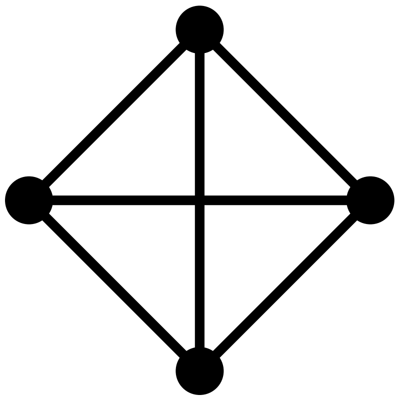
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG
Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems
Institute for Informatics

Smooth Orthogonal Drawings of Planar Graphs

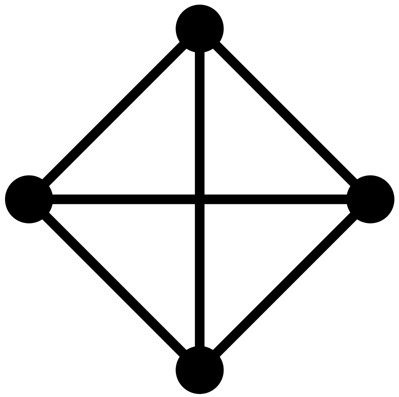
Philipp Kindermann
Chair of Computer Science I
Universität Würzburg

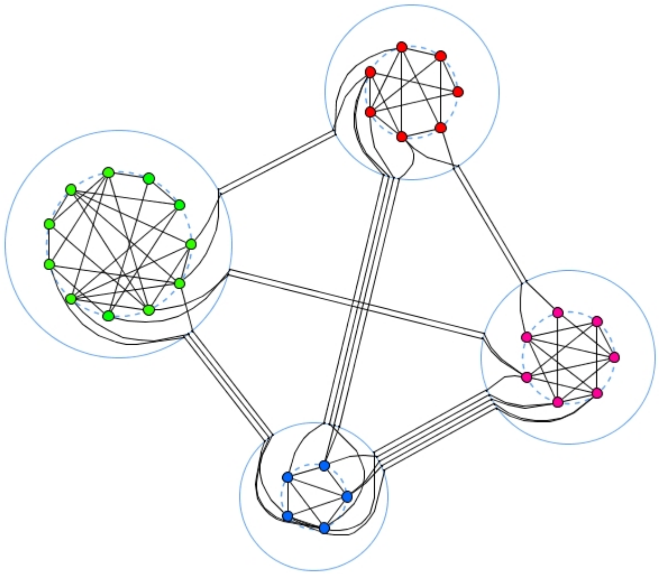
Joint work with
Md. Jawaherul Alam, Michael A. Bekos, Michael Kaufmann,
Stephen G. Kobourov & Alexander Wolff

