Graph parameters for single machine scheduling with time windows

Maher Mallem[1]

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

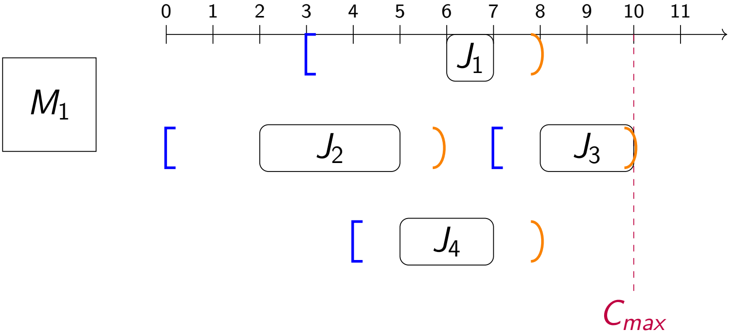
1 April 2026

Outline

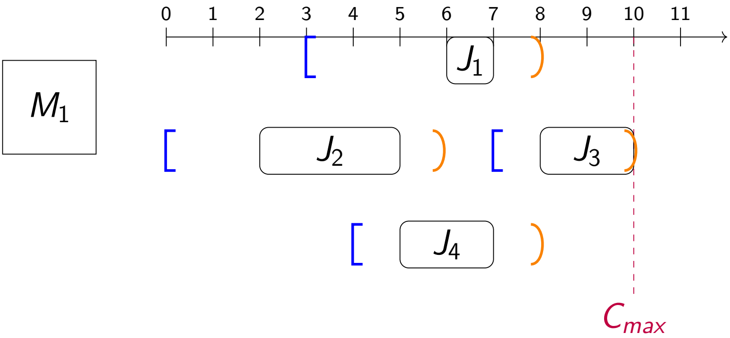
Outline

Introduction to scheduling

- Introduction
- Vertex cover number
- Pathwidth
- Proper level
- Conclusion
- References

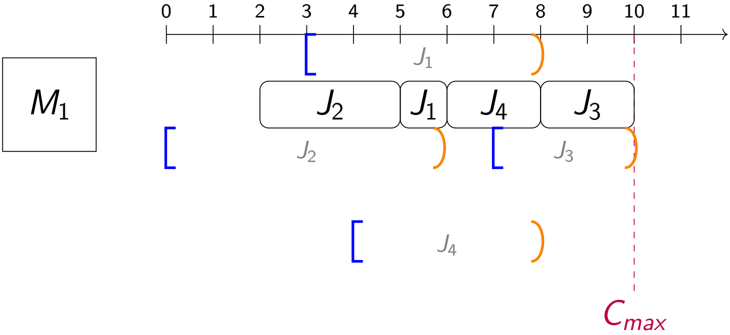


$$1|r_j, d_j|C_{max}$$



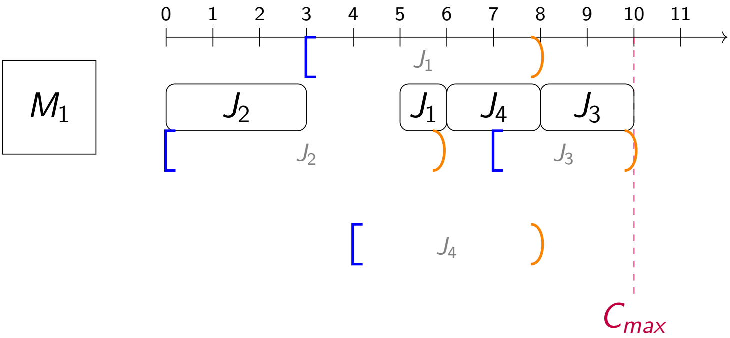
$$1|r_j, d_j|C_{max}$$

Strongly NP-complete [Lenstra et al., 1977].



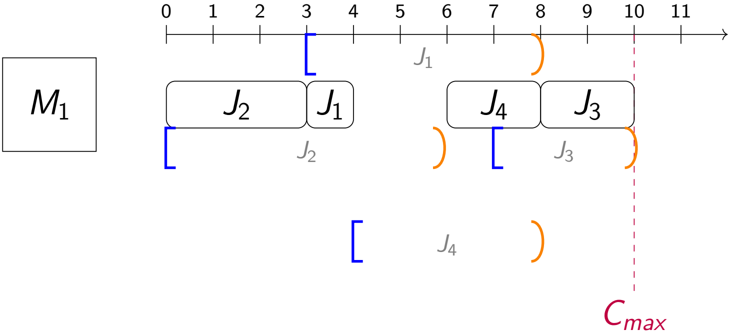
$$1|r_j, d_j|C_{max}$$

Strongly NP-complete [Lenstra et al., 1977].



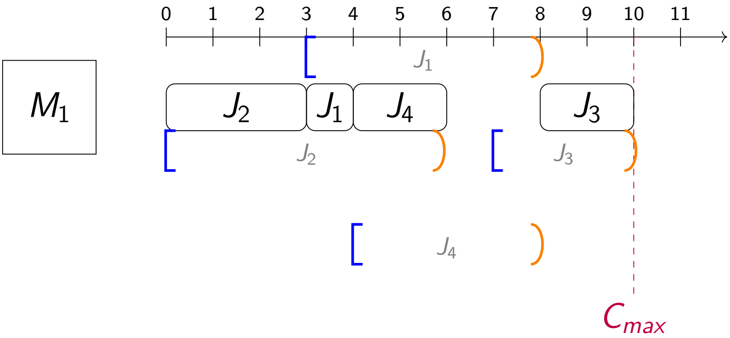
$$1|r_j, d_j|C_{max}$$

Strongly NP-complete [Lenstra et al., 1977].



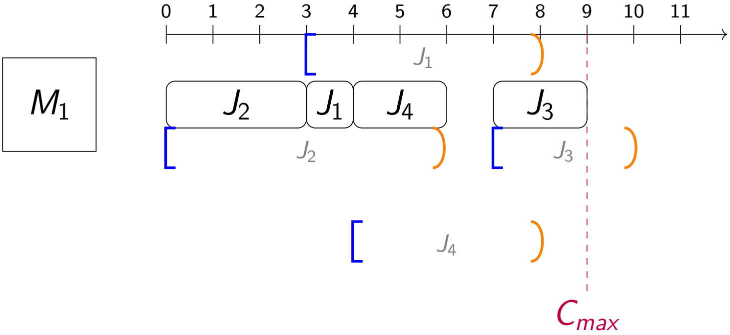
$$1|r_j, d_j|C_{max}$$

Strongly NP-complete [Lenstra et al., 1977].



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

Strongly NP-complete [Lenstra et al., 1977].



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

Strongly NP-complete [Lenstra et al., 1977].

Dynamic programming (baseline)

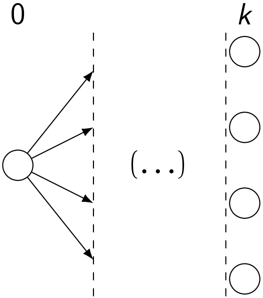
Dynamic programming state (k, S_k)

- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

Dynamic programming (baseline)

Dynamic programming state (k, S_k)

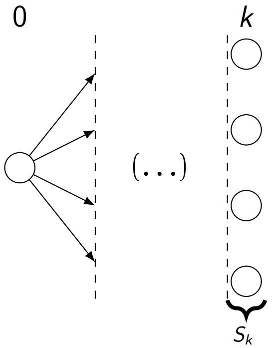
- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline)

Dynamic programming state (k, S_k)

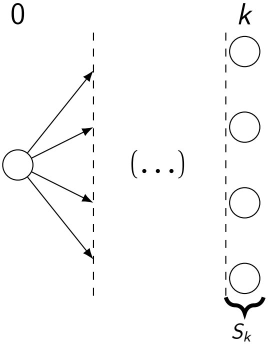
- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline)

Dynamic programming state (k, S_k)

