Parameterized Complexity of Single Machine Scheduling with Job Time Windows

Maher Mallem[1], Claire Hanen[2,3], Alix Munier-Kordon[2]

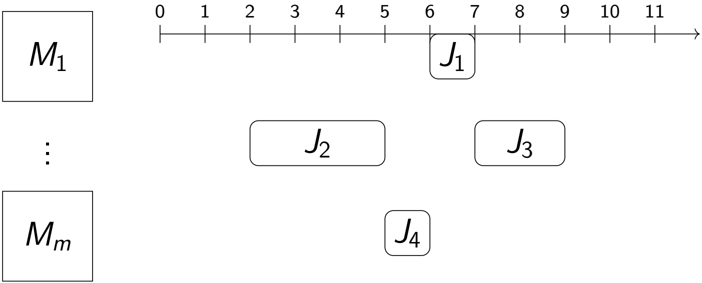
[1]Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France,
[2]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France,
[3]Université Paris Nanterre, F-92000 Nanterre, France,

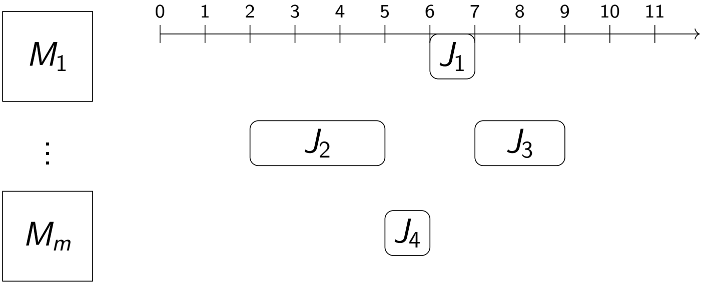
20 March 2025

Outline

Outline

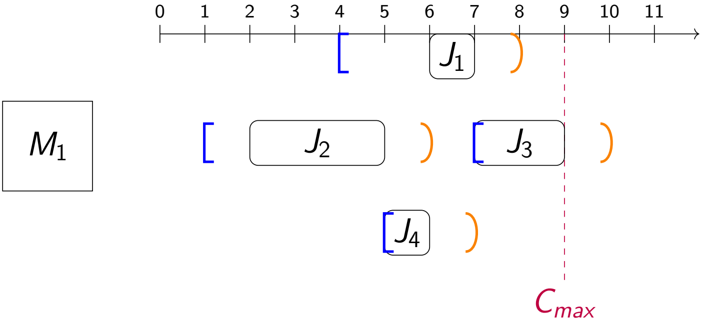
Scheduling





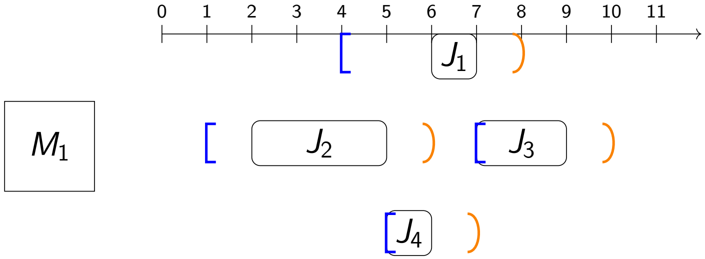
machines | job properties | optimization criterion

Scheduling



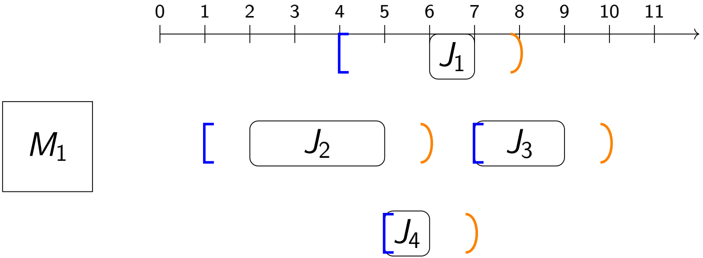
$$1|r_j, d_j|C_{max}$$

Scheduling



$$1|r_j, d_j|\star$$

Scheduling



$1|r_j, d_j|\star$ is strongly NP-hard [LRKB 77].

Parameterized complexity classes

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

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

para-NP

XP

$XNLP$

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

& $f(k) \cdot log(n)$ space

$\cdots$

$W[2]$

$W[1]$

FPT

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

Parameterized complexity classes

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

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

$\boxed{k\text{-COLORING}}$ para-NP

XP

$XNLP$ nondet. $f(k) \cdot poly(|\mathcal{I}|)$ time & $f(k) \cdot log(n)$ space

...

$\boxed{k\text{-DOMINATING SET}}$ $W[2]$

$\boxed{k\text{-CLIQUE}}$ $W[1]$

FPT

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

Parameterized complexity in machine scheduling

- Negative results via *FPT* reductions

- Negative results via *FPT* reductions
  - [Bodlaender, Fellows 95] $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines.

- Negative results via *FPT* reductions
  - [Bodlaender, Fellows 95] $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines.

[Mnich, Wiese 15]

- Negative results via *FPT* reductions
  - [Bodlaender, Fellows 95] $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines.
- MIP with a bounded number of integer variables

[Mnich, Wiese 15]

Parameterized complexity in machine scheduling

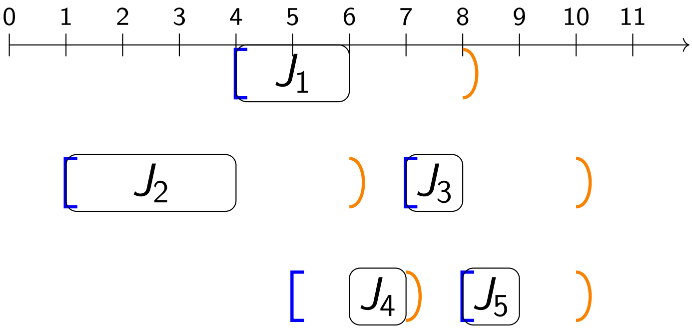
- Negative results via *FPT* reductions
  - [Bodlaender, Fellows 95] $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines.
- MIP with a bounded number of integer variables
  - [Lenstra 83] MILP: [Mnich, Wiese 15]
  - [Dadush et al. 11] MICP
  - [Knop et al. 19] MIMO

Parameterized complexity in machine scheduling

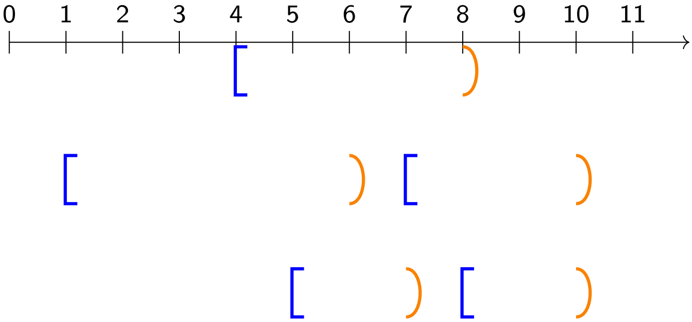
- Negative results via *FPT* reductions
  - [Bodlaender, Fellows 95] $P|prec, p_j = 1|C_{max}$ is W[2]-hard parameterized by the number m of machines.
- MIP with a bounded number of integer variables
  - [Lenstra 83] MILP: [Mnich, Wiese 15]
  - [Dadush et al. 11] MICP
  - [Knop et al. 19] MIMO
- Dynamic Programming

