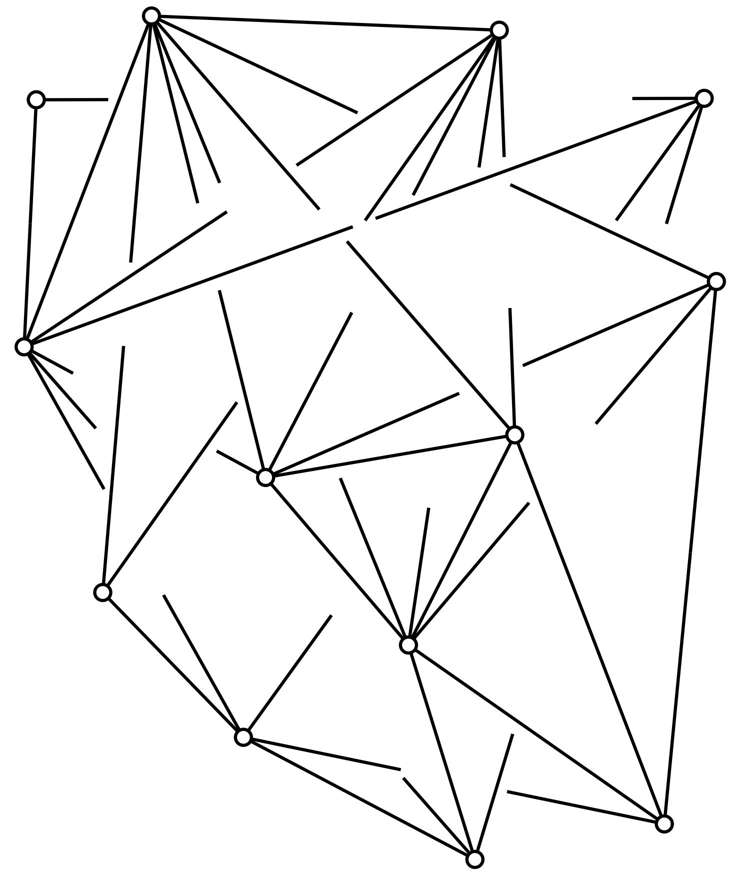
Maximizing Ink in Partial Edge Drawings of k-plane Graphs

Matthias Hummel, Fabian Klute,
Soeren Nickel, *Martin Nöllenburg*



GD 2019 · September 19, 2019

TU WIEN ALGORITHMS AND COMPLEXITY GROUP

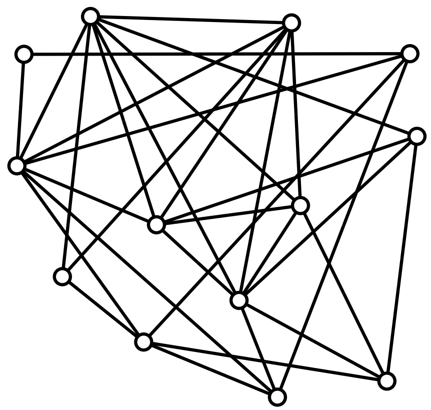
Partial Edge Drawings (PED)

How to draw non-planar graphs?

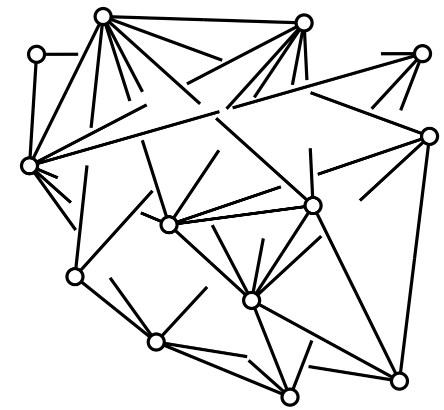
Just *hide* the edge crossings!

[Becker et al. TVCG'95], [Bruckdorfer, Kaufmann FUN'12]

Input:
Straight-line graph drawing with crossings



Output:
"Crossing-free" partial edge drawing (PED)



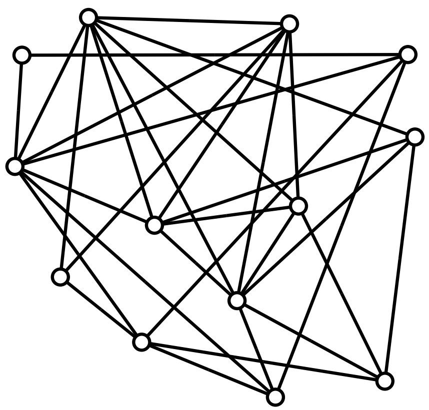
Partial Edge Drawings (PED)

How to draw non-planar graphs?

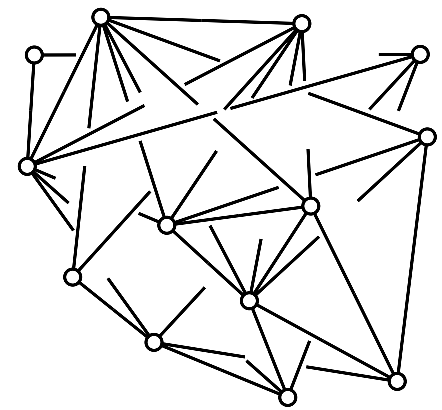
Just *hide* the edge crossings!

[Becker et al. TVCG'95], [Bruckdorfer, Kaufmann FUN'12]

Input:
Straight-line graph drawing with crossings

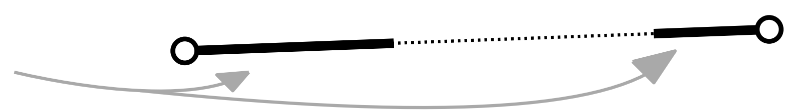


Output:
"Crossing-free" partial edge drawing (PED)



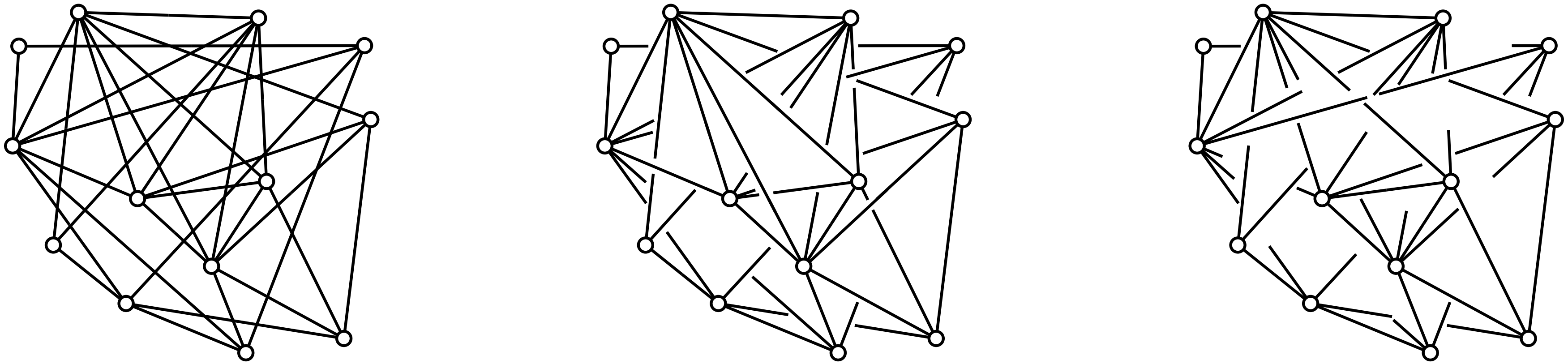
Properties:

- edges are drawn partially with middle part removed
- pairs of opposing **stubs**



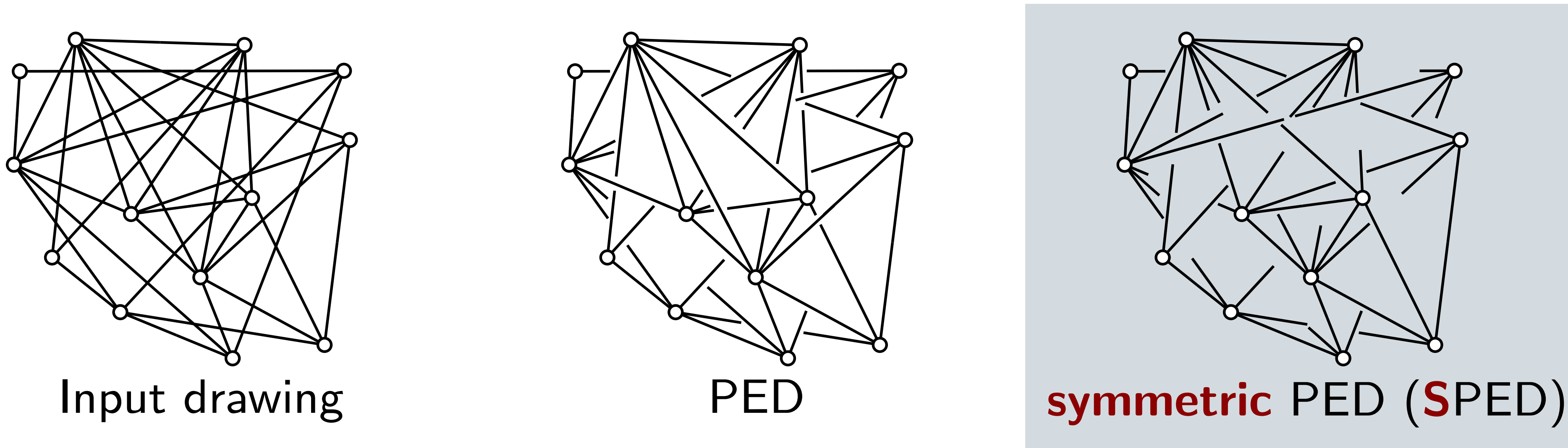
- relies on closure and continuation principles in Gestalt theory
- user studies confirmed that PEDs reduce clutter and remain readable for long enough stubs [Bruckdorfer et al. GD'15], [Burch et al. GD'11]

Symmetric Partial Edge Drawings (SPED)



Input drawing | PED | **symmetric** PED (**S**PED)

Symmetric Partial Edge Drawings (SPED)

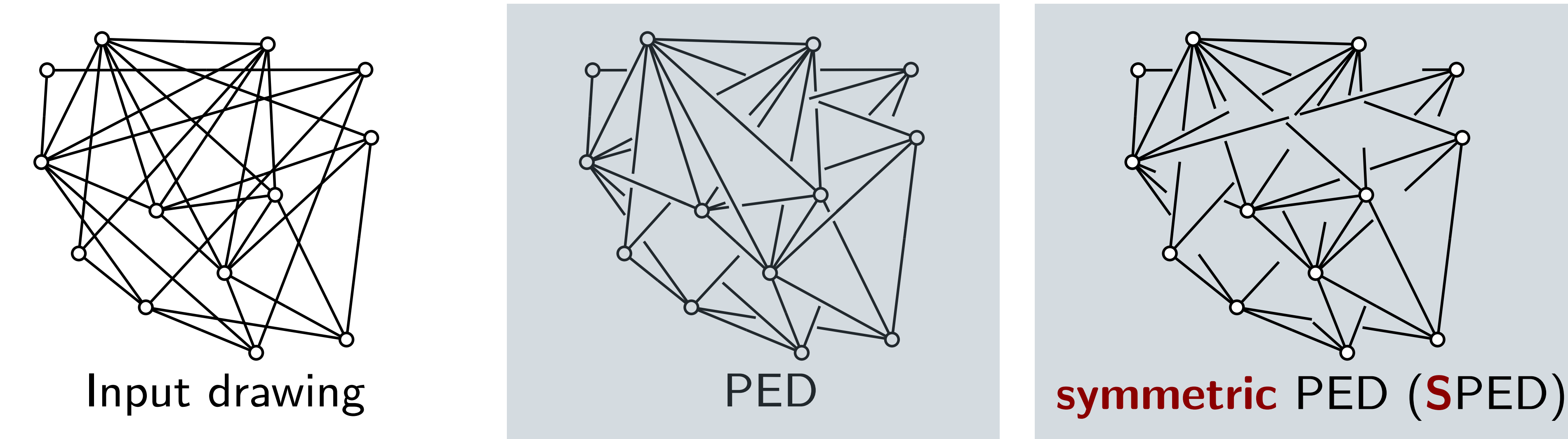


SPED:

- both stubs of an edge have the same length
- identical stub lengths can facilitate finding adjacencies

Symmetric Partial Edge Drawings (SPED)

ac



SPED:

- both stubs of an edge have the same length
- identical stub lengths can facilitate finding adjacencies

Optimization problem: maximize total stub length/drawn ink

$\rightarrow$ **MaxPED** and **MaxSPED**

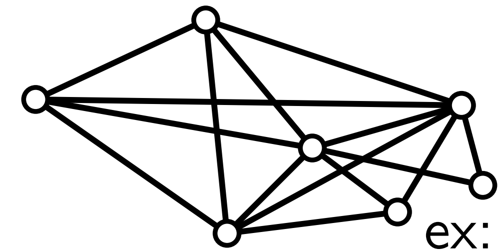
- show as much information as possible without crossings

Overview of Results

Given: k-plane* straight-line drawing Γ
Find: maximum-ink (S)PED of Γ

Overview of Results

Given: k-plane* straight-line drawing Γ
Find: maximum-ink (S)PED of Γ

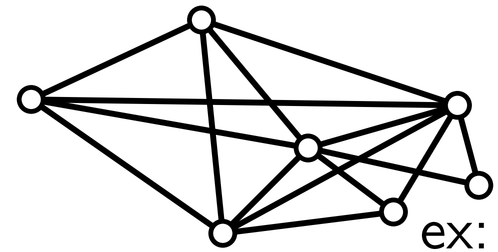


ex: $k = 2$

*: k-plane drawing: every edge crossed by at most k other edges

Overview of Results

Given: k-plane[$\star$] straight-line drawing Γ
Find: maximum-ink (S)PED of Γ

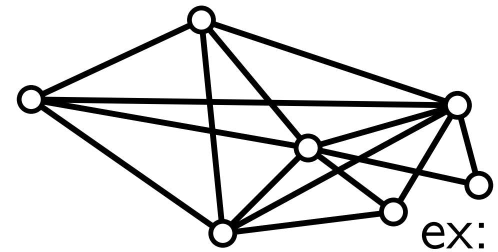

ex: $k = 2$

$\star$: k-plane drawing: every edge crossed by at most k other edges

	$k = 2$	$k = 3$	$k \geq 4$	arbitrary k
MaxSPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]			NP-hard [Bruckdorfer PhD'15]
MaxPED				

Overview of Results

Given: k-plane* straight-line drawing Γ
Find: maximum-ink (S)PED of Γ



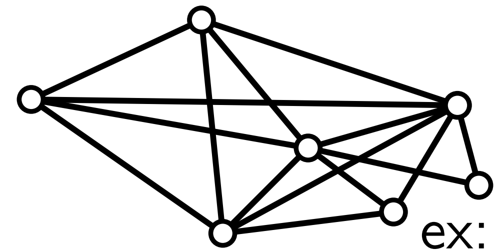
ex: $k = 2$

$\star$: k-plane drawing: every edge crossed by at most k other edges

	$k = 2$	$k = 3$	$k \geq 4$	arbitrary k
MaxSPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]	**NP-hard**	**NP-hard**	NP-hard [Bruckdorfer PhD'15]
MaxPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]		**NP-hard**	

Overview of Results

Given: k-plane* straight-line drawing Γ
Find: maximum-ink (S)PED of Γ



ex: $k = 2$

$\star$: k-plane drawing: every edge crossed by at most k other edges

	$k = 2$	$k = 3$	$k \geq 4$	arbitrary k
MaxSPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]	**NP-hard**	**NP-hard**	NP-hard [Bruckdorfer PhD'15]
MaxPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]		**NP-hard**	

Dynamic Programming if edge intersection graph

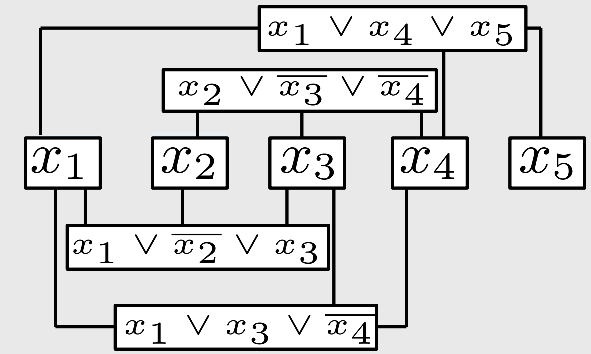
- is a **tree**, or more generally
- has **bounded treewidth**

NP-Hardness

NP-Hardness of MaxSPED

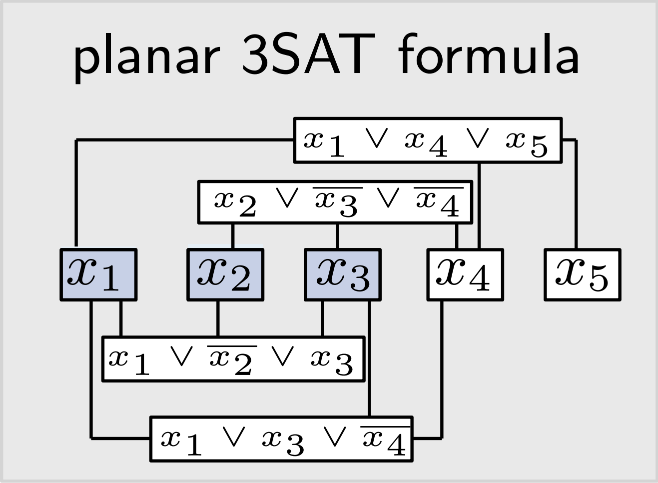
- reduction from PLANAR 3SAT
- gadget-based reduction

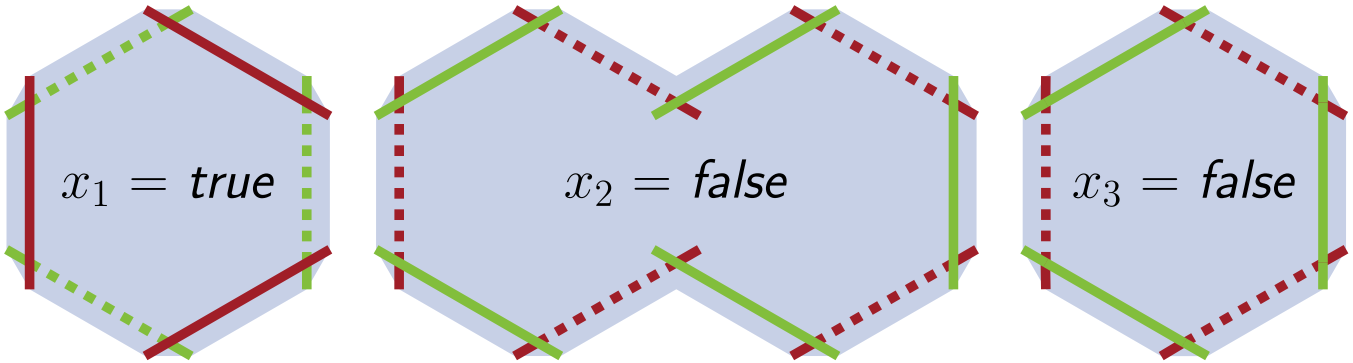
planar 3SAT formula



NP-Hardness of MaxSPED

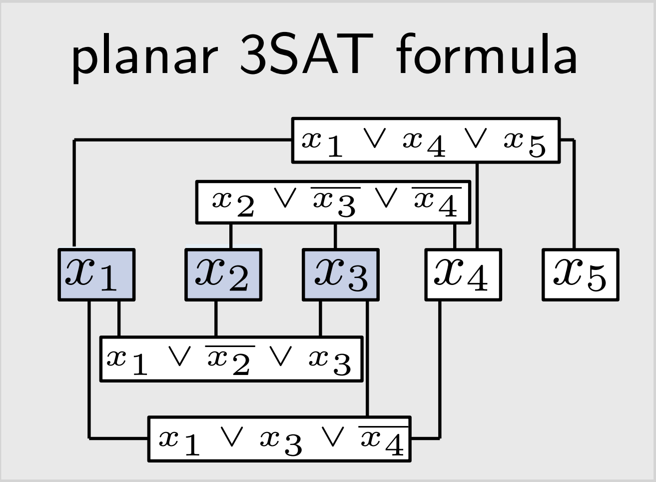
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states

