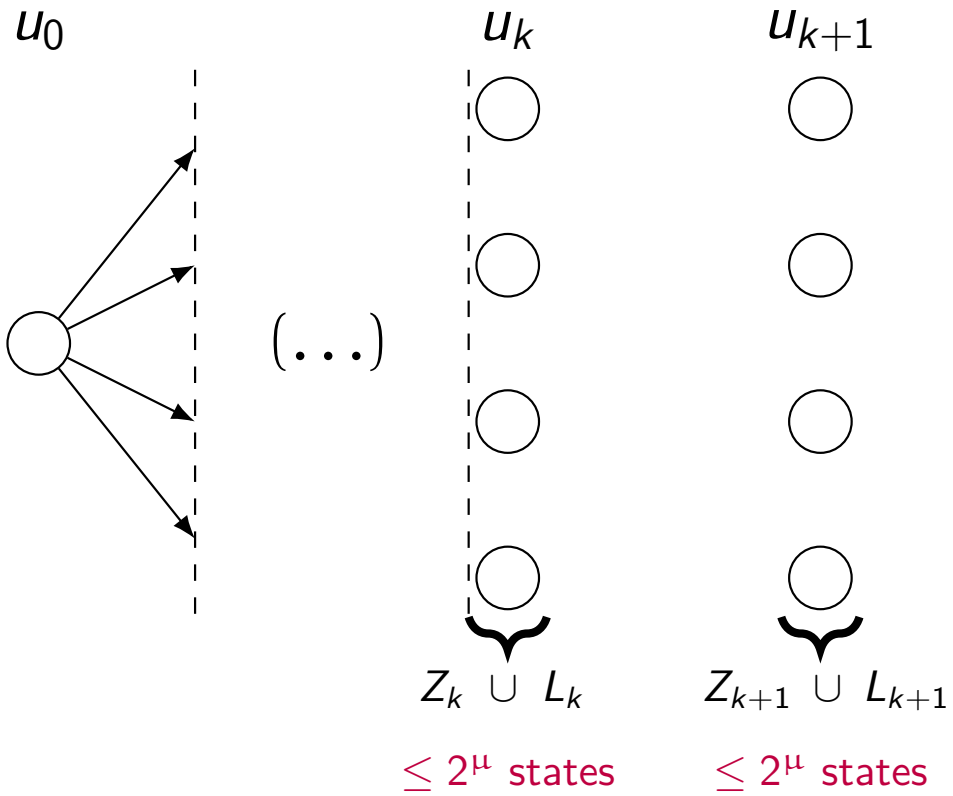


$\leq 2^{\mu}$ states

Dynamic programming with height μ (cont.)

Dynamic programming state (k, L_k)

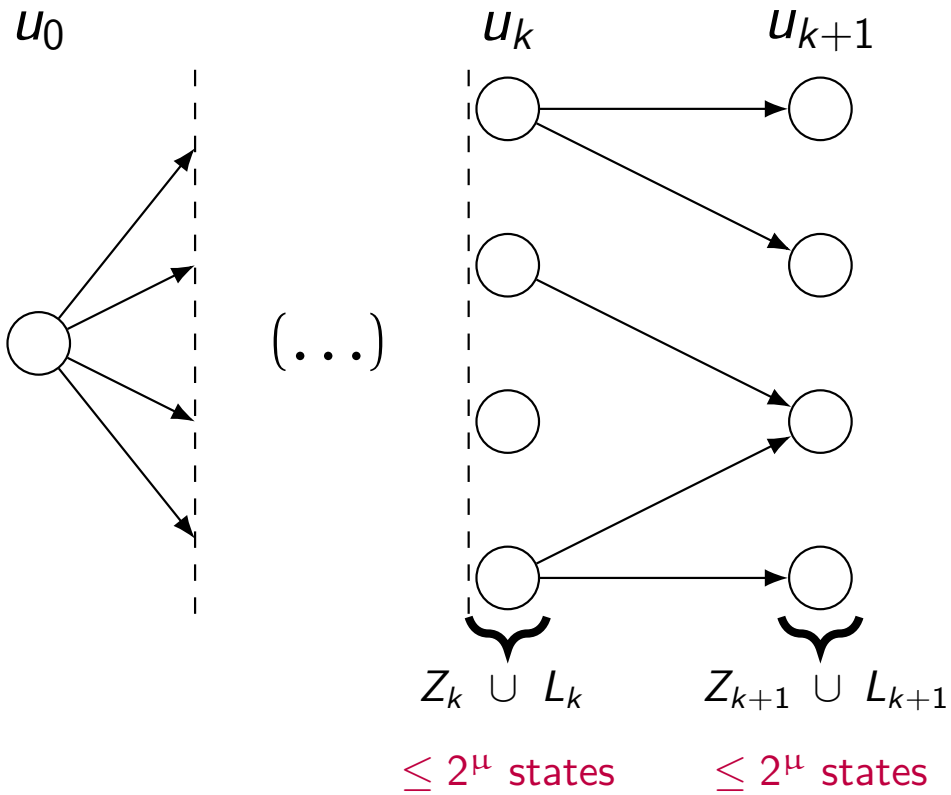
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with height μ (cont.)