Drawings of Graphs

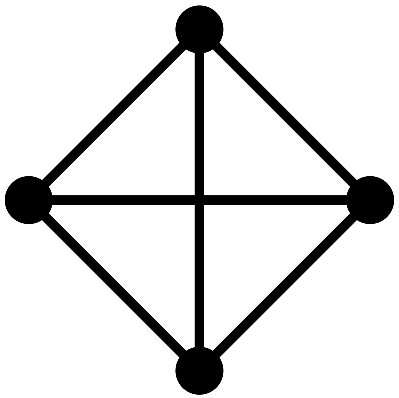


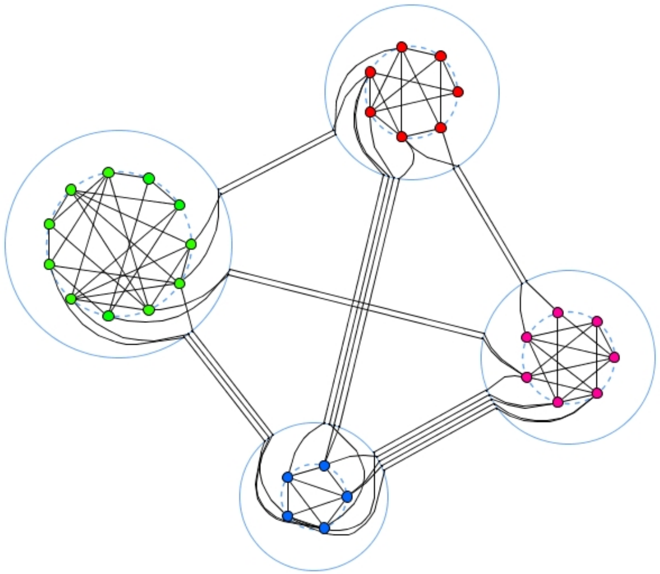
Drawings of Graphs

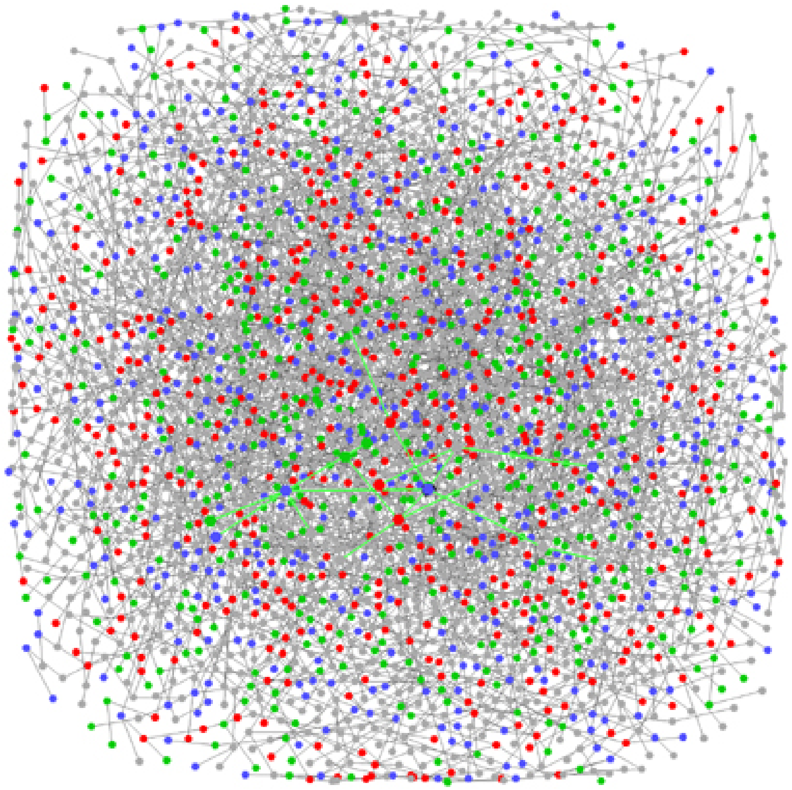




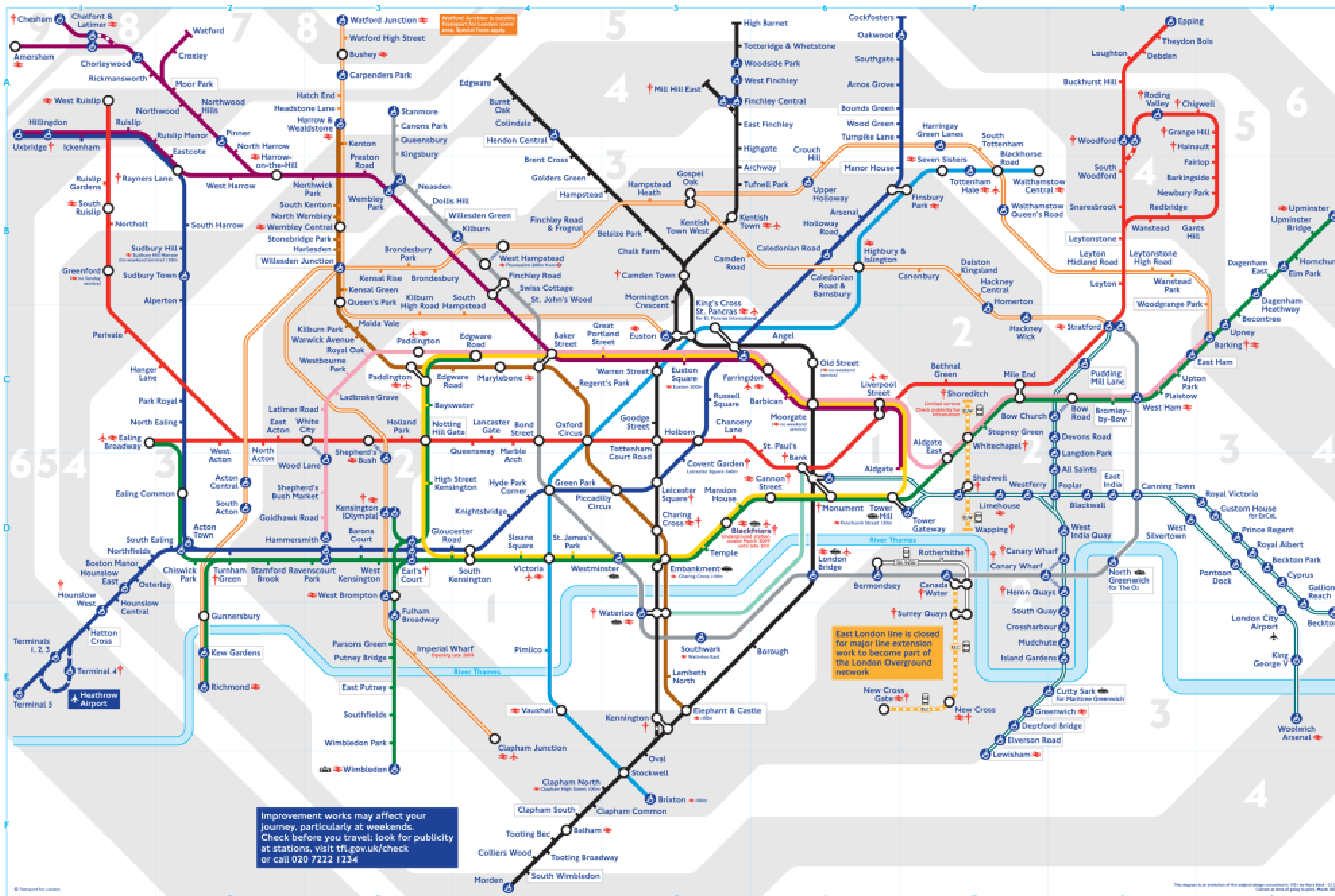
Drawings of Graphs



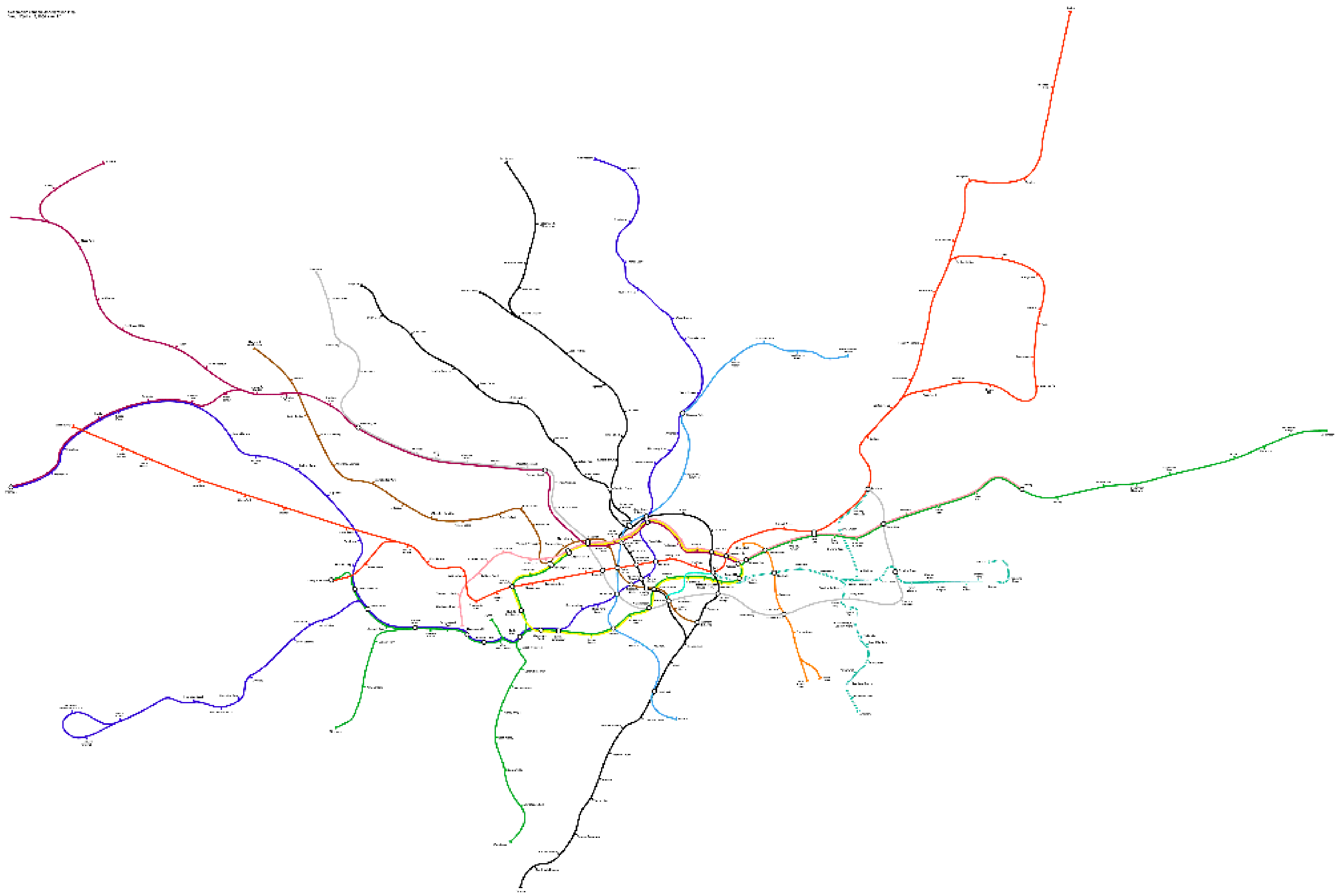




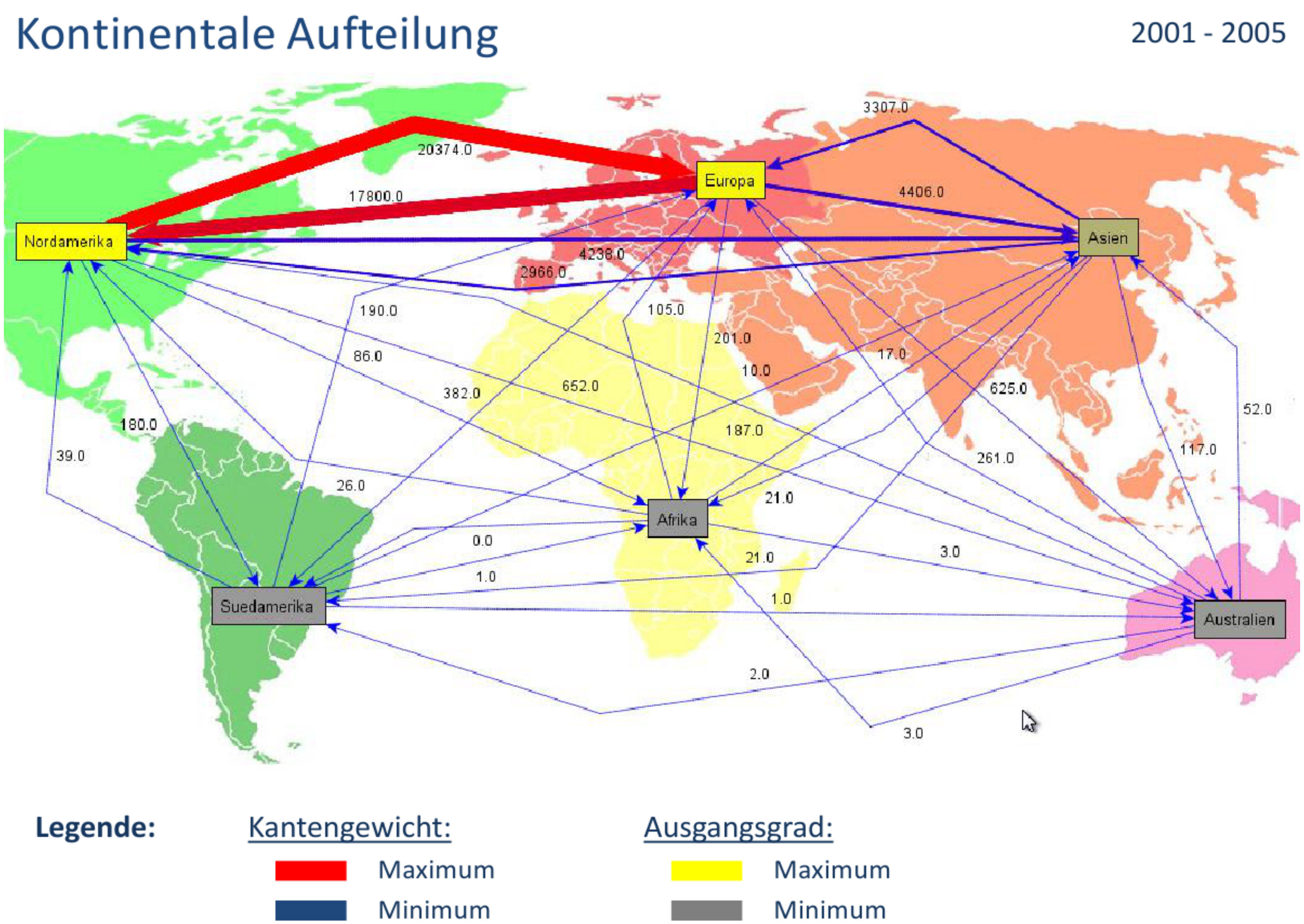
Drawings of Graphs



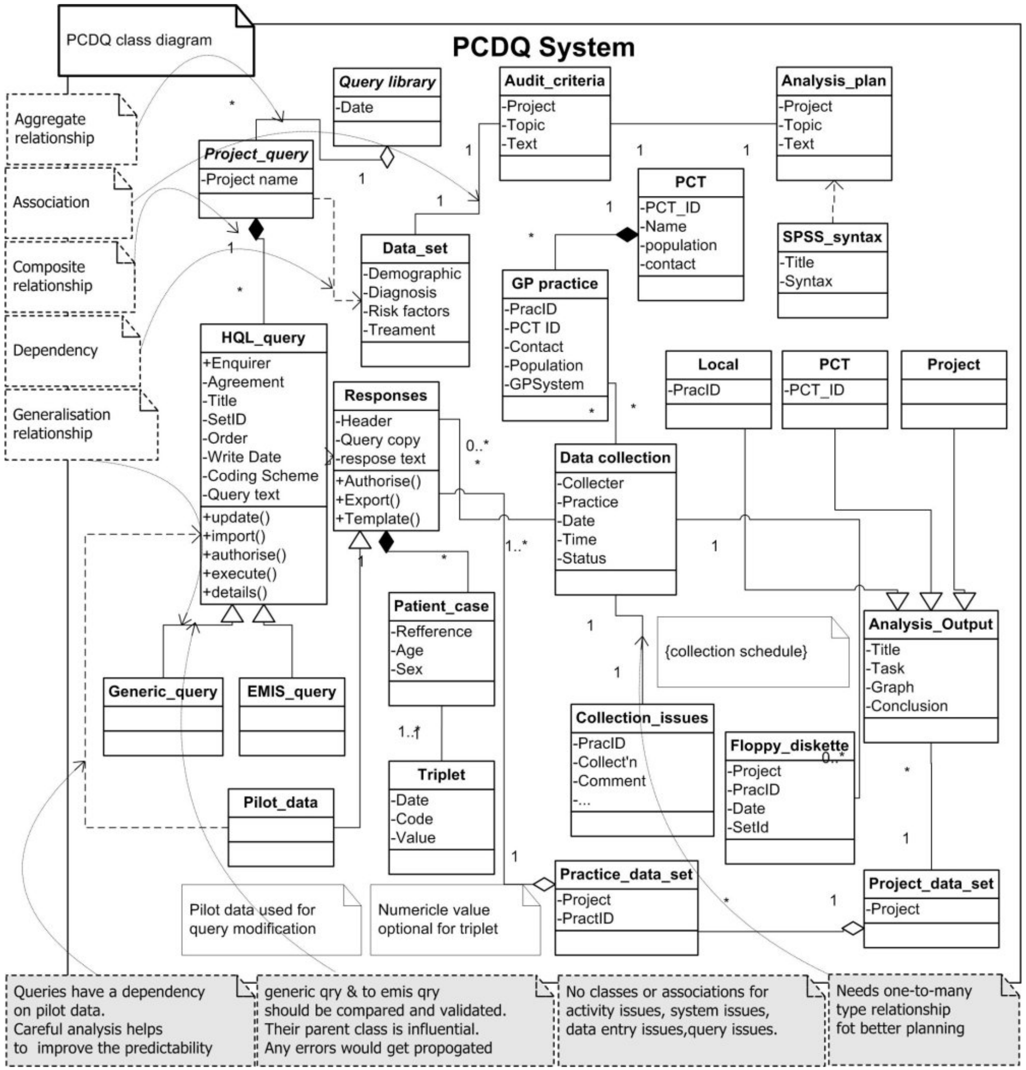
Drawings of Graphs



Drawings of Graphs



Drawings of Graphs



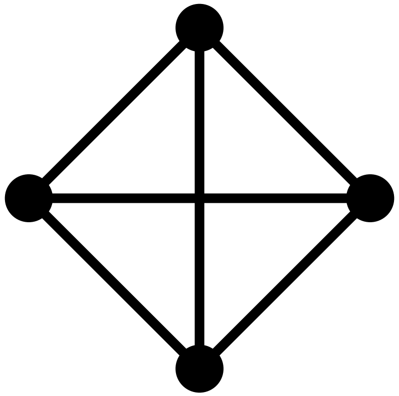
Drawing Styles

$$G = (V, E)$$

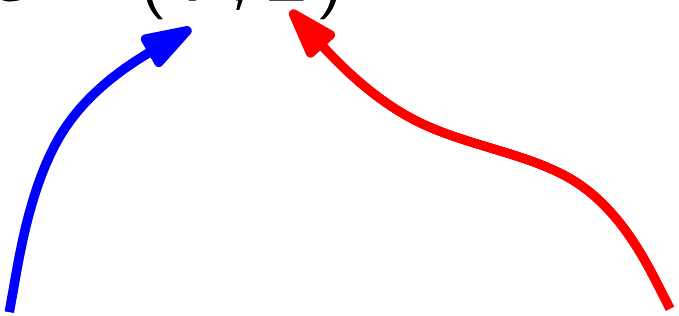
nodes $v \in V$

edge $\{u, v\}$ connecting u and v

Drawing Styles



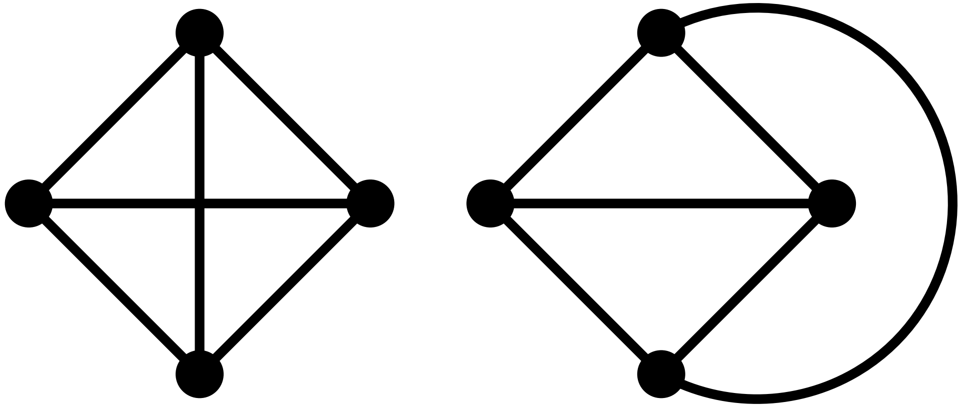
$G = (V, E)$



nodes $v \in V$

edge $\{u, v\}$ connecting u and v

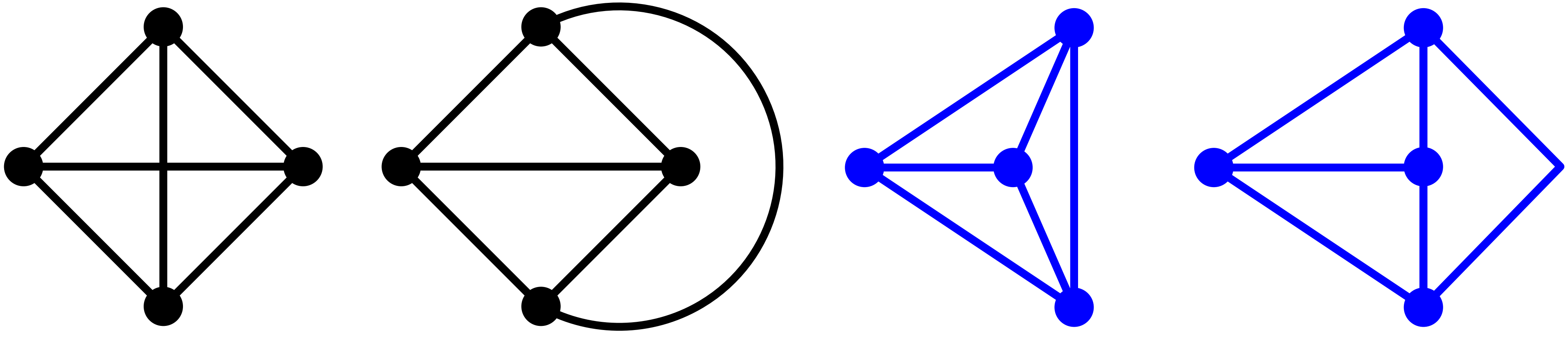
Drawing Styles



$G = (V, E)$

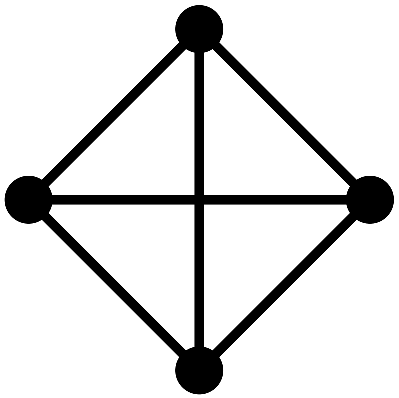
with/without crossings

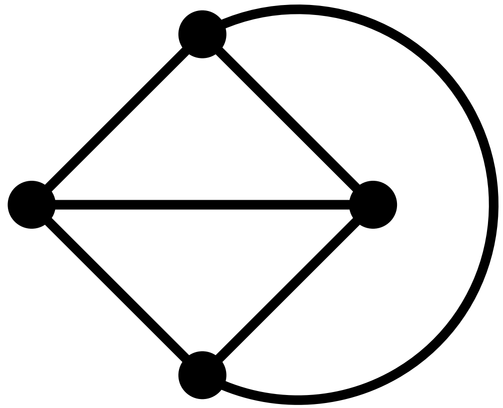
Drawing Styles

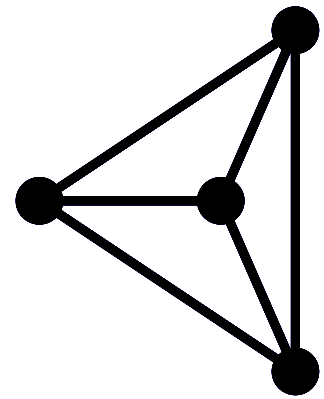


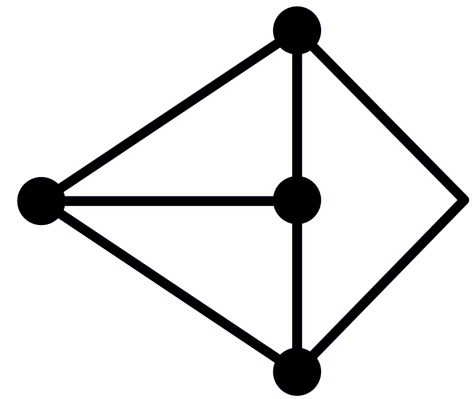
straight-line edges or with *bends*

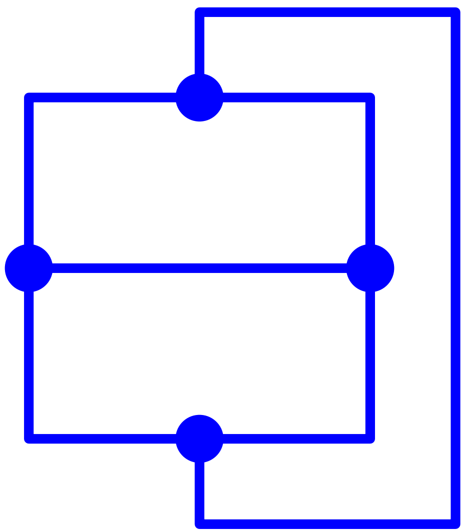
Drawing Styles





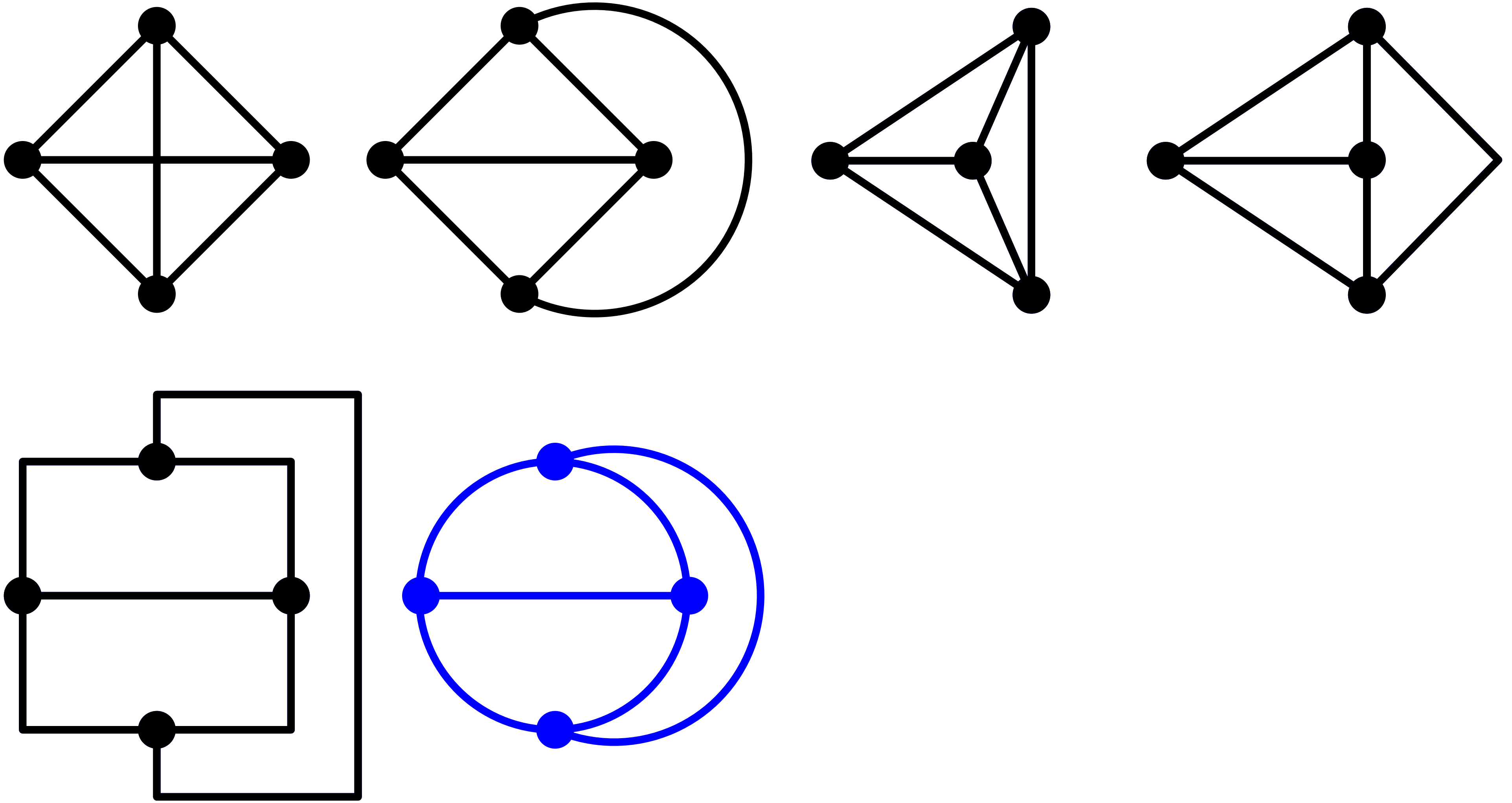






orthogonal

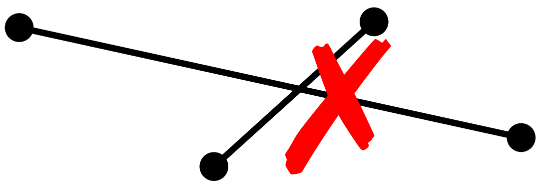
Drawing Styles



curvy edges

Planar Graphs

[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings



Planar Graphs

[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings

[Hopcroft & Tarjan, J. ACM '74]
Let G be a graph with n nodes.
It can be checked in $O(n)$ time whether G is planar.

Planar Graphs

