Maximizing Ink in Partial Edge Drawings of $k$-plane Graphs

Matthias Hummel, Fabian Klute, Soeren Nickel, Martin Nöllenburg
Partial Edge Drawings (PED)

How to draw non-planar graphs?

Input:
Straight-line graph drawing with crossings

Output:
“Crossing-free” partial edge drawing (PED)

Just hide the edge crossings!

[Becker et al. TVCG'95], [Bruckdorfer, Kaufmann FUN'12]
Partial Edge Drawings (PED)

How to draw non-planar graphs?

Just hide the edge crossings!

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

[Bruckdorfer et al. GD’15], [Burch et al. GD’11]

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

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

Input: Straight-line graph drawing with crossings

Output: “Crossing-free” partial edge drawing (PED)
Symmetric Partial Edge Drawings (SPED)

Input drawing

PED

symmetric PED (SPED)
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)

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

Optimization problem: maximize total stub length/drawn ink
→ **MaxPED** and **MaxSPED**

- show as much information as possible without crossings
Overview of Results

**Given:** \(k\)-plane* straight-line drawing \(\Gamma\)

**Find:** maximum-ink (S)PED of \(\Gamma\)
Overview of Results

**Given:** $k$-plane* straight-line drawing $\Gamma$

**Find:** maximum-ink (S)PED of $\Gamma$

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

ex: $k = 2$

$\Gamma$
Overview of Results

**Given:** \( k \)-plane* straight-line drawing \( \Gamma \)

**Find:** maximum-ink (S)PED of \( \Gamma \)

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

| \( k \) | MaxPED | MaxSPED |
|------|--------|---------|
| 2    | \( O(n \log n) \) [Bruckdorfer et al. JGAA'17] | \( O(n \log n) \) [Bruckdorfer et al. JGAA'17] |
| 3    |         |         |
| \( \geq 4 \) |         |         |
| arbitrary \( k \) |         |         |

*NP-hard* [Bruckdorfer PhD'15]
Overview of Results

**Given:** \( k \)-plane* straight-line drawing \( \Gamma \)

**Find:** maximum-ink (S)PED of \( \Gamma \)

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

---

|                  | \( k = 2 \) | \( k = 3 \) | \( k \geq 4 \) | arbitrary \( k \) |
|------------------|-------------|-------------|----------------|-------------------|
| MaxPED           | \( O(n \log n) \) | \text{NP-hard} | \text{NP-hard} | \text{NP-hard} [Bruckdorfer PhD'15] |
| MaxSPED          |             |             | \text{NP-hard} |                   |

[Bruckdorfer et al. JGAA'17]
Overview of Results

Given: $k$-plane* straight-line drawing $\Gamma$

Find: maximum-ink (S)PED of $\Gamma$

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

| $k = 2$ | $k = 3$ | $k \geq 4$ | arbitrary $k$ |
|---------|---------|------------|---------------|
| MaxPED  | MaxPED  | MaxPED     | MaxPED        |
| $O(n \log n)$ | $O(n \log n)$ | $\text{NP-hard}$ | $\text{NP-hard}$ |
| [Bruckdorfer et al. JGAA'17] | [Bruckdorfer PhD'15] | Dynamic Programming if edge intersection graph |
| is a tree, or more generally |
| has bounded treewidth |

Dynamic Programming if edge intersection graph
NP-Hardness
NP-Hardness of MaxSPED

- reduction from Planar 3SAT
- gadget-based reduction
NP-Hardness of MaxSPED

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

Planar 3SAT formula:

\[
\begin{align*}
    &x_1 \lor x_4 \lor x_5 \\
    &x_2 \lor x_3 \lor x_4 \\
    &x_1 \lor x_2 \lor x_3 \\
    &x_1 \lor x_3 \lor x_4
\end{align*}
\]

Variable gadgets:

- \(x_1 = true\)
- \(x_2 = false\)
- \(x_3 = false\)
NP-Hardness of MaxSPED

- reduction from Planar 3SAT
- gadget-based reduction
  - variable gadgets: 2 optimal states
NP-Hardness of MaxSPED

- reduction from \textsc{Planar 3SAT}
- gadget-based reduction
  - variable gadgets: 2 optimal states
NP-Hardness of MaxSPED

- reduction from \textsc{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

\begin{align*}
    x_1 & = \text{true} \\
    x_2 & = \text{false} \\
    x_3 & = \text{false}
\end{align*}
NP-Hardness of MaxSPED

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

planar 3SAT formula

$\begin{align*}
\text{variable gadgets: } & x_1 = \text{true} \\
\text{clause gadgets: } & \\
\end{align*}$
NP-Hardness of MaxSPED

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

planar 3SAT formula

\[ x_1 \lor x_4 \lor x_5 \]
\[ x_2 \lor x_3 \lor x_4 \]
\[ x_1 \lor x_2 \lor x_3 \]
\[ x_1 \lor x_3 \lor x_4 \]

\[ x_1 = \text{true} \]
\[ x_2 = \text{false} \]
\[ x_3 = \text{false} \]
NP-Hardness of MaxSPED

- reduction from \textsc{Planar 3SAT}
- gadget-based reduction
  - variable gadgets: 2 optimal states
  - clause gadgets: 3 optimal states

Planar 3SAT formula:

\[
\begin{align*}
  x_1 &\lor x_2 &\lor x_3 \\
  x_2 &\lor x_3 &\lor x_4 \\
  x_1 &\lor x_2 &\lor x_3 \\
  x_1 &\lor x_3 &\lor x_4 \\
  x_1 &\lor x_4 &\lor x_5 \\
  x_1 &\lor x_3 &\lor x_4
\end{align*}
\]

Variable states:

\[
\begin{align*}
  x_1 &= \text{true} \\
  x_2 &= \text{false} \\
  x_3 &= \text{false} \\
  x_4 &= 5 \\
  x_5 &= \text{true}
\end{align*}
\]
NP-Hardness of MaxSPED

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

Planar 3SAT formula

\[
x_1 \lor x_4 \lor x_5 \\
x_2 \lor x_3 \lor x_4 \\
x_1 \lor x_2 \lor x_3 \\
x_1 \lor x_3 \lor x_4
\]

\[
x_1 = \text{true} \\
x_2 = \text{false} \\
x_3 = \text{false}
\]
NP-Hardness of MaxSPED

- reduction from \textsc{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

planar 3SAT formula

\[
\begin{align*}
x_1 \lor x_4 \lor x_5 \\
x_2 \lor x_3 \lor x_4 \\
x_1 \lor \overline{x_2} \lor x_3 \\
x_1 \lor x_3 \lor x_4 \\
x_1 \lor x_2 \lor x_3 \\
x_1 \lor x_2 \lor x_3 \\
x_1 \lor x_2 \lor x_3 \\
x_1 \lor x_2 \lor x_3
\end{align*}
\]

\[x_1 = \text{false}\]

\[x_2 = \text{false}\]

\[x_3 = \text{false}\]
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

planar 3SAT formula

\[ x_1 \lor x_4 \lor x_5 \]
\[ x_2 \lor x_3 \lor x_4 \]
\[ x_1 \lor x_2 \lor x_3 \]
\[ x_1 \lor x_3 \lor x_4 \]

\[ x_1 = \text{false} \]
\[ x_2 = \text{true} \]
\[ x_3 = \text{false} \]
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

Planar 3SAT formula

$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$

$x_1 = false$, $x_2 = true$, $x_3 = false$
NP-Hardness of MaxSPED

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

![Planar 3SAT formula and gadget-based reduction diagram]
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