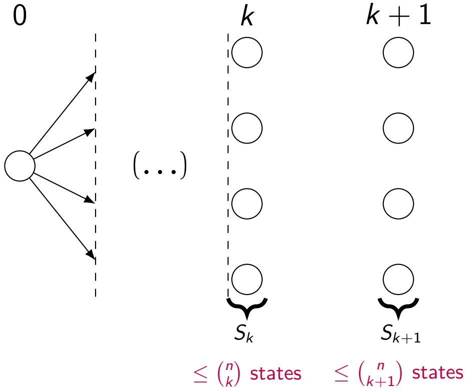
- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



$\leq \binom{n}{k}$ states

Dynamic programming state (k, S_k)

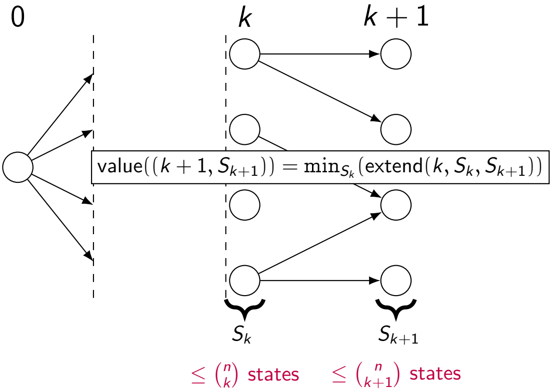
- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline)

Dynamic programming state (k, S_k)

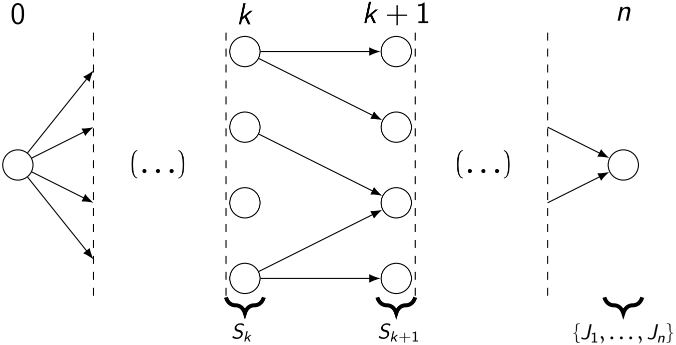
- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline)

Dynamic programming state (k, S_k)

- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

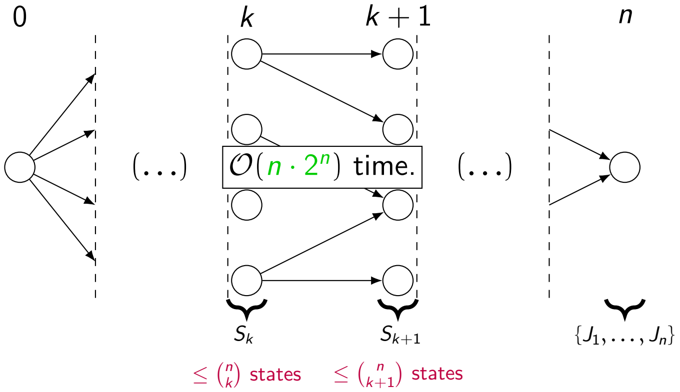


$\leq \binom{n}{k}$ states $\leq \binom{n}{k+1}$ states

Dynamic programming (baseline)

Dynamic programming state (k, S_k)

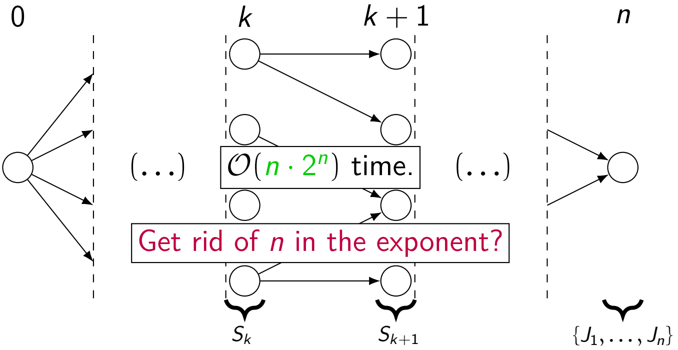
- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline)

Dynamic programming state (k, S_k)

- k = number of completed jobs.
- S_k = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



$\leq \binom{n}{k}$ states $\quad \leq \binom{n}{k+1}$ states

Going beyond NP-hardness

- Problem $\mathcal{P}$, instance $\mathcal{I}$.

- P: deterministic $poly(|\mathcal{I}|)$ time.

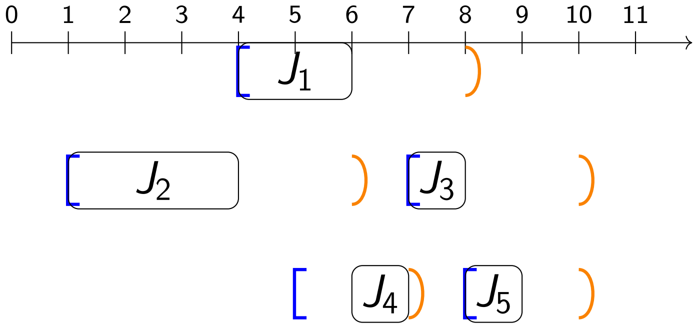
Going beyond NP-hardness

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

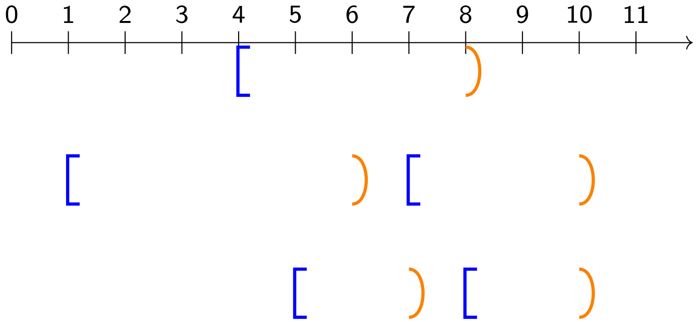
- Problem $\mathcal{P}$, instance $\mathcal{I}$.
  - Parameter $\bar{k}$, instance $\mathcal{I}$ with parameter value $\leq k$.
- P: deterministic $poly(|\mathcal{I}|)$ time.
  - FPT: deterministic $f(k) \cdot poly(|\mathcal{I}|)$ time.
- **Example:** $1|r_j, d_j|C_{max}$ parameterized by n is in FPT.
  - (Note: input size in $\mathcal{O}(n \cdot \log(d_{\max}))$.)