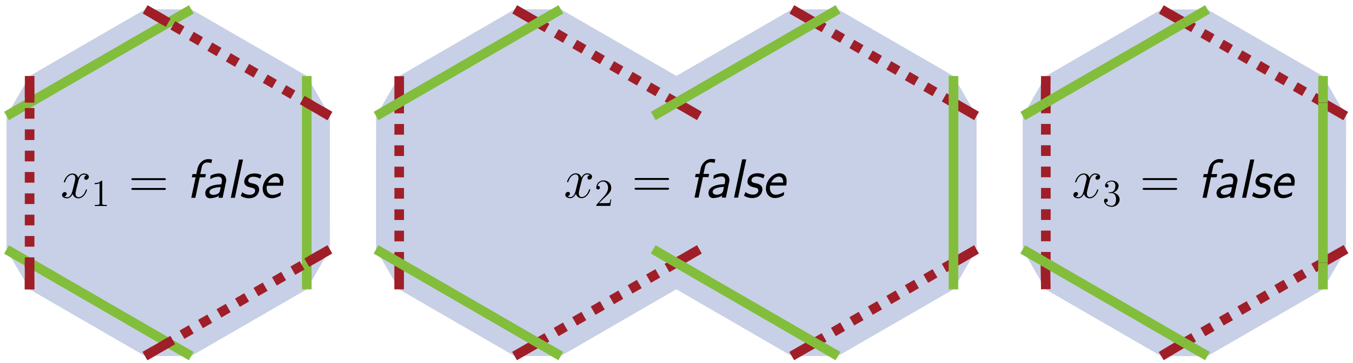




NP-Hardness of MaxSPED

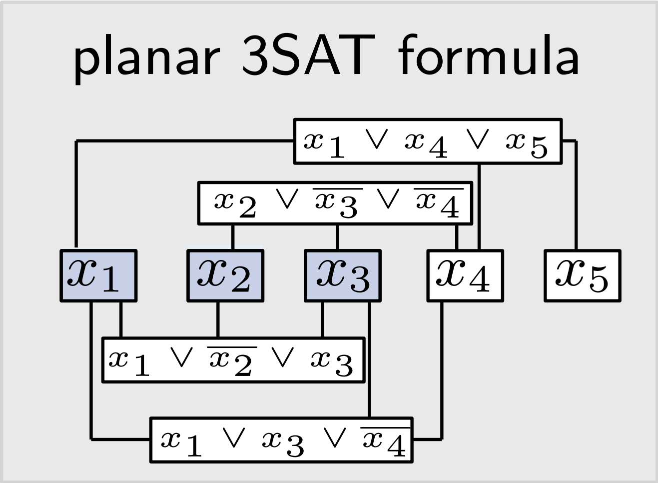
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states

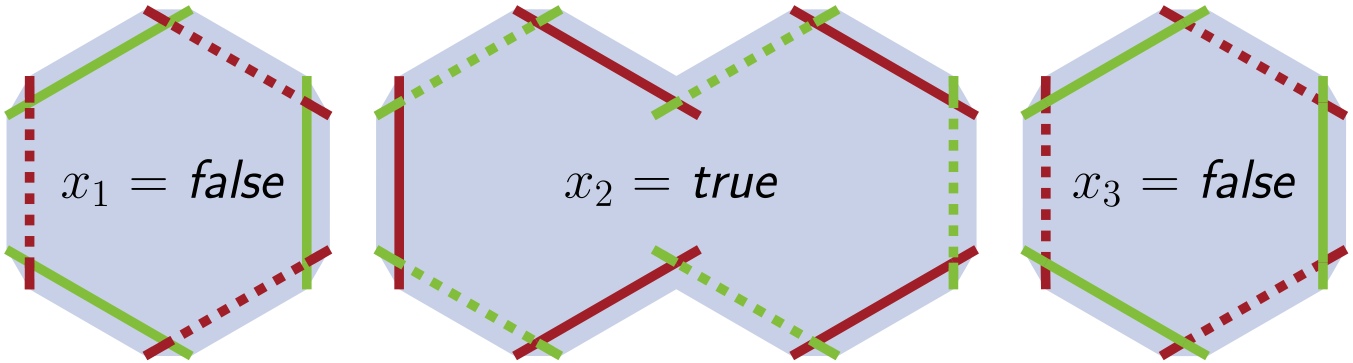




NP-Hardness of MaxSPED

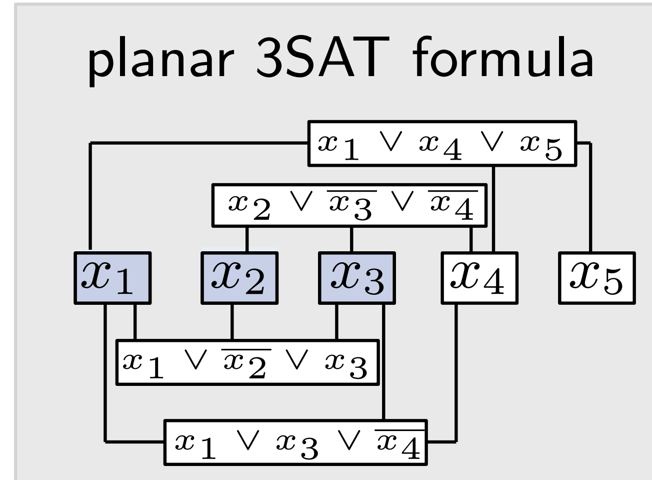
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states

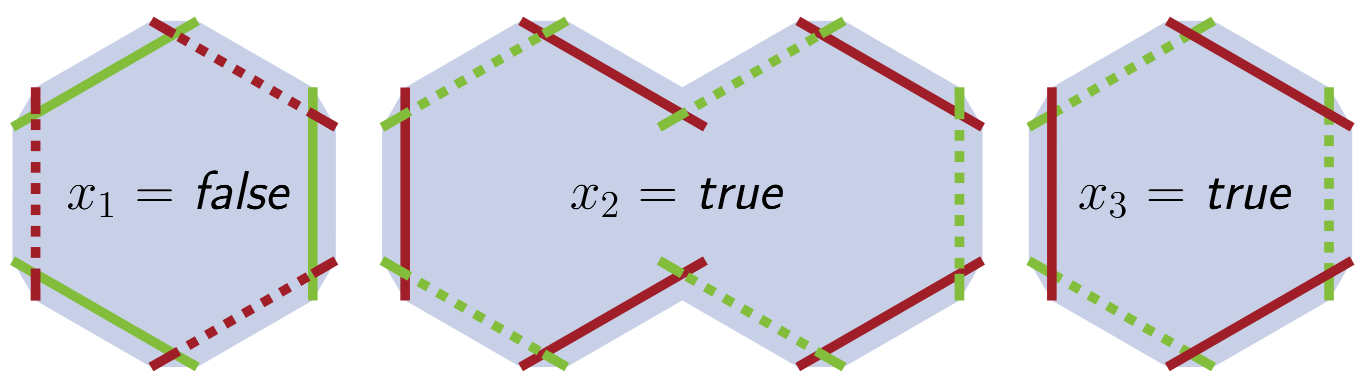




NP-Hardness of MaxSPED

- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states

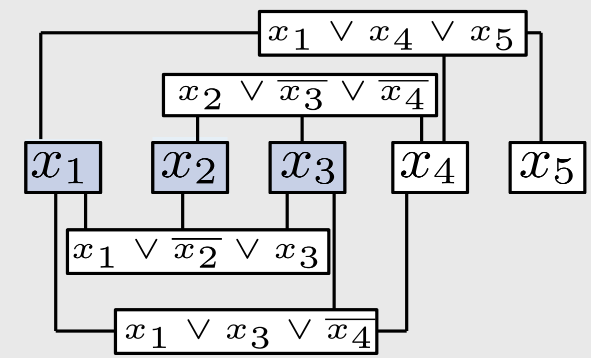


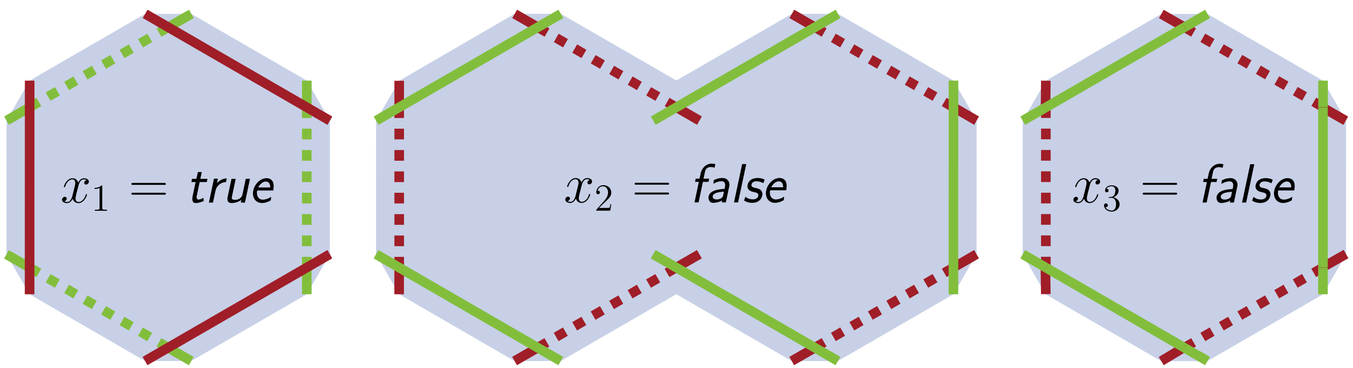


NP-Hardness of MaxSPED

- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states

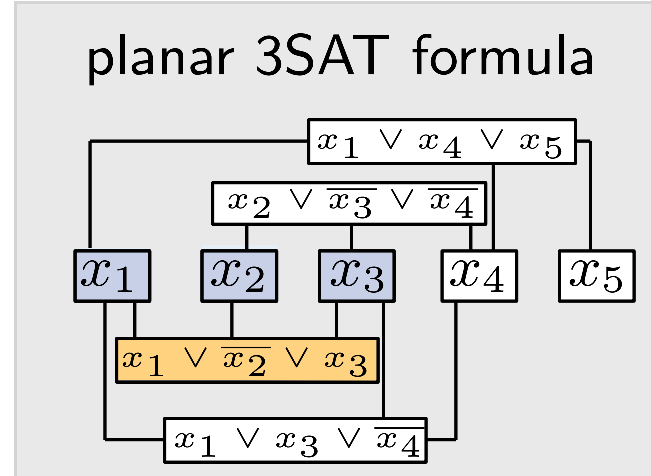
planar 3SAT formula

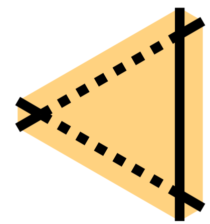


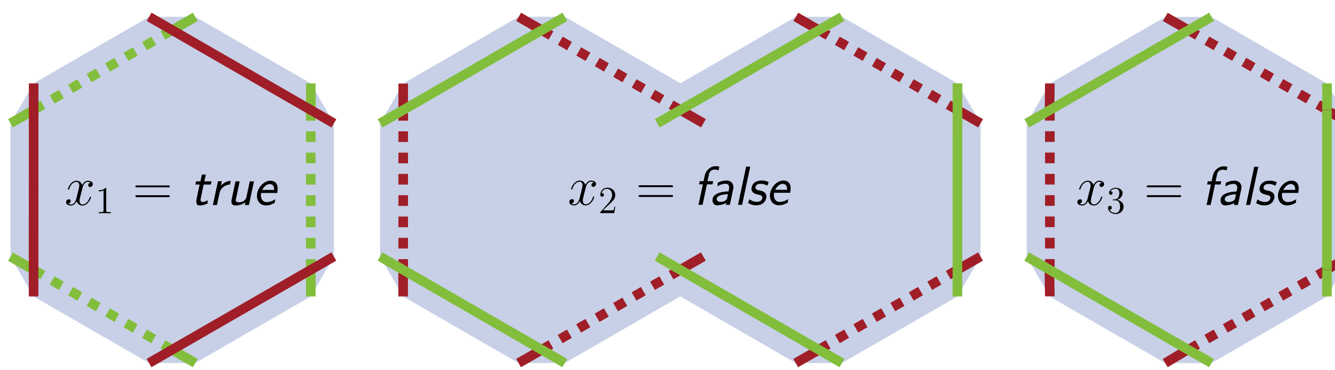


NP-Hardness of MaxSPED

- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states

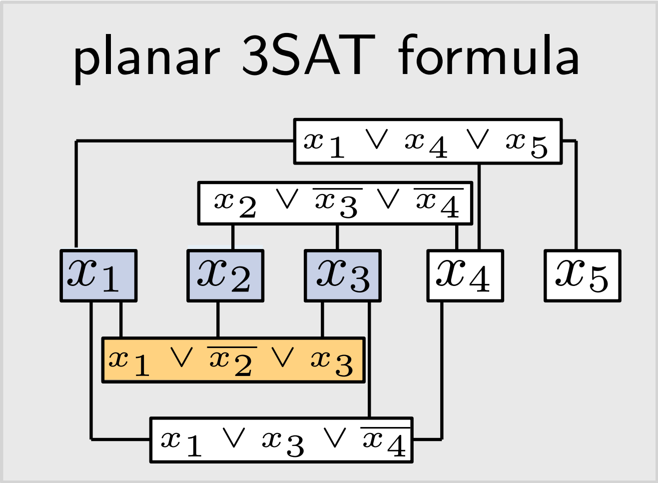


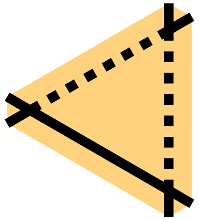


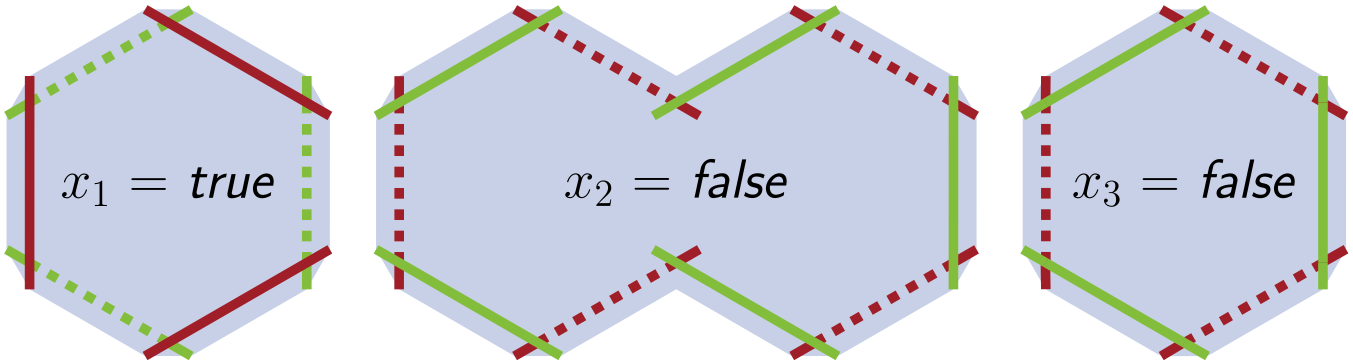


NP-Hardness of MaxSPED

- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states

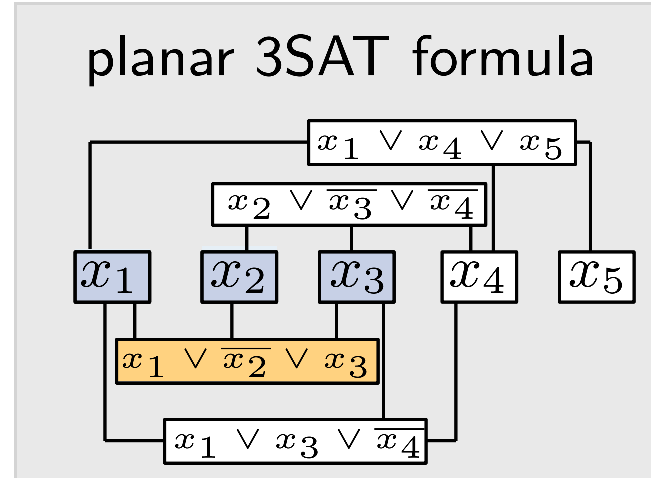


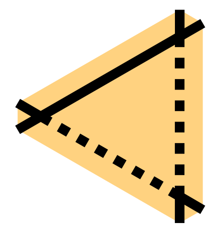


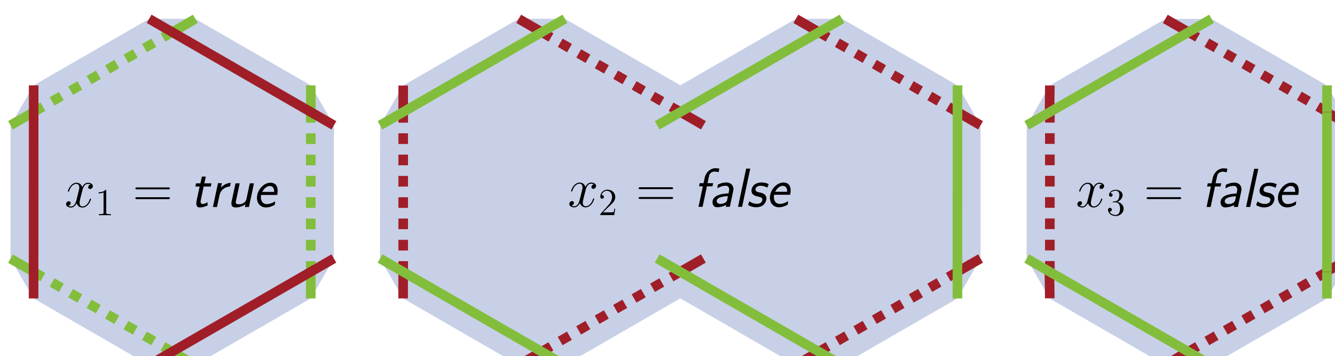


NP-Hardness of MaxSPED

- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states

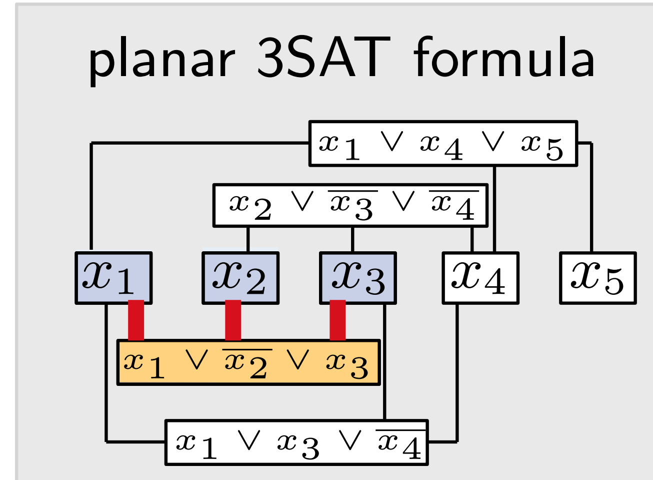


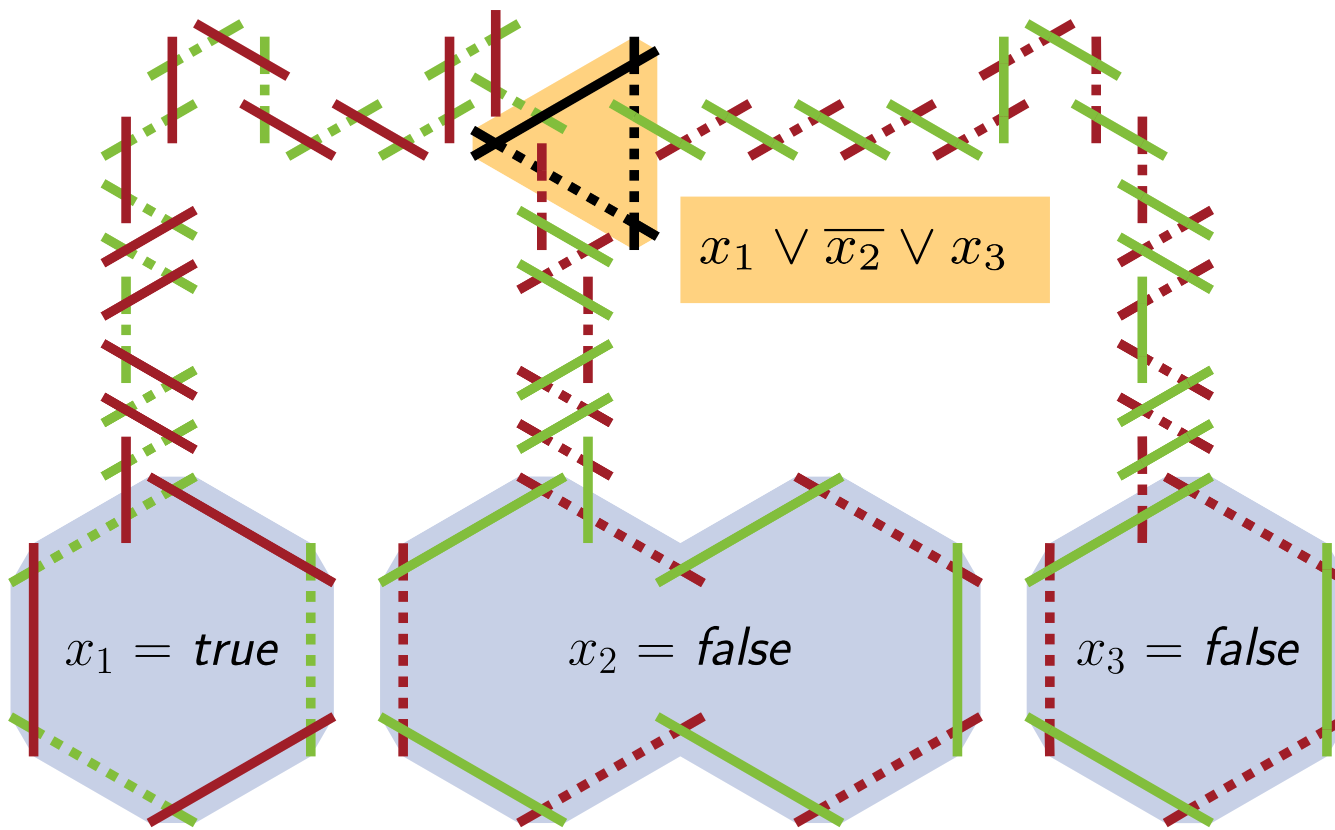




NP-Hardness of MaxSPED

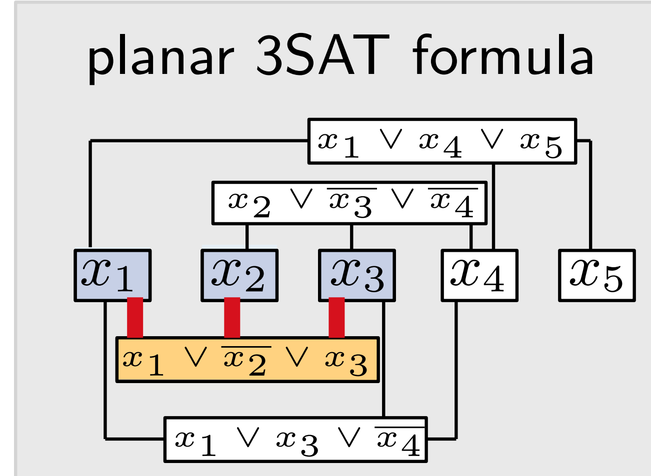
- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states