Dynamic programming state (k, L_k)

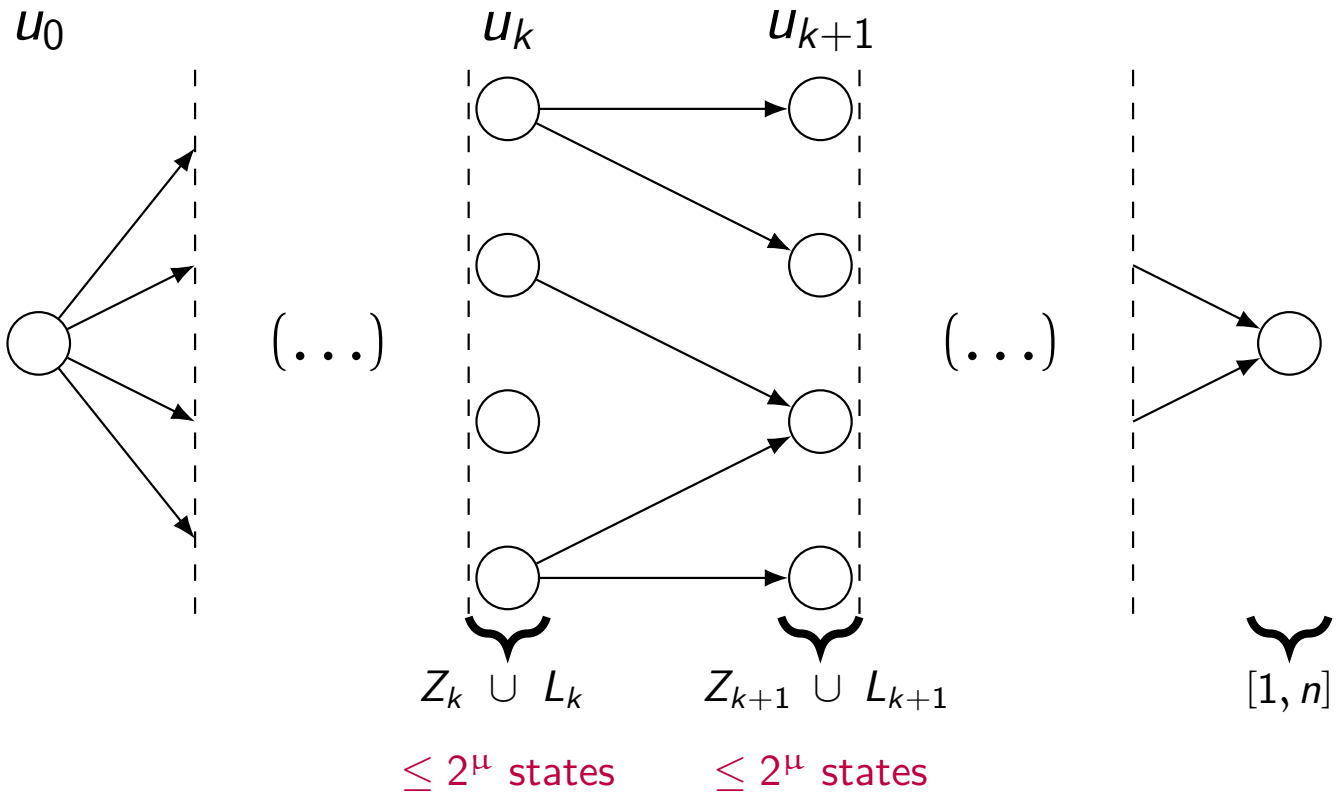
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with height μ (cont.)

Dynamic programming state (k, L_k)

- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before time u_k.
- State value = minimum makespan among the associated partial schedules.



Algorithm 1: main()

▷ Solving $1|r_j, d_j|C_{max}$.

preprocessing() ▷ Compute sequence $(u_k)_{1 \leq k \leq K}$. Compute associated job subsets.

$u_{K+1} \leftarrow \infty$

Create table T with K columns (1 to K) and 2^{μ} slots in each column k.

Initialize T with ∞ everywhere.

$T[1, \emptyset] \leftarrow 0$

for $k = 1$ to $K - 1$ **do**

    for each (L_k, L_{k+1}) **do**

        $T[k+1, L_{k+1}] \leftarrow \min(T[k+1, L_{k+1}], \text{extend}(k, L_k, L_{k+1}))$

return $T[K, \emptyset]$

Algorithm 2: extend(k, L_k, L_{k+1})

1: $Cmax \leftarrow \max(u_k, T[k, L_k])$
2: **if** $Cmax = \infty$ **then return** ∞
▷ Ensure that all jobs in L_k with a deadline higher than u_{k+1} are still in L_{k+1}.
3: **if** $(L_k - Z_{k+1}) \not\subseteq L_{k+1}$ **then return** ∞
▷ Add jobs to the existing partial schedule.
4: $mandatory \leftarrow Z_{k+1} \cap \{J_j \in \mathcal{J}, [u_k, u_{k+1}) \subseteq [r_j, d_j)\}$
5: $add \leftarrow (mandatory \cup L_{k+1}) - L_k$
6: sort(add, nondecreasing d_j)
7: **for** j in add (in order) **do**
8: $Cmax \leftarrow Cmax + p_j$
9: **if** $Cmax > d_j$ **then return** ∞
▷ Ensure that all the added jobs start earlier than u_{k+1}.
10: $last \leftarrow$ last job in add.
11: **if** $Cmax - p_{last} \geq u_{k+1}$ **then return** ∞
12: **return** $Cmax$

Main result

Theorem [Mallem, Hanen, Munier 22]

Single machine scheduling with time windows can be solved in time:

$$\mathcal{O}([(\mu+1)\cdot log(\mu+1)\cdot 4^{\mu}]\cdot n + n^2).$$

Main result

Theorem [Mallem, Hanen, Munier 22]

Single machine scheduling with time windows can be solved in time:

$$\mathcal{O}([(\mu+1)\cdot log(\mu+1)\cdot 4^{\mu}]\cdot n + n^2).$$

- No need to bound p_{max}.