Outline

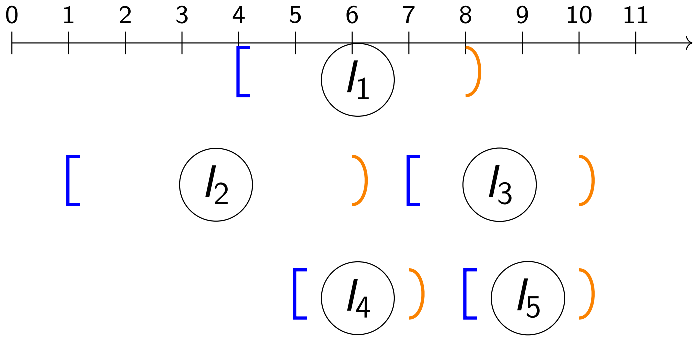
Vertex cover number *vc*



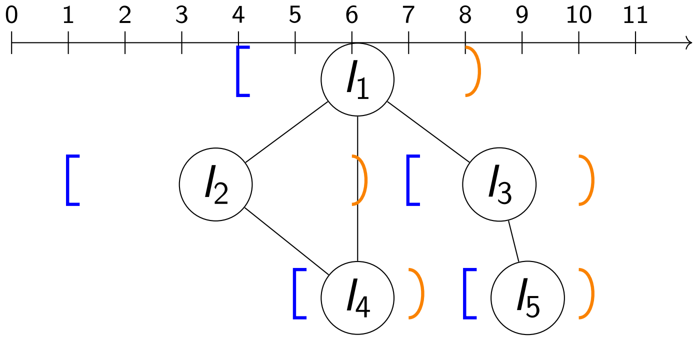
Vertex cover number *vc*

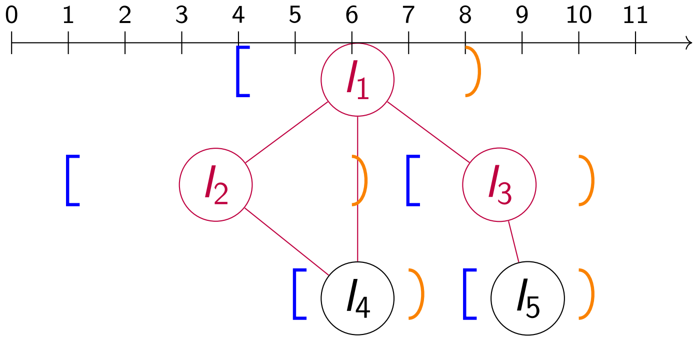


Vertex cover number *vc*

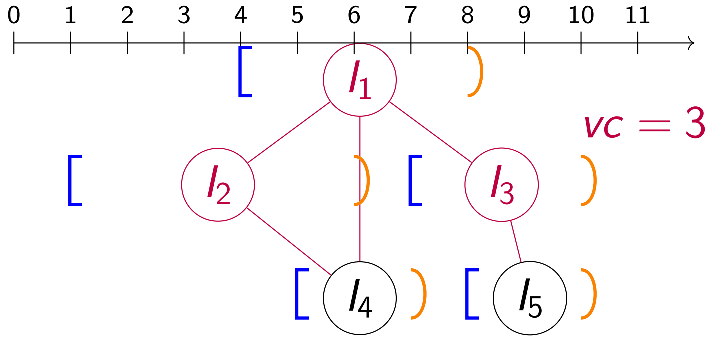


Vertex cover number vc




0
1
2
3
4
5
6
7
8
9
10
11
I1
I2
I3
I4
I5

Vertex cover number vc



Definition

$vc = \min_{\mathcal{VC} \text{ vertex cover}} |\mathcal{VC}|.$

Vertex cover over time windows

Definition

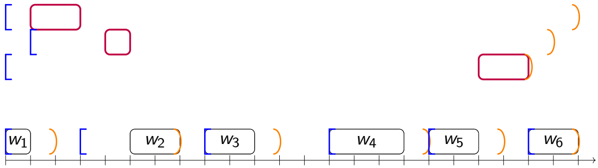
$vc = \min_{\mathcal{VC} \text{ vertex cover}} |\mathcal{VC}|$.

Vertex cover over time windows

Definition

$vc = \min_{\mathcal{VC} \text{ vertex cover}} |\mathcal{VC}|.$

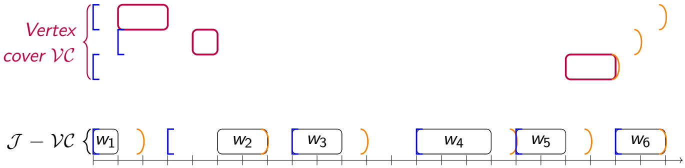
Vertex cover over time windows



Definition

$vc = \min_{\mathcal{VC} \text{ vertex cover}} |\mathcal{VC}|.$

Vertex cover over time windows



Definition

$vc = \min_{\mathcal{VC} \text{ vertex cover}} |\mathcal{VC}|.$

Dynamic programming (baseline reinterpreted)

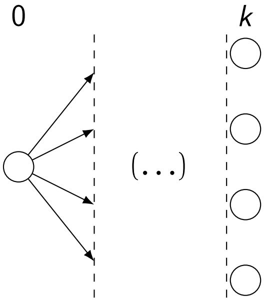
Dynamic programming state (i_k, V_k)

- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

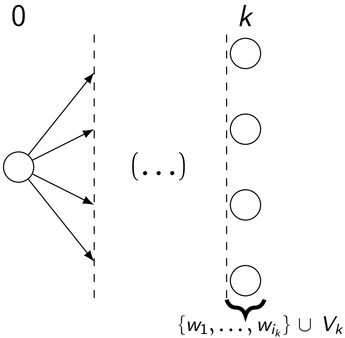
- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \ldots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

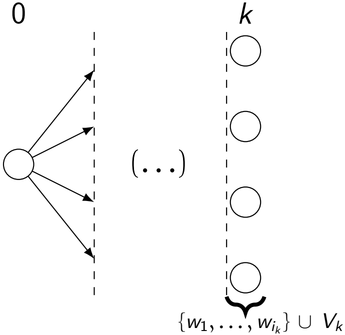
- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

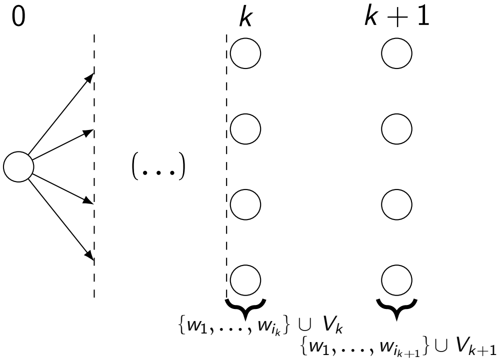


$\leq n \cdot 2^{vc}$ states

Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \ldots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

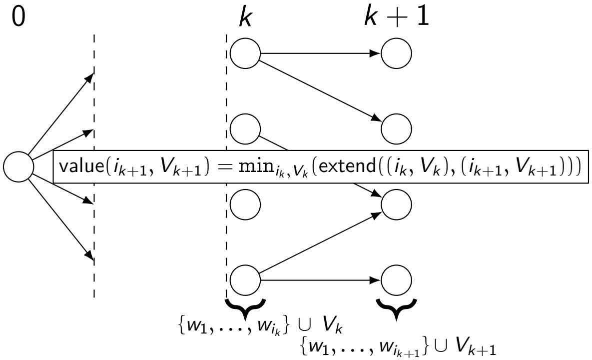


$\leq n \cdot 2^{vc}$ states

Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

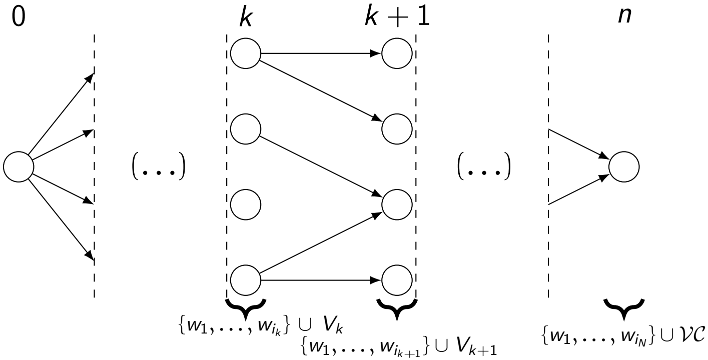


$\leq n \cdot 2^{vc}$ states

Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.