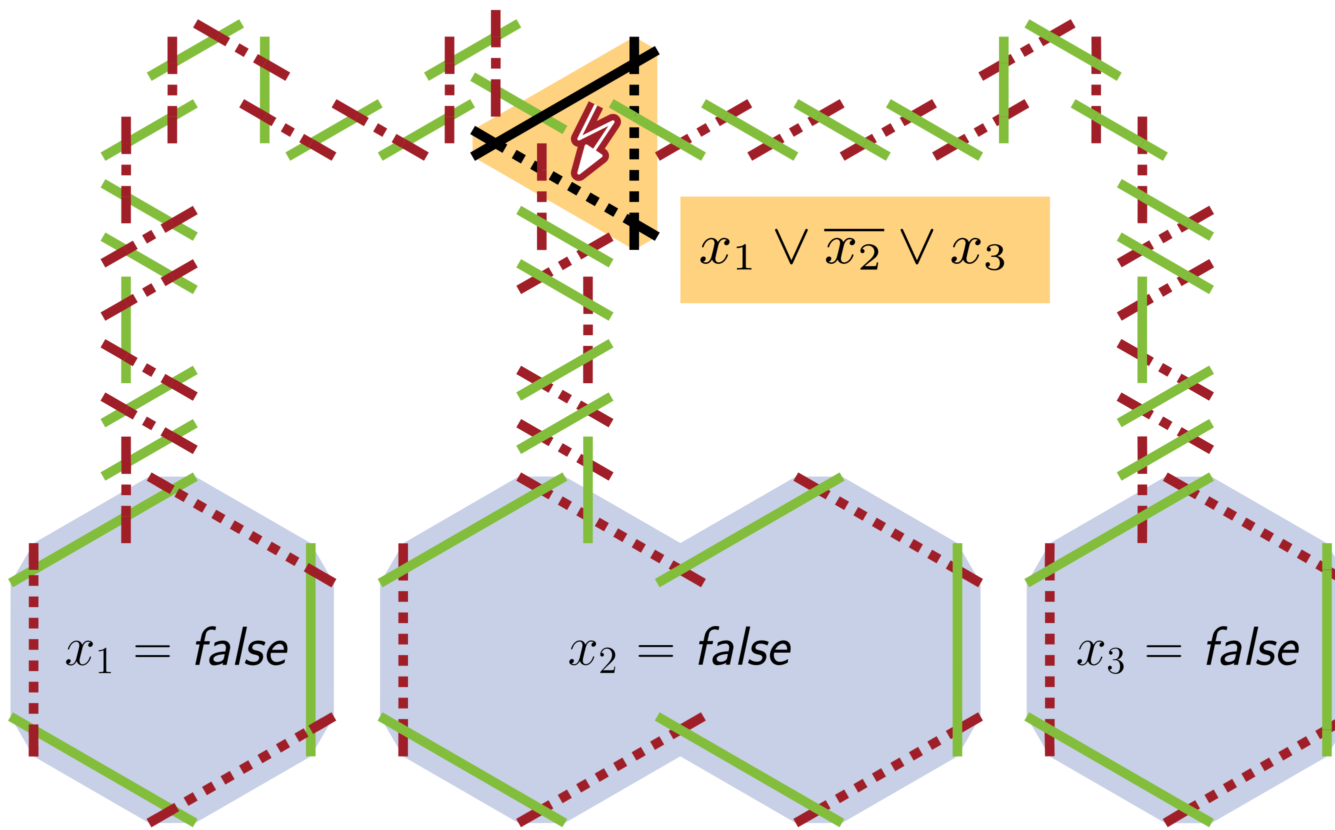




NP-Hardness of MaxSPED

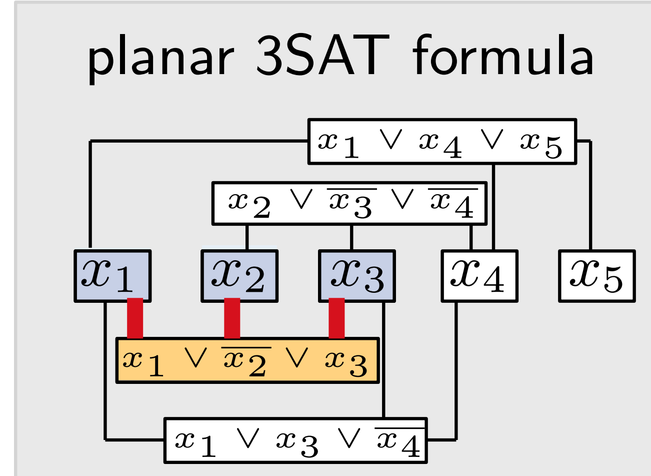
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states

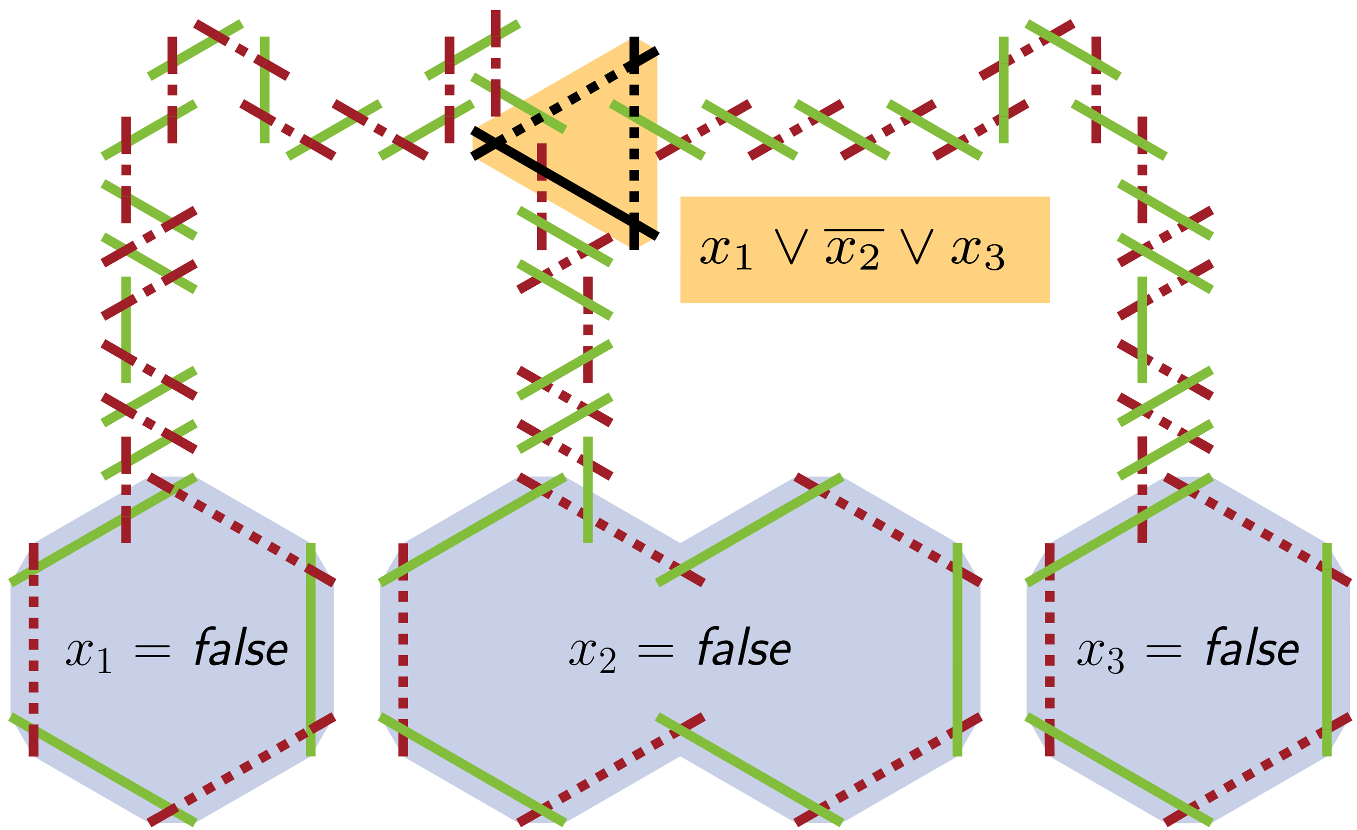




NP-Hardness of MaxSPED

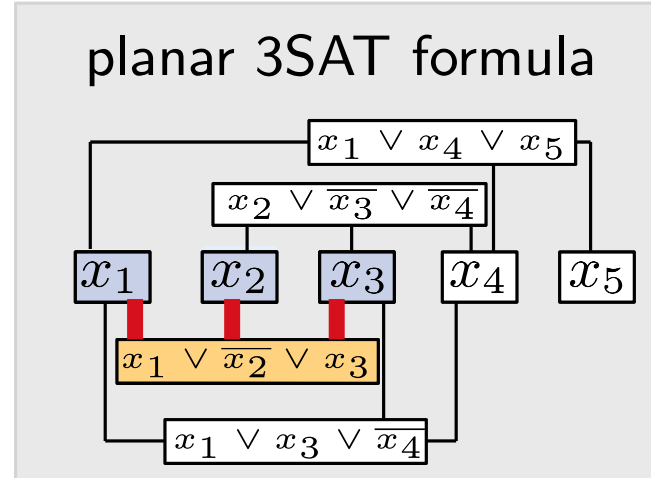
- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states

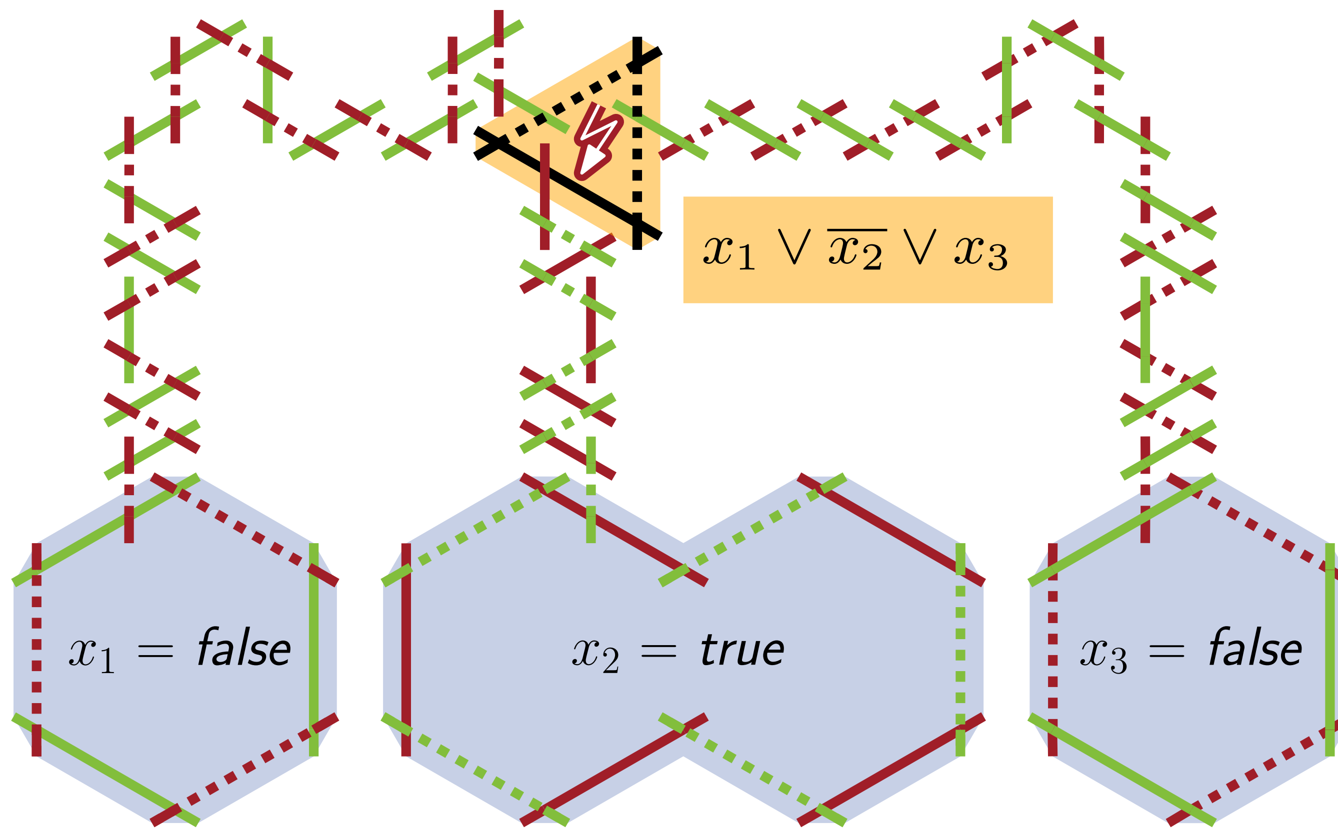




NP-Hardness of MaxSPED

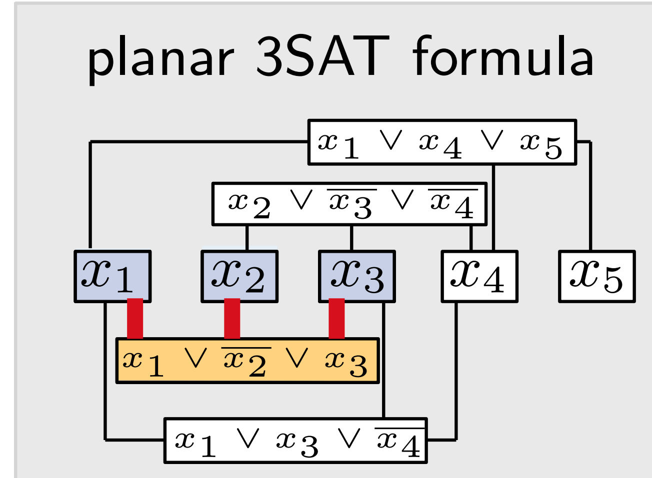
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states

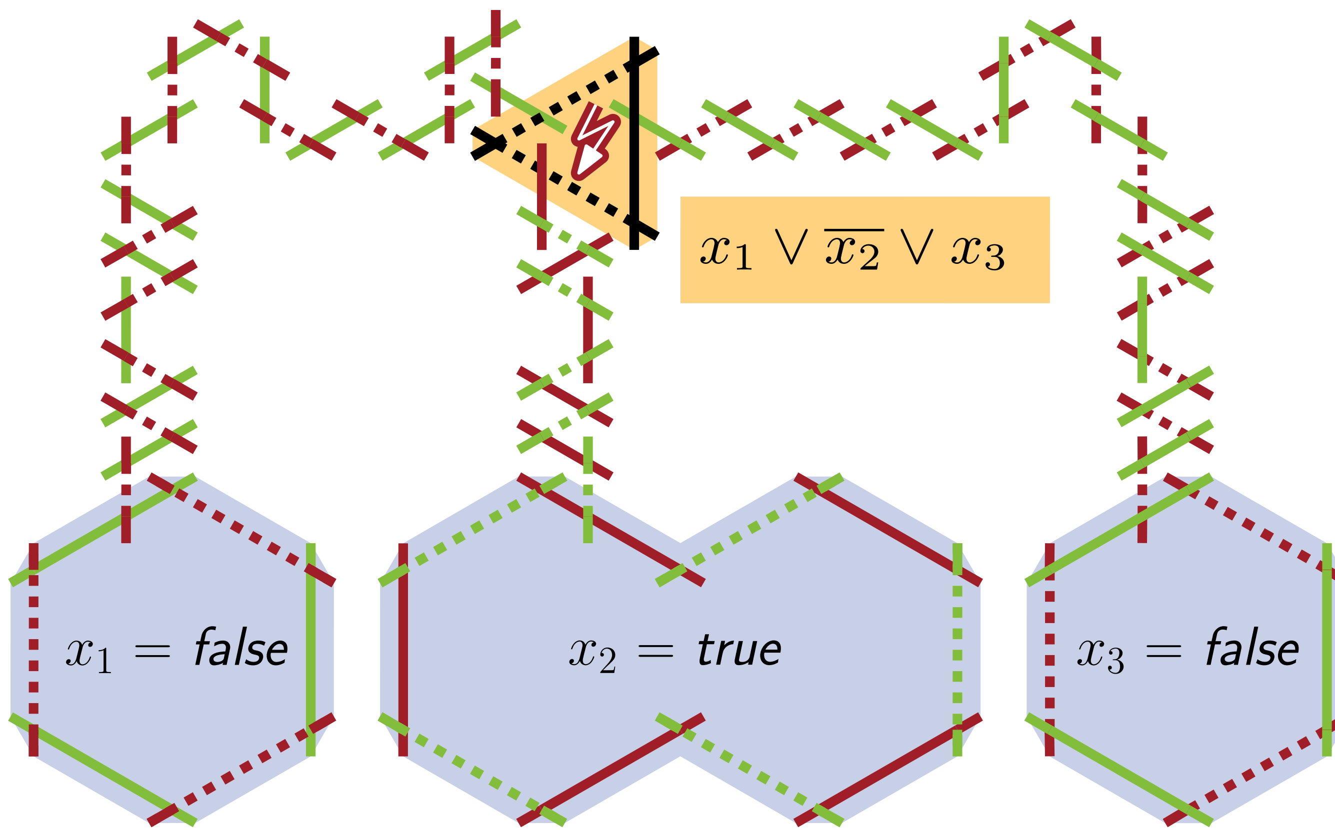




NP-Hardness of MaxSPED

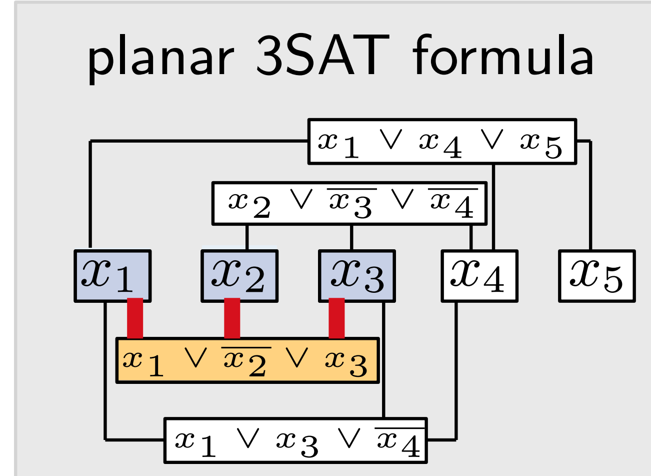
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states