![Planar 3SAT formula]

\[
x_1 \lor x_4 \lor x_5
\]
\[
x_2 \lor x_3 \lor x_4
\]
\[
x_1 \lor x_2 \lor x_3
\]
\[
x_1 \lor x_3 \lor x_4
\]

\[
x_1 = \text{false}
\]
\[
x_2 = \text{true}
\]
\[
x_3 = \text{false}
\]
NP-Hardness of MaxSPED

- reduction from \textsc{Planar }\textsc{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

\begin{align*}
&x_1 &\lor &x_4 &\lor &x_5 \\
&x_2 &\lor &x_3 &\lor &x_4 \\
&x_1 &\lor &x_2 &\lor &x_3 \\
&x_1 &\lor &x_3 &\lor &x_4 \\
\end{align*}
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

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 $\Gamma$.
- Edge $(u, v)$ in $C$ iff $s(u)$ and $s(v)$ cross in $\Gamma$.

**drawing $\Gamma$ of $G = (V, E)$**

**edge intersection graph $C(\Gamma)$ with vertex set $E$**

---

- $C(\Gamma)$ is constructed from $\Gamma$ by drawing each edge as a line segment and connecting vertices if their corresponding segments cross.
- The resulting graph $C(\Gamma)$ represents the intersection of line segments in $\Gamma$.

---

**Diagram:**

1. Left: A drawing $\Gamma$ of a graph $G = (V, E)$ with multiple edges.
2. Right: The corresponding edge intersection graph $C(\Gamma)$, showing only the vertices and edges that result from the intersection of line segments in $\Gamma$.
Edge Intersection Graph

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

| Drawing $\Gamma$ of $G = (V, E)$ | Edge intersection graph $C(\Gamma)$ with vertex set $E$ |
|-------------------------------------|----------------------------------------------------------|
| ![Drawing](image)                    | ![Edge Intersection Graph](image)                       |
Edge Intersection Graph

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

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)

\[ s(u) \]
\[ s(p(u)) \]
\[ s(v_1) \quad s(v_2) \quad s(v_4) \quad s(v_3) \]
\[ \bullet \quad \bullet \quad \bullet \quad \bullet \]
\[ \Rightarrow \quad p(u) \quad v_1 \quad v_2 \quad v_3 \quad v_4 \]
\[ c(u) \]
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

$$s(u), s(p(u))$$

$\Rightarrow$

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

edge crossings induce different relevant stub lengths

$\Rightarrow$

\[
l_0 \rightarrow \text{entire edge (no gap)} \quad l_1, \ldots, l_{\deg(u)} \rightarrow \text{shorter to longer stubs}
\]
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

\[
s(u) \quad s(p(u)) \quad l_2
\]

\[
\begin{align*}
& s(v_1) \quad s(v_2) \quad s(v_4) \quad s(v_3) \\
& c(u) \quad u \quad v_1 \quad v_2 \quad v_3 \quad v_4
\end{align*}
\]
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

\[
\begin{align*}
  s(u) & \quad s(p(u)) \\
  s(v_1) & \quad s(v_2) \\
  s(v_4) & \quad s(v_3)
\end{align*}
\]

\[
\frac{\text{edge crossings induce different relevant stub lengths}}{
\begin{align*}
  l_0 & \quad \text{entire edge (no gap)} \\
  l_1, \ldots, l_{\deg(u)} & \quad \text{shorter to longer stubs}
\end{align*}
\]
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

\[ s(u), s(p(u)), s(v_1), s(v_2), s(v_3), s(v_4) \]

\[ l_4 \text{ edge crossings induce different relevant stub lengths} \]

\[ l_0 \text{ – entire edge (no gap)} \]
\[ l_1, \ldots, l_{\deg(u)} \text{ – shorter to longer stubs} \]
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