[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings

[Hopcroft & Tarjan, J. ACM '74]
Let G be a graph with n nodes.
It can be checked in $O(n)$ time whether G is planar.

[Wagner 1936, Fáry 1948, Stein 1951]
Any planar graph can be drawn with *straight-line* edges.

Planar Graphs

[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings

[Hopcroft & Tarjan, J. ACM '74]
Let G be a graph with n nodes.
It can be checked in $O(n)$ time whether G is planar.

draw?

[Wagner 1936, Fáry 1948, Stein 1951]
Any planar graph can be drawn with *straight-line* edges.

Planar Graphs

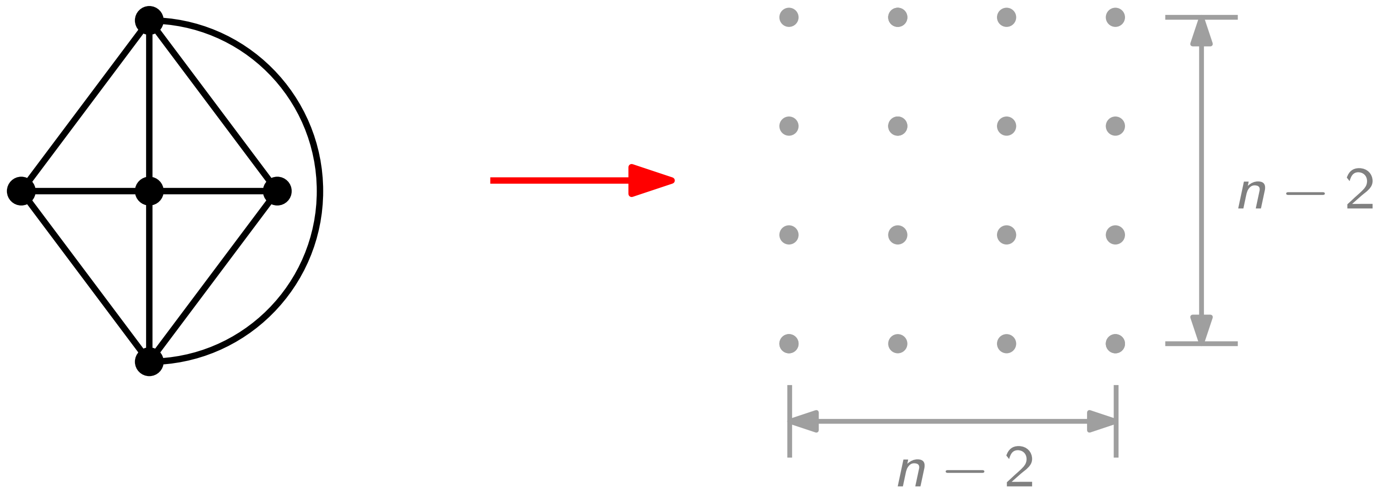
[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings

[Schnyder 1990: *Embedding planar graphs on the grid*]
Any planar graph with n nodes can be drawn on the $(n-2) \times (n-2)$ grid in $O(n)$ time.

Planar Graphs

[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings

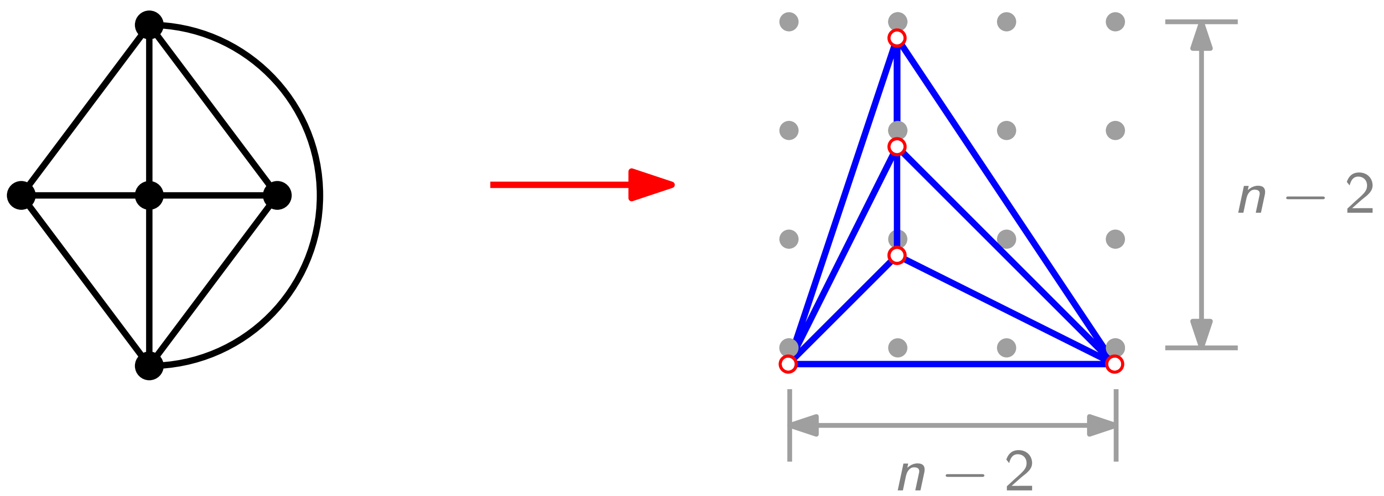
[Schnyder 1990: *Embedding planar graphs on the grid*]
Any planar graph with n nodes can be drawn on the $(n-2) \times (n-2)$ grid in $O(n)$ time.



Planar Graphs

[Def] graph G *planar* $\Leftrightarrow$ G can be drawn without crossings

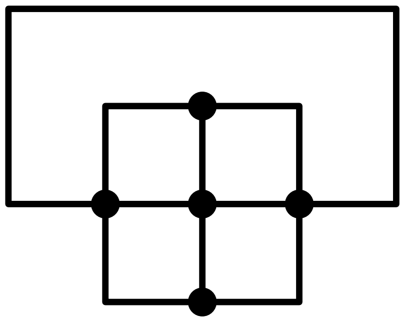
[Schnyder 1990: *Embedding planar graphs on the grid*]
Any planar graph with n nodes can be drawn on the $(n-2) \times (n-2)$ grid in $O(n)$ time.



- nodes on grid points
- compact drawing

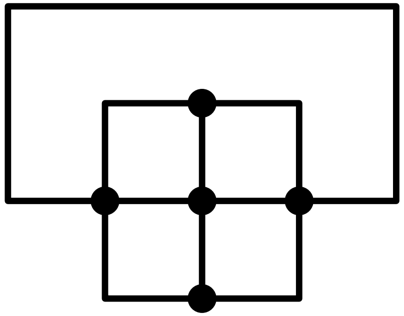
Orthogonal Layouts

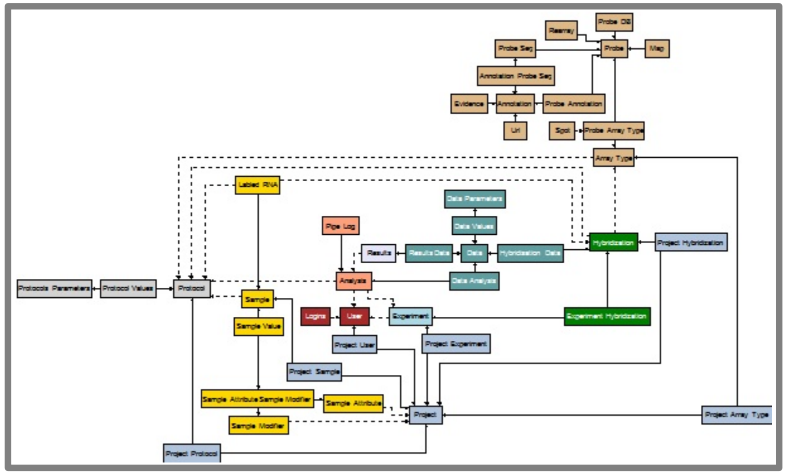
- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications



Orthogonal Layouts

- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications

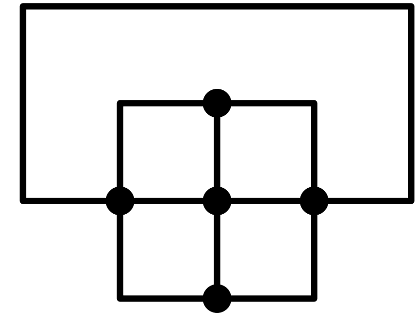


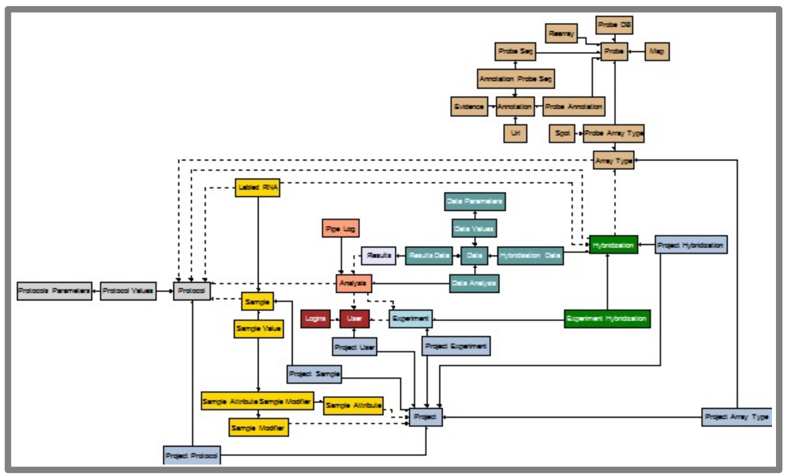


ER diagram in OGDF

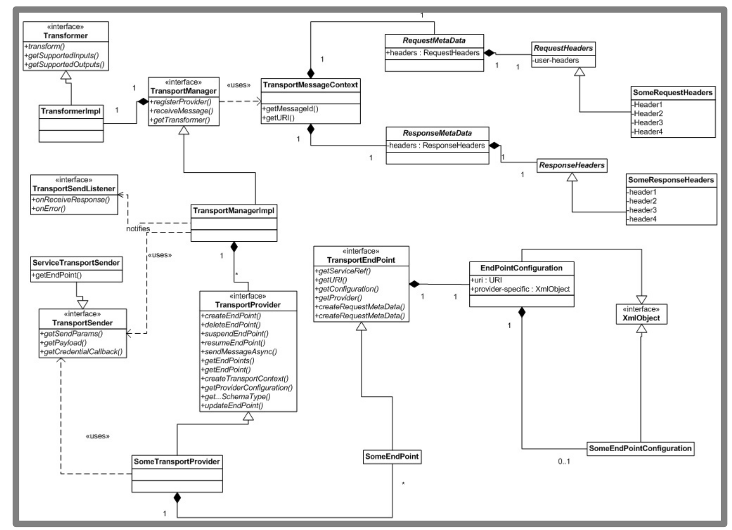
Orthogonal Layouts

- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications





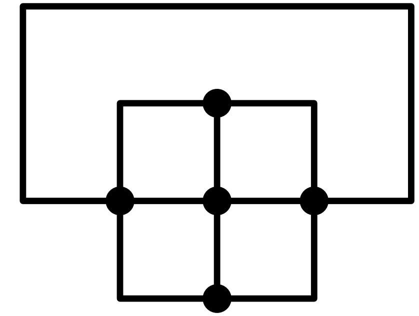
ER diagram in OGDF

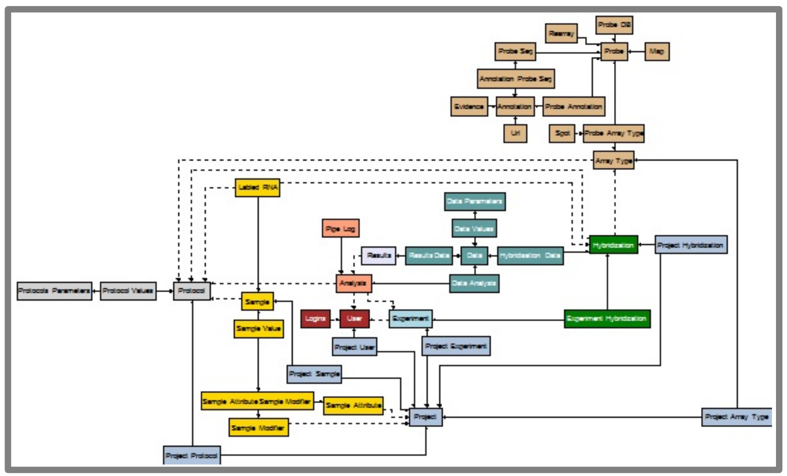


UML diagram by Oracle

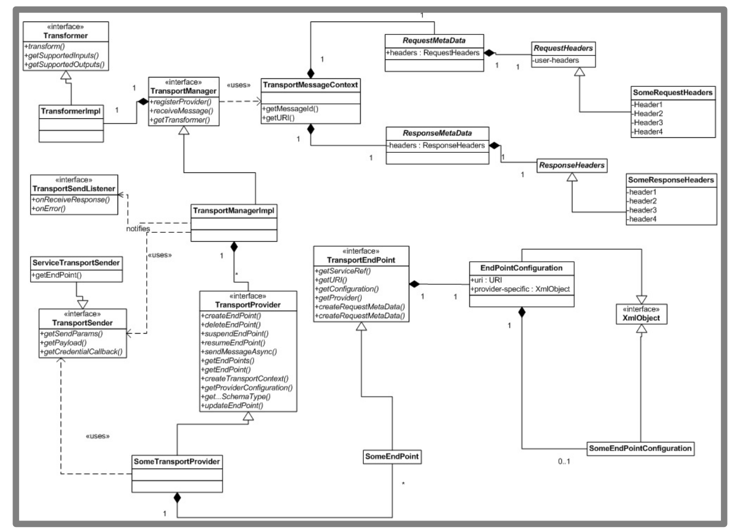
Orthogonal Layouts

- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications

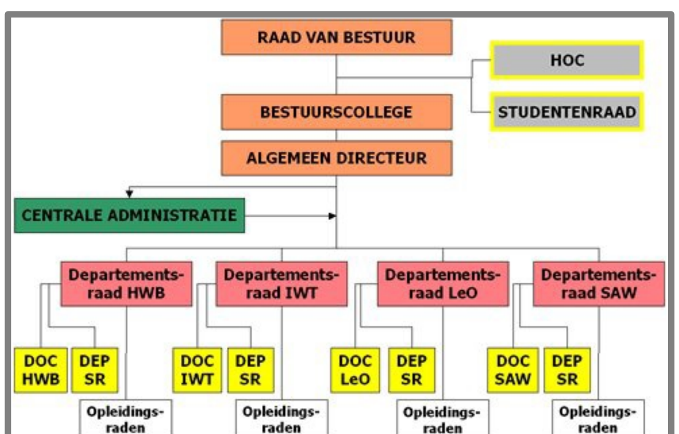




ER diagram in OGDF



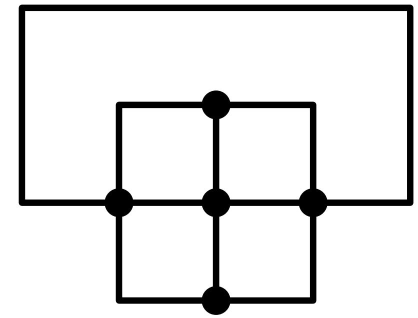
UML diagram by Oracle

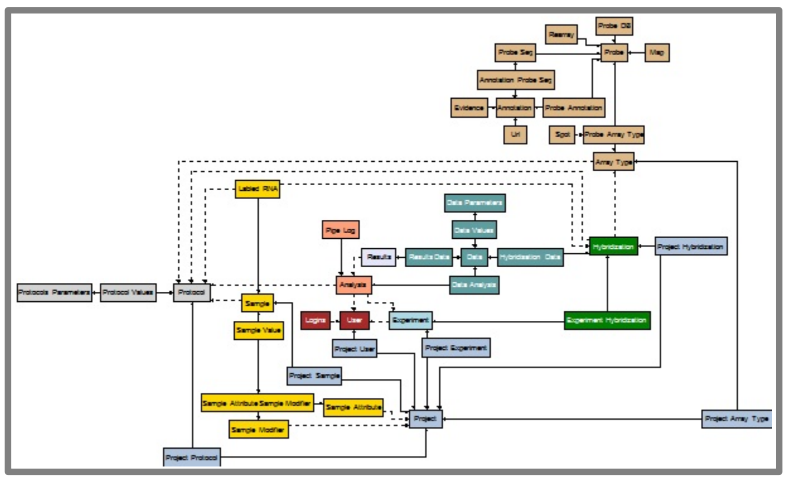


Organigram of HS Limburg

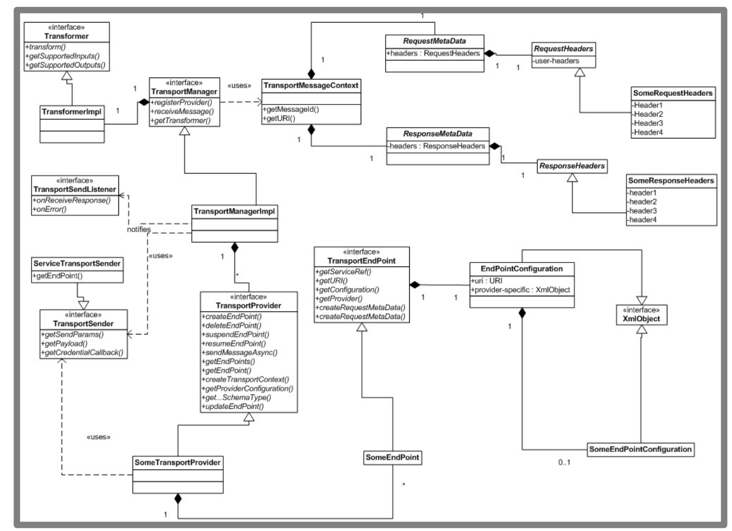
Orthogonal Layouts

- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications

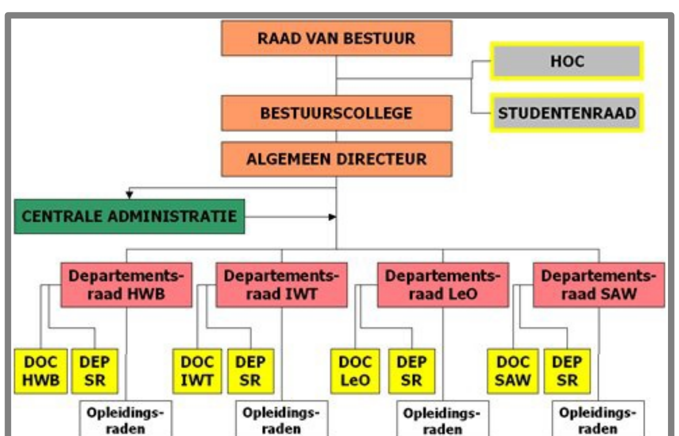




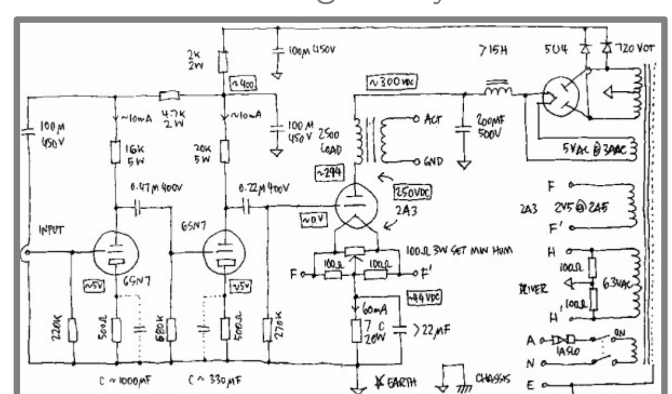
ER diagram in OGDF



UML diagram by Oracle



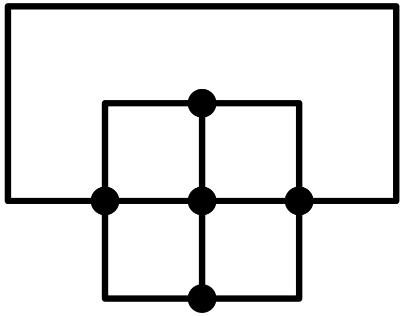
Organigram of HS Limburg



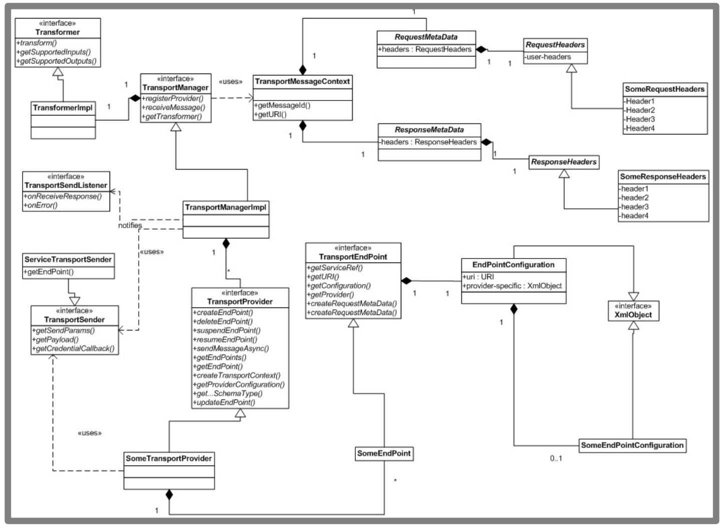
Circuit diagram by Jeff Atwood

Orthogonal Layouts

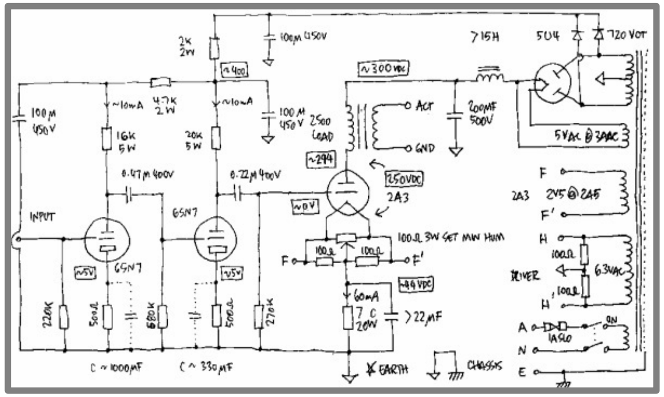
- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications



Fused Grid city layouts



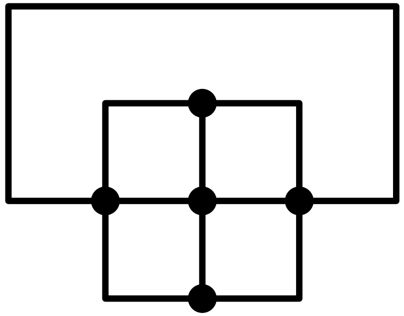
UML diagram by Oracle



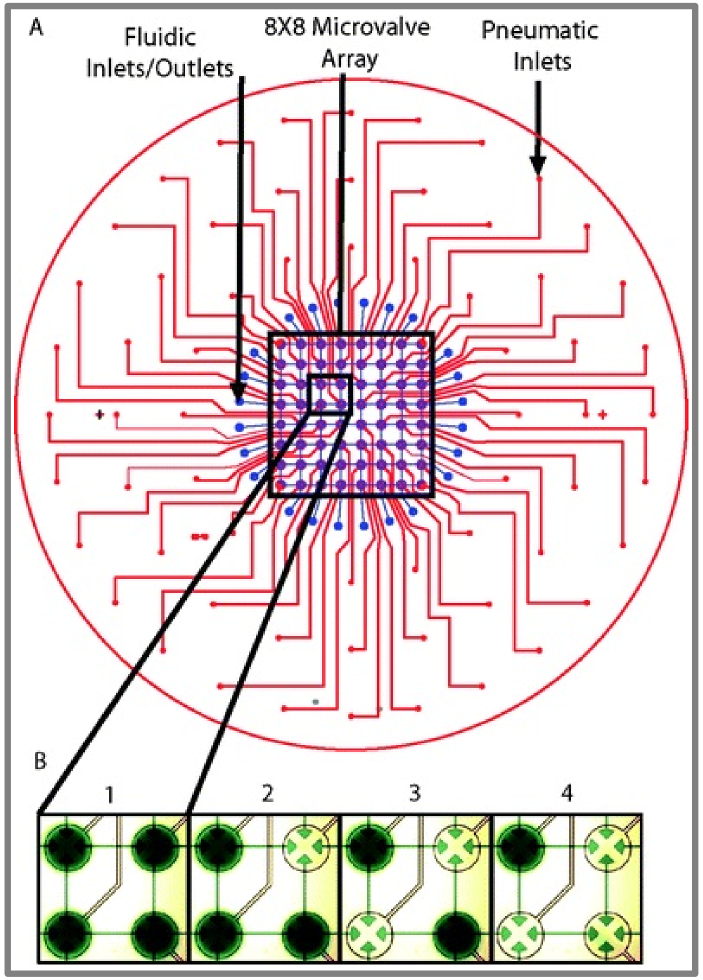
Circuit diagram by Jeff Atwood

Orthogonal Layouts

- all edge segments are horizontal or vertical
- a well-studied drawing convention
- many examples in applications



Fused Grid city layouts



VLSI/PCI chip design

Orthogonal Layouts – Well-Known Results

[Tamassia, SIAM J Comp'87]
Can minimize number of bends for fixed embedding.