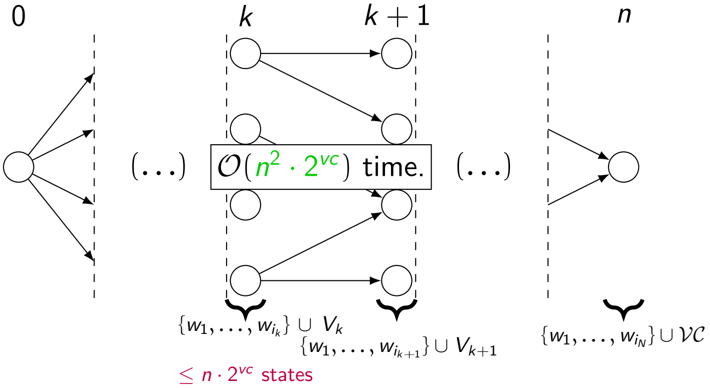


$\leq n \cdot 2^{vc}$ states

Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

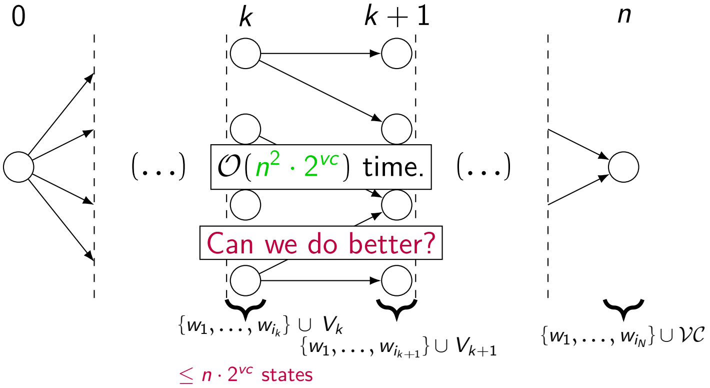
- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming (baseline reinterpreted)

Dynamic programming state (i_k, V_k)

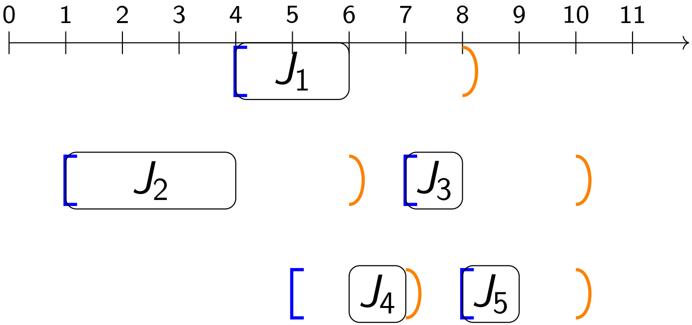
- i_k = index of the latest completed job in $\mathcal{J} - \mathcal{VC}$.
- $\{w_1, \dots w_{i_k}\} \cup V_k$ = subset of completed jobs.
- State value = minimum makespan among the associated partial schedules.



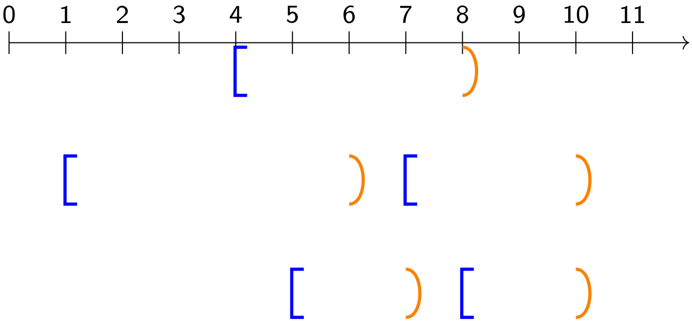
Outline

Pathwidth pw

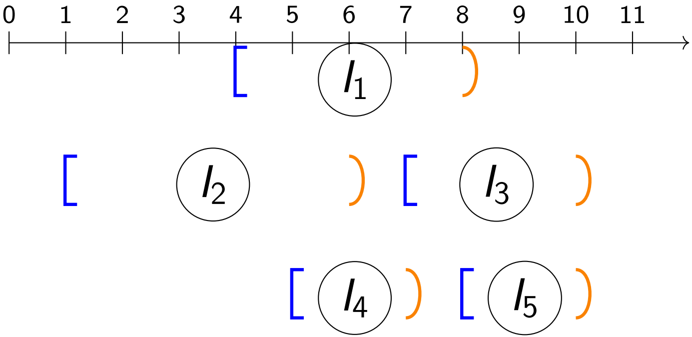
Introduction
Vertex cover number
Pathwidth
Proper level
Conclusion
References



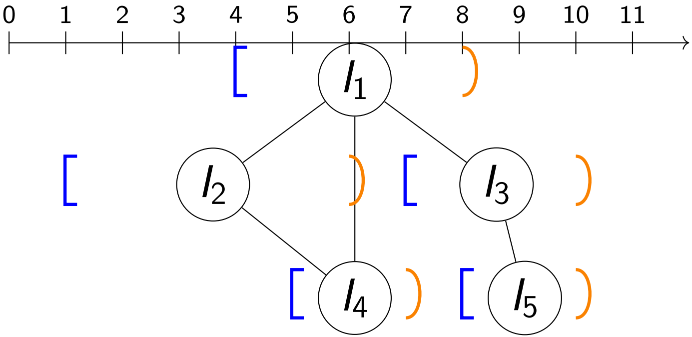
Pathwidth pw



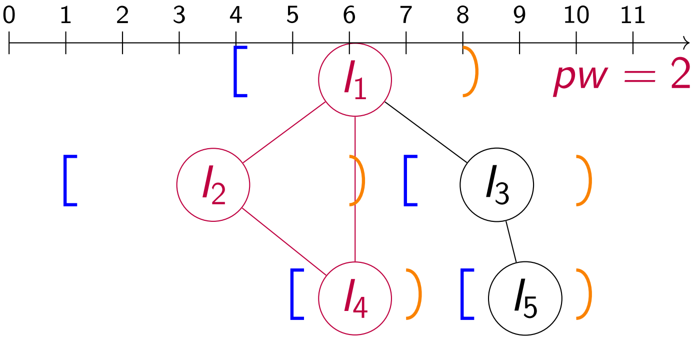
Pathwidth pw



Pathwidth pw



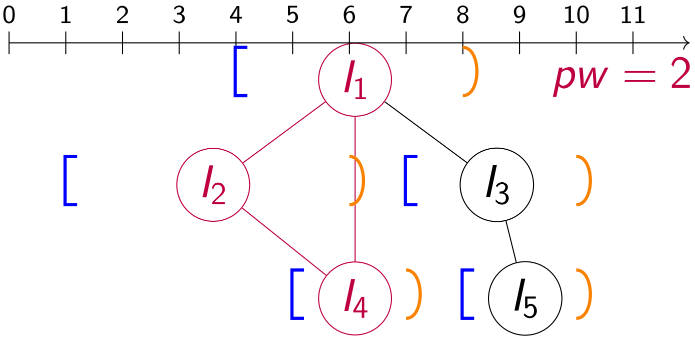
Pathwidth pw



[Bodlaender 93] For any interval graph G:

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

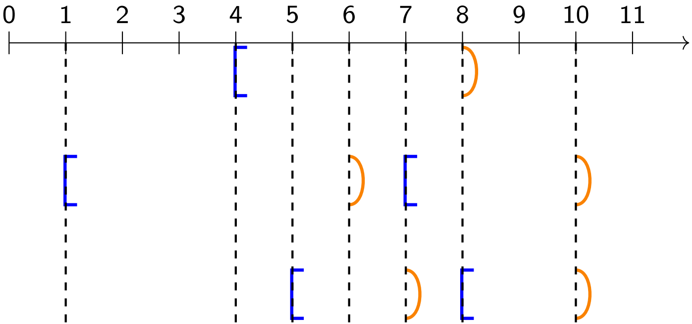
Pathwidth pw



[Bodlaender 93] For any interval graph G:

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

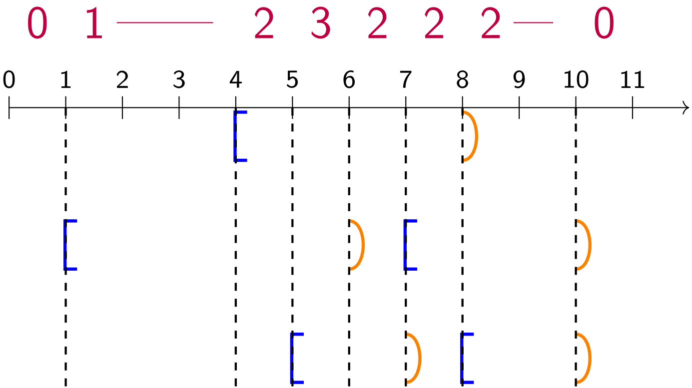
Pathwidth *pw*



[Bodlaender 93] For any interval graph G:

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

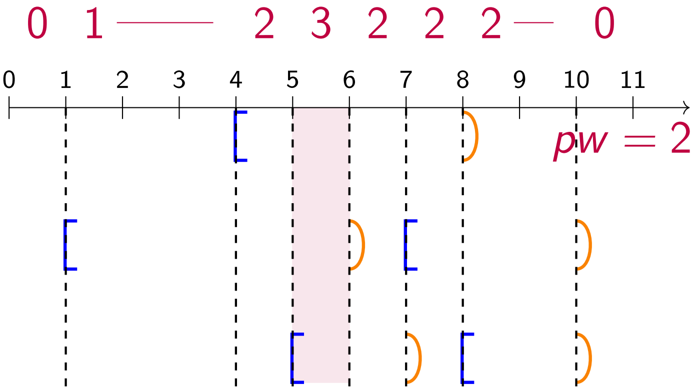
Pathwidth pw



[Bodlaender 93] For any interval graph G:

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

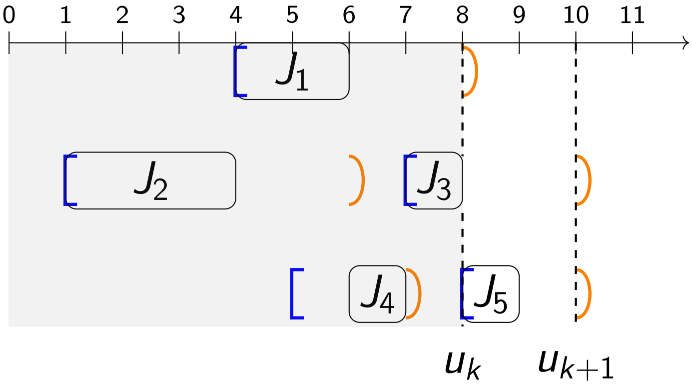
Pathwidth *pw*



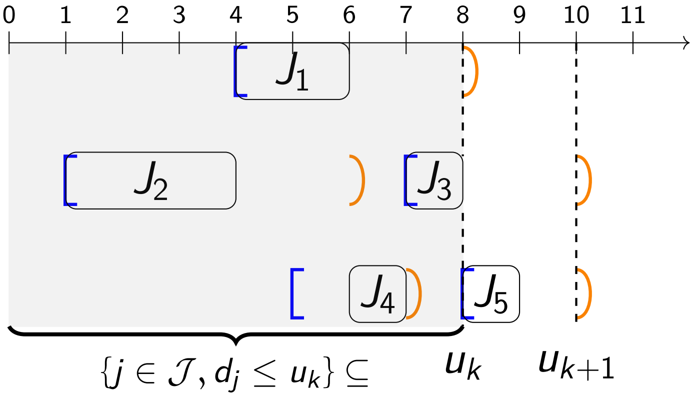
[Bodlaender 93] For any interval graph G:

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

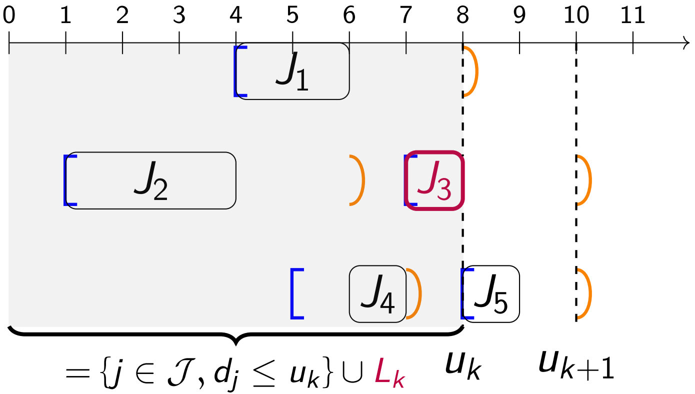
Pathwidth pw



Pathwidth pw

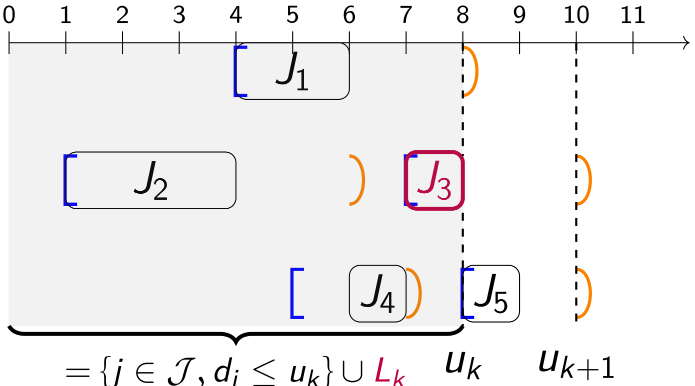


Pathwidth pw



- $J_j \in L_k \implies [u_k, u_{k+1}) \subseteq [r_j, d_j)$: 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

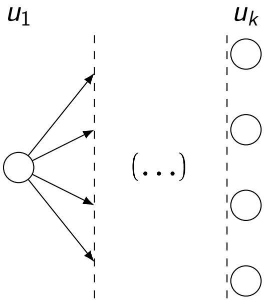
Dynamic programming state (k, L_k)

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

Dynamic programming with pathwidth *pw*

Dynamic programming state (k, L_k)

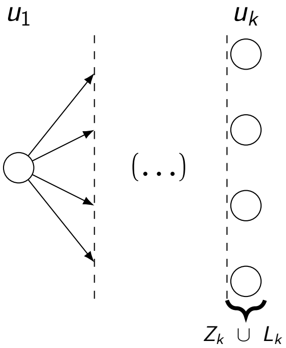
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = subset of jobs completed by time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

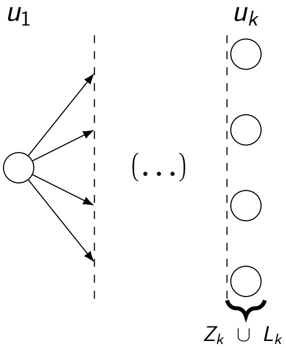
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = subset of jobs completed by time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

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

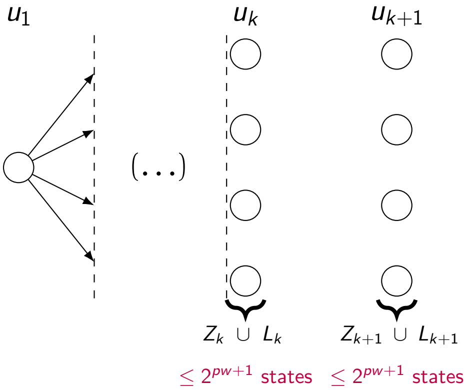


$\leq 2^{pw+1}$ states

Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

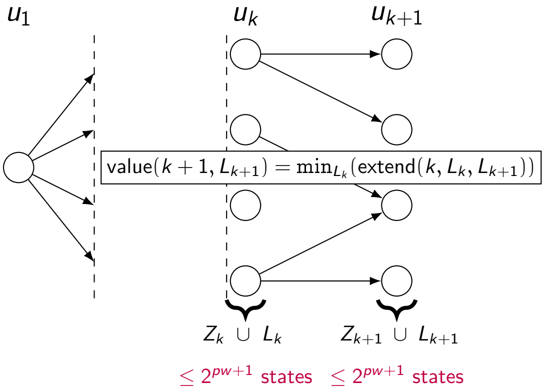
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = subset of jobs completed by time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

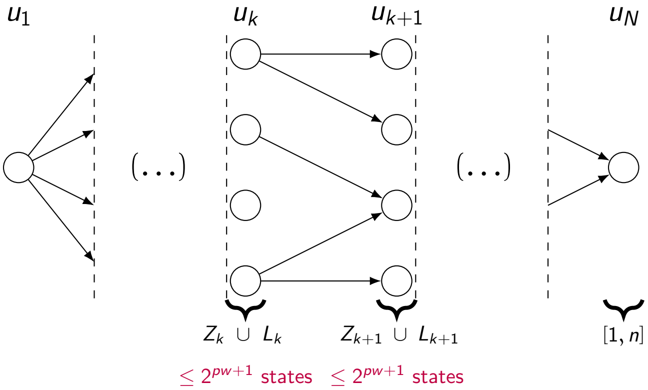
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = subset of jobs completed by time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