Main result

Theorem [Mallem, Hanen, Munier 22]

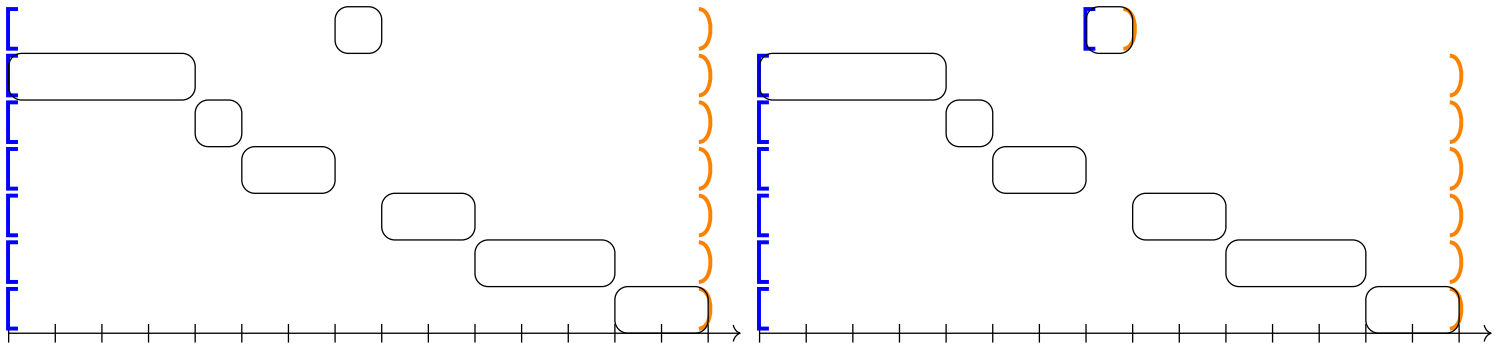
Single machine scheduling with time windows can be solved in time:

$$\mathcal{O}([(\mu+1)\cdot log(\mu+1)\cdot 4^{\mu}]\cdot n + n^2).$$

- No need to bound p_{max}.
- *prec* can be added (almost) for free: $[(\mu+1)^2 \cdot 4^{\mu}]$.

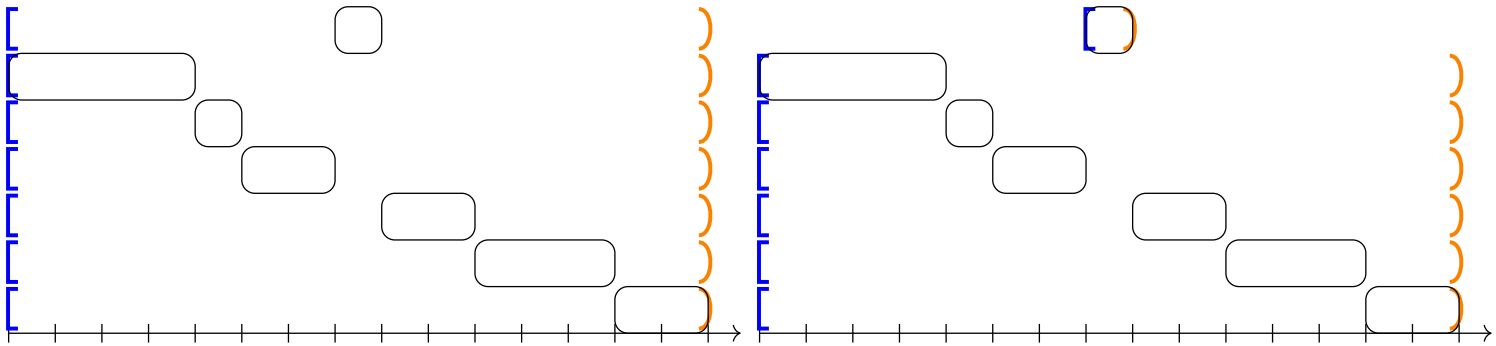
Height limitation

- Two single machine instances with time windows.



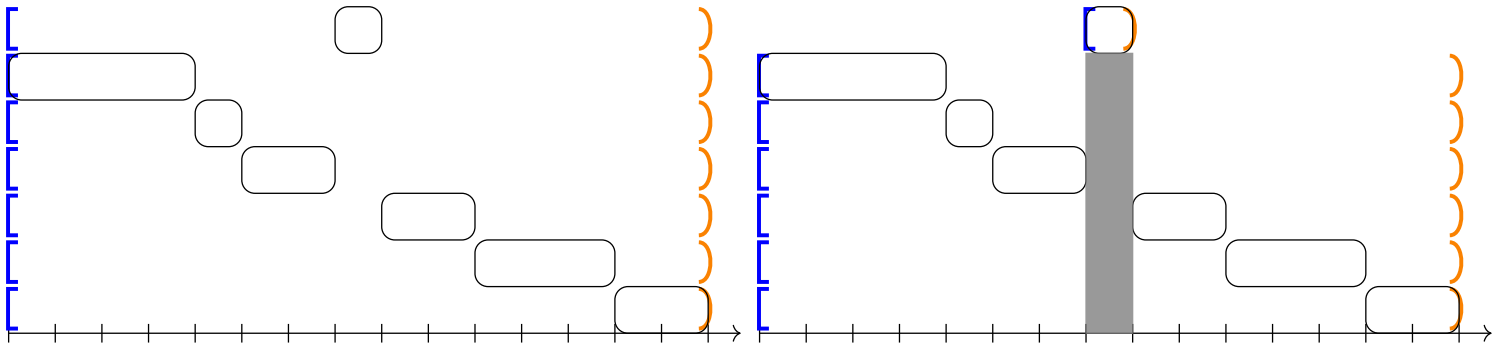
Height limitation

- Two single machine instances with time windows.