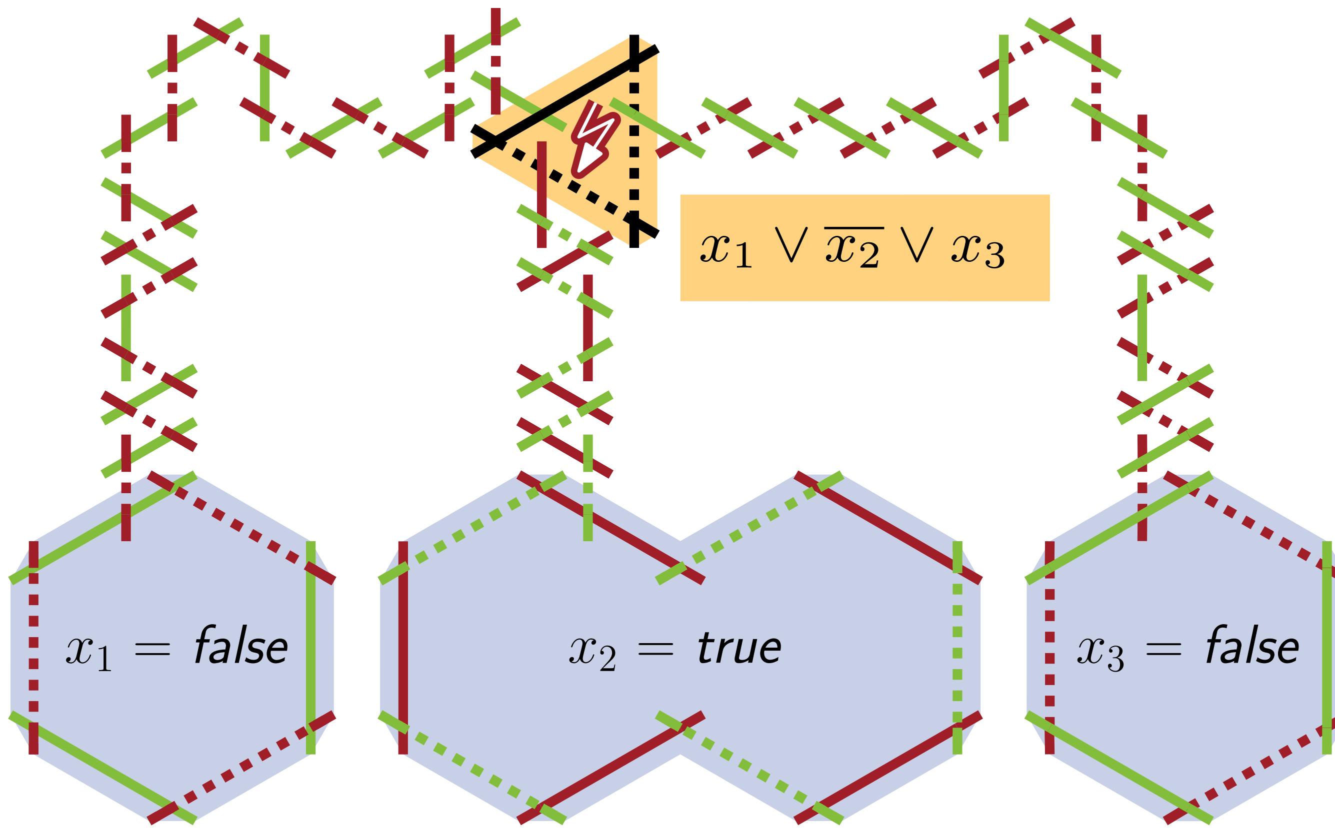




NP-Hardness of MaxSPED

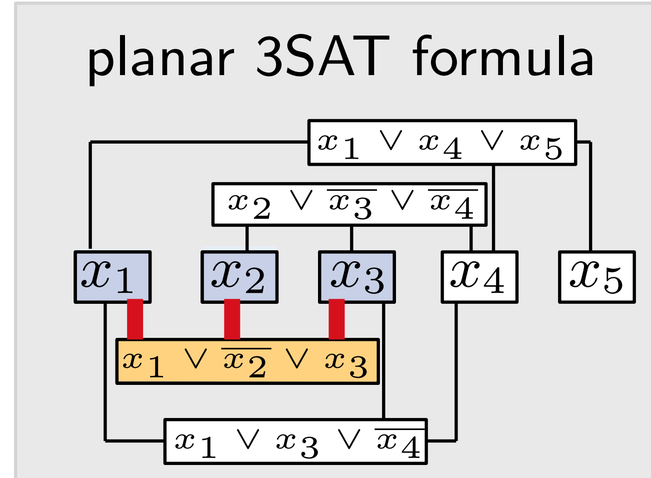
- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states

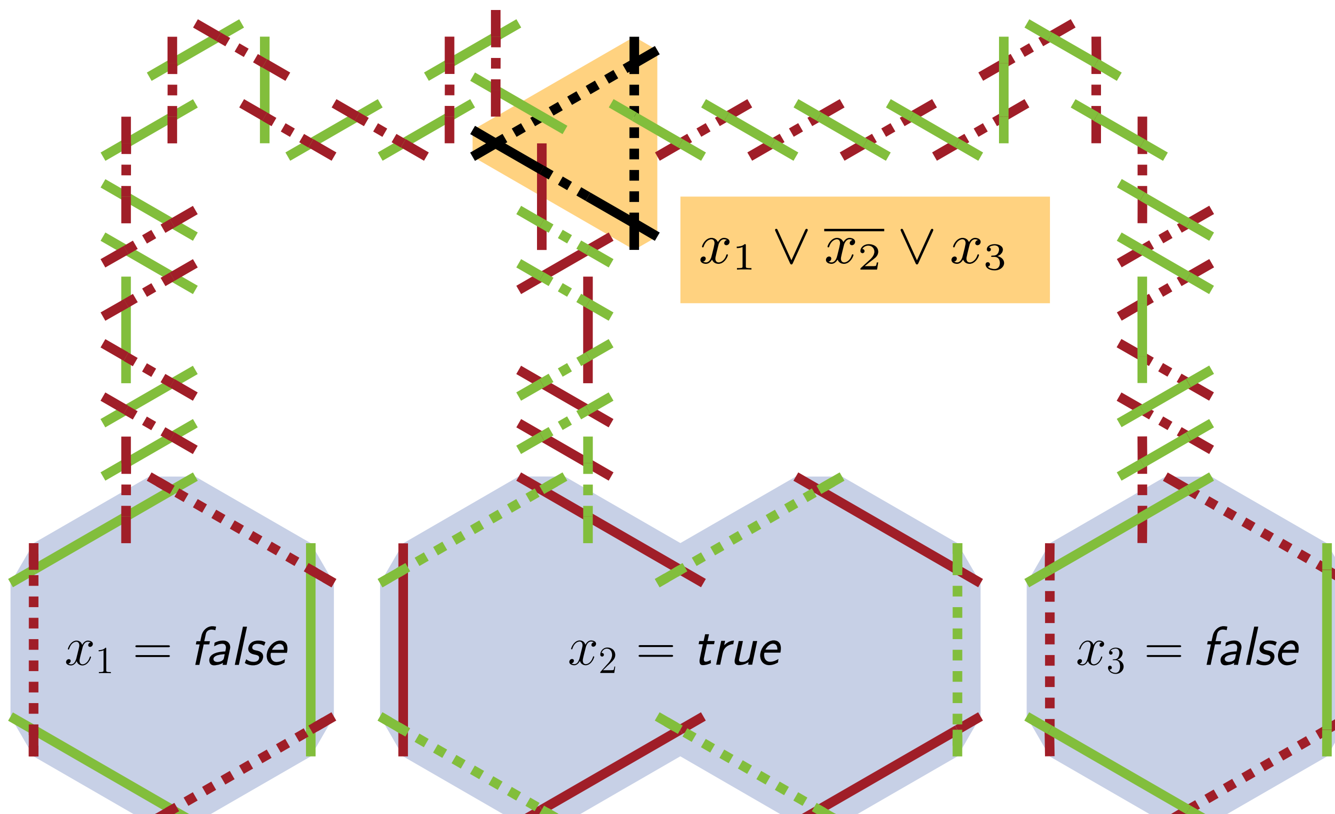




NP-Hardness of MaxSPED

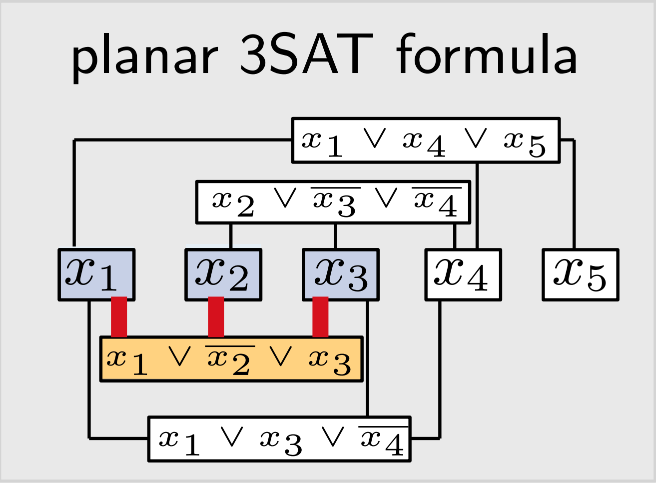
- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states
  - unsatisfied clause loses ink

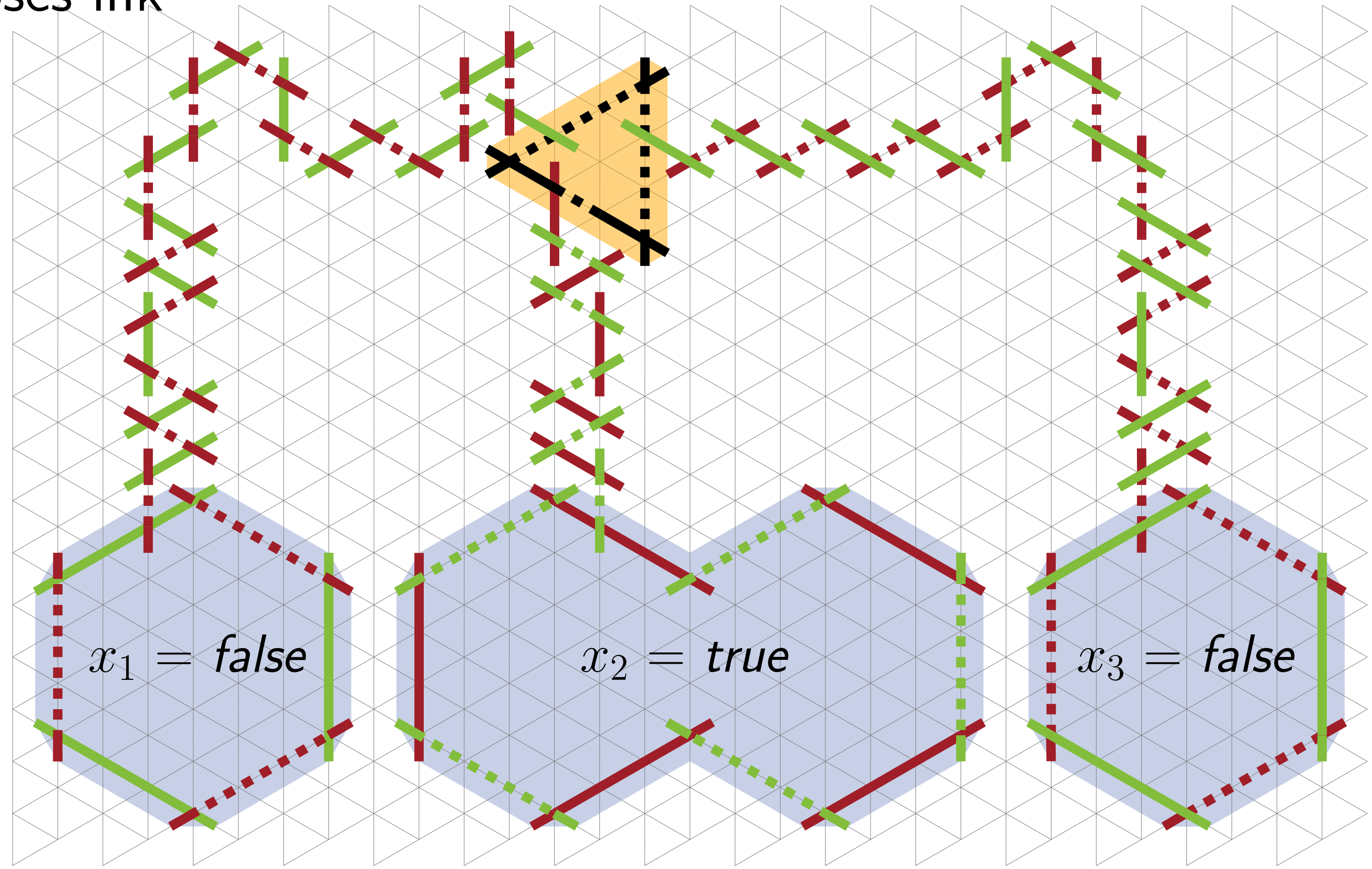




NP-Hardness of MaxSPED

- reduction from PLANAR 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states
  - unsatisfied clause loses ink
  - grid placement

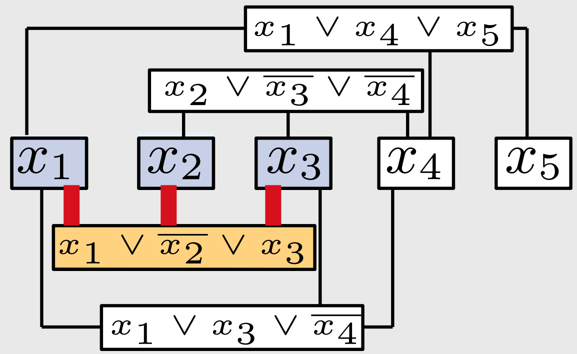


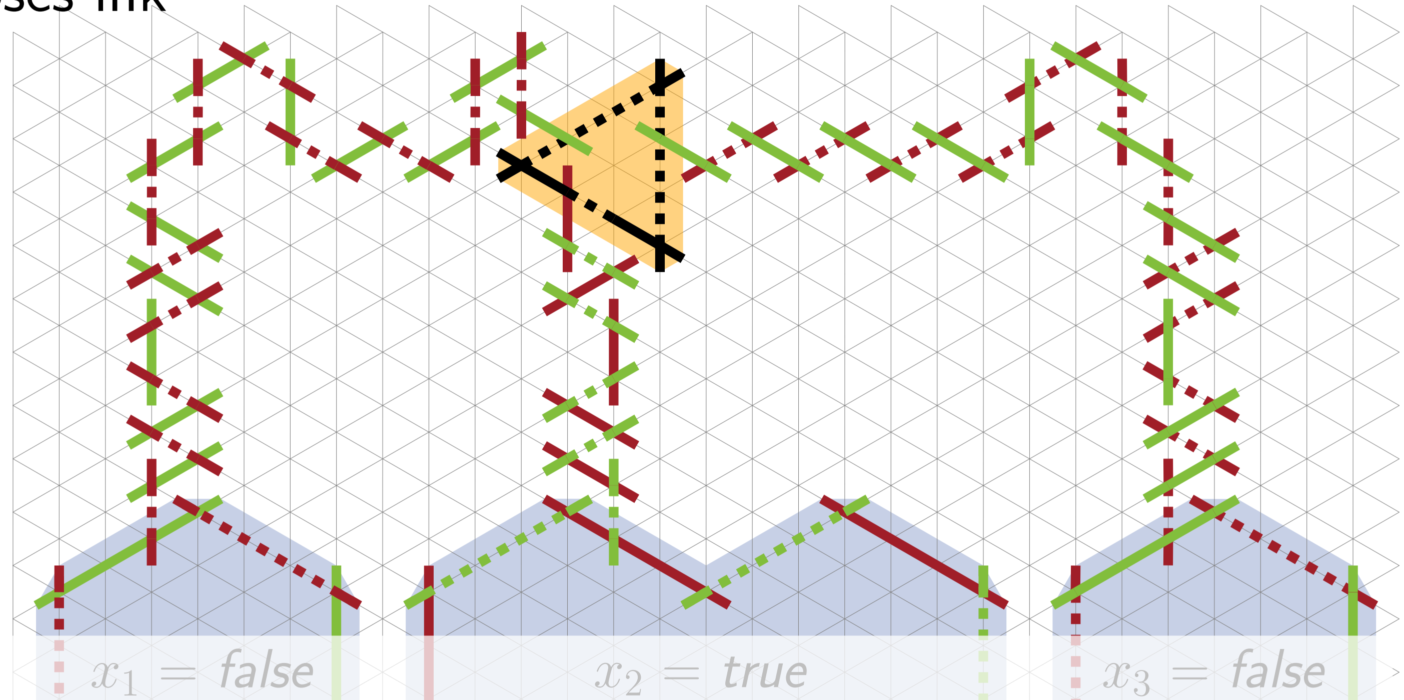


NP-Hardness of MaxSPED

- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states
  - unsatisfied clause loses ink
  - grid placement

planar 3SAT formula

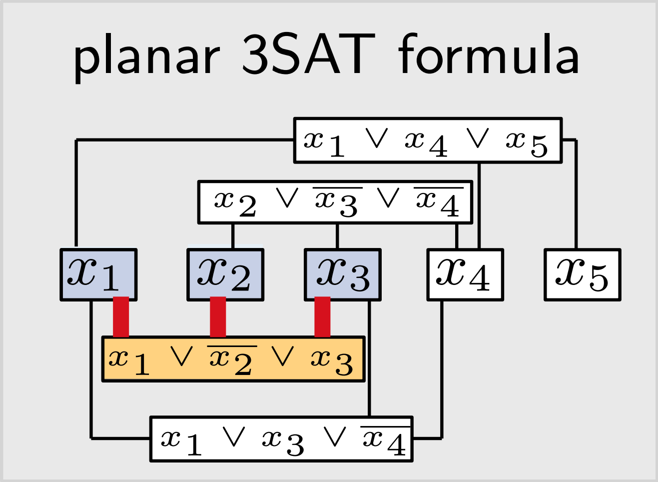




Theorem: MaxSPED is NP-hard for 3-plane input drawings.

NP-Hardness of MaxSPED

- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states
  - literal wires: even length paths, 2 opt. states
  - unsatisfied clause loses ink
  - grid placement



MaxPED: similar proof idea, but non-symmetric stubs require more complex gadgets and up to 4 crossings per edge.

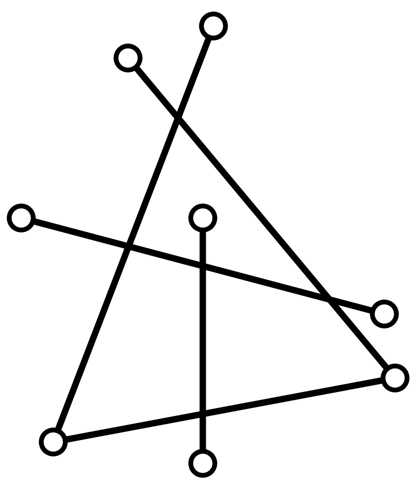
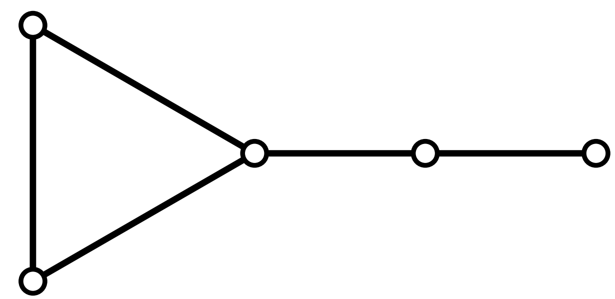
Theorem: MaxPED is NP-hard for 4-plane input drawings.

Theorem: MaxSPED is NP-hard for 3-plane input drawings.