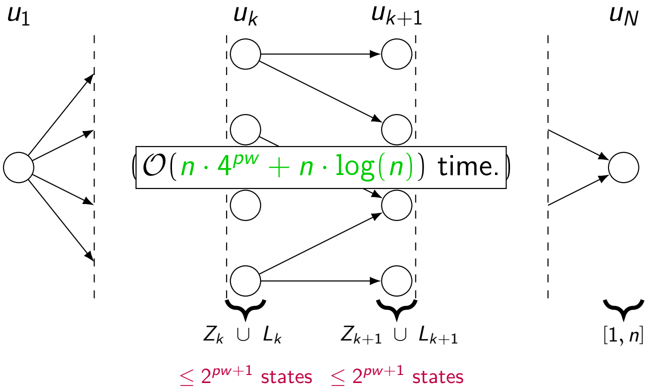
Dynamic programming state (k, L_k)

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



Dynamic programming state (k, L_k)

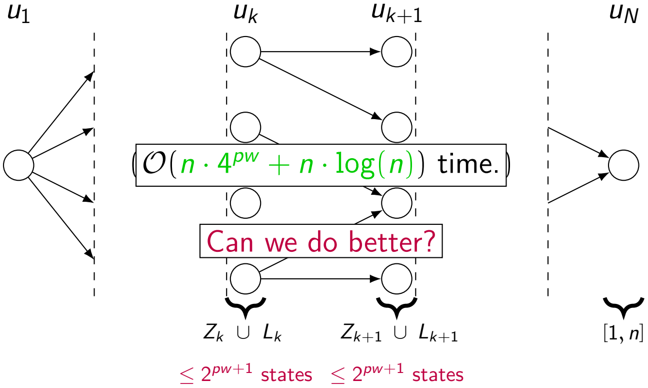
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = subset of jobs completed by time u_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with pathwidth pw

Dynamic programming state (k, L_k)

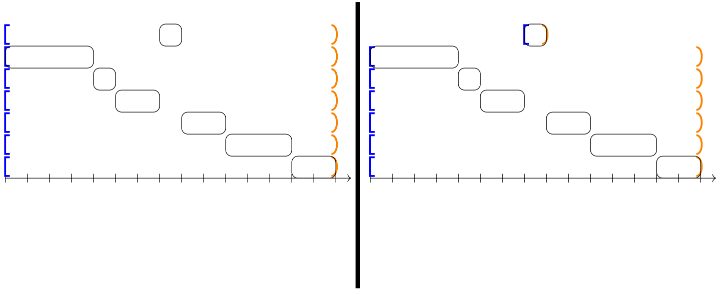
- u_k = associated release date/deadline value.
- $Z_k \cup L_k$ = subset of jobs completed by time u_k.
- State value = minimum makespan among the associated partial schedules.



Outline

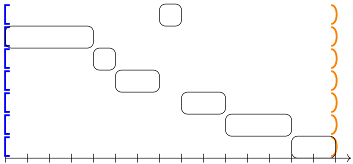
Proper level

- Two single machine instances with time windows.



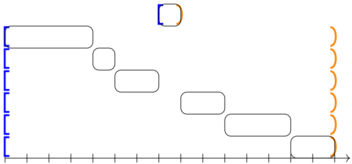
Proper level

- Two single machine instances with time windows.



$$pw = n - 1$$

Interval graph: K_n

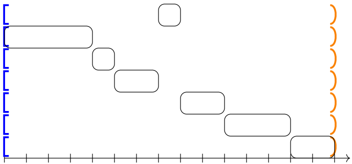


$$pw = n - 1$$

Interval graph: K_n

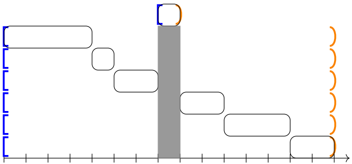
Proper level

- Two single machine instances with time windows.



$pw = n - 1$

Interval graph: K_n

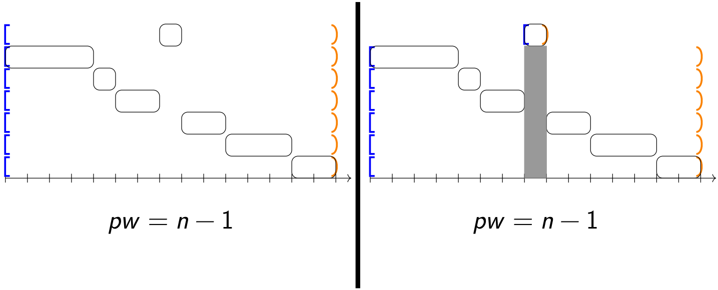


$pw = n - 1$

Interval graph: K_n

Proper level

- Two single machine instances with time windows.

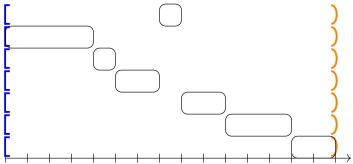


$pw = n - 1$ $\qquad$ $pw = n - 1$

Definition: Proper sets and proper level

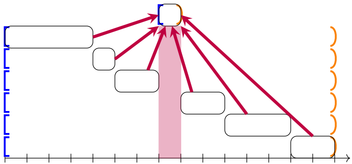
- $\forall J_i \in \mathcal{J}, \Pi_i = \{J_j \in \mathcal{J}, r_j < r_i \wedge d_i < d_j\}$
- $pl = max_{J_i \in \mathcal{J}} \; |\Pi_i|$

- Two single machine instances with time windows.



$pw = n-1$

$pl = 0$



$pw = n-1$

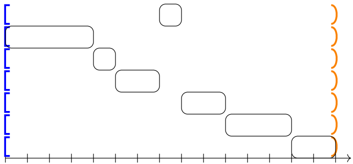
$pl = n-1$

Definition: Proper sets and proper level

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

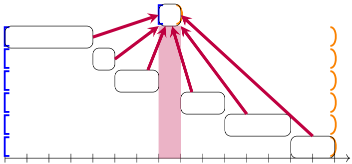
Proper level

- Two single machine instances with time windows.



$pw = n - 1$

$pl = 0$



$pw = n - 1$

$pl = n - 1$

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 et al., 1985].
- $pl = max_{J_i \in \mathcal{J}} |\Pi_i|$ [Proskurowski and Telle, 1999].

The "Earliest Deadline first" rule

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

[Lawler and Moore, 1969] "Earliest Deadline first" rule (ED)

Schedule τ follows the (ED) rule if and only if:

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

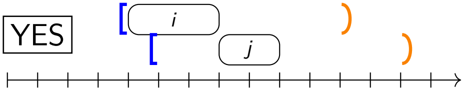
The "Earliest Deadline first" rule

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

[Lawler and Moore, 1969] "Earliest Deadline first" rule (ED)

Schedule τ follows the (ED) rule if and only if:

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



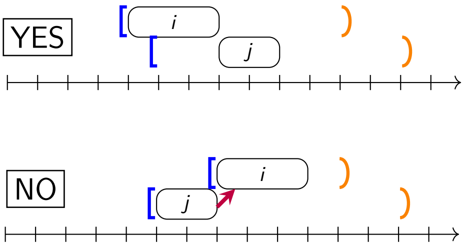
The “Earliest Deadline first” rule

- “$\prec$” = lexicographic order (nondecr. d_j, nondecr. r_j) on $\mathcal{J}$.
- Renaming the jobs of $\mathcal{J}$ in $[1, n]$ accordingly to $\prec$.

[Lawler and Moore, 1969] “Earliest Deadline first” rule (ED)

Schedule τ follows the (ED) rule if and only if:

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



Relaxing the “Earliest Deadline first” rule

Definition: “weak Earliest Deadline first” rule (wED)

Schedule τ follows the (wED) rule if and only if:
$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies$

(1) $\tau(i) < \tau(j)$, or

(2) $j \in \Pi_i$, or

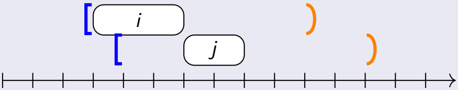
(3) $\exists h \prec i, \tau(j) < \tau(h) < \tau(i)$.