Height limitation

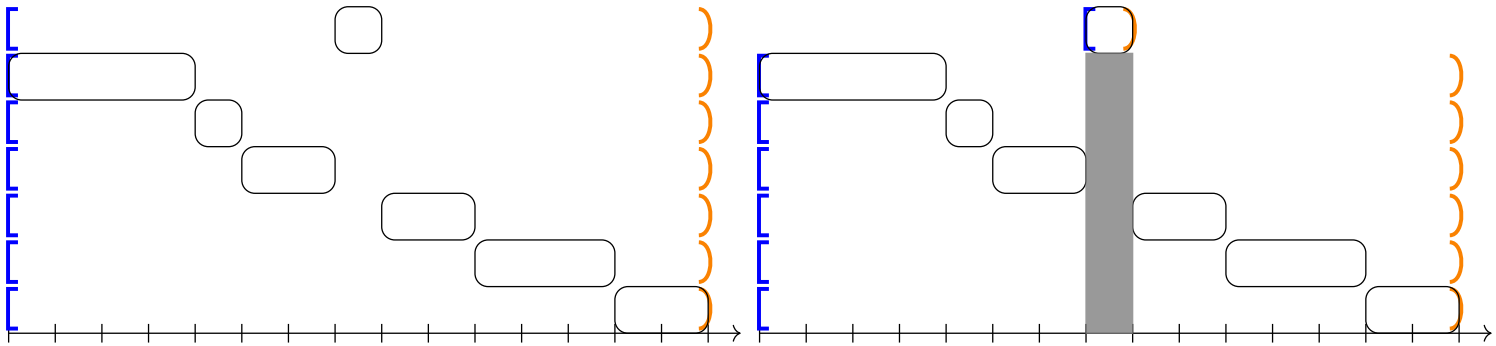
- Two single machine instances with time windows.



$\mu = n$ $\mu = n$

Height limitation

- Two single machine instances with time windows.

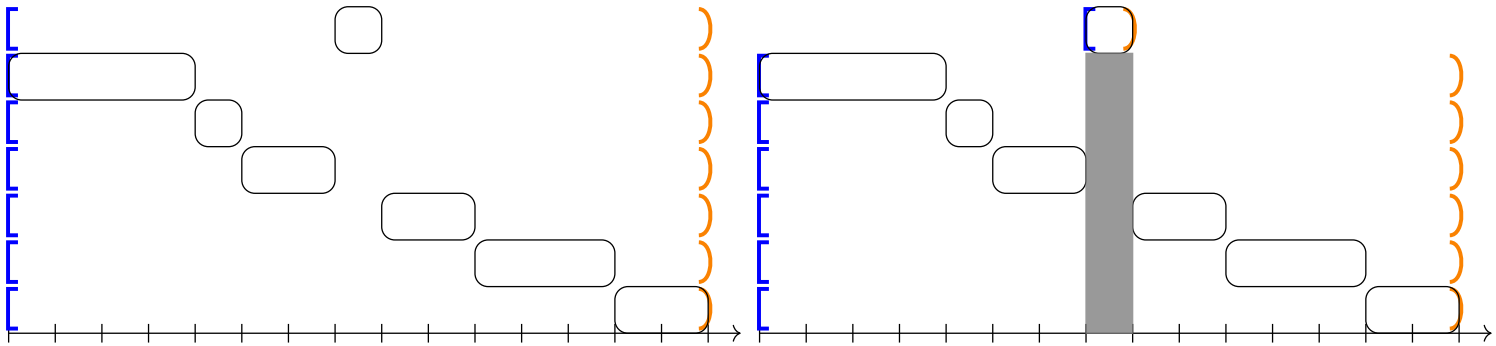


Definition: Proper sets and proper level

- $\forall J_i \in \mathcal{J}, \Pi_i = \{J_j \in \mathcal{J}, r_j < r_i \wedge d_i < d_j\}$
- $q = max_{J_i \in \mathcal{J}} |\Pi_i|$

Height limitation

- Two single machine instances with time windows.