Outline

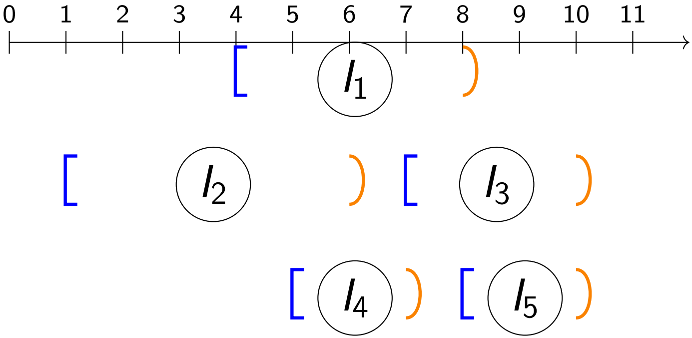
Pathwidth *pw*



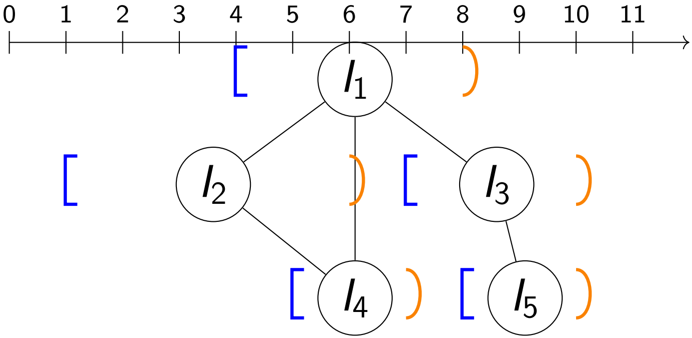
Pathwidth pw



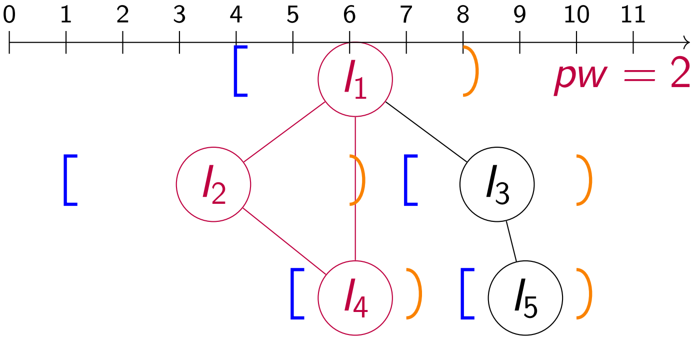
Pathwidth pw



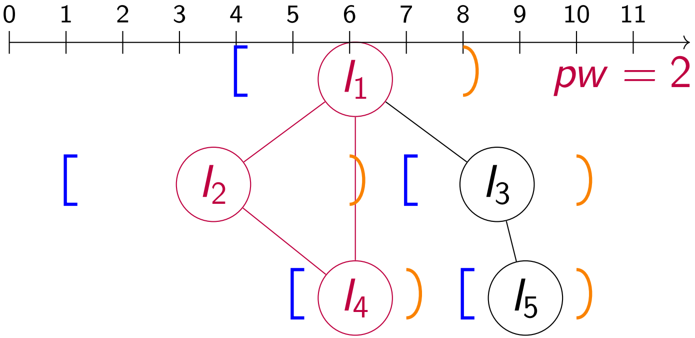
Pathwidth *pw*



Pathwidth pw



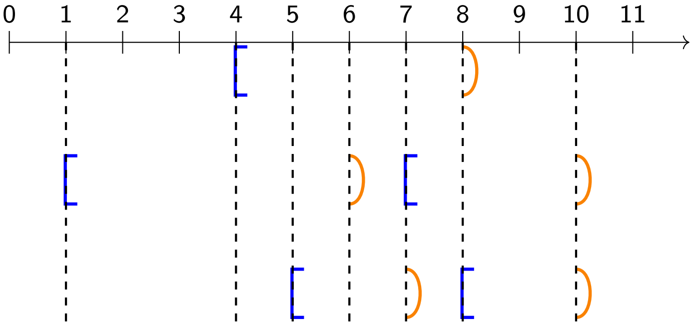
Pathwidth pw



[Bodlaender 93] For any interval graph G:

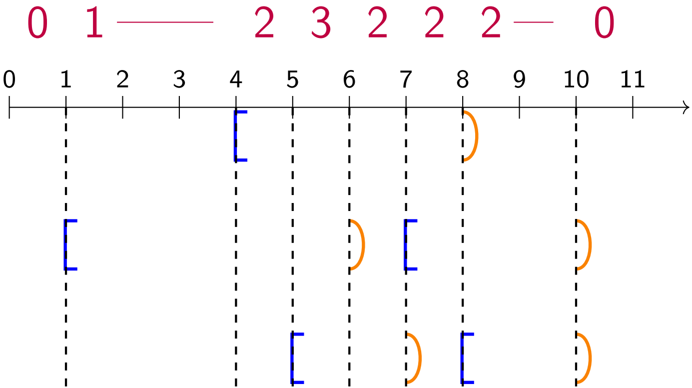
$$\omega(G) = \chi(G) = tw(G) + 1 = pw(G) + 1.$$

Pathwidth pw



[Bodlaender 93] For any interval graph G:

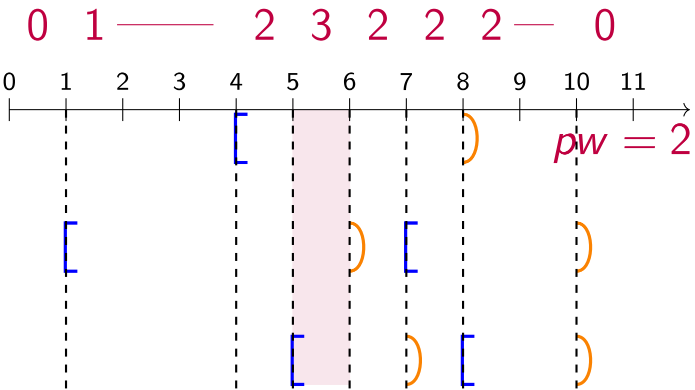
$$\omega(G) = \chi(G) = tw(G) + 1 = pw(G) + 1.$$



[Bodlaender 93] For any interval graph G:

$$\omega(G) = \chi(G) = tw(G) + 1 = pw(G) + 1.$$

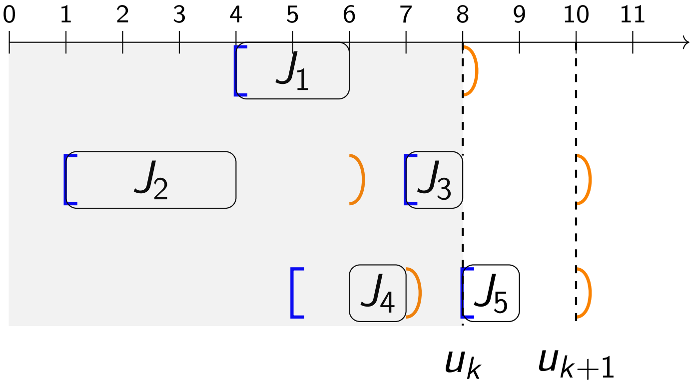
Pathwidth pw



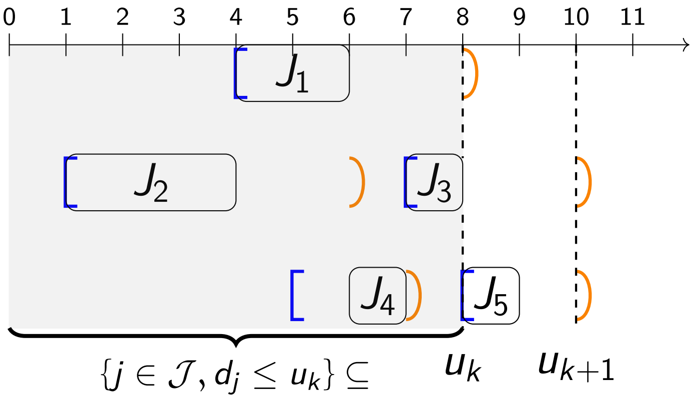
[Bodlaender 93] For any interval graph G:

$$\omega(G) = \chi(G) = tw(G) + 1 = pw(G) + 1.$$

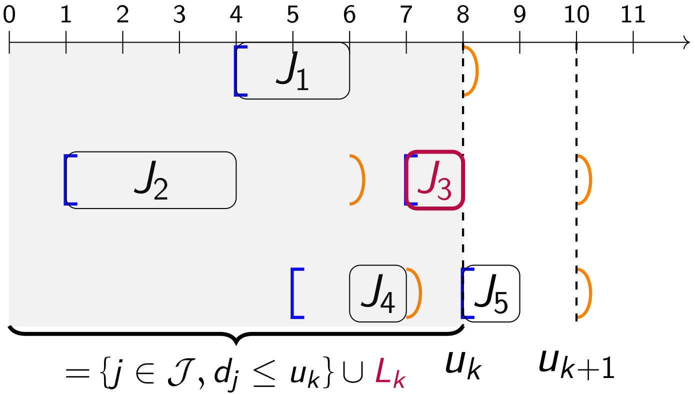
Pathwidth *pw*



Pathwidth pw

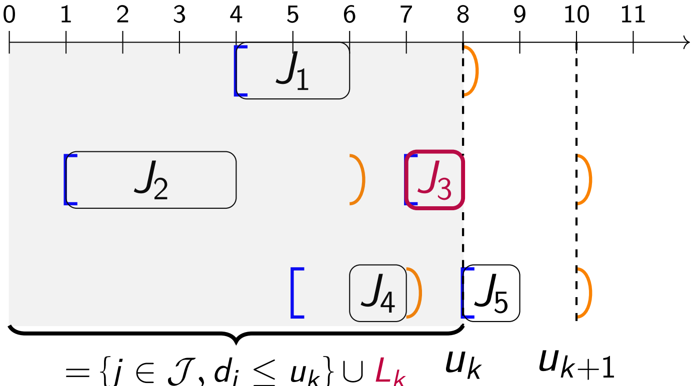


Pathwidth pw



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

Pathwidth *pw*



$$= \underbrace{\{j \in \mathcal{J}, d_j \leq u_k\}}_{= Z_k} \cup L_k$$

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

Dynamic programming with pathwidth pw

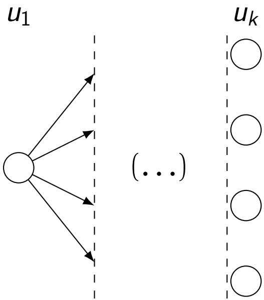
Dynamic programming state (k, L_k)

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

Dynamic programming with pathwidth pw

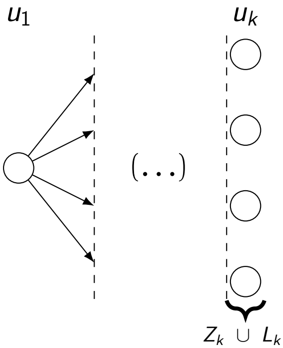
Dynamic programming state (k, L_k)

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



Dynamic programming state (k, L_k)

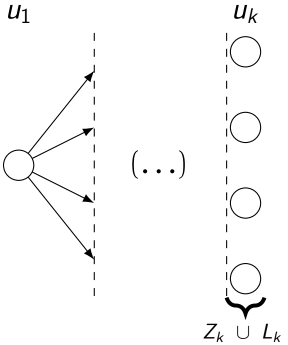
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

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

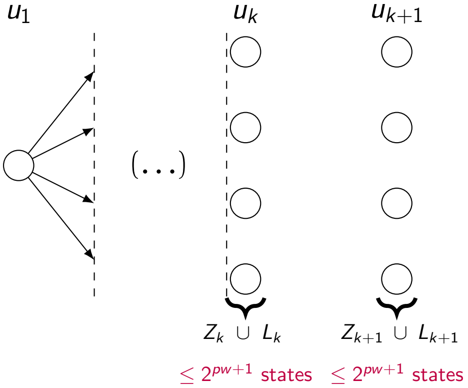


$\leq 2^{pw+1}$ states

Dynamic programming with pathwidth *pw*

Dynamic programming state (k, L_k)

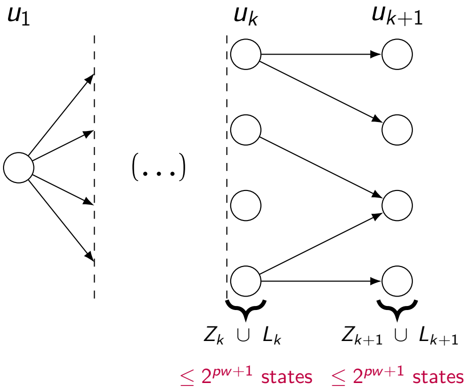
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

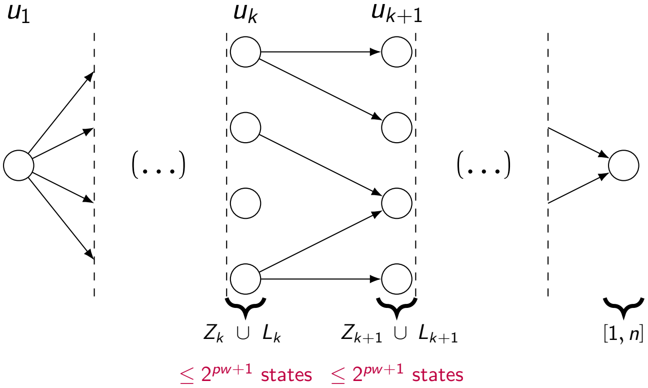
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = set of jobs starting before 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^{pw+1} 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})

$Cmax \leftarrow \max(u_k, T[k, L_k])$
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}.
if $(L_k - Z_{k+1}) \not\subseteq L_{k+1}$ **then return** ∞
▷ Add jobs to the existing partial schedule.
$mandatory \leftarrow Z_{k+1} \cap \{J_j \in \mathcal{J}, [u_k, u_{k+1}) \subseteq [r_j, d_j)\}$
$add \leftarrow (mandatory \cup L_{k+1}) - L_k$
sort(add, nondecreasing d_j)
for j in add (in order) **do**
    $Cmax \leftarrow Cmax + p_j$
    if $Cmax > d_j$ **then return** ∞
▷ Ensure that all the added jobs start earlier than u_{k+1}.
$last \leftarrow$ last job in add.
if $Cmax - p_{last} \geq u_{k+1}$ **then return** ∞
return $Cmax$

Main result

Theorem [Mallem, Hanen, Munier 22]

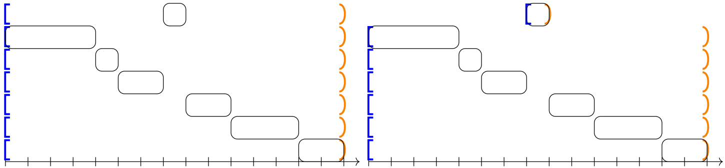
$1|r_j, d_j|C_{max}$ can be solved in time:

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

Outline

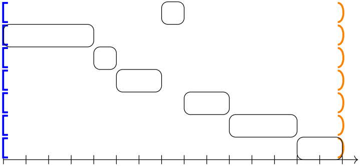
Pathwidth limitation

- Two single machine instances with time windows.