Orthogonal Layouts – Well-Known Results

[Tamassia, SIAM J Comp'87]
Can minimize number of bends for fixed embedding.

[Garg & Tamassia, SIAM J Comp'01]
Without fixed embedding, bend minimization is hard to approx.

Orthogonal Layouts – Well-Known Results

[Tamassia, SIAM J Comp'87]
Can minimize number of bends for fixed embedding.

[Garg & Tamassia, SIAM J Comp'01]
Without fixed embedding, bend minimization is hard to approx.

[Biedl & Kant, CGTA'98], [Liu et al., DAM'98]
Can compute drawing on the $(n \times n)$-grid with $\leq 2n + 2$ bends for any embedding (and ≤ 2 bends/edge – except octahedron)

Orthogonal Layouts – Well-Known Results

[Tamassia, SIAM J Comp'87]
Can minimize number of bends for fixed embedding.

[Garg & Tamassia, SIAM J Comp'01]
Without fixed embedding, bend minimization is hard to approx.

[Biedl & Kant, CGTA'98], [Liu et al., DAM'98]
Can compute drawing on the $(n \times n)$-grid with $\leq 2n + 2$ bends for any embedding (and ≤ 2 bends/edge – except octahedron)

[Bläsius et al., '11]
Given an embedding and a function flex: $E \to \mathbb{N}_{\geq 1}$, can compute a drawing with $\leq$ flex(e) bends/edge (if one exists).

Orthogonal Layouts – Well-Known Results

[Tamassia, SIAM J Comp'87]
Can minimize number of bends for fixed embedding.

[Garg & Tamassia, SIAM J Comp'01]
Without fixed embedding, bend minimization is hard to approx.

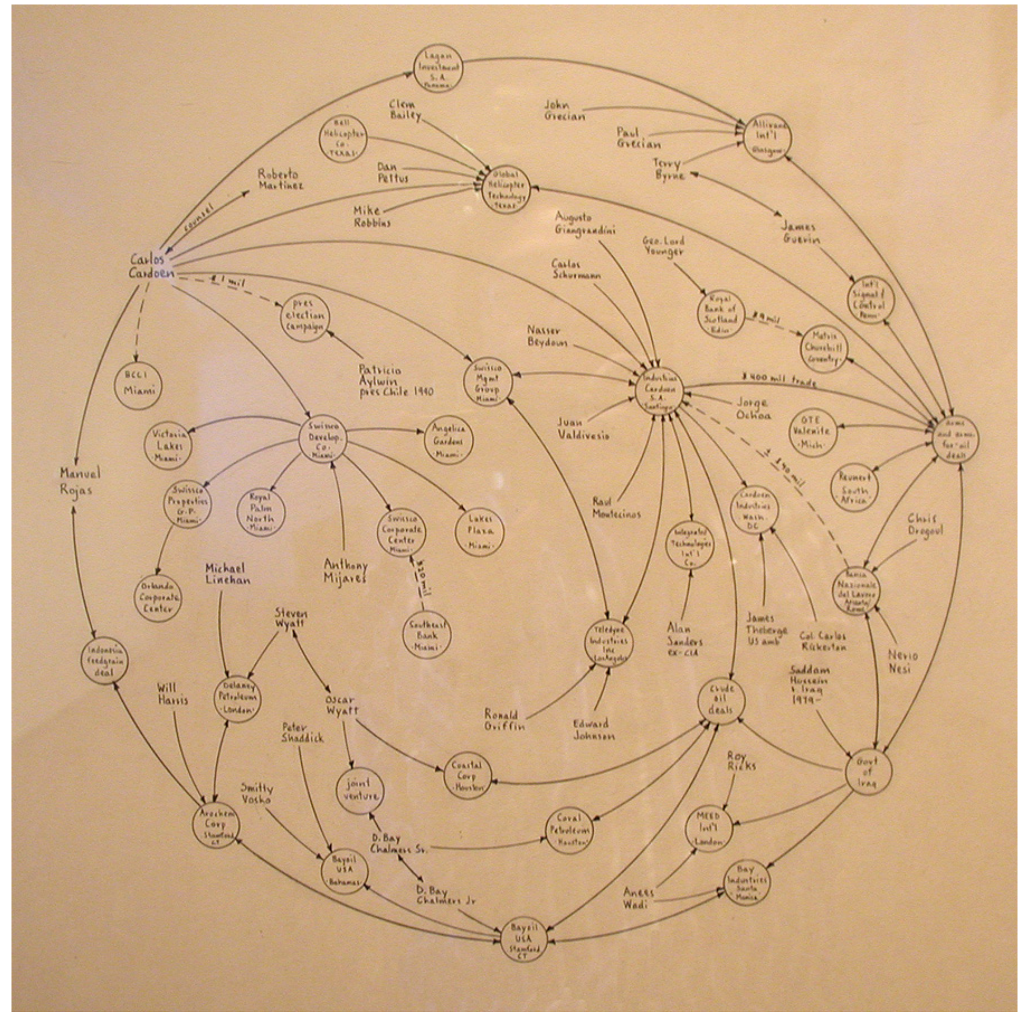
[Biedl & Kant, CGTA'98], [Liu et al., DAM'98]
Can compute drawing on the $(n \times n)$-grid with $\leq 2n + 2$ bends for any embedding (and ≤ 2 bends/edge – except octahedron)

[Bläsius et al., '11]
Given an embedding and a function flex: $E \to \mathbb{N}_{\geq 1}$, can compute a drawing with $\leq$ flex(e) bends/edge (if one exists).

Smooth Drawings

Lombardi drawings

- circular arc edges
- perfect angular resolution



Mark Lombardi
(1951–2000)

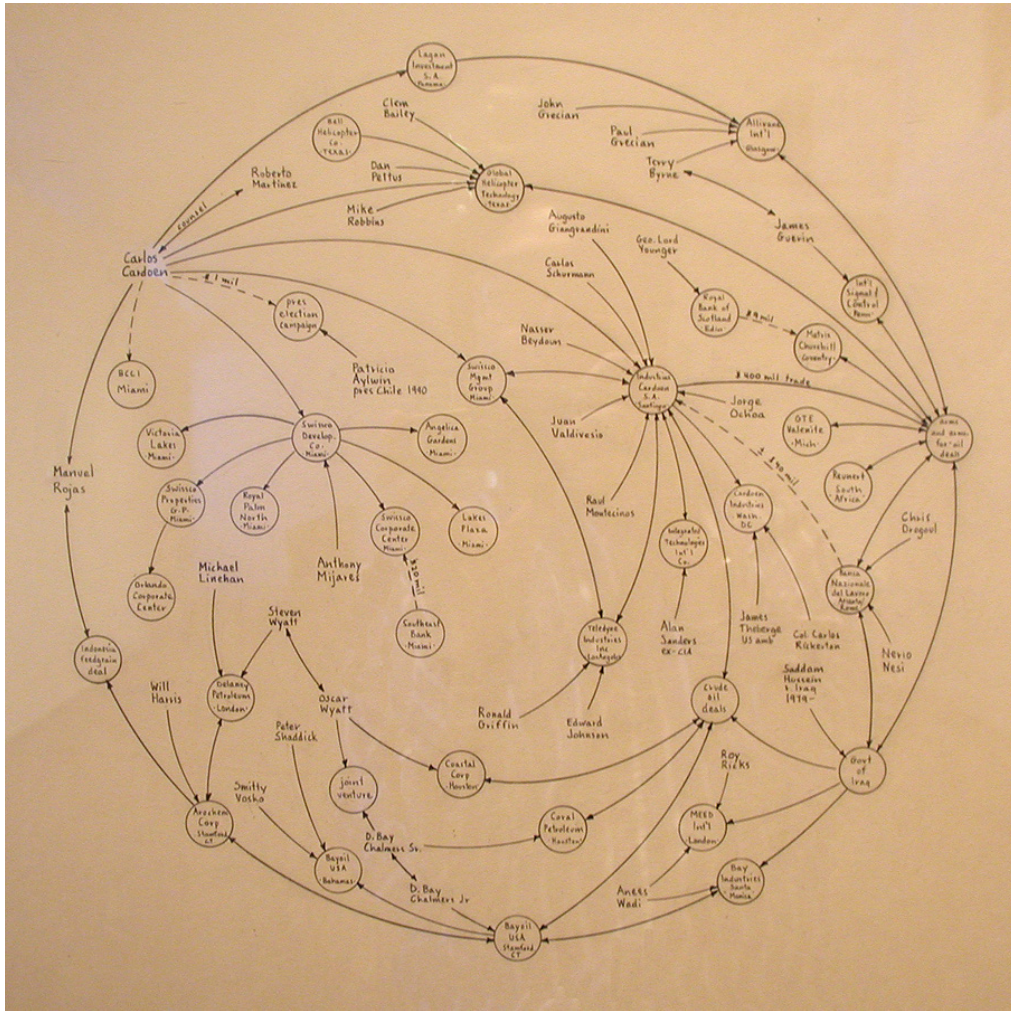
Smooth Drawings

Lombardi drawings

- circular arc edges
- perfect angular resolution

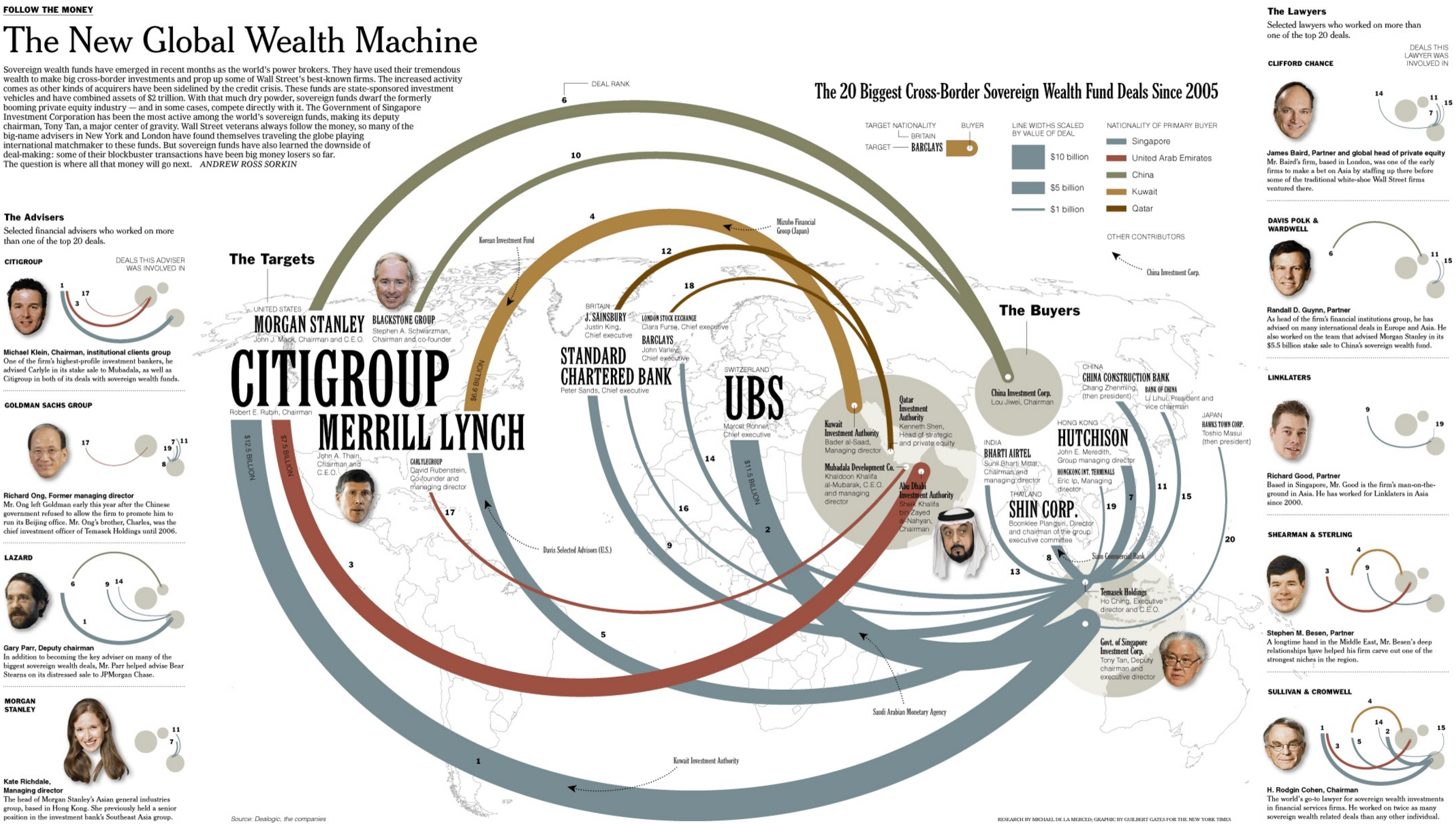
k-Lombardi drawings

- each edge sequence of k circular arcs



Mark Lombardi
(1951–2000)