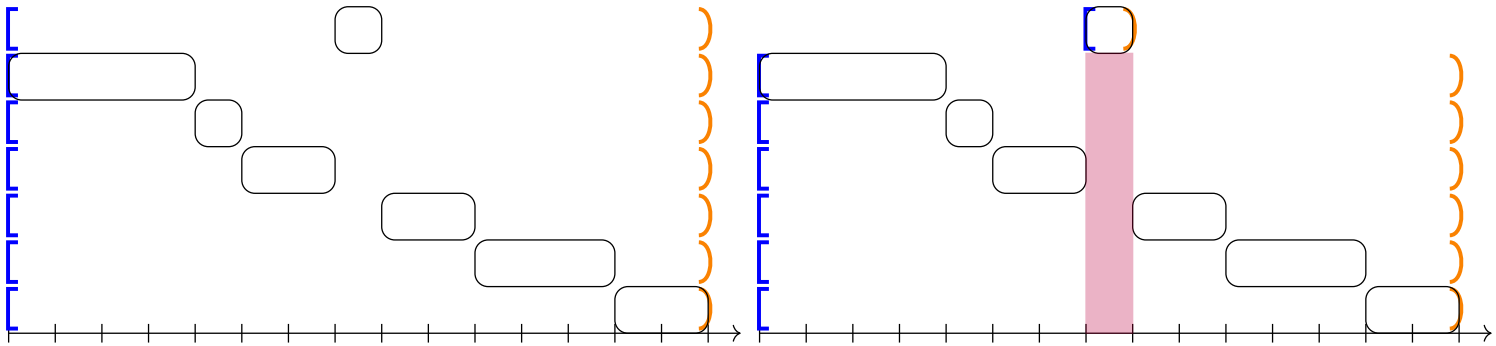


Definition: Proper sets and proper level

- $\forall J_i \in \mathcal{J}, \Pi_i = \{J_j \in \mathcal{J}, r_j < r_i \wedge d_i < d_j\}$ [Erschler, Fontan, Merce 85]
- $q = max_{J_i \in \mathcal{J}} |\Pi_i|$ [Proskurowski, Telle 99]

Height limitation

- Two single machine instances with time windows.



$\mu = n$ $\mu = n$

$q = 0$ $q = n - 1$

Definition: Proper sets and proper level

- $\forall J_i \in \mathcal{J}, \Pi_i = \{J_j \in \mathcal{J}, r_j < r_i \wedge d_i < d_j\}$ [Erschler, Fontan, Merce 85]
- $q = max_{J_i \in \mathcal{J}} |\Pi_i|$ [Proskurowski, Telle 99]

Extending the Earliest Deadline rule

- $\prec$ = lexicographic order (non decreasing d_j, non decreasing r_j) on $\mathcal{J}$.
- Renaming the jobs of $\mathcal{J}$ in $[1, n]$.

Definition: Earliest Deadline rule (ED) [Lawler, Moore 69]

Schedule τ follows the (ED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies \tau(i) < \tau(j).$$

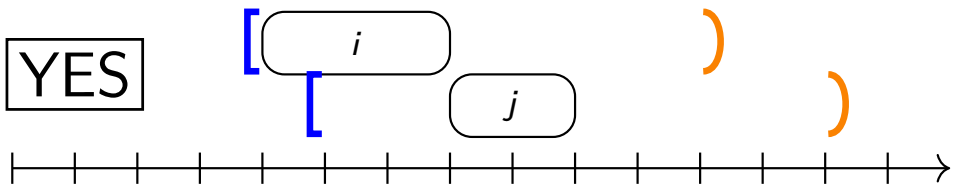
Extending the Earliest Deadline rule

- $\prec$ = lexicographic order (non decreasing d_j, non decreasing r_j) on $\mathcal{J}$.
- Renaming the jobs of $\mathcal{J}$ in $[1, n]$.

Definition: Earliest Deadline rule (ED) [Lawler, Moore 69]

Schedule τ follows the (ED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies \tau(i) < \tau(j).$$



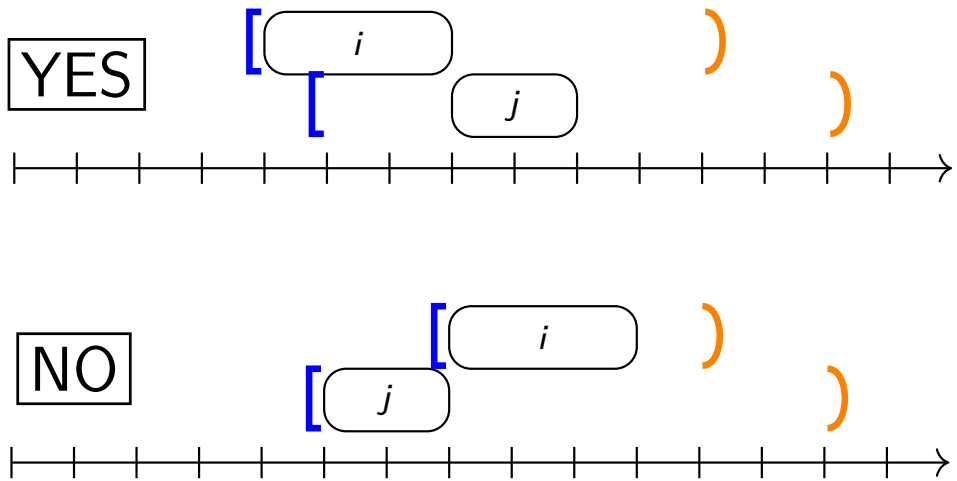
Extending the Earliest Deadline rule

- $\prec$ = lexicographic order (non decreasing d_j, non decreasing r_j) on $\mathcal{J}$.
- Renaming the jobs of $\mathcal{J}$ in $[1, n]$.