\[ s(u) \quad s(p(u)) \quad \begin{array}{c}
\vdots
s(v_1) \quad s(v_2) \\
\vdots
s(v_4) \\
\vdots
s(v_3)
\end{array} \quad l_5 \quad \Rightarrow \quad \begin{array}{c}
\begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \quad \circ \\
\vdots
\vdots
\vdots
\vdots
\vdots
\end{array} \\
\begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \quad \circ \\
\vdots
\vdots
\vdots
\vdots
\vdots
\end{array} \\
\begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \quad \circ \\
\vdots
\vdots
\vdots
\vdots
\vdots
\end{array} \\
\begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \quad \circ \\
\vdots
\vdots
\vdots
\vdots
\vdots
\end{array} \\
\begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \quad \circ \\
\vdots
\vdots
\vdots
\vdots
\vdots
\end{array}
\end{array} \quad p(u) \quad u \quad v_1 \quad v_2 \quad v_3 \quad v_4 \quad c(u)

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

\[ s(u) \quad s(p(u)) \quad s(v_1) \quad s(v_2) \quad s(v_4) \quad s(v_3) \]

- 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

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

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

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

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

\[ \text{short}(v) = \max\{T_1(v), \ldots, T_p(v)\} \]

... stubs not affecting the parent

\[ \text{long}(v) = \max\{T_0(v), \ldots, T_{\deg(v)}(v)\} \]

... 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:

\[ v_1 : T_0(v_1) = l_0 = 3 \]
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:

\[ T_0(v_1) = l_0 = 3 \]
\[ T_1(v_1) = l_1 = 2 \]
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:

\begin{align*}
V_1 &: \quad T_0(v_1) = 3 \\
& \quad T_1(v_1) = 2 \\
V_2 &: \quad T_0(v_2) = l_0 = 5 \\
& \quad T_1(v_2) = l_1 = 4 \\
V_3 : &\quad \text{...}
\end{align*}
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:

\[
\begin{align*}
T_0(v_1) &= 3 \\
T_1(v_1) &= 2 \\
T_0(v_2) &= 5 \\
T_1(v_2) &= 4 \\
v_3 : T_0(v_3) &= l_0 = 4 \\
v_3 : T_1(v_3) &= l_1 = 2
\end{align*}
\]
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:

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

Intersection Graph:

\[ v_1 : T_0(v_1) = 3 \]
\[ T_1(v_1) = 2 \]
\[ v_2 : T_0(v_2) = 5 \]
\[ T_1(v_2) = 4 \]
\[ v_3 : T_0(v_3) = 4 \]
\[ T_1(v_3) = 2 \]
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:

\[
\begin{align*}
T_0(u_1) & = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) = 21 \\
T_1(u_1) & = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 14
\end{align*}
\]

Intersection Graph:

\[
\begin{align*}
v_1: & \quad T_0(v_1) = 3 \\
& \quad T_1(v_1) = 2 \\
v_2: & \quad T_0(v_2) = 5 \\
& \quad T_1(v_2) = 4 \\
v_3: & \quad T_0(v_3) = 4 \\
& \quad T_1(v_3) = 2
\end{align*}
\]
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:**

\[ T_0(u_1) = l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) = 21 \]
\[ T_1(u_1) = l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 14 \]
\[ u_1 : T_2(u_2) = l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 15 \]

**Intersection Graph:**

\[ v_1 : \begin{array}{c} T_0(v_1) = 3 \\
T_1(v_1) = 2 \end{array} \]
\[ v_2 : \begin{array}{c} T_0(v_2) = 5 \\
T_1(v_2) = 4 \end{array} \]
\[ v_3 : \begin{array}{c} T_0(v_3) = 4 \\
T_1(v_3) = 2 \end{array} \]
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:

\[
\begin{align*}
T_0(u_1) &= l_0 + \text{short}(v_1) + \text{short}(v_2) + \text{short}(v_3) = 21 \\
T_1(u_1) &= l_1 + \text{long}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 14 \\
u_1: T_2(u_2) &= l_2 + \text{short}(v_1) + \text{long}(v_2) + \text{long}(v_3) = 15 \\
T_3(u_3) &= l_3 + \text{short}(v_1) + \text{short}(v_2) + \text{long}(v_3) = 16
\end{align*}
\]

\[
\begin{array}{c|c}
\text{vertex} & T_0(v) \quad T_1(v) \\
\hline
v_1 & 3 \quad 2 \\
v_2 & 5 \quad 4 \\
v_3 & 4 \quad 2 \\
\end{array}
\]
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:

- \( v_1 : T_{0}(v_1) = 3 \)
- \( T_{1}(v_1) = 2 \)
- \( v_2 : T_{0}(v_2) = 5 \)
- \( T_{1}(v_2) = 4 \)
- \( v_3 : T_{0}(v_3) = 4 \)
- \( T_{1}(v_3) = 2 \)

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

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

\[
\begin{align*}
v_1: T_0(v_1) &= 3 \\
& \quad T_1(v_1) = 2 \\
v_2: T_0(v_2) &= 5 \\
& \quad T_1(v_2) = 4 \\
v_3: T_0(v_3) &= 4 \\
& \quad T_1(v_3) = 2 \\
\end{align*}
\]

\[ \text{short}(u_1) = \max\{T_1, T_2, T_3\} = T_3(u_1) = 16 \]
**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:**

  \[
  \begin{align*}
  v_1 &: T_0(v_1) = 3 \\
  & T_1(v_1) = 2
  \\
  v_2 &: T_0(v_2) = 5 \\
  & T_1(v_2) = 4
  \\
  v_3 &: T_0(v_3) = 4 \\
  & T_1(v_3) = 2
  \end{align*}
  \]

- **Graph:**

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

- **Long:**

  \[
  \text{long}(u_1) = \max\{T_0, \ldots, T_4\} = T_4(u_1) = 22
  \]
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:

\[
\begin{align*}
\begin{array}{c|c}
\text{vertex} & T_0(\text{vertex}) \\
v_1 & 3 \\
v_2 & 5 \\
v_3 & 4 \\
u_1 & 2 \\
u_2 & 2 \\
u_3 & 2 \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
u_1 : & \quad T_0(u_1) = 3 \\
& \quad T_1(u_1) = 2 \\
v_2 : & \quad T_0(v_2) = 5 \\
& \quad T_1(v_2) = 4 \\
v_3 : & \quad T_0(v_3) = 4 \\
& \quad T_1(v_3) = 2 \\
u_1 : & \quad \text{short}(u_1) = T_3 = 16 \\
& \quad \text{long}(u_1) = T_4 = 22 \\
u_2 : & \quad T_0(u_2) = 5 \\
& \quad T_1(u_2) = 2
\end{align*}
\]
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:**

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

\[ T_0(r) = l_0 + \text{short}(w) = 45 \]
\[ T_1(r) = l_1 + \text{long}(w) = 44 \]
Dynamic Programming: Example

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

**Input:**

![Graph](image)

**Intersection Graph:**

|       | \( T_0(v_1) \) | \( T_1(v_1) \) |
|-------|----------------|----------------|
| \( v_1 \) | 3              | 2              |
| \( v_2 \) | 5              | 4              |
| \( v_3 \) | 4              | 2              |

|       | \( T_0(u_2) \) | \( T_1(u_2) \) |
|-------|----------------|----------------|
| \( u_2 \) | 5              | 2              |

|       | \( T_0(w) \) | \( T_1(w) \) |
|-------|--------------|--------------|
| \( w \) | 33           | 38           |

\[ T_0(r) = l_0 + \text{short}(w) = 45 \]
\[ T_1(r) = l_1 + \text{long}(w) = 44 \]
Dynamic Programming: Example

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

Input:

Intersection Graph:

\[
\begin{array}{l}
\text{Input:} \\
\text{Intersection Graph:}
\end{array}
\]

\[
\begin{array}{l}
T_0(u_1) = 3 \\
T_1(u_1) = 2 \\
T_0(v_2) = 5 \\
T_1(v_2) = 4 \\
T_0(u_2) = 5 \\
T_1(u_2) = 2 \\
T_0(v_3) = 4 \\
T_1(v_3) = 2 \\
T_0(v_1) = 3 \\
T_1(v_1) = 2 \\
T_0(w) = 33 \\
T_1(w) = 38 \\
T_0(r) = 45 \\
T_1(r) = 44 \\
\end{array}
\]

\[ r : T_0(r) = l_0 + \text{short}(w) = 45 \quad \leftarrow \text{backtracking} \]

\[ r : T_1(r) = l_1 + \text{long}(w) = 44 \]
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:

\[
T_0(r) = l_0 + \text{short}(w) = 45 \\
T_1(r) = l_1 + \text{long}(w) = 44
\]

\[v_1: \begin{array}{c}
T_0(v_1) = 3 \\
T_1(v_1) = 2
\end{array}
\]

\[v_2: \begin{array}{c}
T_0(v_2) = 5 \\
T_1(v_2) = 4
\end{array}
\]

\[v_3: \begin{array}{c}
T_0(v_3) = 4 \\
T_1(v_3) = 2
\end{array}
\]

\[u_1: \begin{array}{c}
\text{short}(u_1) = T_3 = 16 \\
\text{long}(u_1) = T_4 = 22
\end{array}
\]

\[u_2: \begin{array}{c}
T_0(u_2) = 5 \\
T_1(u_2) = 2
\end{array}
\]

\[w: \begin{array}{c}
\text{short}(w) = T_1 = 33 \\
\text{long}(w) = T_0 = 38
\end{array}
\]
Running Time Analysis

- Recurrence can be solved naively in $O(mk^2)$ time for $m$ segments in the $k$-plane input drawing $\Gamma$
- 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 $\Gamma$
- 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 $\Gamma$
- 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 $\tau$ 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 $\Gamma$ 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 $\tau$ 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 $\Gamma$ 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 $\Gamma$ with $m$ edges whose intersection graph has treewidth $\tau$, 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 $\tau$ 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 $\Gamma$ 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 $\Gamma$ with $m$ edges whose intersection graph has treewidth $\tau$, 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 $k$ |
|---|---|---|---|
| MaxPED | $O(n \log n)$ | NP-hard | NP-hard [Bruckdorfer PhD’15] |
|  | [Bruckdorfer et al. JGAA’17] | **Dynamic Programming** if edge intersection graph is a tree, or more generally has bounded treewidth | |
## Conclusion

| $k$       | MaxPED      | $k = 3$      | $k \geq 4$      | arbitrary $k$      |
|-----------|-------------|--------------|-----------------|-------------------|
| $k = 2$   | $O(n \log n)$ [Bruckdorfer et al. JGAA’17] | Dynamic Programming if edge intersection graph | | NP-hard [Bruckdorfer PhD’15] |
| $k = 3$   | NP-hard     |              |                 |                   |
| $k \geq 4$|              |              |                 |                   |

### open questions:
- complexity of MaxPED for $k = 3$
- algorithms/complexity for deciding existence of $\delta$-HPEDs
## Conclusion

| $k = 2$          | $k = 3$          | $k \geq 4$        | arbitrary $k$          |
|------------------|------------------|-------------------|------------------------|
| MaxPED            | MaxPED           | $O(n \log n)$     | $O(n \log n)$          |
| $\text{NP-hard}$ | $\text{NP-hard}$ |                   | [Bruckdorfer et al. JGAA'17] |

### Open Questions:

- Complexity of MaxPED for $k = 3$
- Algorithms/complexity for deciding existence of $\delta$-HPEDs

**Dynamic Programming** if edge intersection graph
- is a **tree**, or more generally
- has **bounded treewidth**

**Thank You!**