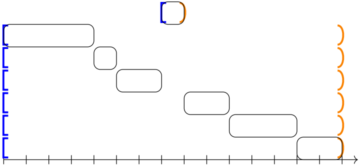


Pathwidth limitation

- Two single machine instances with time windows.



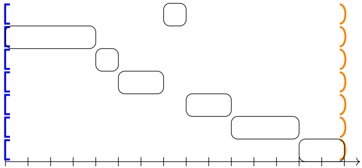
$pw = n - 1$



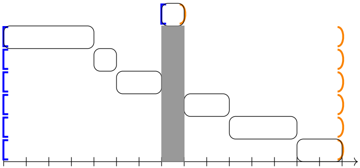
$pw = n - 1$

Pathwidth limitation

- Two single machine instances with time windows.



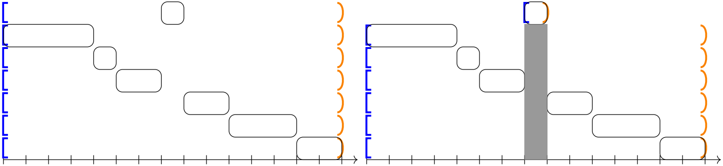
$pw = n - 1$



$pw = n - 1$

Pathwidth limitation

- Two single machine instances with time windows.



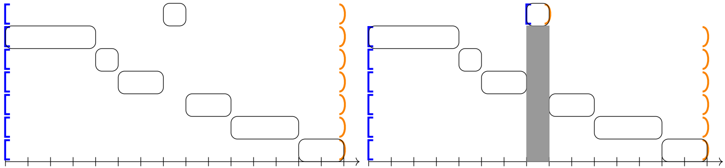
$pw = n - 1$ $\qquad$ $pw = n - 1$

Definition: Proper sets and proper level

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

Pathwidth limitation

- Two single machine instances with time windows.



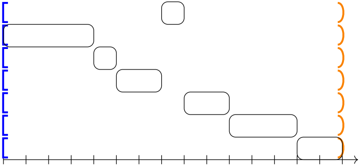
$pw = n - 1$ $\qquad$ $pw = n - 1$

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]

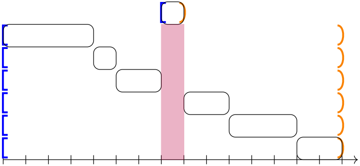
Pathwidth limitation

- Two single machine instances with time windows.



$pw = n - 1$

$q = 0$



$pw = n - 1$

$q = n - 1$

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:

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

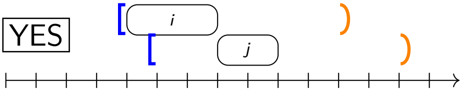
Extending the Earliest Deadline rule

- $\prec$ = 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:

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



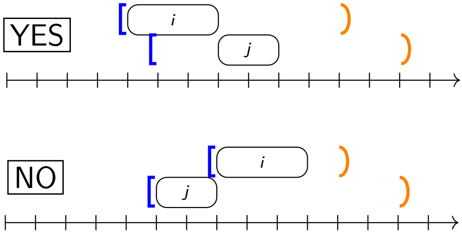
Extending the Earliest Deadline rule

- $\prec$ = 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)

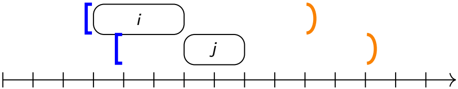
Schedule τ follows the (wED) rule if:

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

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

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

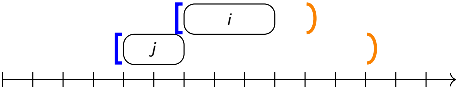


Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

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

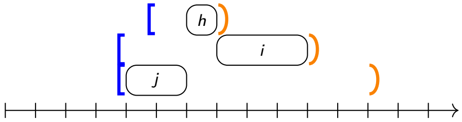


Extending the Earliest Deadline rule (cont.)

Definition: weak Earliest Deadline rule (wED)

Schedule τ follows the (wED) rule if:

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

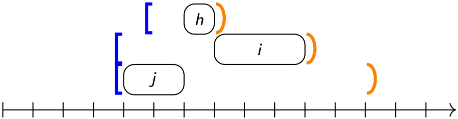


Extending the Earliest Deadline rule (cont.)

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

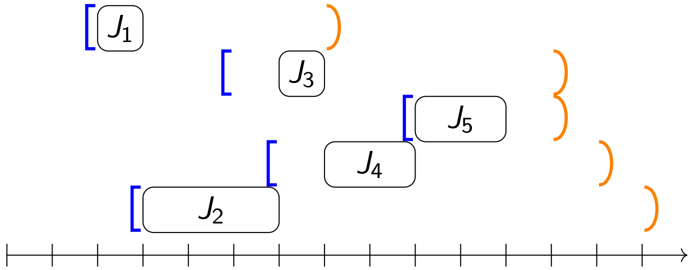
Definition: Summit

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

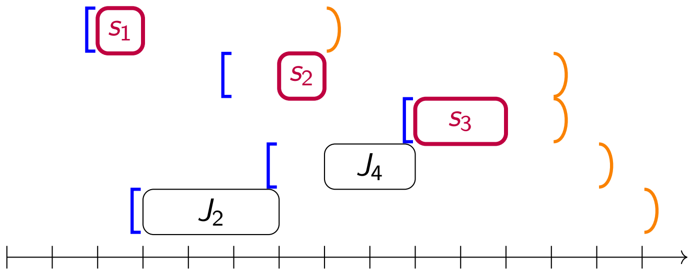
Definition: Summit

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

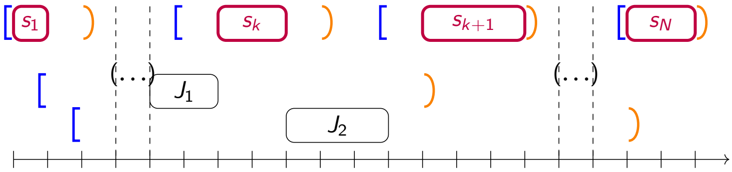
A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.
The summits of the instance are named $s_1 \prec \ldots \prec s_N$ $(N \leq n)$.

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
  $\implies$ τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$.

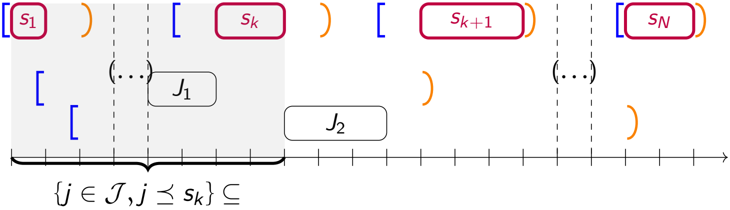


Summit-based schedule enumeration (cont.)