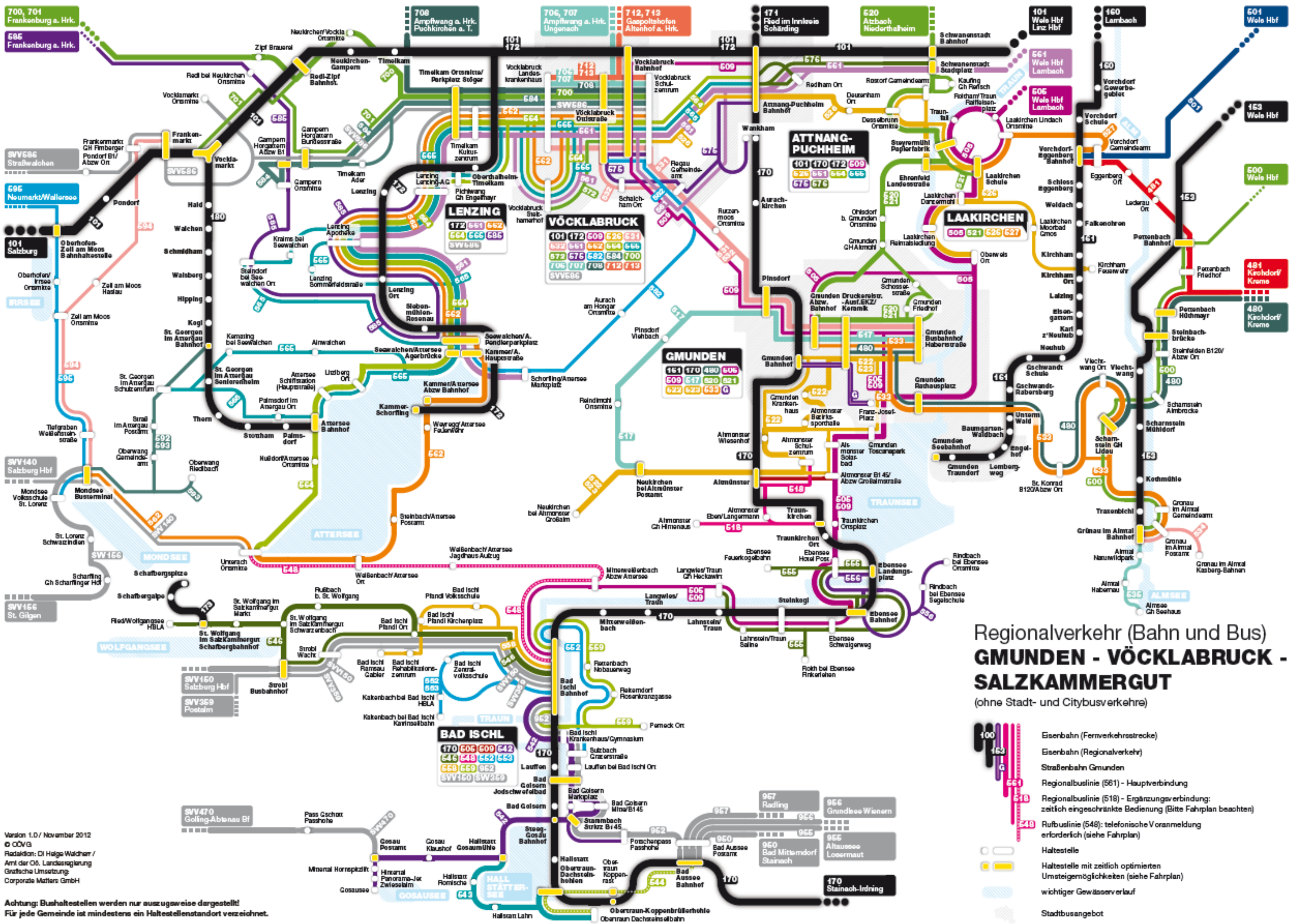
Smooth Drawings



Smooth Drawings

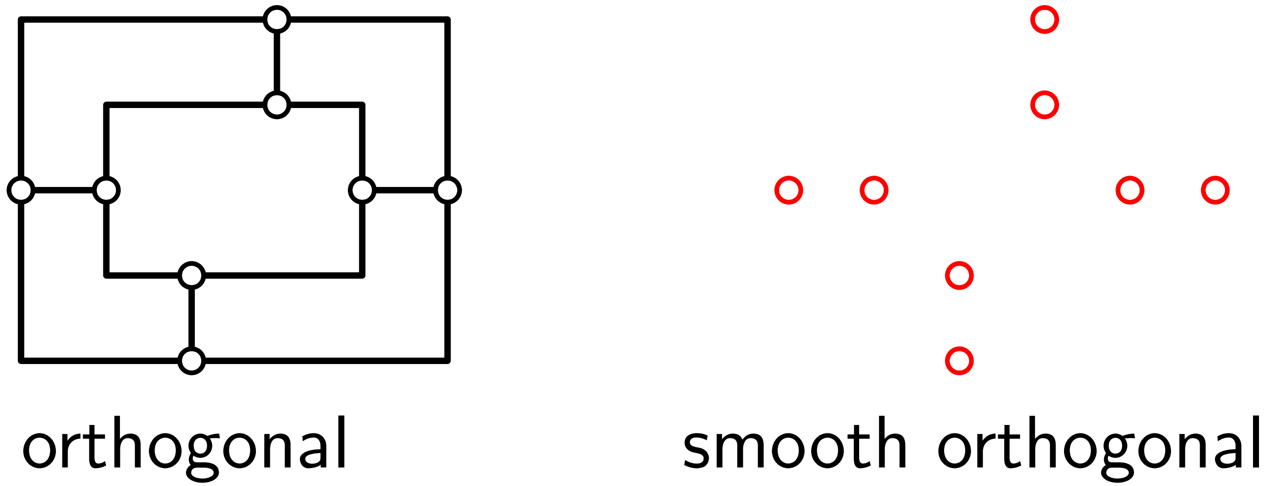
City model; plan

Smooth Drawings



Smooth Orthogonal Layouts

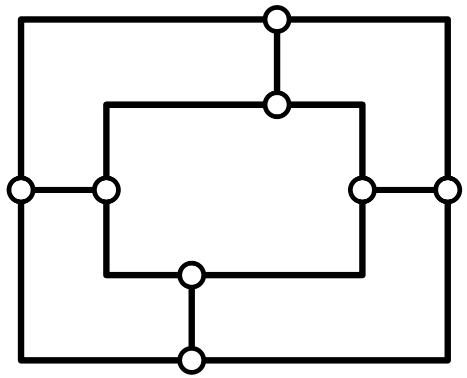
Combine both worlds:



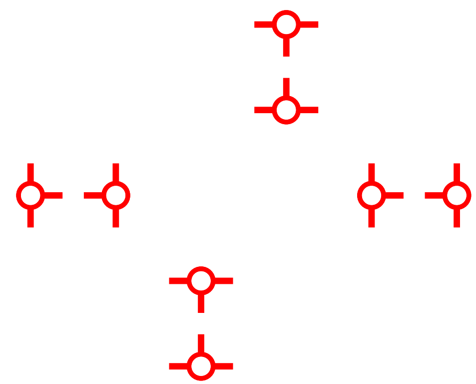
Smooth Orthogonal Layouts

Combine both worlds:

- edges leave and enter vertices horizontally or vertically



orthogonal

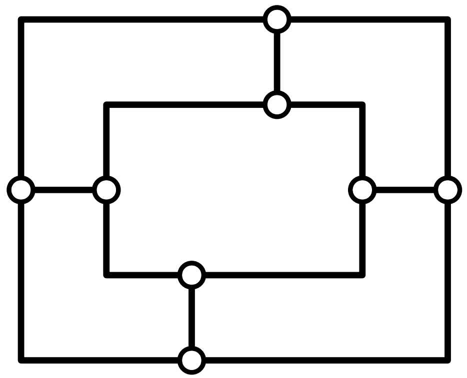


smooth orthogonal

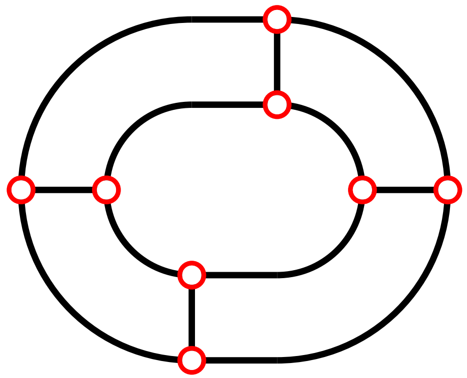
Smooth Orthogonal Layouts

Combine both worlds:

- edges leave and enter vertices horizontally or vertically
- each edge is drawn as a sequence of axis-aligned line segments and circular-arc segments without bends



orthogonal

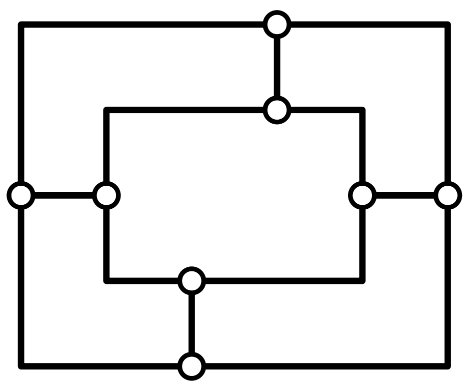


smooth orthogonal

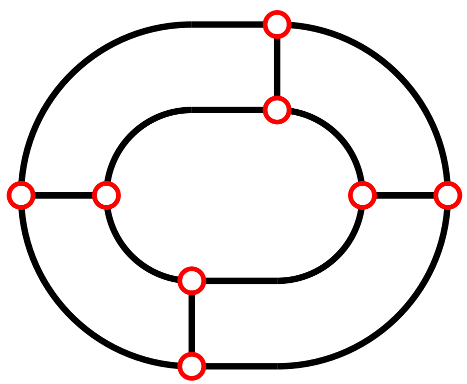
Smooth Orthogonal Layouts

Combine both worlds:

- edges leave and enter vertices horizontally or vertically
- each edge is drawn as a sequence of axis-aligned line segments and circular-arc segments without bends
- there are no edge-crossings (for planar graphs)