Definition: Earliest Deadline rule (ED) [Lawler, Moore 69]

Schedule τ follows the (ED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies \tau(i) < \tau(j).$$



Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

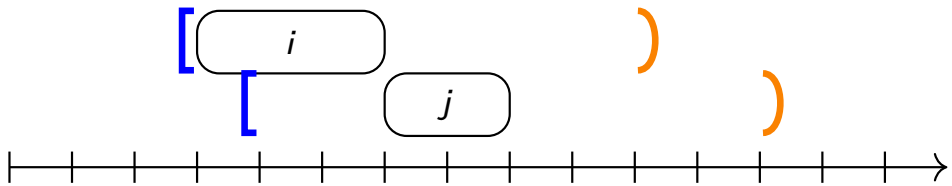
$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies [\tau(i) < \tau(j)] \vee [j \in \Pi_i] \vee [\exists h \prec i, \tau(j) < \tau(h) < \tau(i)].$$

Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies [\tau(i) < \tau(j)] \vee [j \in \Pi_i] \vee [\exists h \prec i, \tau(j) < \tau(h) < \tau(i)].$$

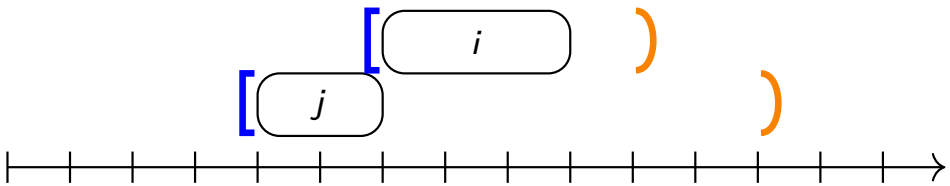


Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies [\tau(i) < \tau(j)] \vee [j \in \Pi_i] \vee [\exists h \prec i, \tau(j) < \tau(h) < \tau(i)].$$

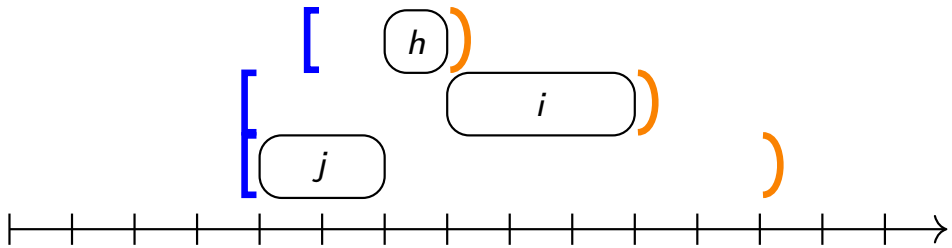


Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies [\tau(i) < \tau(j)] \vee [j \in \Pi_i] \vee [\exists h \prec i, \tau(j) < \tau(h) < \tau(i)].$$

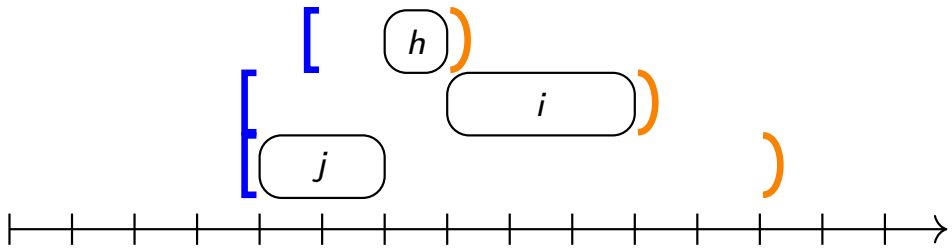


Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

$$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies [\tau(i) < \tau(j)] \vee [j \in \Pi_i] \vee [\exists h \prec i, \tau(j) < \tau(h) < \tau(i)].$$



Proposition

On $1|r_j, d_j|C_{max}$ the (wED)-following schedules are dominant.

Summit-based schedule enumeration

Definition: Summit

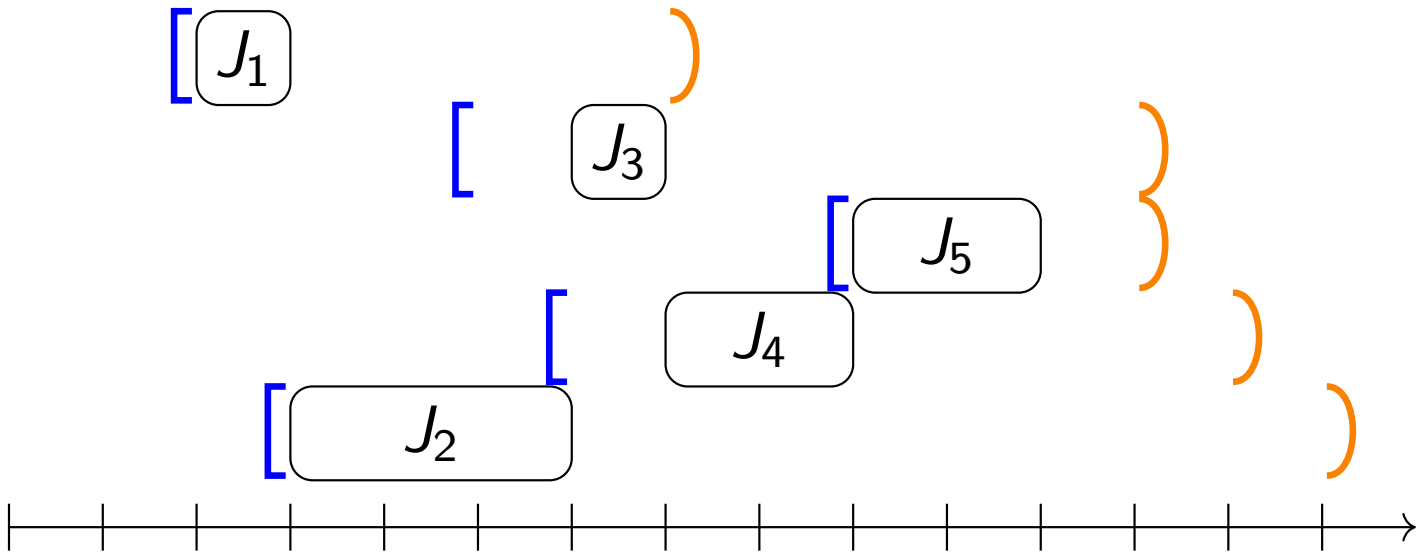
A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.