Relaxing the “Earliest Deadline first” rule

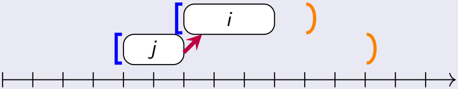
Definition: “weak Earliest Deadline first” rule (wED)

Schedule τ follows the (wED) rule if and only if:
$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies$

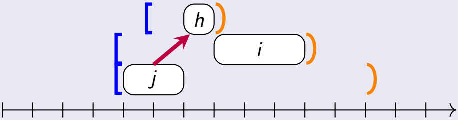
(1) $\tau(i) < \tau(j)$, or



(2) $j \in \Pi_i$, or



(3) $\exists h \prec i, \tau(j) < \tau(h) < \tau(i)$.

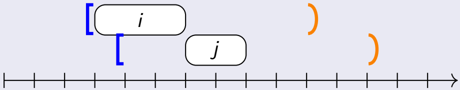


Relaxing the "Earliest Deadline first" rule

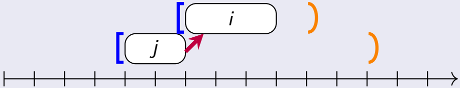
Definition: "weak Earliest Deadline first" rule (wED)

Schedule τ follows the (wED) rule if and only if:
$\forall (i,j) \in \mathcal{J}^2, i \prec j \implies$

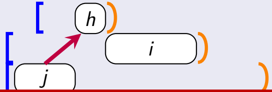
(1) $\tau(i) < \tau(j)$, or



(2) $j \in \Pi_i$, or



(3) $\exists h \prec i, \tau(j) < \tau(h) < \tau(i)$.



Proposition

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

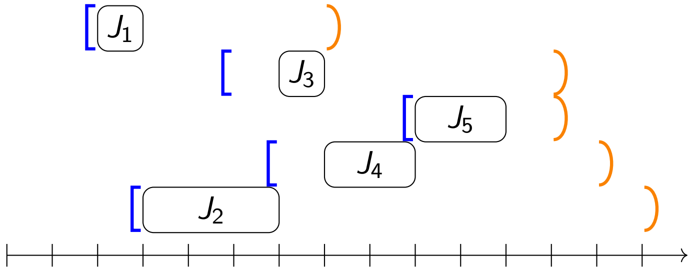
Definition: Summit

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

Summits

Definition: Summit

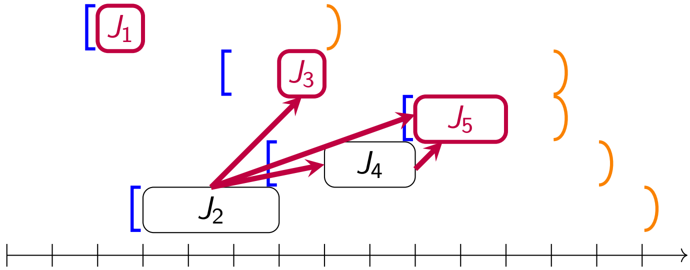
A summit is a job i in $\mathcal{J}$ such that: $\forall j \in \mathcal{J}, i \notin \Pi_j$.



Summits

Definition: Summit

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



Lemma

Let $s_1 \prec \ldots \prec s_N$ be the summits of the instance ($N \leq n$).
If a schedule τ follows the (wED) rule, then:

(i) τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$,

(ii) if $i \prec s_k$ then $\tau(i) < \tau(s_k)$,

(iii) if $s_k \prec j$ and $\tau(j) < \tau(s_k)$ then $j \in \Pi_{s_k}$.

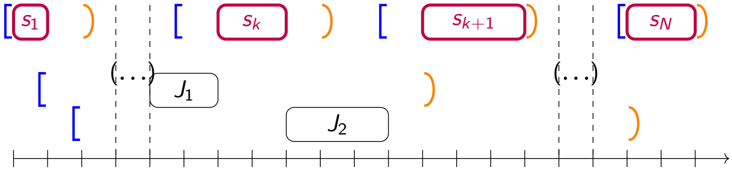
Lemma

Let $s_1 \prec \ldots \prec s_N$ be the summits of the instance ($N \leq n$).
If a schedule τ follows the (wED) rule, then:

(i) τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$,

(ii) if $i \prec s_k$ then $\tau(i) < \tau(s_k)$,

(iii) if $s_k \prec j$ and $\tau(j) < \tau(s_k)$ then $j \in \Pi_{s_k}$.



Summit-based schedule enumeration

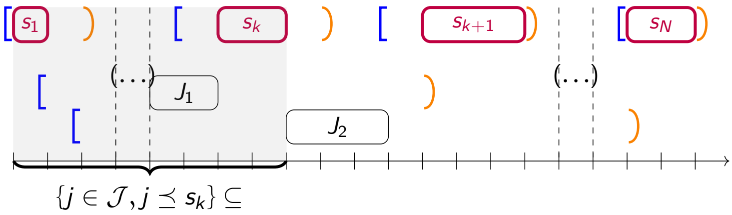
Lemma

Let $s_1 \prec \ldots \prec s_N$ be the summits of the instance ($N \leq n$).
If a schedule τ follows the (wED) rule, then:

(i) τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$,

(ii) if $i \prec s_k$ then $\tau(i) < \tau(s_k)$,

(iii) if $s_k \prec j$ and $\tau(j) < \tau(s_k)$ then $j \in \Pi_{s_k}$.



Summit-based schedule enumeration

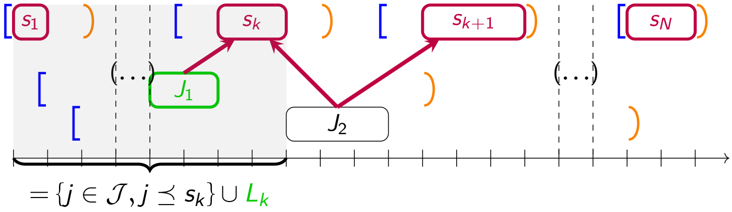
Lemma

Let $s_1 \prec \ldots \prec s_N$ be the summits of the instance ($N \leq n$).
If a schedule τ follows the (wED) rule, then:

(i) τ is of the form $T_1 s_1 \ldots s_k T_{k+1} s_{k+1} \ldots s_N T_{N+1}$,

(ii) if $i \prec s_k$ then $\tau(i) < \tau(s_k)$,

(iii) if $s_k \prec j$ and $\tau(j) < \tau(s_k)$ then $j \in \Pi_{s_k}$.



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

Dynamic programming with proper level pl

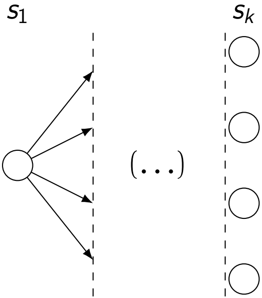
Dynamic programming state (k, L_k)

- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.