Algorithms

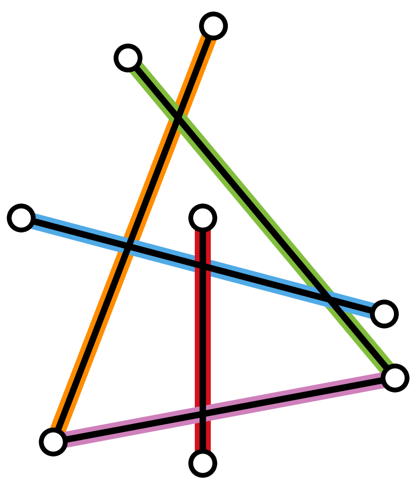
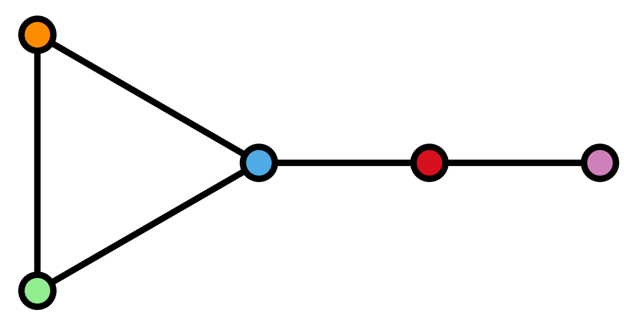
Edge Intersection Graph

- intersection graph $C(\Gamma)$: every vertex u corresponds to one segment $s(u)$ in Γ
- edge (u, v) in C iff $s(u)$ and $s(v)$ cross in Γ

drawing Γ of $G = (V, E)$	edge intersection graph $C(\Gamma)$ with vertex set E
 $\rightarrow$	

Edge Intersection Graph

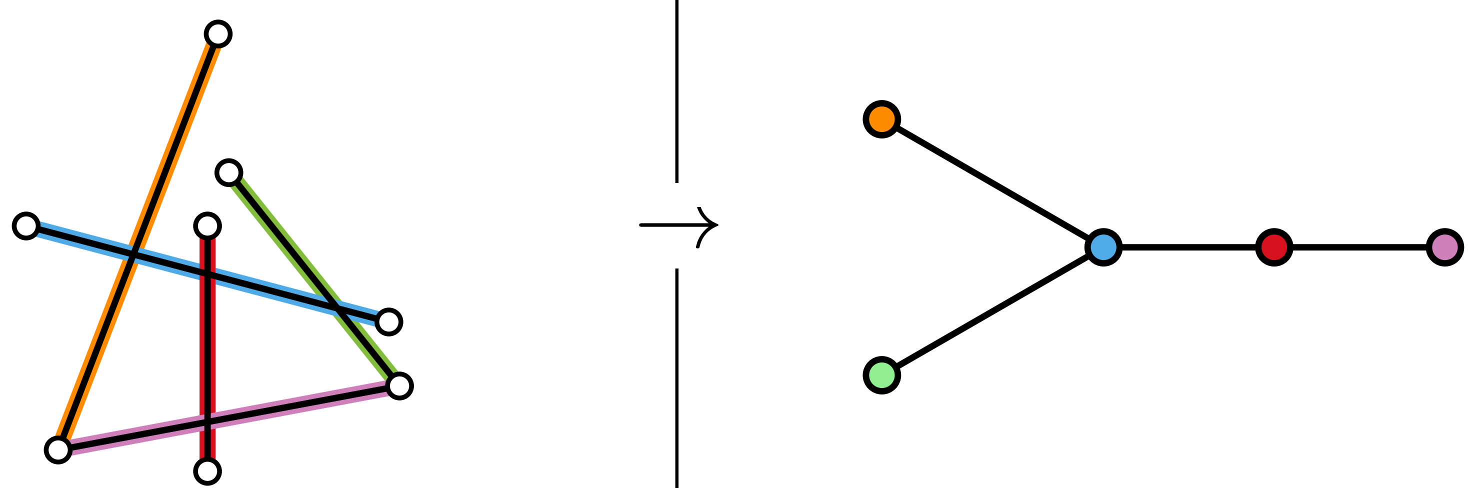
- intersection graph $C(\Gamma)$: every vertex u corresponds to one segment $s(u)$ in Γ
- edge (u, v) in C iff $s(u)$ and $s(v)$ cross in Γ

drawing Γ of $G = (V, E)$	edge intersection graph $C(\Gamma)$ with vertex set E
 $\rightarrow$	

Edge Intersection Graph

- intersection graph $C(\Gamma)$: every vertex u corresponds to one segment $s(u)$ in Γ
- edge (u, v) in C iff $s(u)$ and $s(v)$ cross in Γ

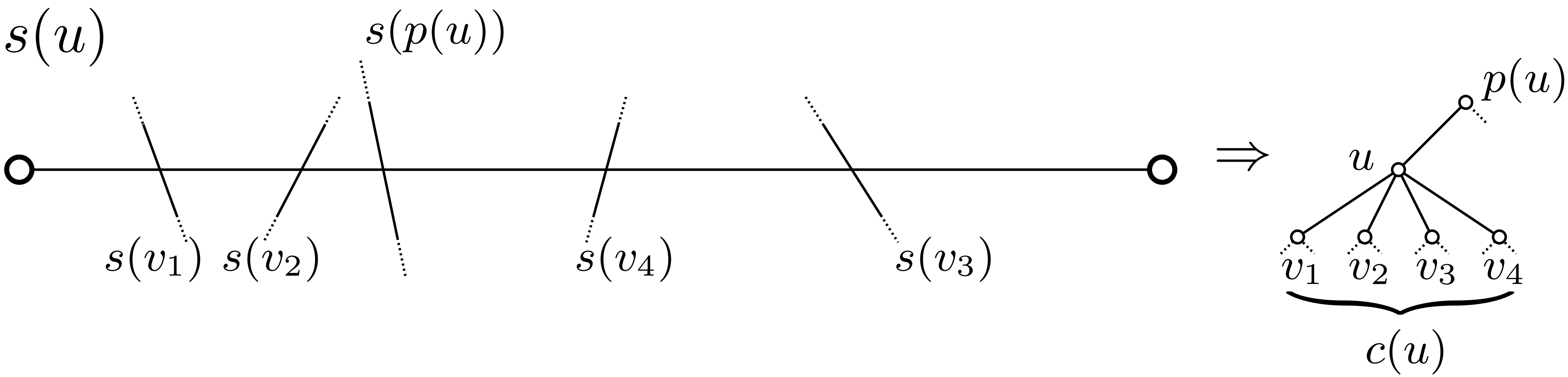
drawing Γ of $G = (V, E)$	edge intersection graph $C(\Gamma)$ with vertex set E



First assumption: C is a tree

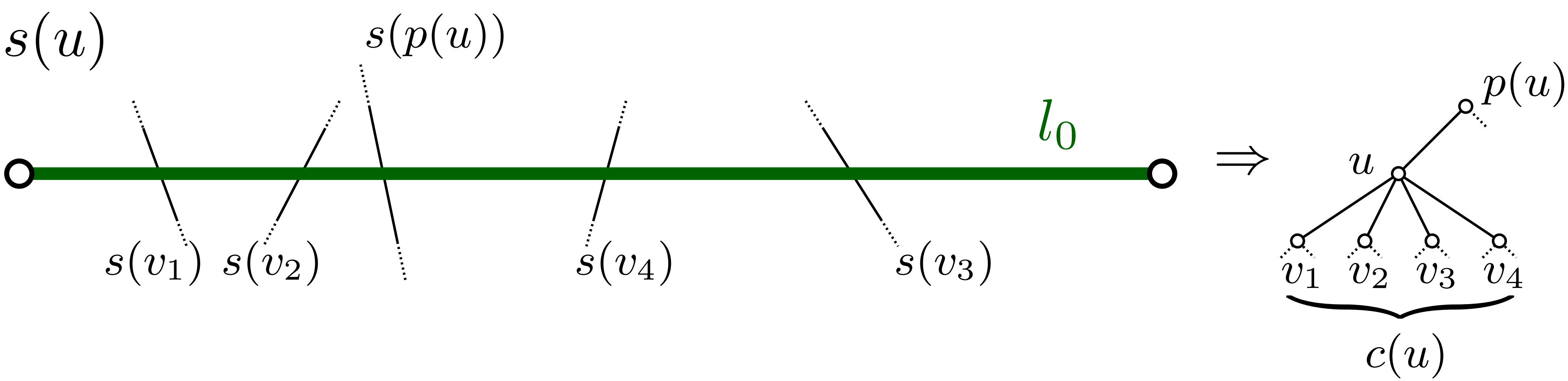
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$



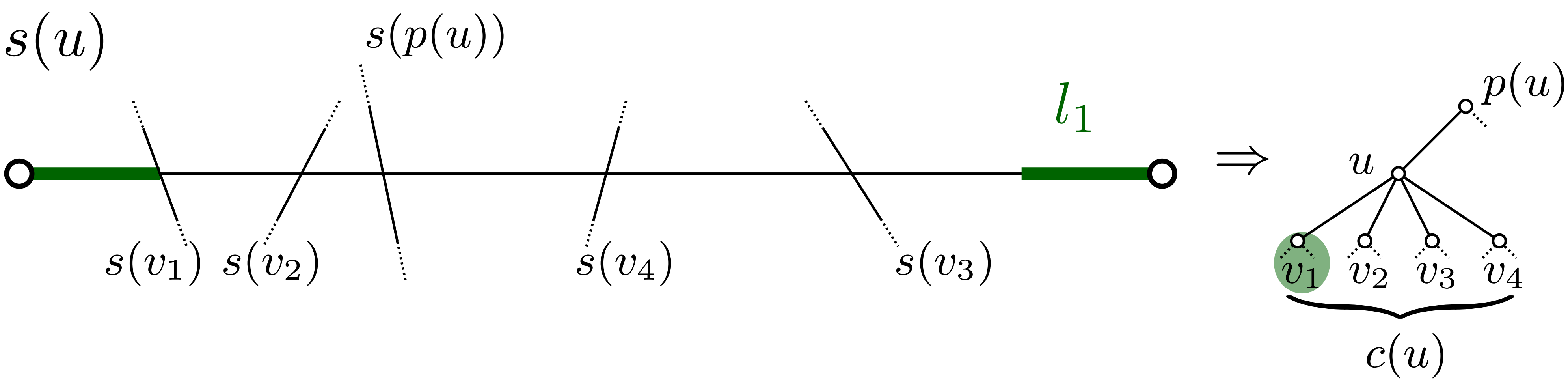
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)



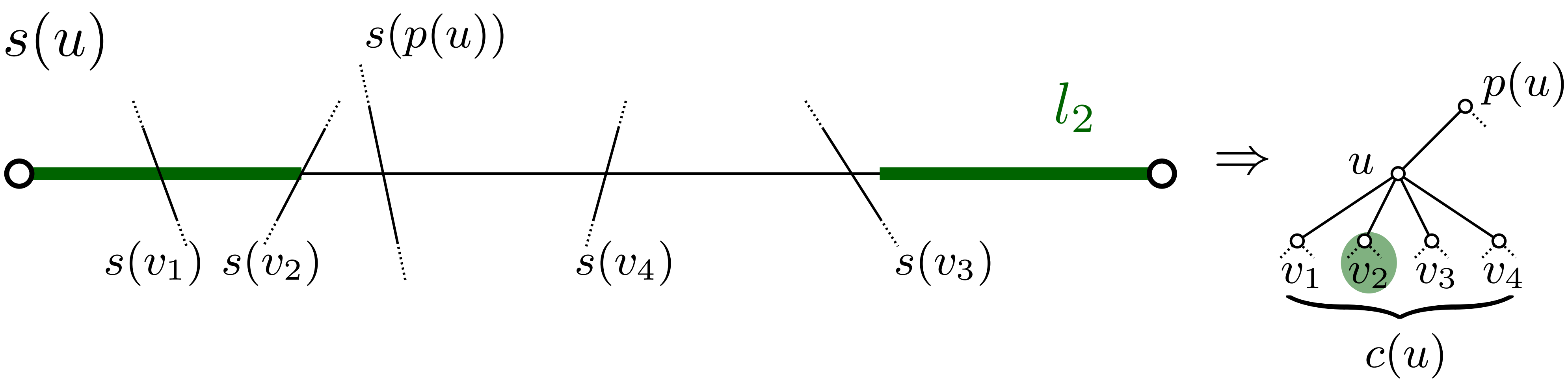
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



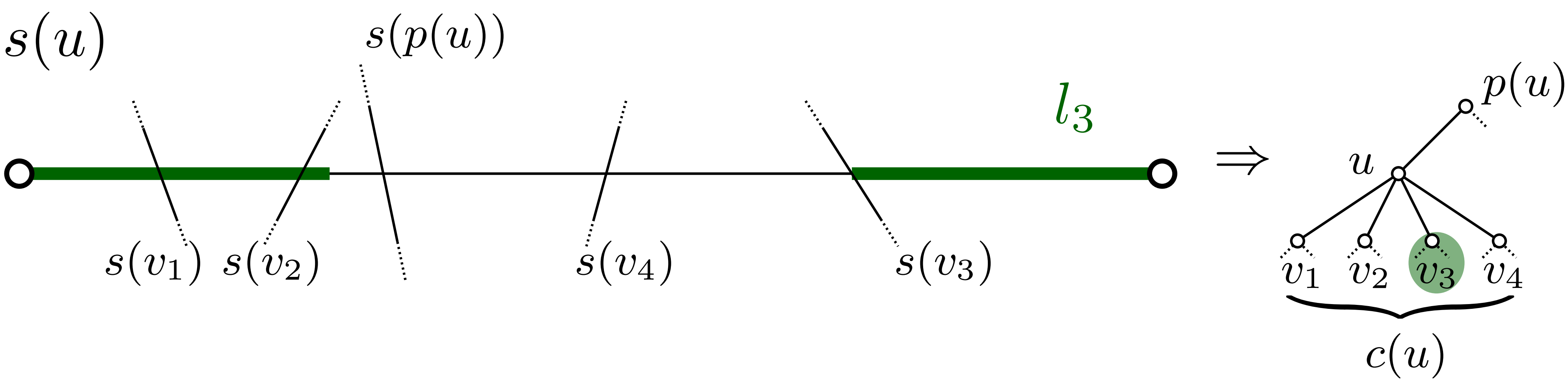
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



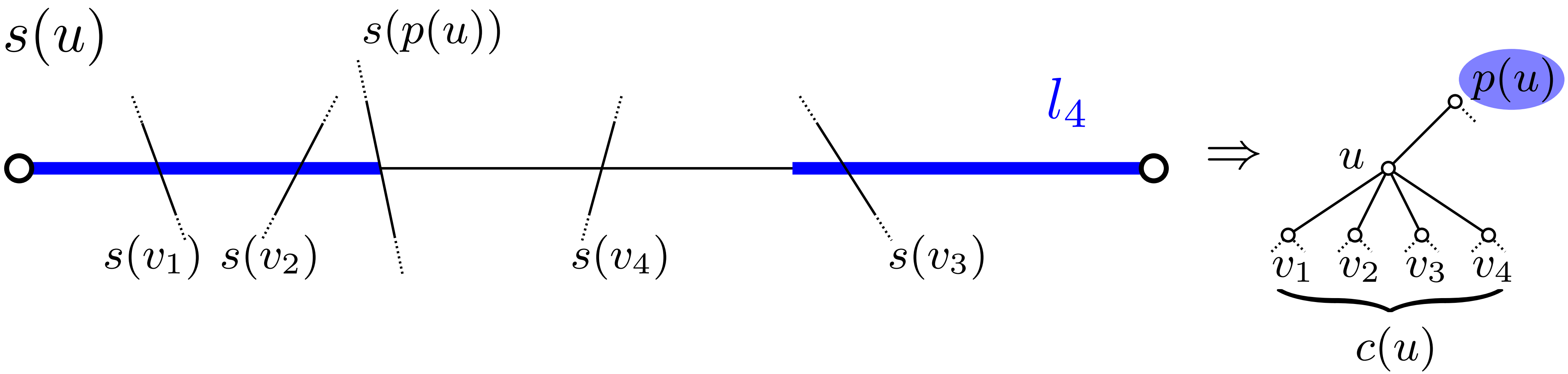
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



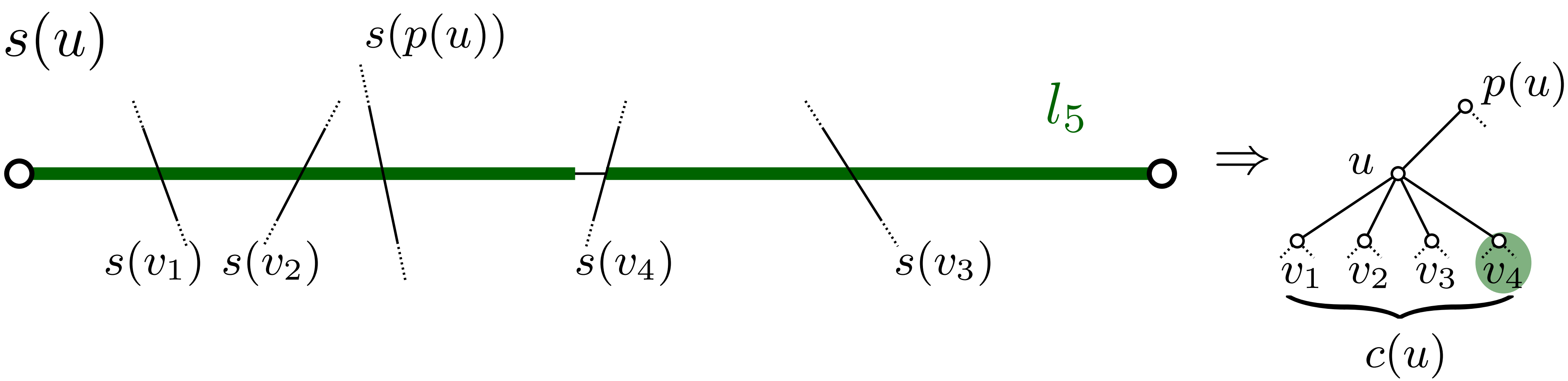
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



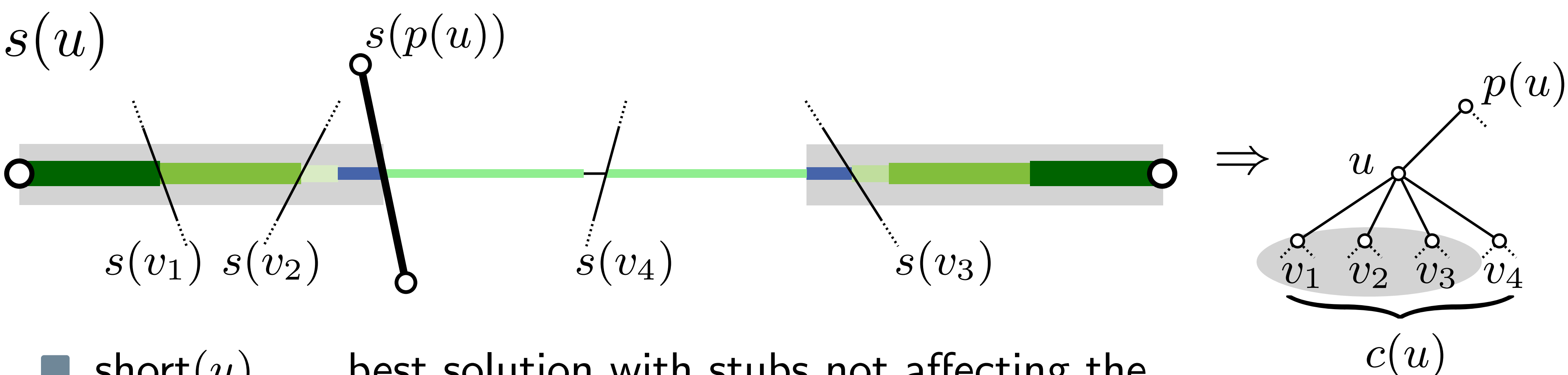
Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