The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.

Summit-based schedule enumeration

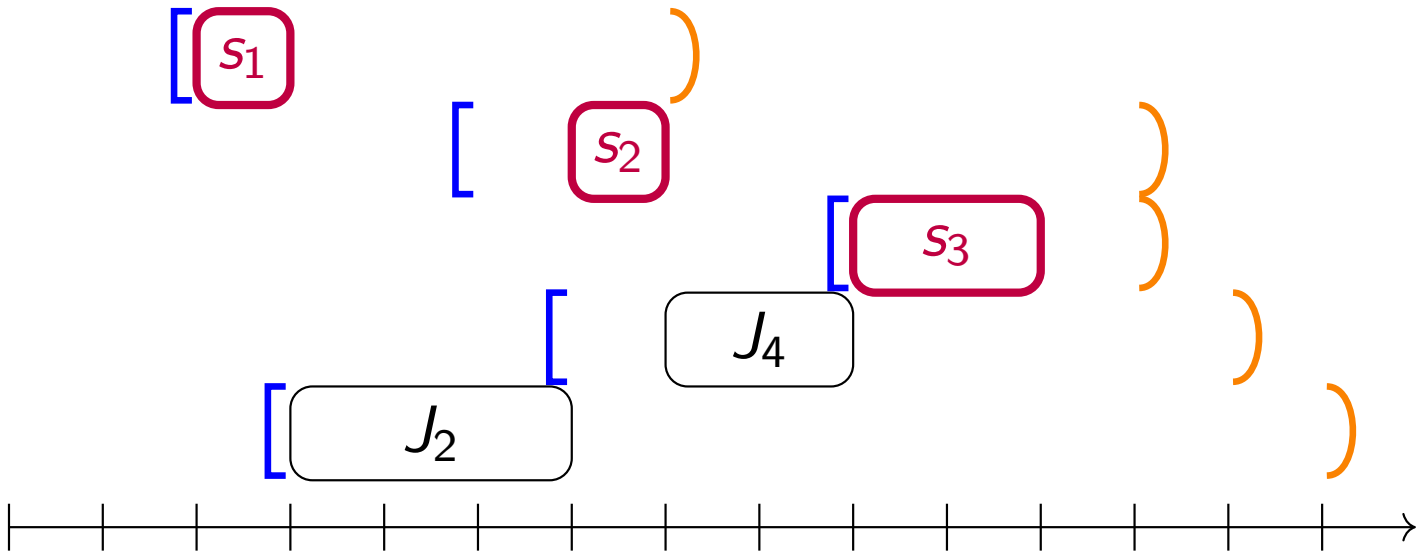
Definition: Summit

A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.



Definition: Summit

A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.



Summit-based schedule enumeration (cont.)

Definition: Summit

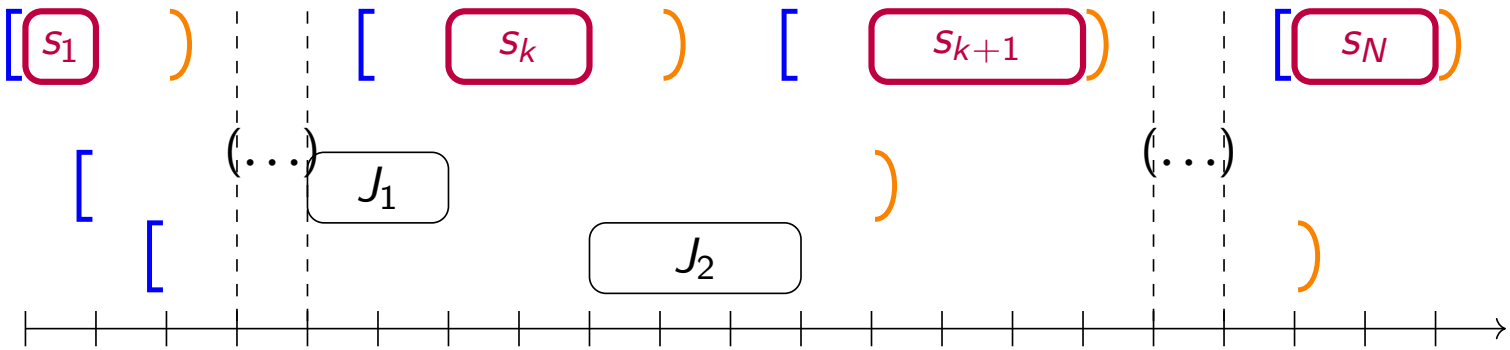
A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.

Summit-based schedule enumeration (cont.)

Definition: Summit

A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.

- Schedule τ follows the (wED) rule
  $\Longrightarrow$ τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$.