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 $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$.

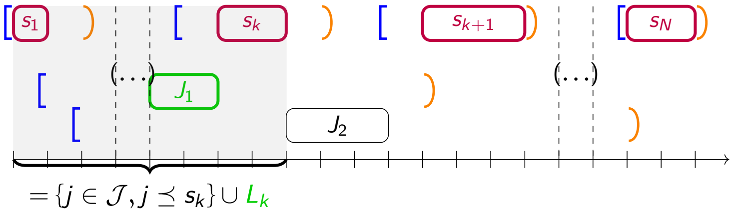


Summit-based schedule enumeration (cont.)

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.

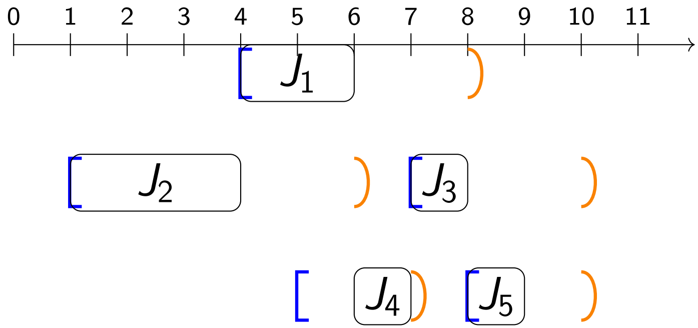
Theorem [Mallem, Hanen, Munier 24]

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

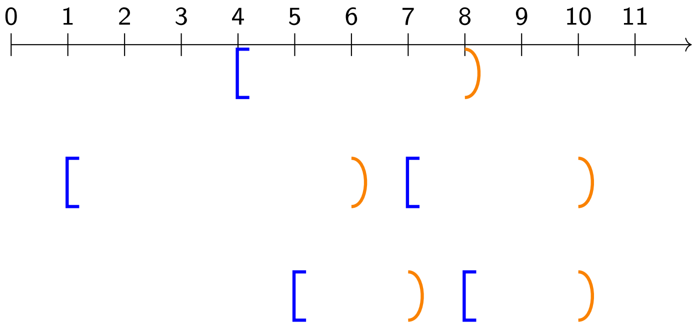
$$\mathcal{O}([(q+1) \cdot log(q+1) \cdot 4^q] \cdot n + n^2).$$

Outline

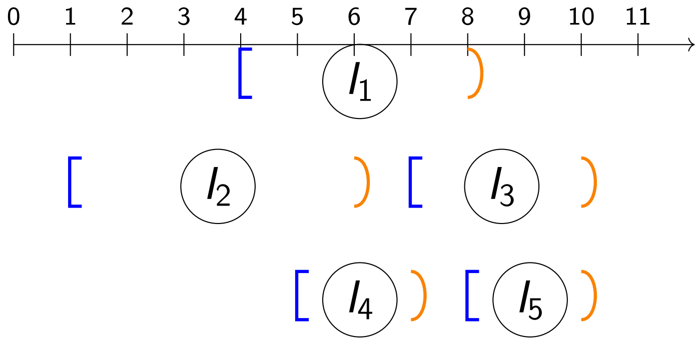
Pathwidth pw and vertex cover number vc



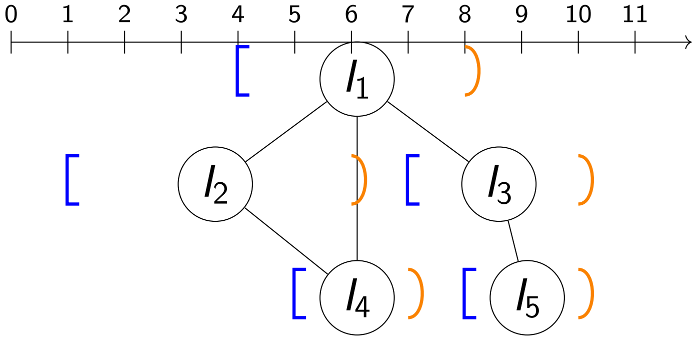
Pathwidth pw and vertex cover number vc



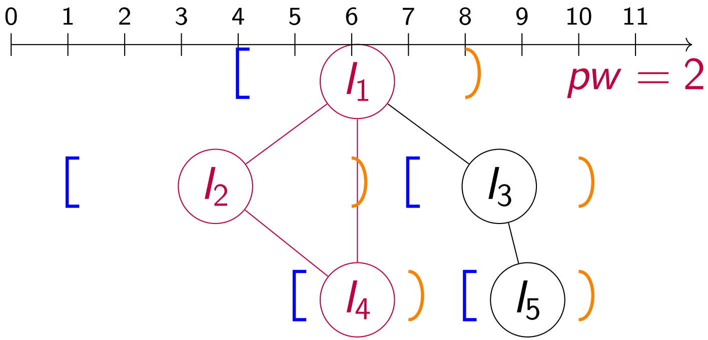
Pathwidth *pw* and vertex cover number *vc*



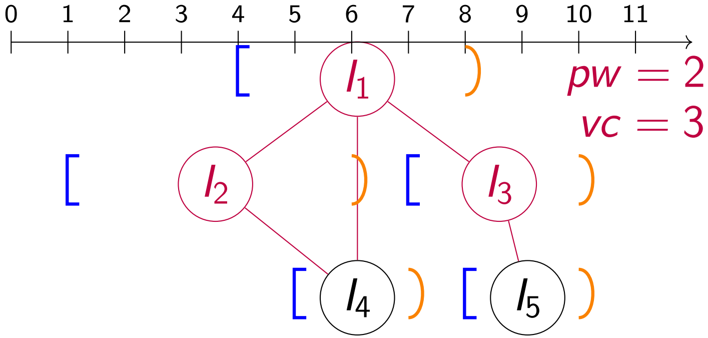
Pathwidth *pw* and vertex cover number *vc*



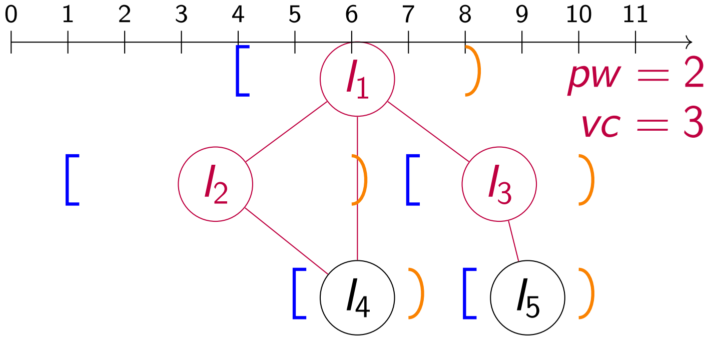
Pathwidth pw and vertex cover number vc



Pathwidth *pw* and vertex cover number *vc*



Pathwidth pw and vertex cover number vc



Claim

$$pw \leq vc$$

Polynomial kernel overview

Proposition

$1|r_j, d_j|\star$ has a polynomial kernel w.r.t. parameter vc.

Polynomial kernel overview

Proposition

$1|r_j, d_j|\star$ has a polynomial kernel w.r.t. parameter vc.