Discretized Stub Lengths for MaxSPED

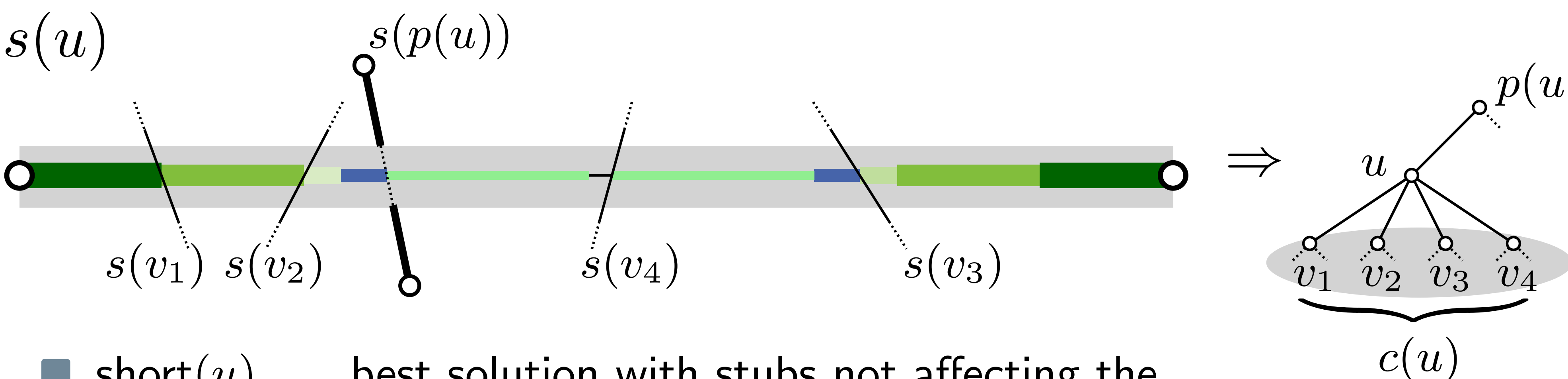
- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



- short(u) ... best solution with stubs not affecting the stubs of the parent

Discretized Stub Lengths for MaxSPED

- pick arbitrary root for tree $C(\Gamma)$
- edge crossings induce different relevant stub lengths
  - l_0 – entire edge (no gap)
  - $l_1, \ldots, l_{\deg(u)}$ – shorter to longer stubs



- short(u) ... best solution with stubs not affecting the stubs of the parent
- long(u) ... best solution with stubs possibly affecting the stubs of the parent

Dynamic Programming: Recurrence

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \mathrm{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \mathrm{long}(v) & \text{otherwise.} \end{cases}$$

Dynamic Programming: Recurrence

$T_i(u)$ . . . maximum ink value for subtree rooted at u s.t. $s(u)$ has stubs of length $l_i(u)$

$s(u)$ . . . segment for vertex u

$l_i(u)$ . . . i-th stub length of $s(u)$

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise.} \end{cases}$$

Dynamic Programming: Recurrence

$T_i(u)$ ... maximum ink value for subtree rooted at u s.t. $s(u)$ has stubs of length $l_i(u)$

$s(u)$ ... segment for vertex u

$l_i(u)$ ... i-th stub length of $s(u)$

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise.} \end{cases}$$

$c(u)$ ... set of children of u

$\text{short}(v) = \max\{T_1(v), ..., T_p(v)\}$
... stubs not affecting the parent

$\text{long}(v) = \max\{T_0(v), ..., T_{deg(v)}(v)\}$
... all stubs (possibly affecting the parent)

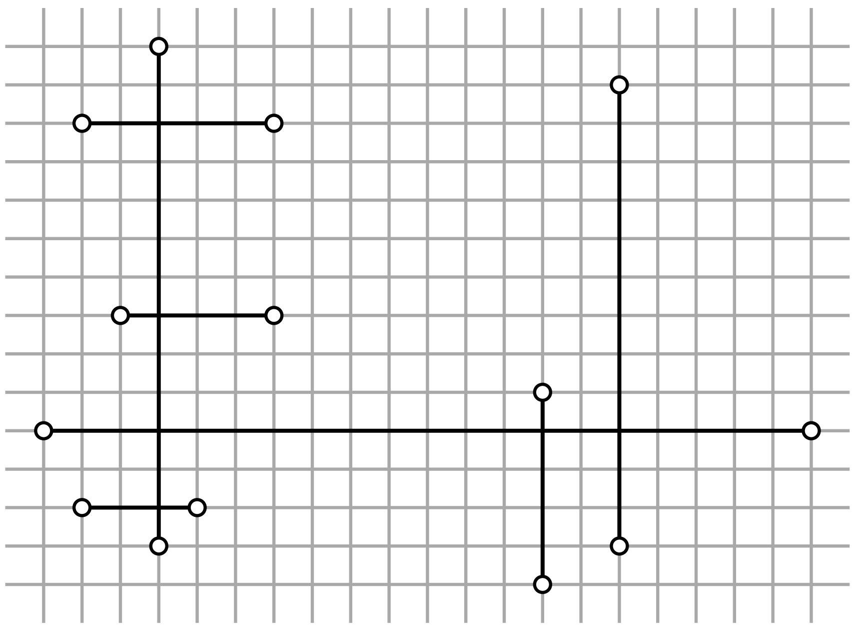
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

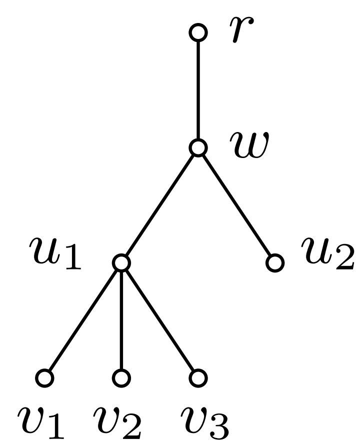
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



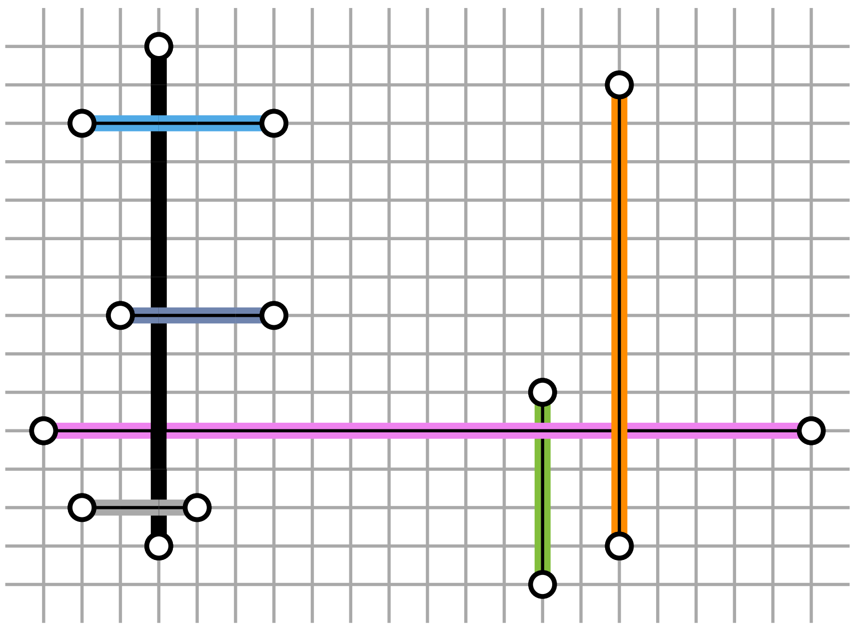
Intersection Graph:



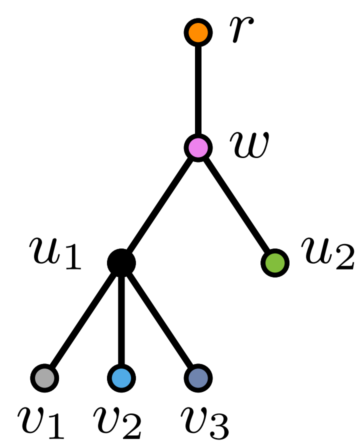
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



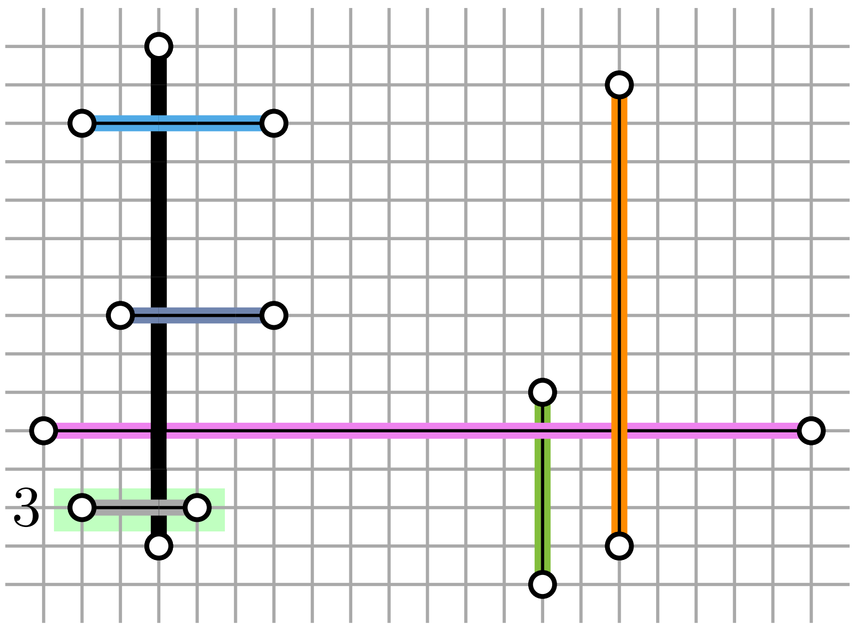
Intersection Graph:



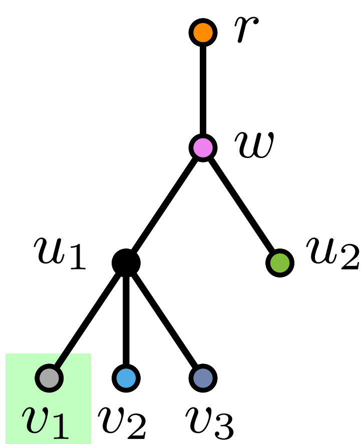
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:

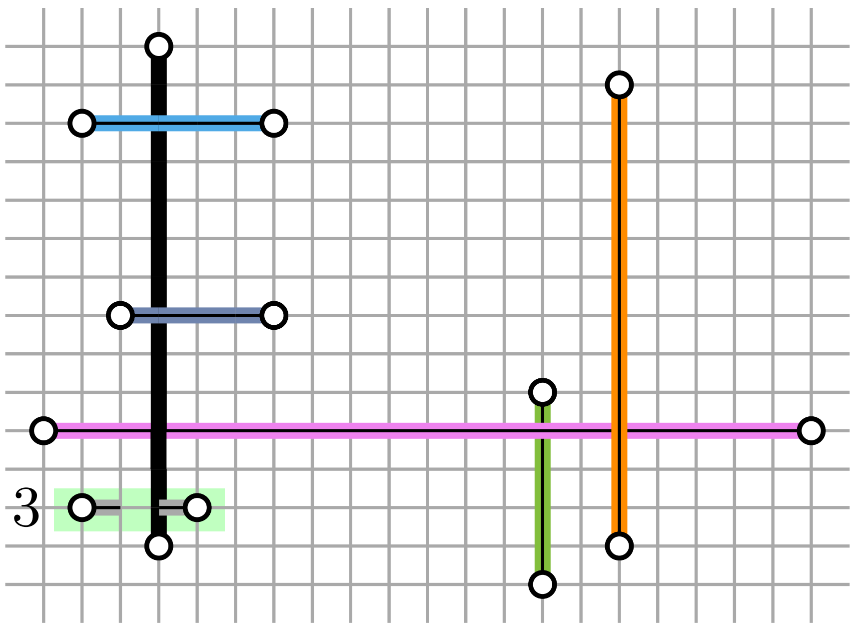


$$v_1: \; T_0(v_1) = l_0 \quad = 3$$

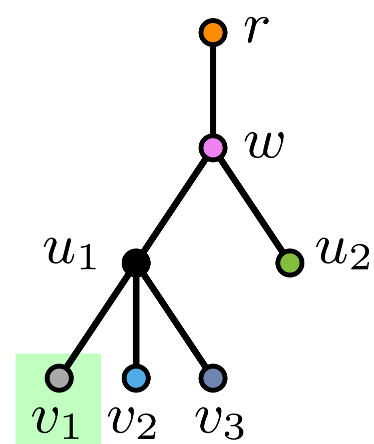
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:

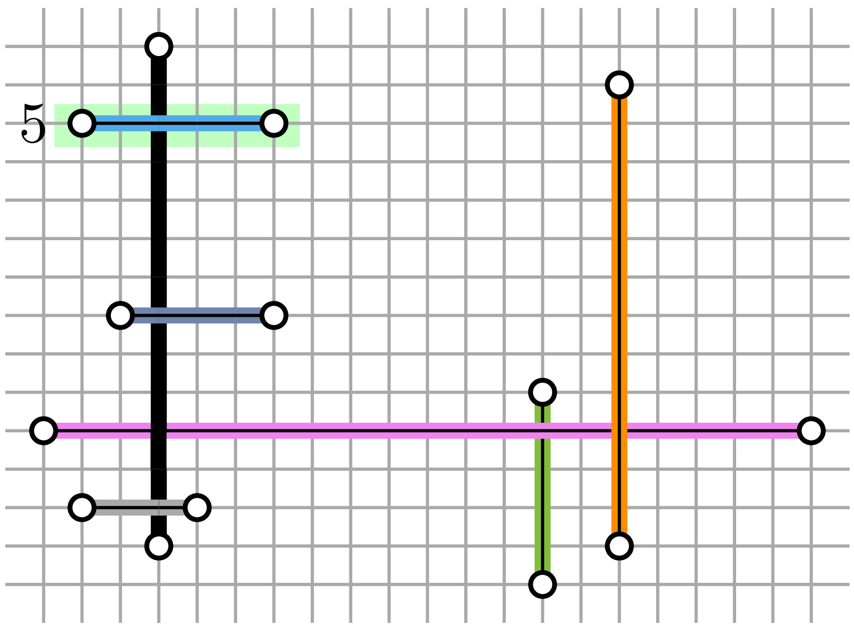


$$v_1 : \begin{matrix} T_0(v_1) = l_0 & = 3 \\ T_1(v_1) = l_1 & = 2 \end{matrix}$$

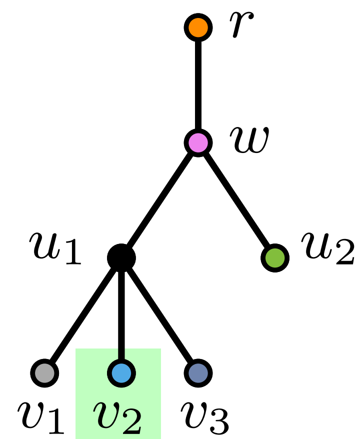
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



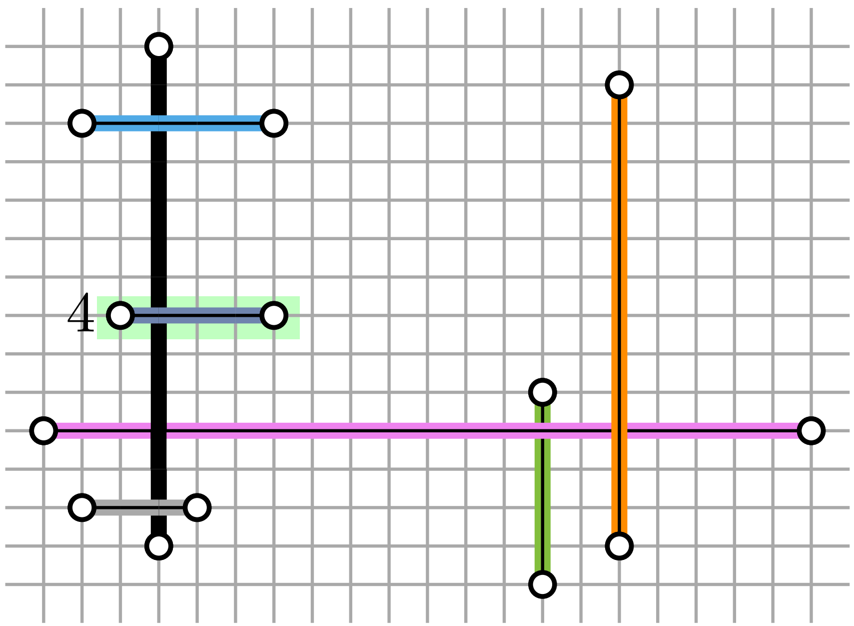
$$v_1 : \begin{array}{ll} T_0(v_1) & = 3 \\ T_1(v_1) & = 2 \end{array}$$

$$v_2 : \begin{array}{lll} T_0(v_2) = l_0 & = 5 \\ T_1(v_2) = l_1 & = 4 \end{array}$$

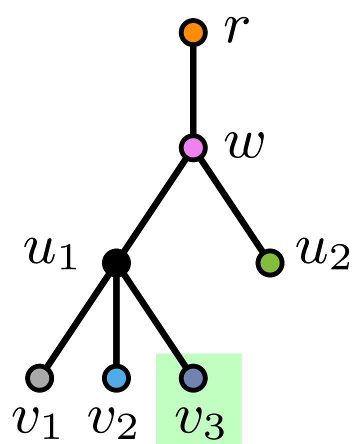
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



$$v_1 : \begin{aligned} T_0(v_1) &= 3 \\ T_1(v_1) &= 2 \end{aligned}$$

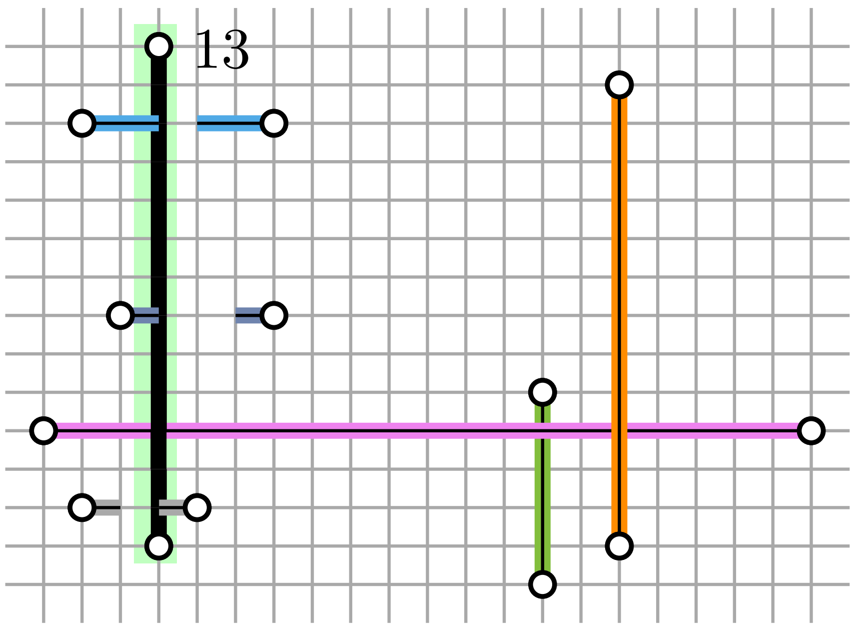
$$v_2 : \begin{aligned} T_0(v_2) &= 5 \\ T_1(v_2) &= 4 \end{aligned}$$

$$v_3 : \begin{aligned} T_0(v_3) = l_0 &= 4 \\ T_1(v_3) = l_1 &= 2 \end{aligned}$$

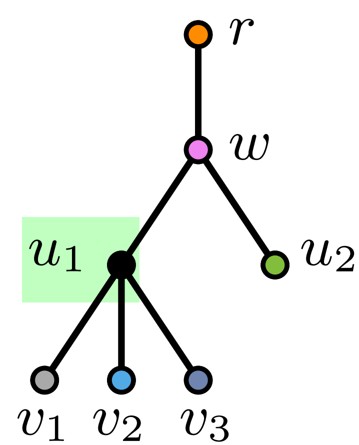
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$