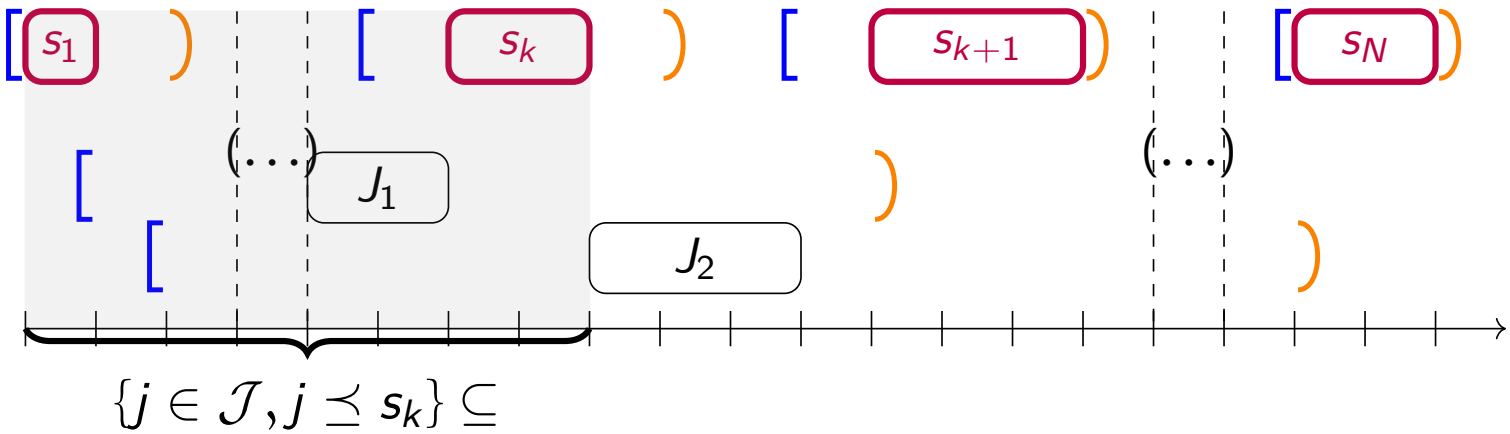


Summit-based schedule enumeration (cont.)

Definition: Summit

A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.

- Schedule τ follows the (wED) rule
  $\Longrightarrow$ τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$.

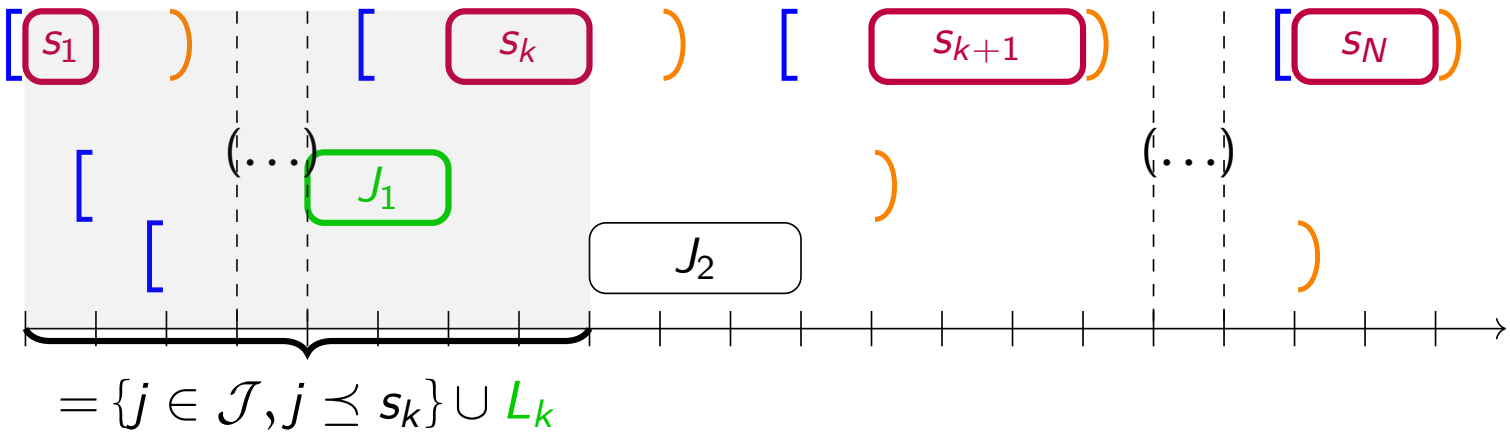


Summit-based schedule enumeration (cont.)

Definition: Summit

A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.

- Schedule τ follows the (wED) rule
  $\implies$ τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$.



- $j \in L_k \implies j \in \Pi_{s_k}$: at most q such jobs j.

Main result

Theorem [Mallem, Hanen, Munier 24]

Single machine scheduling with time windows can be solved in time:

$$\mathcal{O}([(q+1) \cdot log(q+1) \cdot 4^q] \cdot n + n^2).$$

Main result

Theorem [Mallem, Hanen, Munier 24]

Single machine scheduling with time windows can be solved in time:

$$\mathcal{O}([(q+1) \cdot log(q+1) \cdot 4^q] \cdot n + n^2).$$

- No need to bound p_{max}.

Main result

Theorem [Mallem, Hanen, Munier 24]

Single machine scheduling with time windows can be solved in time:

$$\mathcal{O}([(q+1) \cdot log(q+1) \cdot 4^q] \cdot n + n^2).$$

- No need to bound p_{max}.
- *prec* can be added ("almost") for free: $[(q+1)^2 \cdot 4^q]$.

Outline

Conclusion

- An introduction to scheduling and parameterized complexity.

Conclusion

- An introduction to scheduling and parameterized complexity.
- A theoretical approach to (eventually) give insight on real-life problems.

Conclusion

- An introduction to scheduling and parameterized complexity.
- A theoretical approach to (eventually) give insight on real-life problems.
- Feel free to contact me!
  - M7 306
  - maher.mallem@ens-lyon.fr
  - Discord