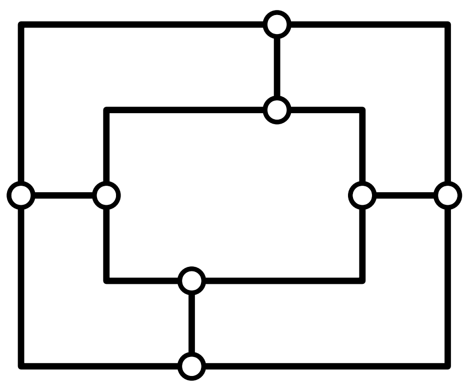


orthogonal

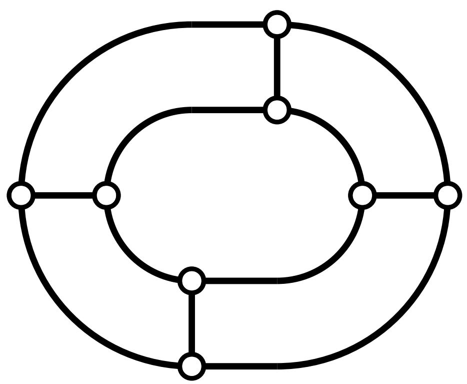


smooth orthogonal

Edge Complexity



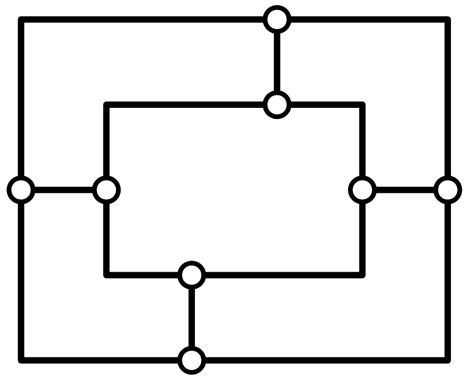
orthogonal



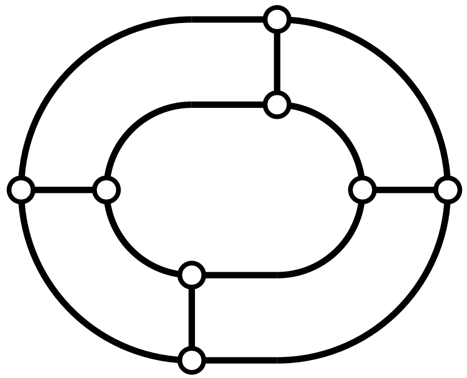
smooth orthogonal

Edge Complexity

complexity of an edge: number of arcs



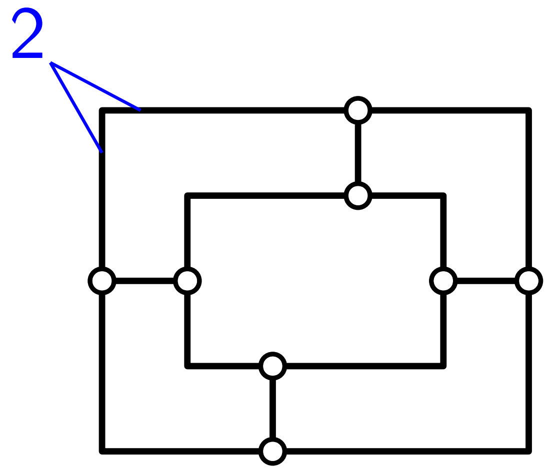
orthogonal



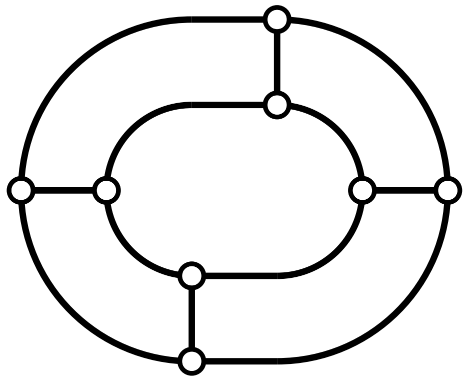
smooth orthogonal

Edge Complexity

complexity of an edge: number of arcs



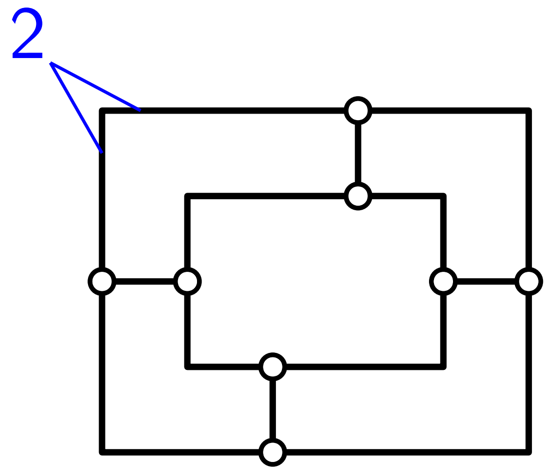
orthogonal



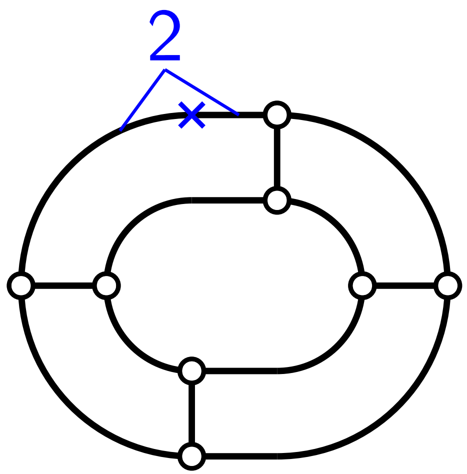
smooth orthogonal

Edge Complexity

complexity of an edge: number of arcs



orthogonal

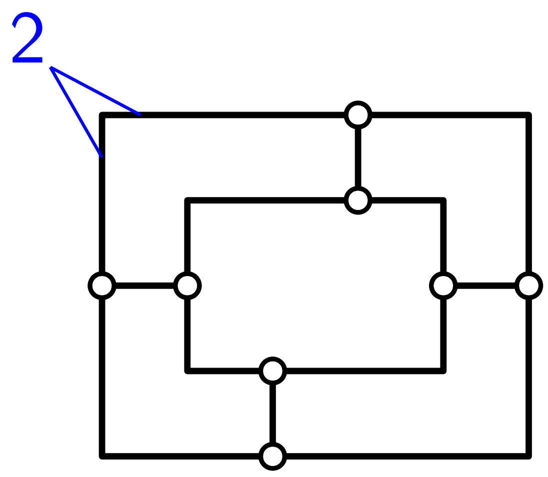


smooth orthogonal

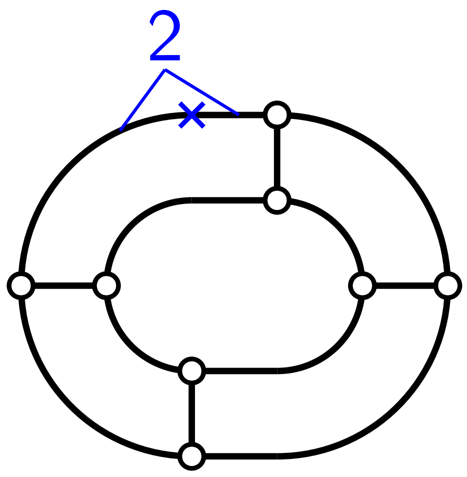
Edge Complexity

complexity of an edge: number of arcs

edge complexity: maximum complexity over all edges



orthogonal

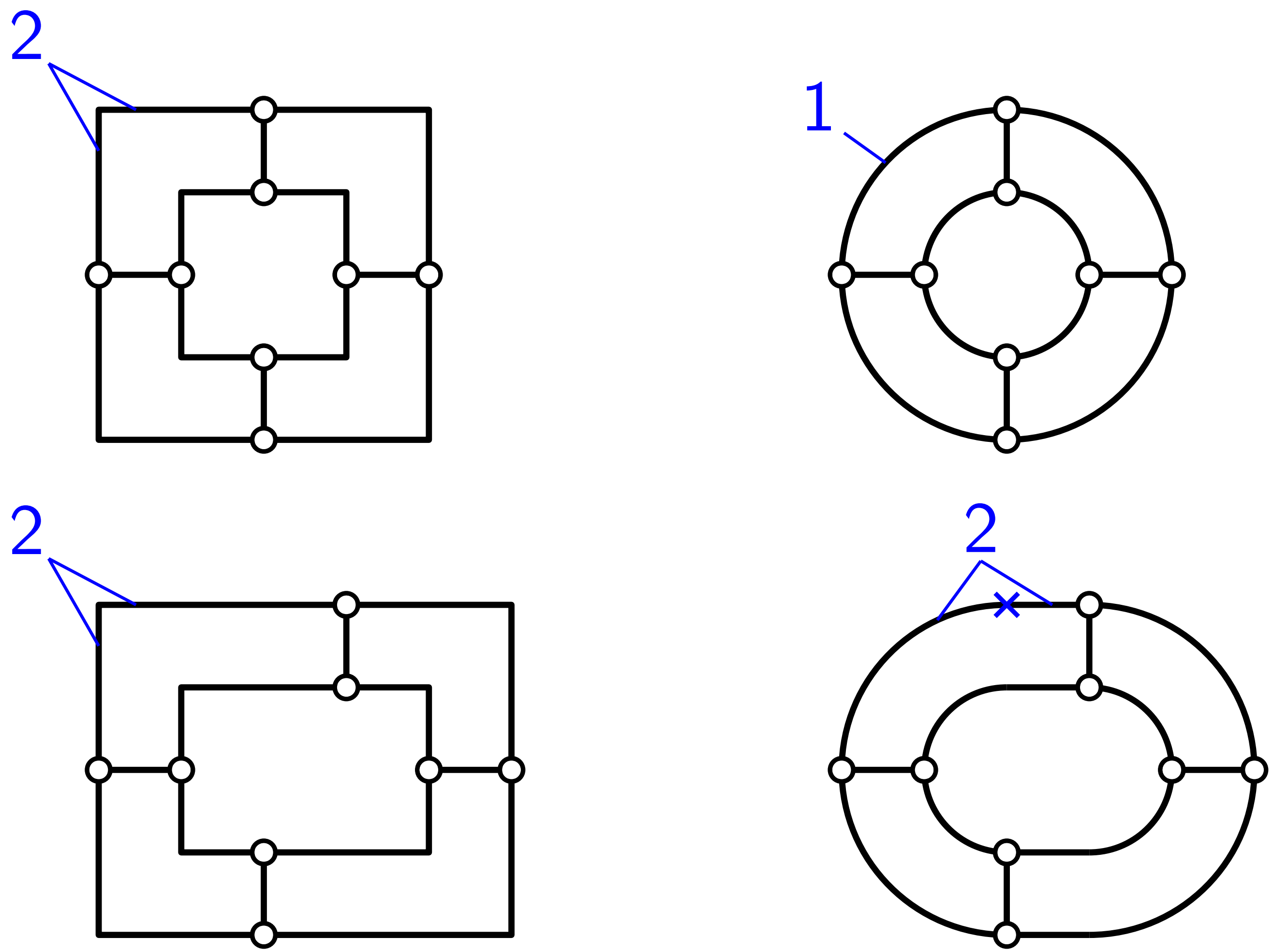


smooth orthogonal

Edge Complexity

complexity of an edge: number of arcs

edge complexity: maximum complexity over all edges



orthogonal smooth orthogonal

Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t

Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering

Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.

Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

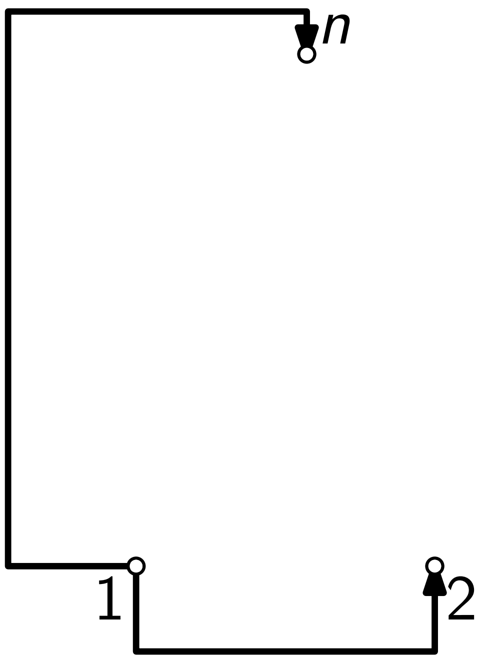
out:	$\uparrow$	$\rightarrow$	$\leftarrow$
in:	$\downarrow$	$\leftarrow$	$\rightarrow$

Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

out:	$\uparrow$	$\rightarrow$	$\leftarrow$
in:	$\downarrow$	$\leftarrow$	$\rightarrow$

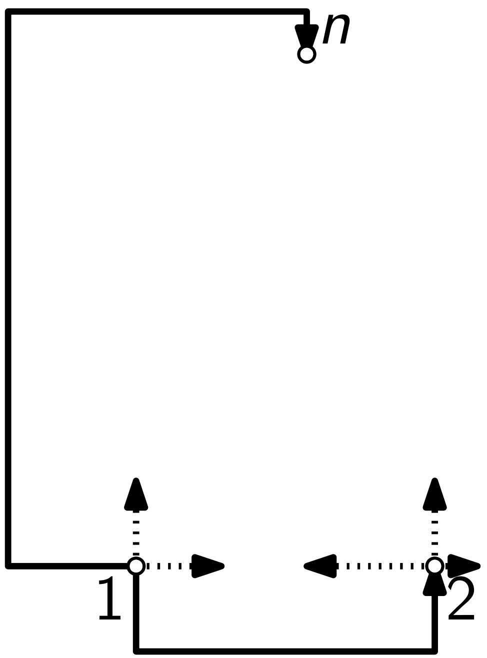


Liu et al. Algorithm

biconnected 4-planar graph → orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

out:	↑	→	←
in:	↓	←	→

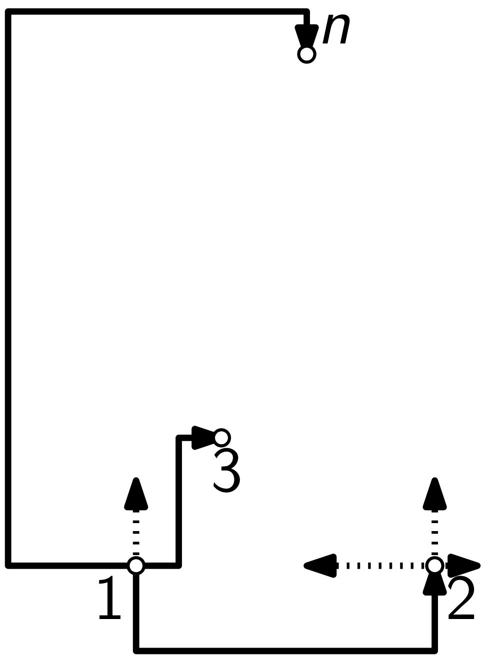


Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

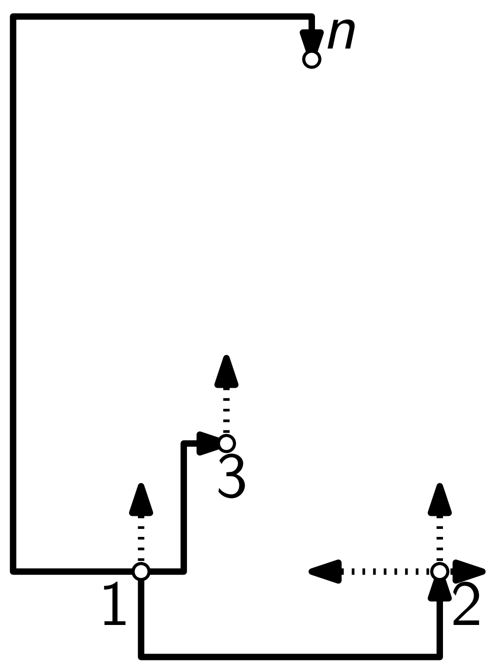
out:	$\uparrow$	$\rightarrow$	$\leftarrow$
in:	$\downarrow$	$\leftarrow$	$\rightarrow$



Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

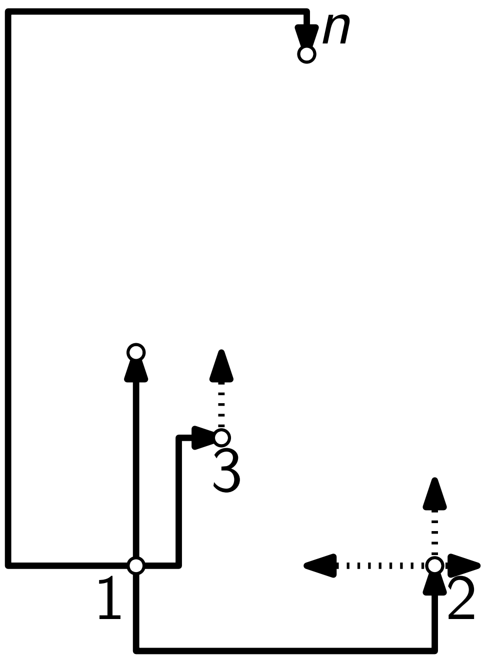
- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:
  out: $\uparrow$ $\rightarrow$ $\leftarrow$
  in: $\downarrow$ $\leftarrow$ $\rightarrow$



Liu et al. Algorithm

biconnected 4-planar graph → orthogonal complexity-3 layout

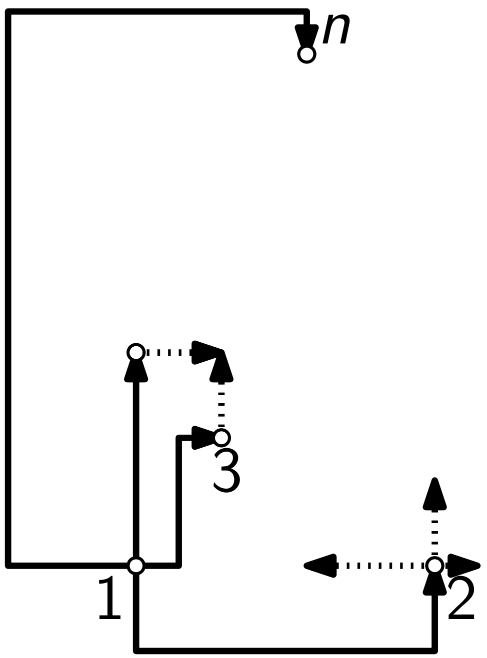
- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:
  out: $\uparrow$ $\rightarrow$ $\leftarrow$
  in: $\downarrow$ $\leftarrow$ $\rightarrow$



Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:
  out: $\uparrow$ $\rightarrow$ $\leftarrow$
  in: $\downarrow$ $\leftarrow$ $\rightarrow$