Dynamic programming with proper level pl

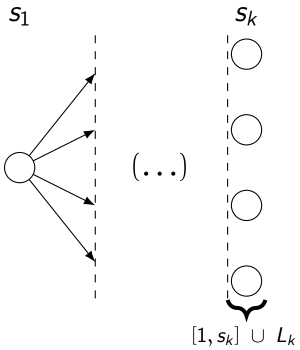
Dynamic programming state (k, L_k)

- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming state (k, L_k)

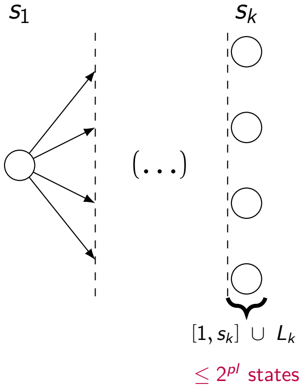
- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with proper level pl

Dynamic programming state (k, L_k)

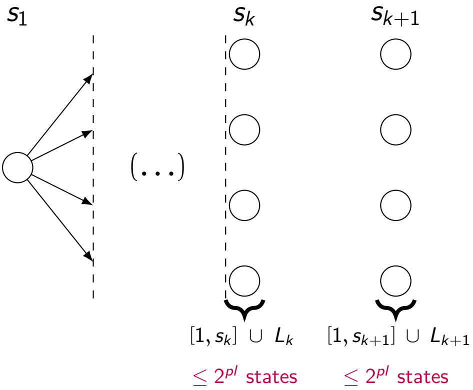
- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with proper level pl

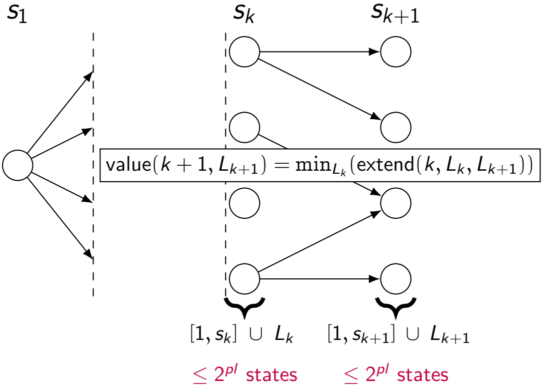
Dynamic programming state (k, L_k)

- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming state (k, L_k)

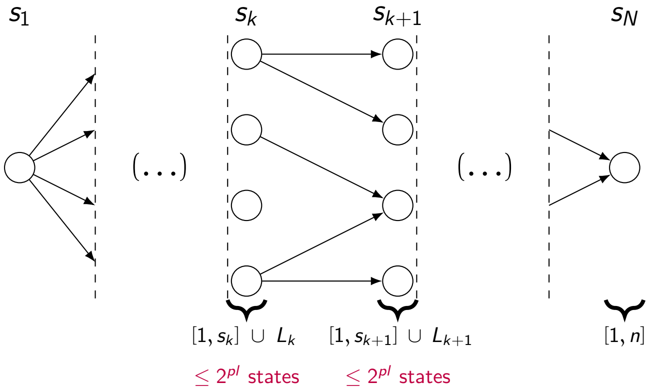
- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with proper level pl

Dynamic programming state (k, L_k)

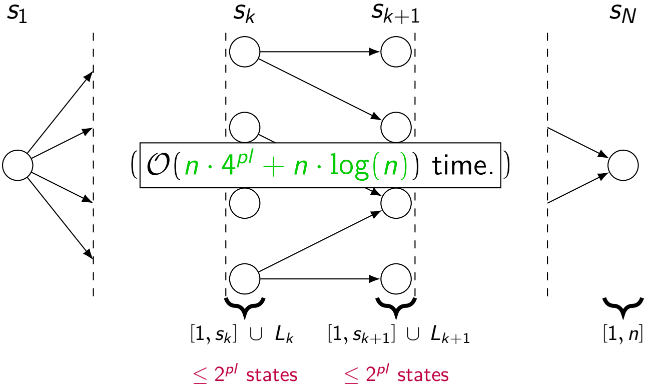
- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with proper level pl

Dynamic programming state (k, L_k)

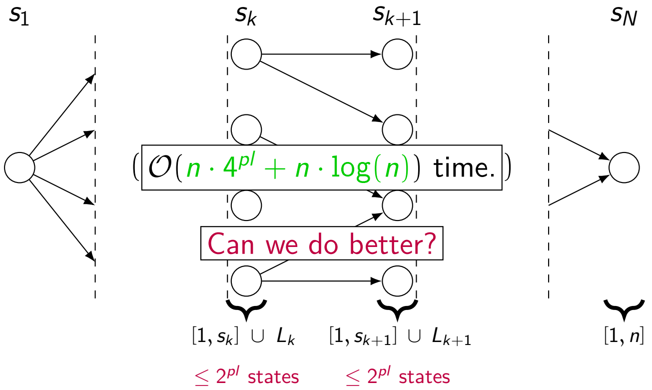
- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Dynamic programming with proper level pl

Dynamic programming state (k, L_k)

- s_k = associated summit.
- $[1, s_k] \cup L_k$ = subset of jobs completed up to summit s_k.
- State value = minimum makespan among the associated partial schedules.



Outline

1. Introduction
2. Vertex cover number
3. Pathwidth
4. Proper level
5. Conclusion
6. References

Takeaway

- Better understanding of scheduling problems.
- More efficient exact algorithms.

Conclusion

Takeaway

- Better understanding of scheduling problems.
- More efficient exact algorithms.

Going further

- $4^{pl} \longrightarrow 2^{pl}$?
- Lower bound on $f(k)$: **fine-grained complexity**.
- Poly-time compression to $g(k)$ size: **kernelization**.
- Parameterized hardness: para-NP, W-hierarchy, XNLP.

Conclusion

Takeaway

- Better understanding of scheduling problems.
- More efficient exact algorithms.

Going further

- $4^{pl} \longrightarrow 2^{pl}$?
- Lower bound on $f(k)$: **fine-grained complexity**.
- Poly-time compression to $g(k)$ size: **kernelization**.
- Parameterized hardness: para-NP, W-hierarchy, XNLP.

Recent news

- IWOCA (with Bodlaender): scheduling with precedence delays.
- FUN!

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

$$pl \longrightarrow pw \longrightarrow vc$$

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

$$
\begin{tikzcd}
 & cw \arrow[dr] & & & \#r_j+\#d_j \arrow[dr] & \\
k\text{-}nested \arrow[r] & pl \arrow[r] & pw \arrow[r] \arrow[dr] & vc \arrow[rr] & & D \\
 & & p_{max} \arrow[r] & pw + p_{max} \arrow[r] & \sigma \arrow[ur] &
\end{tikzcd}
$$

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

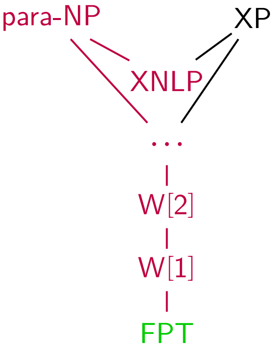
cw: clique-width of the time window interval graph.

p_{max}: maximum processing time.

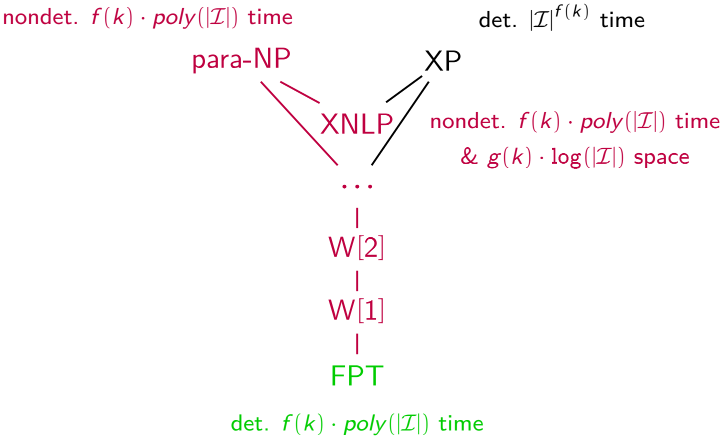
$\#r_j+\#d_j$: number of distinct release date and deadline values.

σ slack: $\max_j(d_j - r_j - p_j)$.

D: maximum deadline.


para-NP
XP
XNLP
...
W[2]
W[1]
FPT

Parameterized complexity classes



References I

Erschler, J., Fontan, G., and Merce, C. (1985).
Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine.
RAIRO-Operations Research, 19(1):15–26.

Flum, J. and Grohe, M. (1998).
Parameterized Complexity Theory.
Springer.

Lawler, E. L. and Moore, J. M. (1969).
A functional equation and its application to resource allocation and sequencing problems.
Management science, 16(1):77–84.

Lenstra, J., Rinnooy Kan, A., and Brucker, P. (1977).
Complexity of machine scheduling problems.
Ann. of Discrete Math., 1:343–362.

Proskurowski, A. and Telle, J. A. (1999).
Classes of graphs with restricted interval models.
Discrete Mathematics & Theoretical Computer Science, 3.