Instance I n jobs size $\mathcal{O}(n \cdot log(max_j(d_j)))$	$\xrightarrow{(1)}$	Instance I' $\mathcal{O}(vc^2)$ jobs size $\mathcal{O}(vc^2 \cdot log(max_j(d_j)))$	$\xrightarrow{(2)}$	Instance I'' $\mathcal{O}(vc^2)$ jobs size $\mathcal{O}(vc^8)$

Polynomial kernel overview

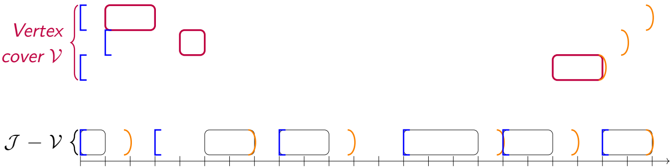
Proposition

$1|r_j, d_j|\star$ has a polynomial kernel w.r.t. parameter vc.

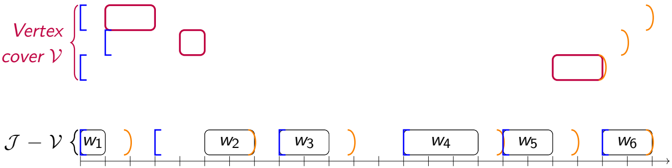
Instance I		Instance I'		Instance I''
n jobs	$\xrightarrow{(1)}$	$\mathcal{O}(vc^2)$ jobs	$\xrightarrow{(2)}$	$\mathcal{O}(vc^2)$ jobs
size $\mathcal{O}(n \cdot log(max_j(d_j)))$		size $\mathcal{O}(vc^2 \cdot log(max_j(d_j)))$		size $\mathcal{O}(vc^8)$

- (1) Delete *most* jobs.
  - Use a reduction rule.
  - Time $\mathcal{O}(n^2)$.
- (2) Reduce time values.
  - Translate into a BIP and use a theorem from [Frank, Tardos 87].
  - Time $\mathcal{O}(n^{18})$.

(1) Viewing non-vertex covering jobs as gaps

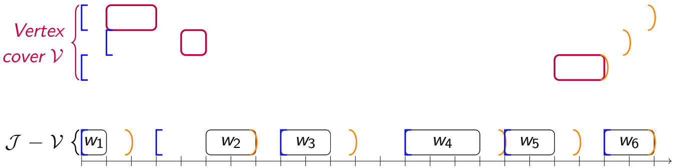


(1) Viewing non-vertex covering jobs as gaps



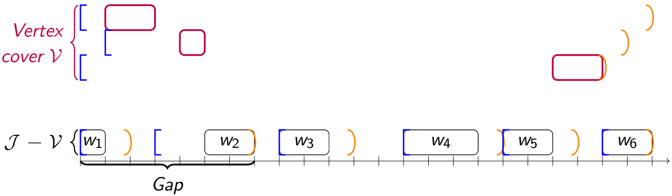
- $\mathcal{J}-\mathcal{V}$: jobs $w_1, w_2, \ldots, w_{|\mathcal{J}-\mathcal{V}|}$.

(1) Viewing non-vertex covering jobs as gaps



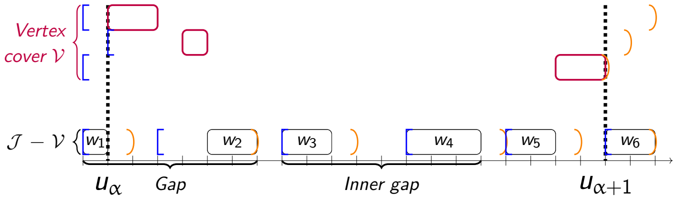
- $\mathcal{J}-\mathcal{V}$: jobs $w_1, w_2, \ldots, w_{|\mathcal{J}-\mathcal{V}|}$.
- Gap $=$ two consecutive jobs w_i, w_{i+1}.

(1) Viewing non-vertex covering jobs as gaps



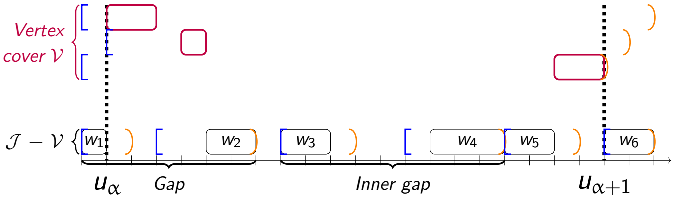
- $\mathcal{J} - \mathcal{V}$: jobs $w_1, w_2, \ldots, w_{|\mathcal{J}-\mathcal{V}|}$.
- Gap $=$ two consecutive jobs w_i, w_{i+1}.

(1) Viewing non-vertex covering jobs as gaps



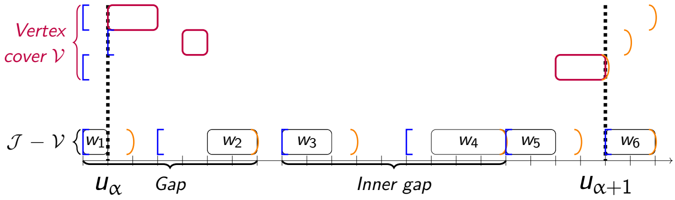
- $\mathcal{J} - \mathcal{V}$: jobs $w_1, w_2, \ldots, w_{|\mathcal{J}-\mathcal{V}|}$.
- Gap $=$ two consecutive jobs w_i, w_{i+1}.
  - Inner gap: time windows fully included in $[u_\alpha, u_{\alpha+1})$.

(1) Viewing non-vertex covering jobs as gaps



- $\mathcal{J} - \mathcal{V}$: jobs $w_1, w_2, \ldots, w_{|\mathcal{J}-\mathcal{V}|}$.
- Gap = two consecutive jobs w_i, w_{i+1}.
  - Inner gap: time windows fully included in $[u_\alpha, u_{\alpha+1})$.
- Maximum capacity: w_i at the earliest, w_{i+1} at the latest.

(1) Viewing non-vertex covering jobs as gaps

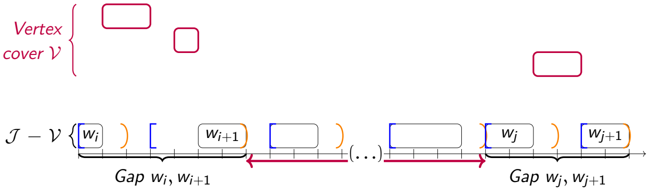


- $\mathcal{J} - \mathcal{V}$: jobs $w_1, w_2, \dots, w_{|\mathcal{J}-\mathcal{V}|}$.
- Gap = two consecutive jobs w_i, w_{i+1}.
  - Inner gap: time windows fully included in $[u_\alpha, u_{\alpha+1})$.
- Maximum capacity: w_i at the earliest, w_{i+1} at the latest.
- $vc_\alpha = |\{J_j \in \mathcal{V}, [u_\alpha, u_{\alpha+1}) \subseteq [r_j, d_j)\}|$.

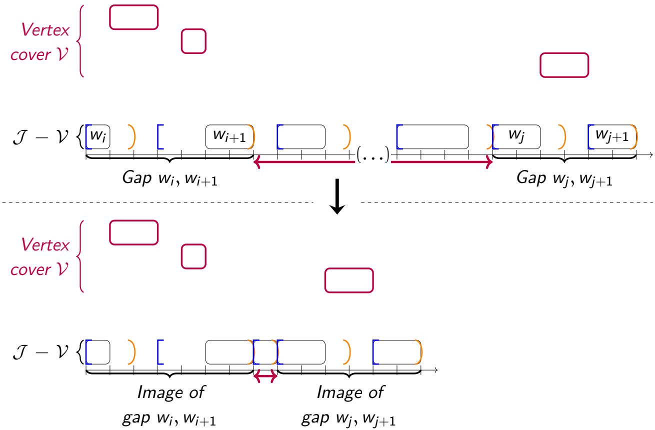
Reduction rule

In each interval $[u_\alpha, u_{\alpha+1})$, only keep the $3vc_\alpha$ inner gaps with highest maximum capacity.