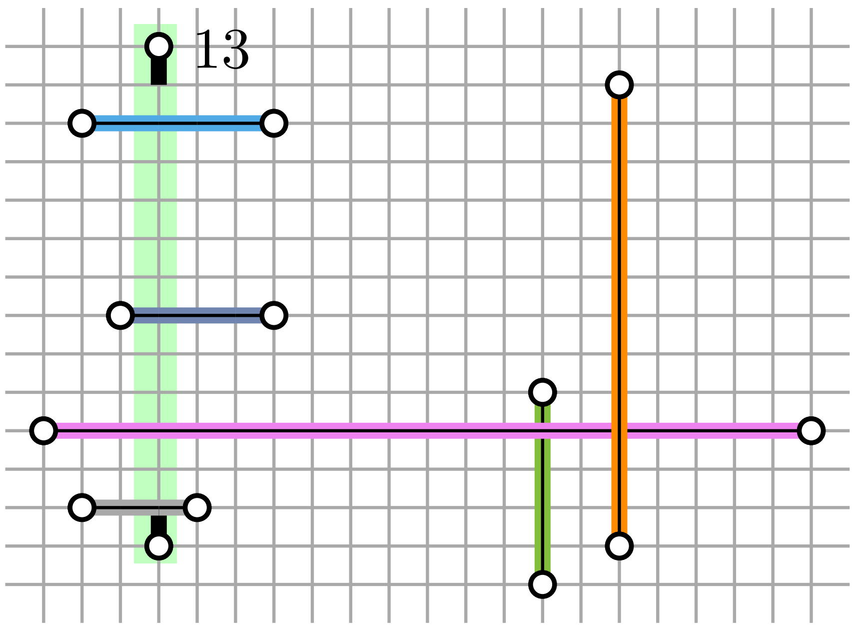
$$T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) \quad = 21$$

u_1 :

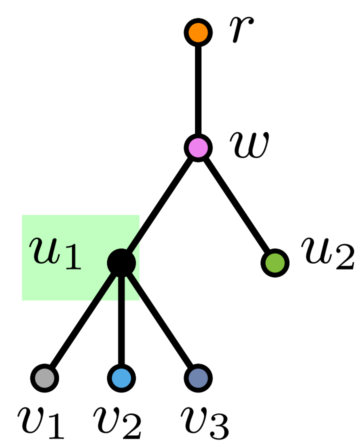
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$

$$T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) \quad = 21$$

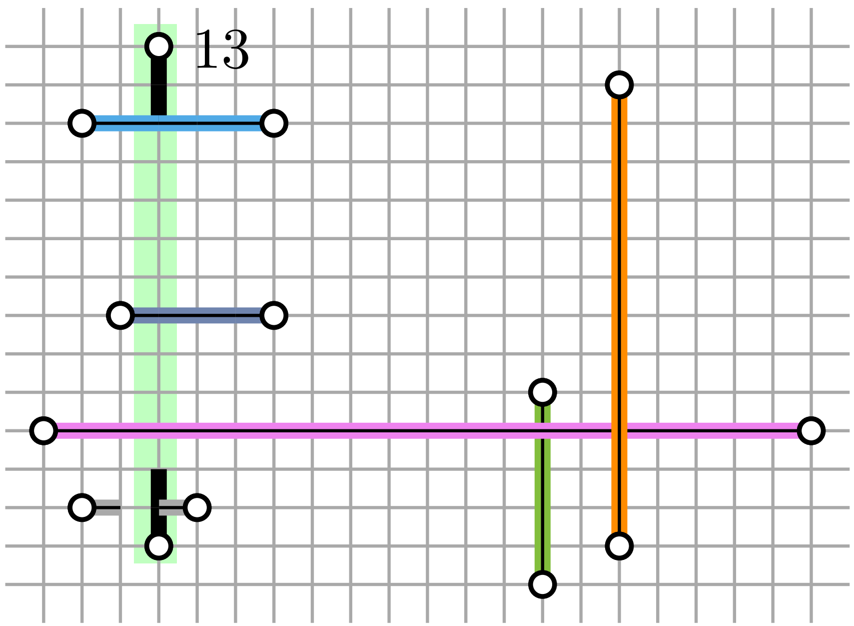
$$T_1(u_1) = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) \quad = 14$$

u_1 :

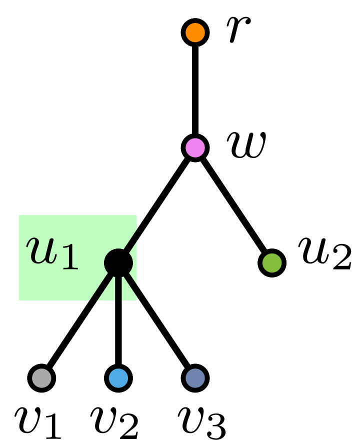
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$

$$T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) = 21$$

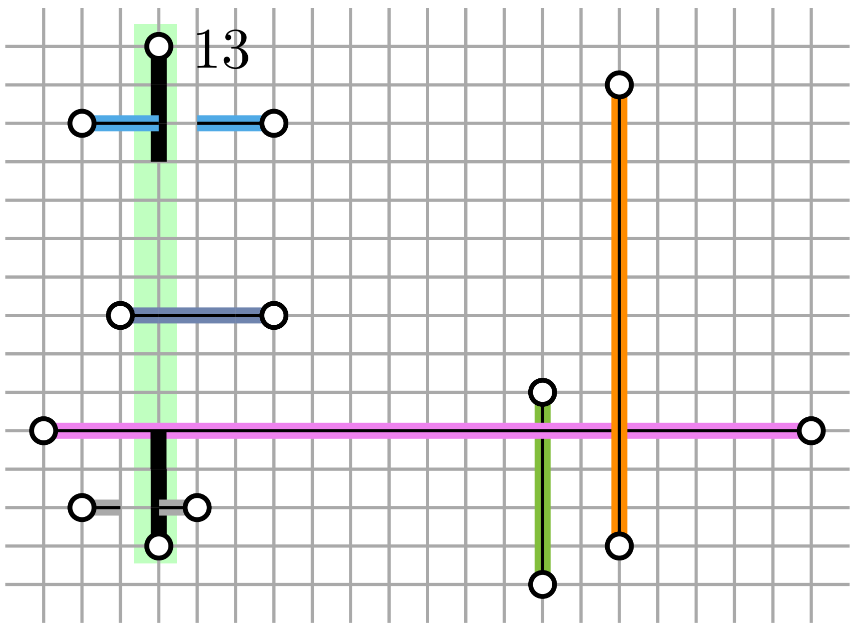
$$T_1(u_1) = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 14$$

$$u_1 : T_2(u_2) = l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 15$$

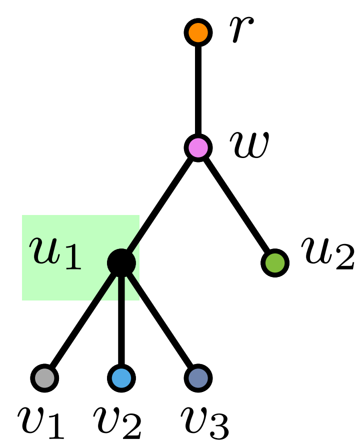
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



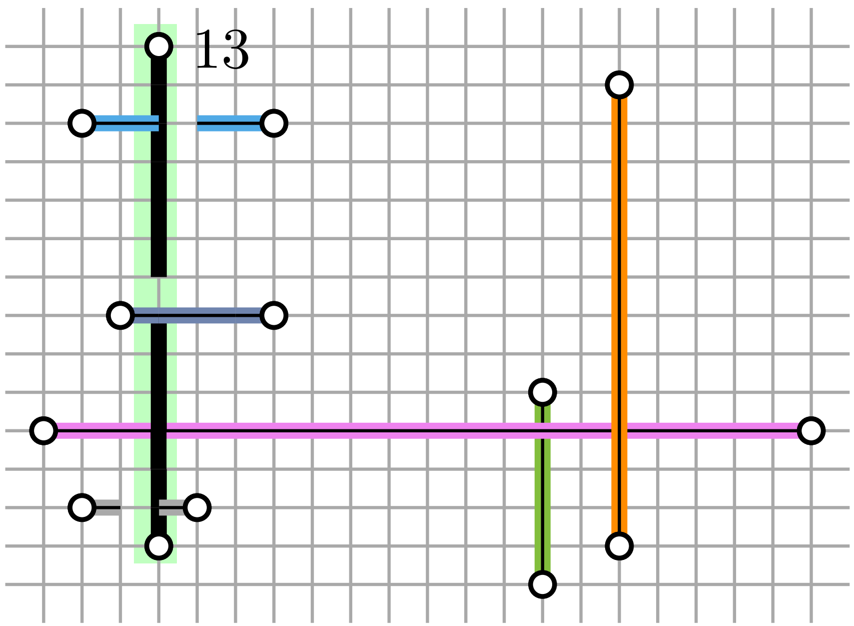
v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$

$$T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) \quad = 21$$
$$T_1(u_1) = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) \quad = 14$$
$$u_1 : T_2(u_2) = l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) \quad = 15$$
$$T_3(u_3) = l_3 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) \quad = 16$$

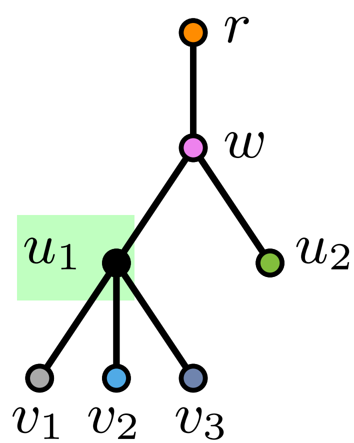
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



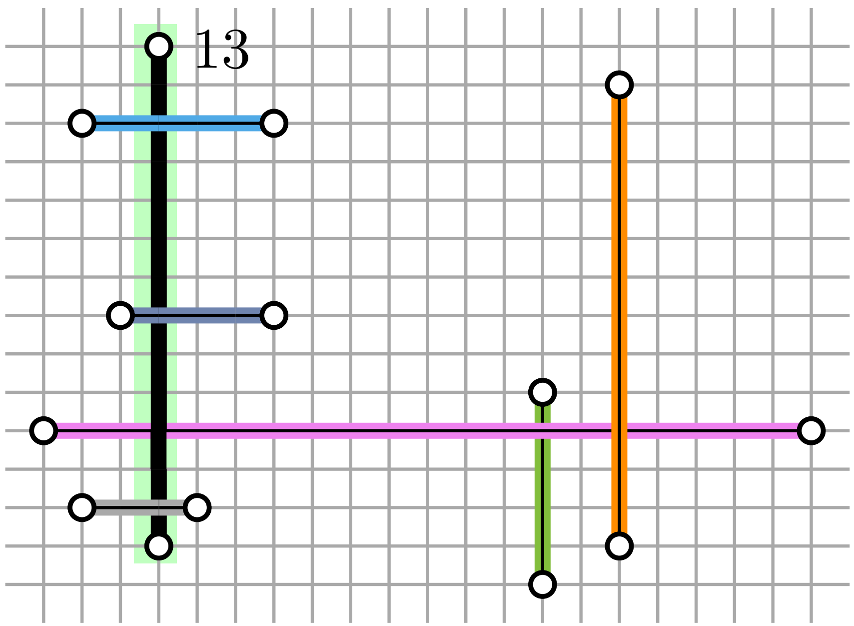
v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$

$$\begin{aligned} & T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) && = 21 \\ & T_1(u_1) = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) && = 14 \\ u_1 : \; & T_2(u_2) = l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) && = 15 \\ & T_3(u_3) = l_3 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) && = 16 \\ & T_4(u_4) = l_4 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) && = 22 \end{aligned}$$

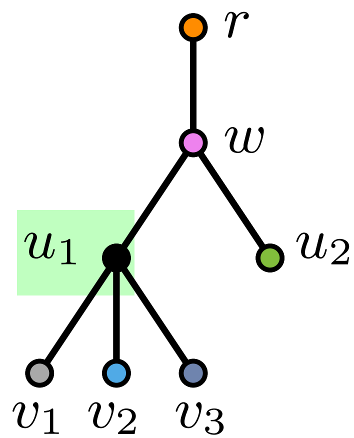
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$

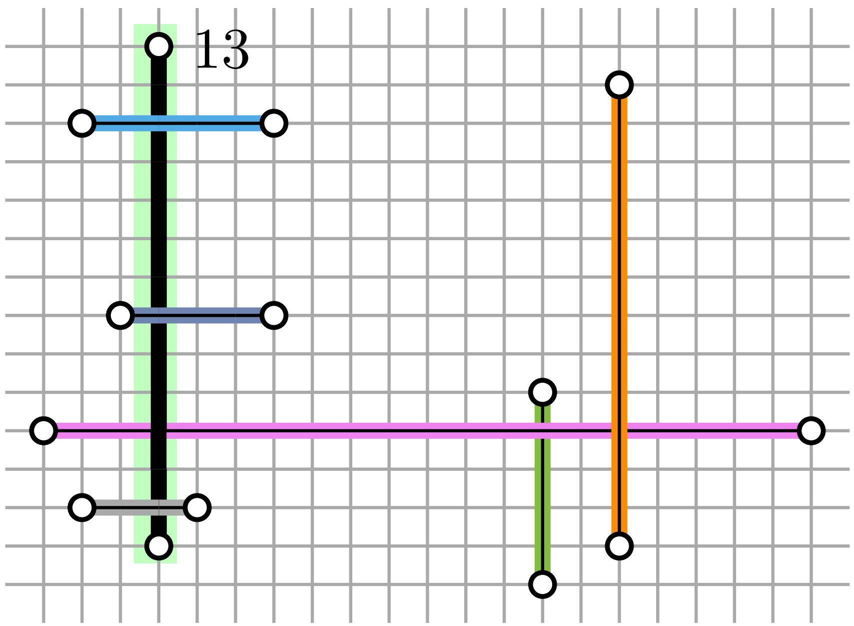
$$\begin{aligned} & T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) && = 21 \\ & T_1(u_1) = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) && = 14 \\ u_1 : \; & T_2(u_2) = l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) && = 15 \\ & T_3(u_3) = l_3 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) && = 16 \\ & T_4(u_4) = l_4 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) && = 22 \end{aligned}$$

$\left.\begin{matrix} T_1 \\ T_2 \\ T_3 \end{matrix}\right\}$ $\text{short}(u_1) = \max\{T_1, T_2, T_3\} = T_3(u_1) = 16$

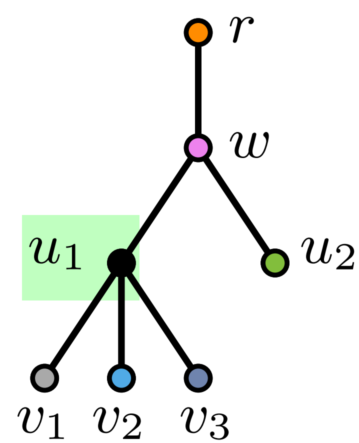
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



$$v_1 : \begin{aligned} T_0(v_1) &= 3 \\ T_1(v_1) &= 2 \end{aligned}$$

$$v_2 : \begin{aligned} T_0(v_2) &= 5 \\ T_1(v_2) &= 4 \end{aligned}$$

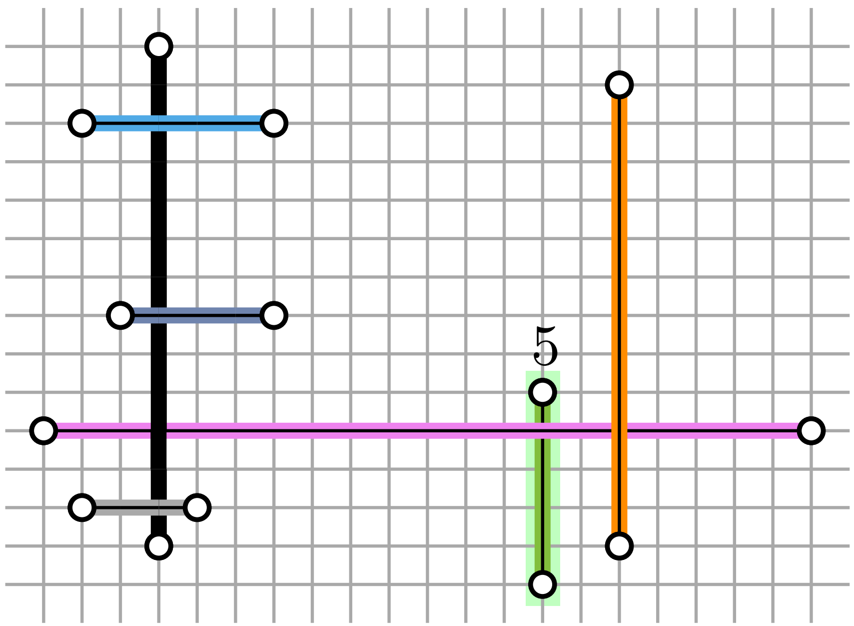
$$v_3 : \begin{aligned} T_0(v_3) &= 4 \\ T_1(v_3) &= 2 \end{aligned}$$

$$u_1 : \left. \begin{aligned} T_0(u_1) &= l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) && = 21 \\ T_1(u_1) &= l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) && = 14 \\ T_2(u_2) &= l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) && = 15 \\ T_3(u_3) &= l_3 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) && = 16 \\ T_4(u_4) &= l_4 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) && = 22 \end{aligned} \right\} \begin{aligned} &\text{long}(u_1) = \\ &\max\{T_0, \ldots, T_4\} = \\ &T_4(u_1) = 22 \end{aligned}$$

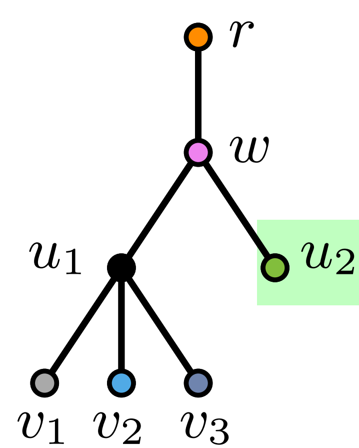
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 : $T_0(v_1) = 3$, $T_1(v_1) = 2$

v_2 : $T_0(v_2) = 5$, $T_1(v_2) = 4$

v_3 : $T_0(v_3) = 4$, $T_1(v_3) = 2$

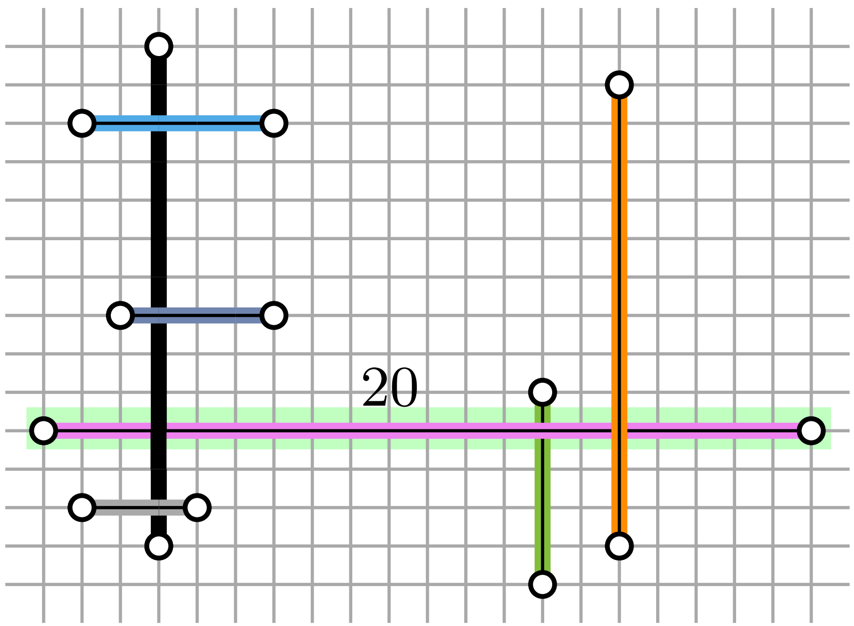
u_1 : $\text{short}(u_1) = T_3 = 16$, $\text{long}(u_1) = T_4 = 22$

$$u_2 : \begin{aligned} T_0(u_2) &= l_0 &= 5 \\ T_1(u_2) &= l_1 &= 2 \end{aligned}$$

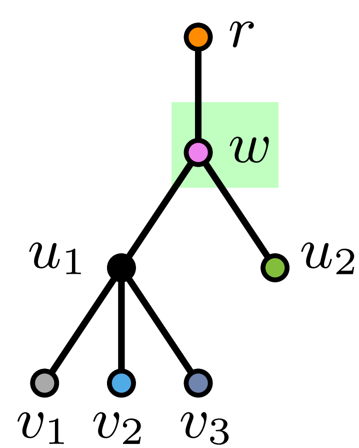
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