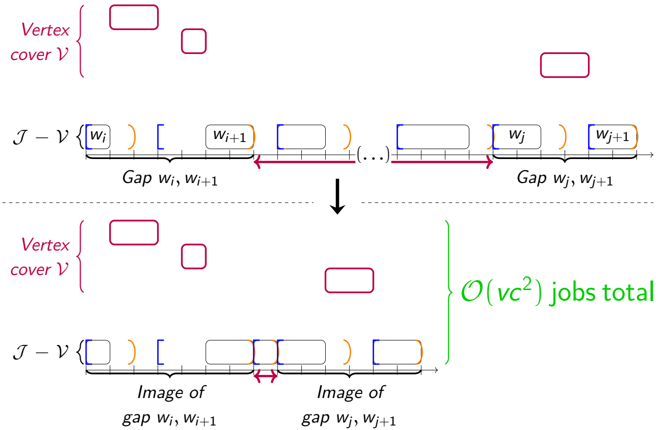
(1) Deleting unnecessary inner gaps



(1) Deleting unnecessary inner gaps



(1) Deleting unnecessary inner gaps



(2) Reducing time values

(2) Reducing time values

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w_1', \ldots, w_d')$ of positive integers such that:

- $w_i' \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N - 1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w_i' x_i \geq 0.$$

(2) Reducing time values

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w'_1, \ldots, w'_d)$ of positive integers such that:

- $w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N - 1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w'_i x_i \geq 0.$$

- [Lasserre, Queyranne 92] Binary IP equivalent to $1|r_j, d_j|\star$.
  - At most $n + 2$ variables equal to one in each equality of any solution.

(2) Reducing time values

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w'_1, \ldots, w'_d)$ of positive integers such that:

- $w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N-1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w'_i x_i \geq 0.$$

- [Lasserre, Queyranne 92] Binary IP equivalent to $1|r_j, d_j|\star$.
  - At most $n+2$ variables equal to one in each equality of any solution.
- Use the theorem on instance I' with $\tilde{n} = \mathcal{O}(vc^2)$ jobs.
  - $d = 3\tilde{n}$, $\mathtt{w} = (p_1, r_1, d_1, \ldots, p_{\tilde{n}}, r_{\tilde{n}}, d_{\tilde{n}})$, $N = \tilde{n} + 3$.

(2) Reducing time values

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w'_1, \ldots, w'_d)$ of positive integers such that:

- $w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N - 1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w'_i x_i \geq 0.$$

- [Lasserre, Queyranne 92] Binary IP equivalent to $1|r_j, d_j|\star$.
  - At most $n + 2$ variables equal to one in each equality of any solution.
- Use the theorem on instance I' with $\tilde{n} = \mathcal{O}(vc^2)$ jobs.
  - $d = 3\tilde{n}$, $\mathtt{w} = (p_1, r_1, d_1, \ldots, p_{\tilde{n}}, r_{\tilde{n}}, d_{\tilde{n}})$, $N = \tilde{n} + 3$.
  - Time $\mathcal{O}(vc^{18}) = \mathcal{O}(n^{18})$.
  - Each reduced time value taking space $\mathcal{O}(vc^6)$.

Outline

No polynomial kernel w.r.t. pathwidth pw

Proposition

Unless $NP \subseteq coNP/poly$ there is no polynomial kernel of $1|r_j, d_j|\star$ w.r.t. pathwidth pw.

No polynomial kernel w.r.t. pathwidth pw

Proposition

Unless $NP \subseteq coNP/poly$ there is no polynomial kernel of $1|r_j, d_j|\star$ w.r.t. pathwidth pw.

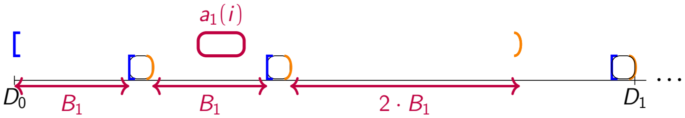
- *Proof:* Cross-composition from PARTITION.
  - $(P_1, \ldots, P_t) \mapsto I$
  - $pw \leq poly(max_i(|P_i|) + log(t))$

No polynomial kernel w.r.t. pathwidth pw

Proposition

Unless $NP \subseteq coNP/poly$ there is no polynomial kernel of $1|r_j, d_j|\star$ w.r.t. pathwidth pw.

- *Proof:* Cross-composition from PARTITION.
  - $(P_1, \dots, P_t) \mapsto I$
  - $pw \leq poly(max_i(|P_i|) + log(t))$

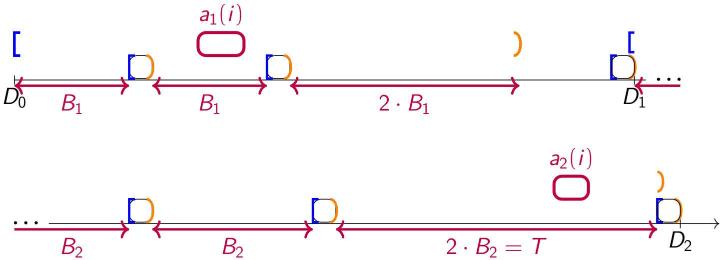


No polynomial kernel w.r.t. pathwidth pw

Proposition

Unless $NP \subseteq coNP/poly$ there is no polynomial kernel of $1|r_j, d_j|\star$ w.r.t. pathwidth pw.

- *Proof:* Cross-composition from PARTITION.
  - $(P_1, \ldots, P_t) \mapsto I$
  - $pw \leq poly(max_i(|P_i|) + log(t))$

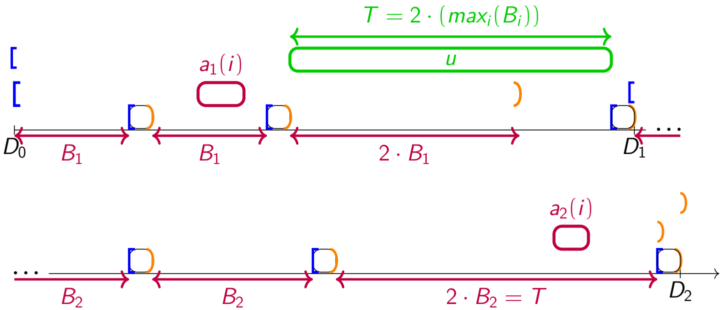


No polynomial kernel w.r.t. pathwidth *pw*

Proposition

Unless $NP \subseteq coNP/poly$ there is no polynomial kernel of $1|r_j, d_j|\star$ w.r.t. pathwidth pw.

- *Proof:* Cross-composition from PARTITION.
  - $(P_1, \ldots, P_t) \mapsto I$
  - $pw \leq poly(max_i(|P_i|) + log(t))$



Algorithm 1 NaiveKernel($1|r_j, d_j|\star$) with pathwidth pw

Input: an instance $\mathcal{I}$ of $1|r_j, d_j|\star$.
Compute pw in time $\mathcal{O}(n \cdot log(n))$.
if $|\mathcal{I}| \leq 2^{pw}$ **then**
  return $\mathcal{I}$
else
  Use the *FPT* algorithm from [MHMK 22] in time $\mathcal{O}((pw+1)^2 \cdot 4^{pw} \cdot n + n \cdot log(n)) = \mathcal{O}(|\mathcal{I}|^2 \cdot log(|\mathcal{I}|) \cdot n)$.
  if the output is YES **then**
    return a dummy YES instance of size $\mathcal{O}(1)$.
  else
    return a dummy NO instance of size $\mathcal{O}(1)$.

An exponential size kernel w.r.t. pathwidth pw

Algorithm 1 NaiveKernel($1|r_j, d_j|\star$) with pathwidth pw

Input: an instance $\mathcal{I}$ of $1|r_j, d_j|\star$.
Compute pw in time $\mathcal{O}(n \cdot log(n))$.
if $|\mathcal{I}| \leq 2^{pw}$ **then**
  return $\mathcal{I}$
else
  Use the *FPT* algorithm from [MHMK 22] in time $\mathcal{O}((pw+1)^2 \cdot 4^{pw} \cdot n + n \cdot log(n)) = \mathcal{O}(|\mathcal{I}|^2 \cdot log(|\mathcal{I}|) \cdot n)$.
  if the output is YES **then**
    return a dummy YES instance of size $\mathcal{O}(1)$.
  else
    return a dummy NO instance of size $\mathcal{O}(1)$.

An exponential size kernel w.r.t. pathwidth pw (cont.)

Algorithm 2 Kernel$(1|r_j, d_j|\star)$ with pathwidth pw

Input: an instance $\mathcal{I}$ of $1|r_j, d_j|\star$.
Compute pw and vc in time $\mathcal{O}(n \cdot log(n))$.
if $vc \leq 2^{pw}$ **then**
    return the polynomial kernel of $\mathcal{I}$ w.r.t. vc.
else
    Use the *FPT* algorithm from [MHMK 22] in time $\mathcal{O}((pw+1)^2 \cdot 4^{pw} \cdot n + n \cdot log(n)) = \mathcal{O}(vc^2 \cdot log(vc) \cdot n)$.
    if the output is YES **then**
        return a dummy YES instance of size $\mathcal{O}(1)$.
    else
        return a dummy NO instance of size $\mathcal{O}(1)$.

Outline

Conclusion

Parameter	$1\|r_j, d_j\|\star$
q	FPT Kernel of exponential size No polynomial kernel
pw	
vc	FPT Polynomial kernel

Conclusion

Parameter	$1\|r_j, d_j\|\star$
q	FPT Kernel of exponential size No polynomial kernel
pw	
vc	FPT Polynomial kernel

Future work

- Investigate other scheduling problems with time windows.
  - Postdoc: $1|periodic_L(r_j, d_j)|C_{max}$

Conclusion

Parameter	$1\|r_j, d_j\|\star$
q	FPT Kernel of exponential size No polynomial kernel
pw	
vc	FPT Polynomial kernel

Future work

- Investigate other scheduling problems with time windows.
  - Postdoc: $1|periodic_L(r_j, d_j)|C_{max}$
- Consider parameters between pathwidth and vertex cover.
  - $pw \longrightarrow$ treedepth $\longrightarrow$ vertex integrity $\longrightarrow vc$

Parameter	$1\|r_j, d_j\|\star$
q	FPT Kernel of exponential size No polynomial kernel
pw	
vc	FPT Polynomial kernel

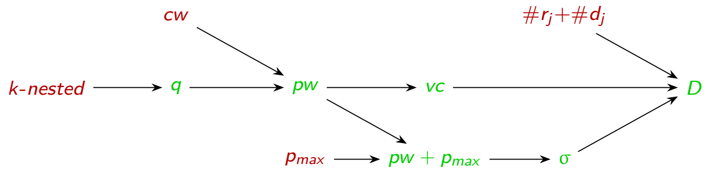
Future work

- Investigate other scheduling problems with time windows.
  - Postdoc: $1|periodic_L(r_j, d_j)|C_{max}$
- Consider parameters between pathwidth and vertex cover.
  - $pw \longrightarrow$ treedepth $\longrightarrow$ vertex integrity $\longrightarrow vc$
- Improve the existing algorithms.

Parameter hierarchy on $1|r_j, d_j|\star$

$$q \longrightarrow pw \longrightarrow vc$$

Parameter hierarchy on $1|r_j, d_j|\star$



k-nested: depth of the forest inferred by the "strictly included on both ends" relation.

cw: clique-width of the time window interval graph.

p_{max}: maximum processing time.

$\#r_j+\#d_j$: number of distinct release date and deadline values.

σ slack: $\max_j(d_j - r_j - p_j)$.

D: maximum deadline.

(2) Reducing time values (with the BIP)

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w_1', \ldots, w_d')$ of positive integers such that:

- $w_i' \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N - 1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w_i' x_i \geq 0.$$

(2) Reducing time values (with the BIP)

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w'_1, \ldots, w'_d)$ of positive integers such that:

- $w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N - 1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w'_i x_i \geq 0.$$

- [Lasserre, Queyranne 92] Binary IP equivalent to $1|r_j, d_j|\star$.

$$BIP(I) : \begin{cases} \text{One job per position} & \sum_{j=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq i \leq n. \\ \text{One position per job} & \sum_{i=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq j \leq n. \\ \text{Time window checks} & \sum_{j=1}^{n} r_j \cdot x_{i_1,j} + \sum_{i=i_1}^{i_2} \sum_{j=1}^{n} p_j \cdot x_{i,j} \leq \sum_{j=1}^{n} d_j \cdot x_{i_2,j} \\ & \text{for all } 1 \leq i_1 \leq i_2 \leq n. \end{cases}$$

(2) Reducing time values (with the BIP)

Theorem [Frank, Tardos 87]

Let $\mathtt{w} = (w_1, \dots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathtt{w}' = (w'_1, \dots, w'_d)$ of positive integers such that:

- $w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,
- for every $\mathtt{x} = (x_1, \dots, x_d)$ vector of integers such that $\sum_{i=1}^{d} |x_i| \leq N - 1$:

$$\sum_{i=1}^{d} w_i x_i \geq 0 \text{ if and only if } \sum_{i=1}^{d} w'_i x_i \geq 0.$$

- [Lasserre, Queyranne 92] Binary IP equivalent to $1|r_j, d_j|\star$.

$$BIP(I) : \begin{cases} \text{One job per position} & \sum_{j=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq i \leq n. \\ \text{One position per job} & \sum_{i=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq j \leq n. \\ \text{Time window checks} & \sum_{j=1}^{n} r_j \cdot x_{i_1,j} + \sum_{i=i_1}^{i_2} \sum_{j=1}^{n} p_j \cdot x_{i,j} \leq \sum_{j=1}^{n} d_j \cdot x_{i_2,j} \\ & \text{for all } 1 \leq i_1 \leq i_2 \leq n. \end{cases}$$

- Use the theorem on instance I' with $\tilde{n} = \mathcal{O}(vc^2)$ jobs.
  - $d = 3\tilde{n}$, $\mathtt{w} = (p_1, r_1, d_1, \dots, p_{\tilde{n}}, r_{\tilde{n}}, d_{\tilde{n}})$, $N = \tilde{n} + 3$.