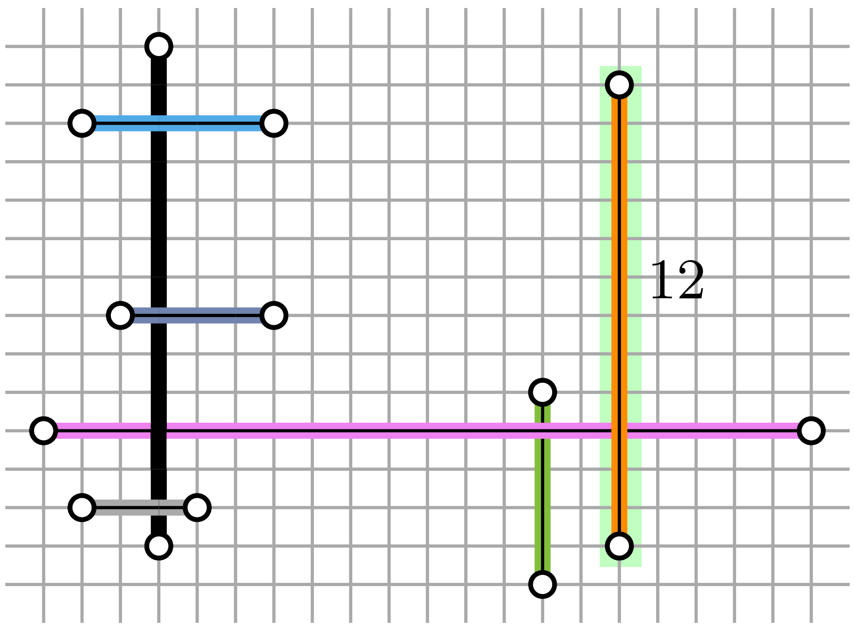
v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$
u_1 :	$\text{short}(u_1) = T_3$	$= 16$
	$\text{long}(u_1) = T_4$	$= 22$
u_2 :	$T_0(u_2)$	$= 5$
	$T_1(u_2)$	$= 2$

$$w: \begin{aligned} T_0(w) &= l_0 + \text{short}(u_1) + \text{short}(u_2) &&= 38 \\ T_1(w) &= l_1 + \text{long}(u_1) + \text{long}(u_2) &&= 33 \\ T_2(w) &= l_2 + \text{short}(u_1) + \text{long}(u_2) &&= 31 \\ T_3(w) &= l_3 + \text{short}(u_1) + \text{long}(u_2) &&= 35 \end{aligned}$$

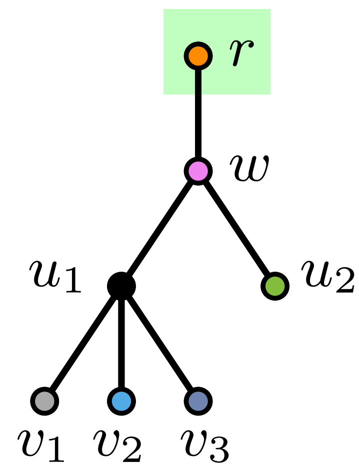
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 : $T_0(v_1) = 3$, $T_1(v_1) = 2$

v_2 : $T_0(v_2) = 5$, $T_1(v_2) = 4$

v_3 : $T_0(v_3) = 4$, $T_1(v_3) = 2$

u_1 : $\text{short}(u_1) = T_3 = 16$, $\text{long}(u_1) = T_4 = 22$

u_2 : $T_0(u_2) = 5$, $T_1(u_2) = 2$

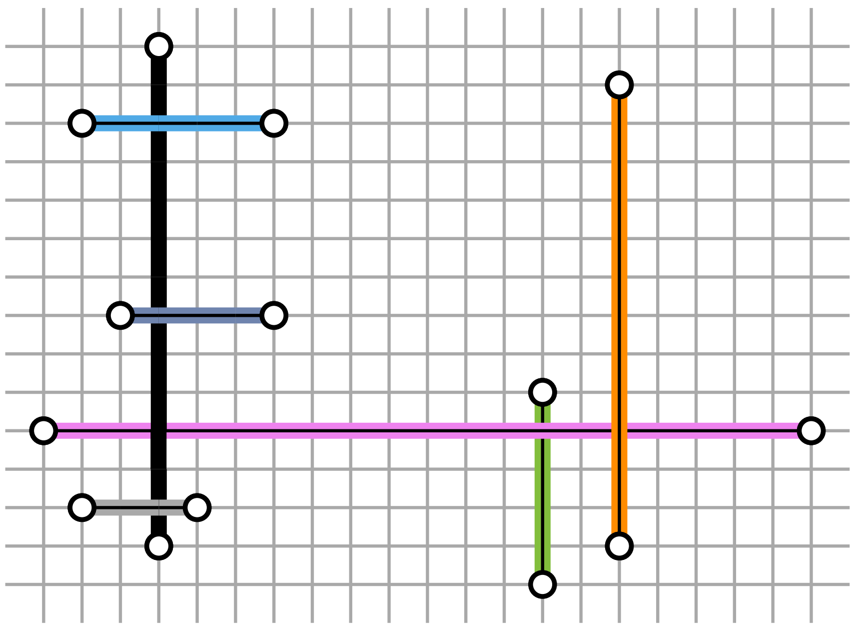
w : $\text{short}(w) = T_1 = 33$, $\text{long}(w) = T_0 = 38$

$$r: \begin{aligned} T_0(r) &= l_0 + \text{short}(w) &= 45 \\ T_1(r) &= l_1 + \text{long}(w) &= 44 \end{aligned}$$

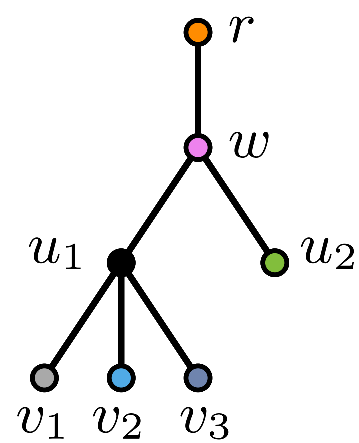
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 : $T_0(v_1) = 3$, $T_1(v_1) = 2$

v_2 : $T_0(v_2) = 5$, $T_1(v_2) = 4$

v_3 : $T_0(v_3) = 4$, $T_1(v_3) = 2$

u_1 : $\text{short}(u_1) = T_3 = 16$, $\text{long}(u_1) = T_4 = 22$

u_2 : $T_0(u_2) = 5$, $T_1(u_2) = 2$

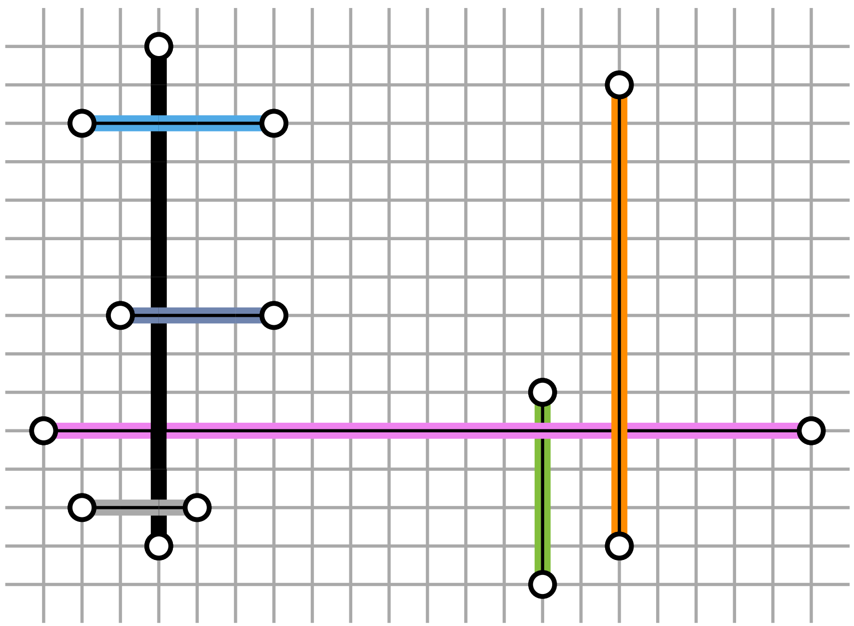
w : $\text{short}(w) = T_1 = 33$, $\text{long}(w) = T_0 = 38$

$$r : \begin{aligned} T_0(r) &= l_0 + \text{short}(w) &= 45 \\ T_1(r) &= l_1 + \text{long}(w) &= 44 \end{aligned}$$

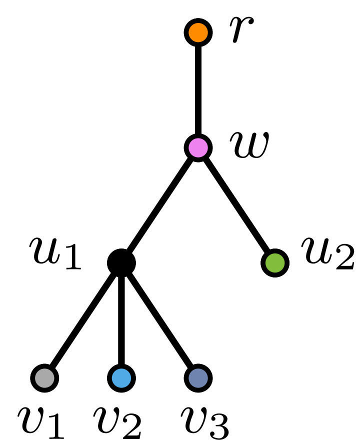
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

Input:



Intersection Graph:



v_1 :	$T_0(v_1)$	$= 3$
	$T_1(v_1)$	$= 2$
v_2 :	$T_0(v_2)$	$= 5$
	$T_1(v_2)$	$= 4$
v_3 :	$T_0(v_3)$	$= 4$
	$T_1(v_3)$	$= 2$
u_1 :	$\text{short}(u_1) = T_3$	$= 16$
	$\text{long}(u_1) = T_4$	$= 22$
u_2 :	$T_0(u_2)$	$= 5$
	$T_1(u_2)$	$= 2$
w :	$\text{short}(w) = T_1$	$= 33$
	$\text{long}(w) = T_0$	$= 38$

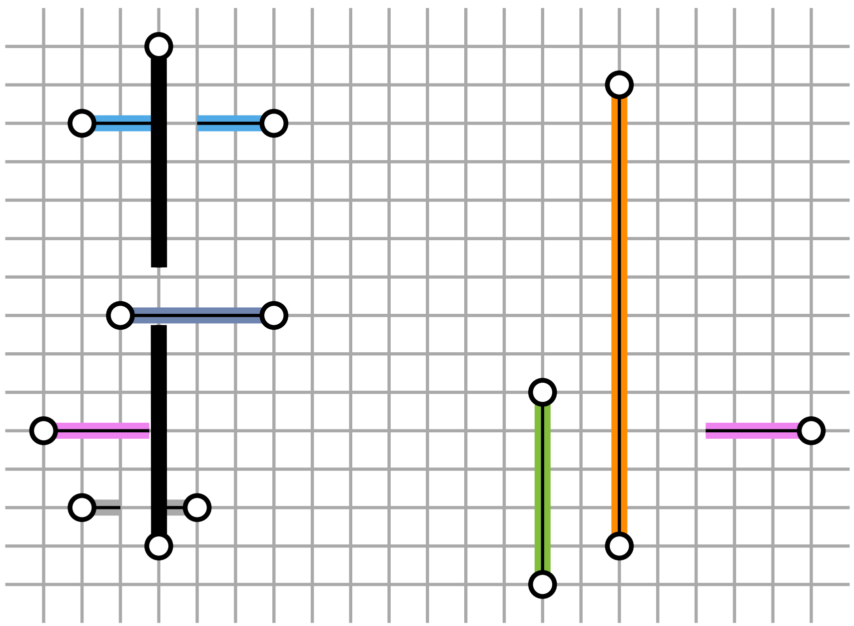
r : $T_0(r) = l_0 + \text{short}(w) = 45$ ← backtracking

$T_1(r) = l_1 + \text{long}(w) = 44$

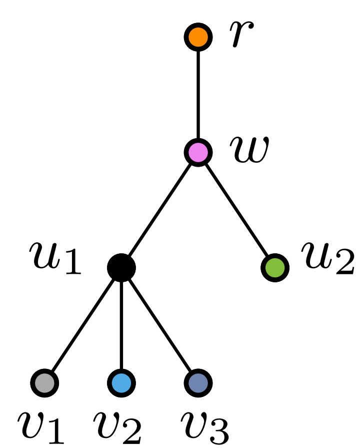
Dynamic Programming: Example

$$T_i(u) = l_i(u) + \sum_{v \in c(u)} \begin{cases} \text{short}(v) & \text{if } s(u) \text{ with length } l_i(u) \text{ intersects } s(v) \\ \text{long}(v) & \text{otherwise} \end{cases}$$

MaxSPED:



Intersection Graph:



v_1 : $T_0(v_1) = 3$, $T_1(v_1) = 2$

v_2 : $T_0(v_2) = 5$, $T_1(v_2) = 4$

v_3 : $T_0(v_3) = 4$, $T_1(v_3) = 2$

u_1 : $\text{short}(u_1) = T_3 = 16$, $\text{long}(u_1) = T_4 = 22$

u_2 : $T_0(u_2) = 5$, $T_1(u_2) = 2$

w : $\text{short}(w) = T_1 = 33$, $\text{long}(w) = T_0 = 38$

$$r : \begin{aligned} T_0(r) &= l_0 + \text{short}(w) &= 45 \\ T_1(r) &= l_1 + \text{long}(w) &= 44 \end{aligned}$$

Running Time Analysis

- recurrence can be solved naively in $O(mk^2)$ time for m segments in the k-plane input drawing Γ
- can be improved to $O(mk)$ time using dependencies in the order of the stub lengths
- intersection graph $C(\Gamma)$ is a tree with $O(m)$ edges and can be computed in $O(m \log m)$ time

Running Time Analysis

- recurrence can be solved naively in $O(mk^2)$ time for m segments in the k-plane input drawing Γ
- can be improved to $O(mk)$ time using dependencies in the order of the stub lengths
- intersection graph $C(\Gamma)$ is a tree with $O(m)$ edges and can be computed in $O(m \log m)$ time

Theorem: MaxSPED can be solved in $O(mk + m \log m)$ time for a k-plane input drawing whose intersection graph is a tree.

Running Time Analysis

- recurrence can be solved naively in $O(mk^2)$ time for m segments in the k-plane input drawing Γ
- can be improved to $O(mk)$ time using dependencies in the order of the stub lengths
- intersection graph $C(\Gamma)$ is a tree with $O(m)$ edges and can be computed in $O(m \log m)$ time

Theorem: MaxSPED can be solved in $O(mk + m \log m)$ time for a k-plane input drawing whose intersection graph is a tree.

MaxPED: similar algorithm idea, but non-symmetric stubs require k^2 pairs of stub lengths.

Theorem: MaxPED can be solved in $O(mk^2 + m \log m)$ time for k-plane input drawing with tree intersection graph.

Bounded Treewidth

If the edge intersection graph $C(\Gamma)$ has bounded treewidth τ then a more complex dynamic programming idea can be used.

- each node (bag) of a *nice* tree decomposition of C has at most $\tau + 1$ vertices; for a k-plane drawing Γ it is sufficient to store maximum ink values for at most $(k+1)^{\tau+1}$ stub sets
- perform bottom-up dynamic programming in the nice tree decomposition, which has $O(\tau m)$ nodes
- the operations for one stub set require at most $O(k\tau)$ time

Bounded Treewidth

If the edge intersection graph $C(\Gamma)$ has bounded treewidth τ then a more complex dynamic programming idea can be used.

- each node (bag) of a *nice* tree decomposition of C has at most $\tau + 1$ vertices; for a k-plane drawing Γ it is sufficient to store maximum ink values for at most $(k+1)^{\tau+1}$ stub sets
- perform bottom-up dynamic programming in the nice tree decomposition, which has $O(\tau m)$ nodes
- the operations for one stub set require at most $O(k\tau)$ time

Theorem: For a k-plane drawing Γ with m edges whose intersection graph has treewidth τ, MaxSPED can be solved in $O(m(k+1)^{\tau+2}\tau^2 + m \log m)$ time.

Bounded Treewidth

If the edge intersection graph $C(\Gamma)$ has bounded treewidth τ then a more complex dynamic programming idea can be used.

- each node (bag) of a *nice* tree decomposition of C has at most $\tau + 1$ vertices; for a k-plane drawing Γ it is sufficient to store maximum ink values for at most $(k+1)^{\tau+1}$ stub sets
- perform bottom-up dynamic programming in the nice tree decomposition, which has $O(\tau m)$ nodes
- the operations for one stub set require at most $O(k\tau)$ time

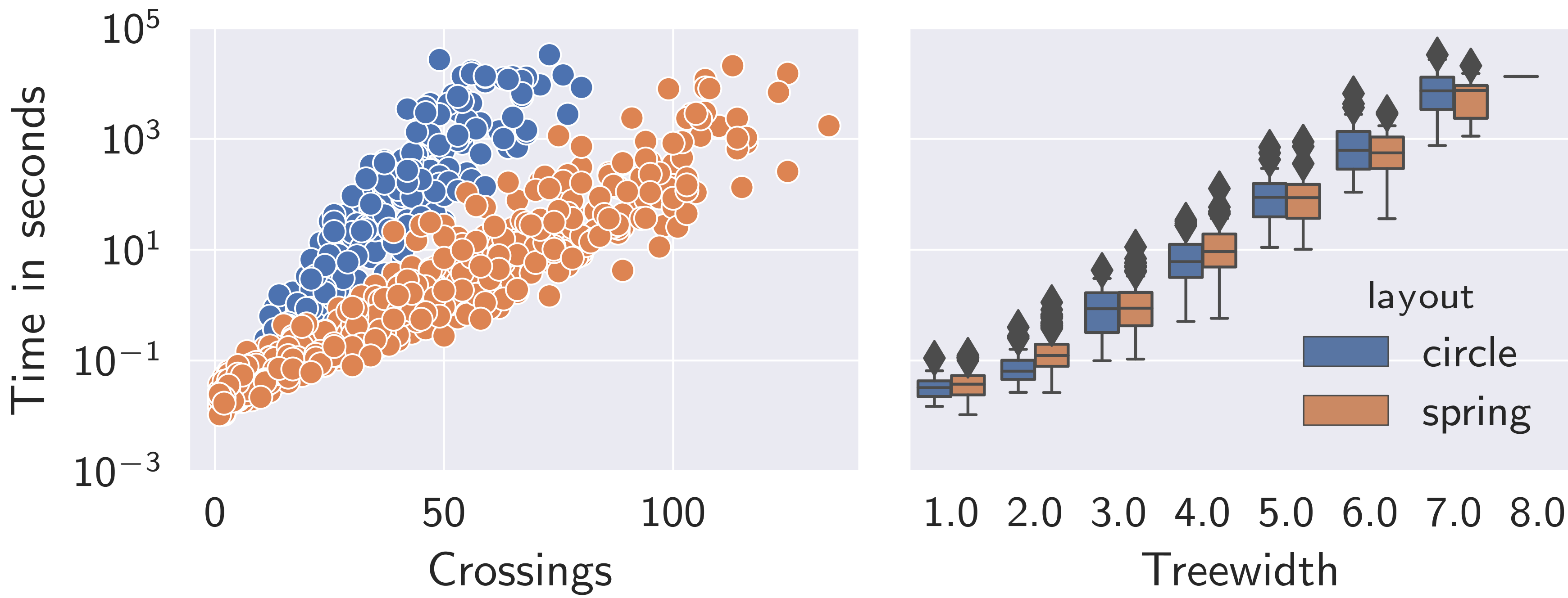
Theorem: For a k-plane drawing Γ with m edges whose intersection graph has treewidth τ, MaxSPED can be solved in $O(m(k+1)^{\tau+2}\tau^2 + m \log m)$ time.

The algorithm can be adapted to solve MaxPED with an increase by a factor of k in the running time.

Experiments

We implemented the treewidth-based algorithms for MaxSPED in Python and performed some proof-of-concept experiments.

- used "htd" library to compute nice tree decomposition
- 800 random graphs with 40 vertices and 40–75 edges
- spring and circular layouts from NetworkX and graphviz



Conclusion

	$k = 2$	$k = 3$	$k \geq 4$	arbitrary k
MaxSPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]	**NP-hard**		NP-hard [Bruckdorfer PhD'15]
MaxPED			**NP-hard**	

Dynamic Programming if edge intersection graph

- is a **tree**, or more generally
- has **bounded treewidth**

Conclusion

	$k = 2$	$k = 3$	$k \geq 4$	arbitrary k
MaxSPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]	**NP-hard**	**NP-hard**	NP-hard [Bruckdorfer PhD'15]
MaxPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]	**?**	**NP-hard**	

Dynamic Programming if edge intersection graph

- is a **tree**, or more generally
- has **bounded treewidth**

open questions:

- complexity of MaxPED for $k = 3$
- algorithms/complexity for deciding existence of δ-HPEDs

Conclusion

	$k = 2$	$k = 3$	$k \geq 4$	arbitrary k
MaxSPED	$O(n \log n)$ [Bruckdorfer et al. JGAA'17]	**NP-hard**		NP-hard [Bruckdorfer PhD'15]
MaxPED		**?**	**NP-hard**	

Dynamic Programming if edge intersection graph

- is a **tree**, or more generally
- has **bounded treewidth**

open questions:

- complexity of MaxPED for $k = 3$
- algorithms/complexity for deciding existence of δ-HPEDs

Thank You!