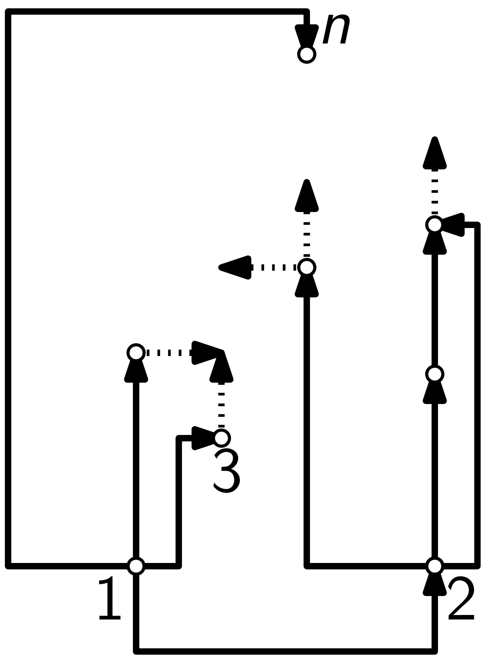


Liu et al. Algorithm

biconnected 4-planar graph → orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

out:	$\uparrow$	$\rightarrow$	$\leftarrow$
in:	$\downarrow$	$\leftarrow$	$\rightarrow$

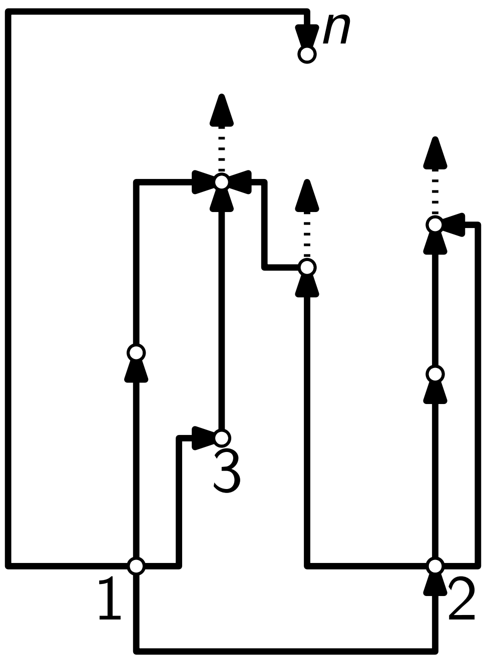


Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

out:	$\uparrow$	$\rightarrow$	$\leftarrow$
in:	$\downarrow$	$\leftarrow$	$\rightarrow$

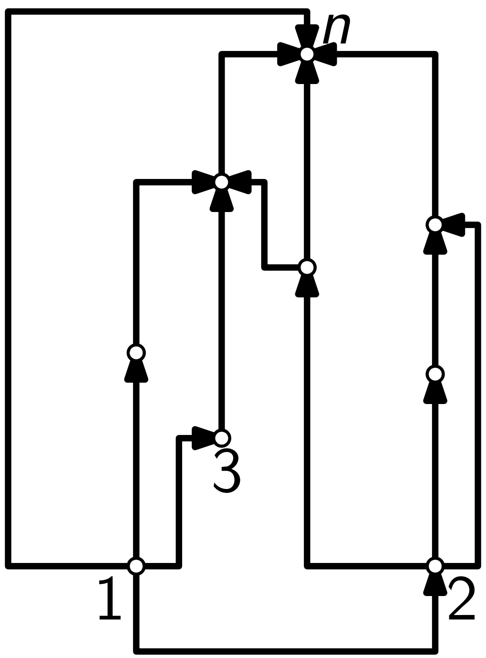


Liu et al. Algorithm

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

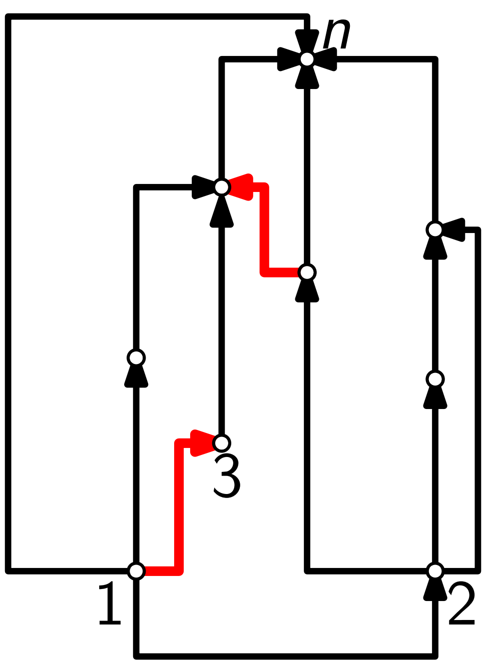
out:	$\uparrow$	$\rightarrow$	$\leftarrow$
in:	$\downarrow$	$\leftarrow$	$\rightarrow$



Liu et al. Algorithm

biconnected 4-planar graph → orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:
  out: $\uparrow \quad \rightarrow \quad \leftarrow$
  in: $\downarrow \quad \leftarrow \quad \rightarrow$
- Eliminate S-shapes

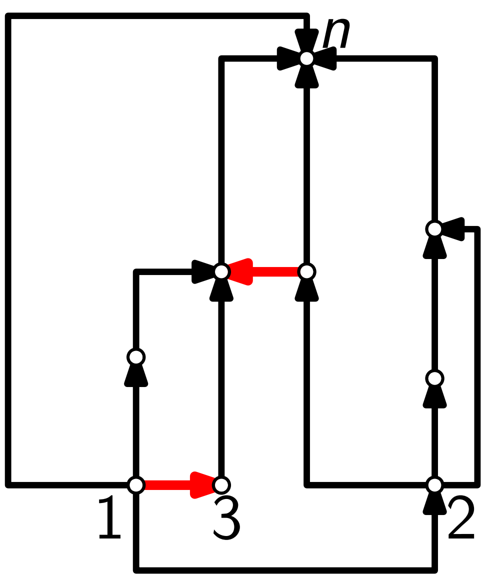


Liu et al. Algorithm

biconnected 4-planar graph → orthogonal complexity-3 layout

- choose vertices s and t
- place vertices by their st-numbering
- Invariant:
  When we fix an endpoint of edge e, we associate e with a column.
- Use ports in this order:

out:	↑	→	←
in:	↓	←	→

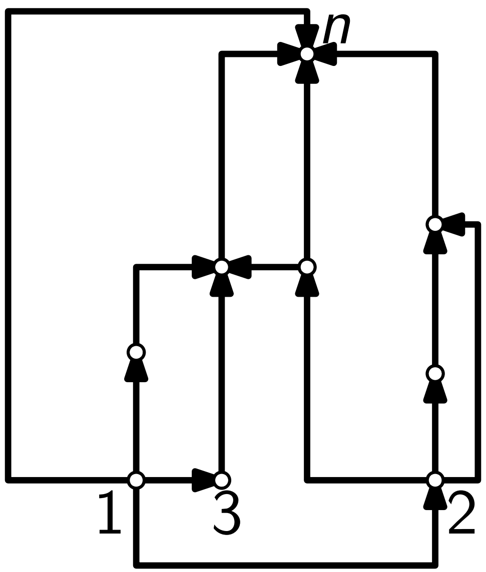


- Eliminate S-shapes

SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout

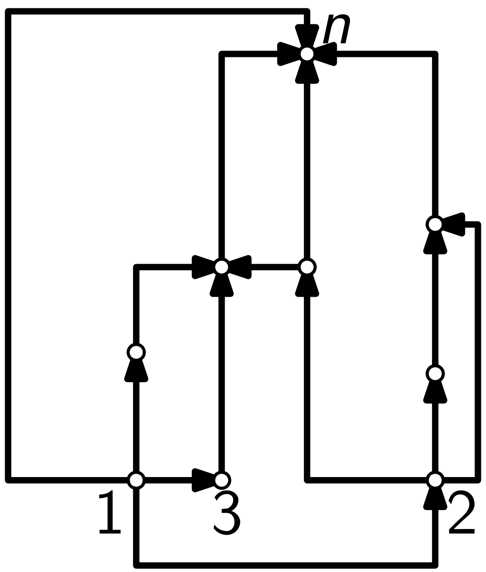
smooth



SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout
smooth

| → |

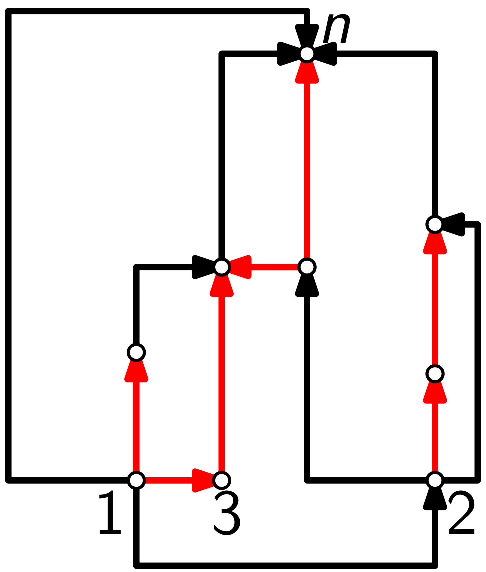


SC_2-Layouts

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-2 layout

smooth

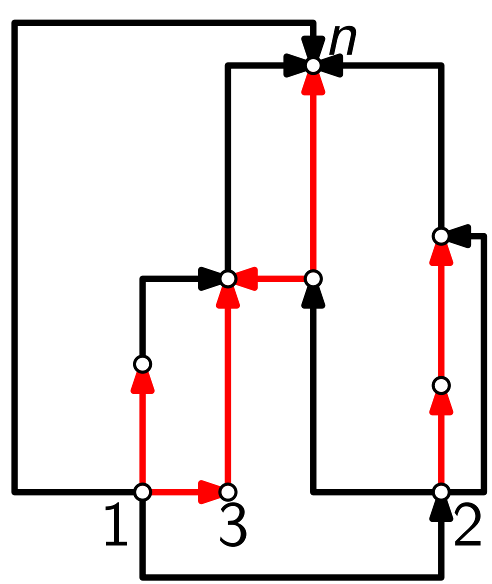
| $\rightarrow$ |



SC$_2$-Layouts

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-2 layout

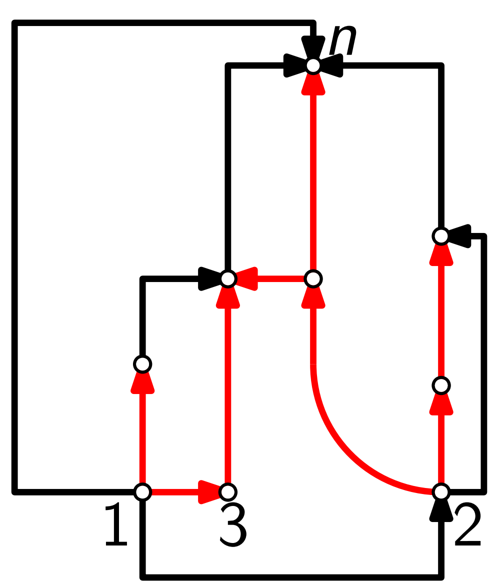
smooth



SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout

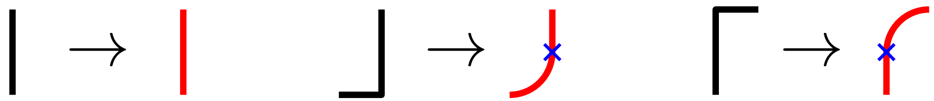
smooth

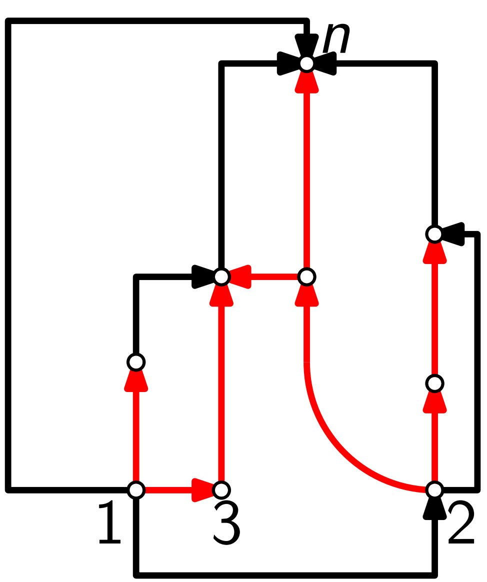


SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout

smooth

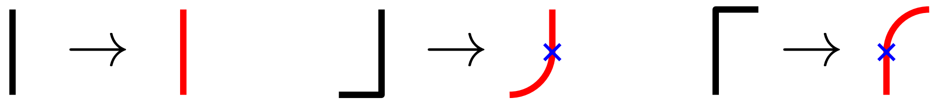


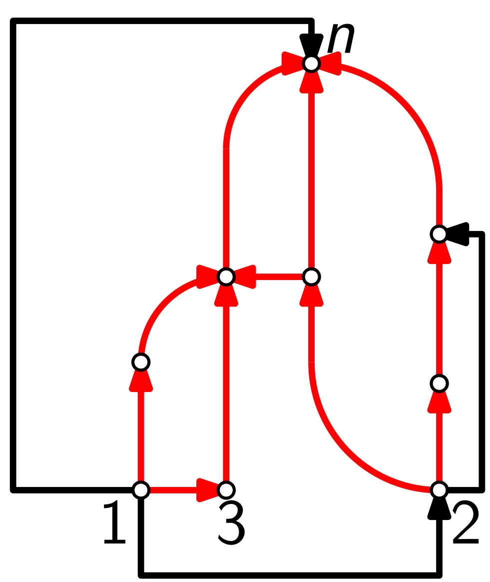


SC$_2$-Layouts

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-2 layout

smooth

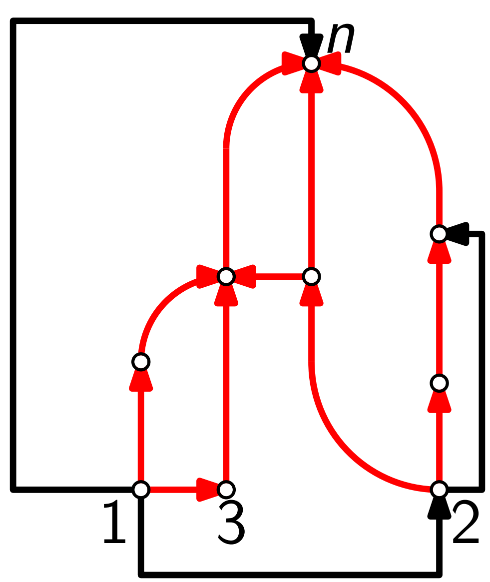




SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout

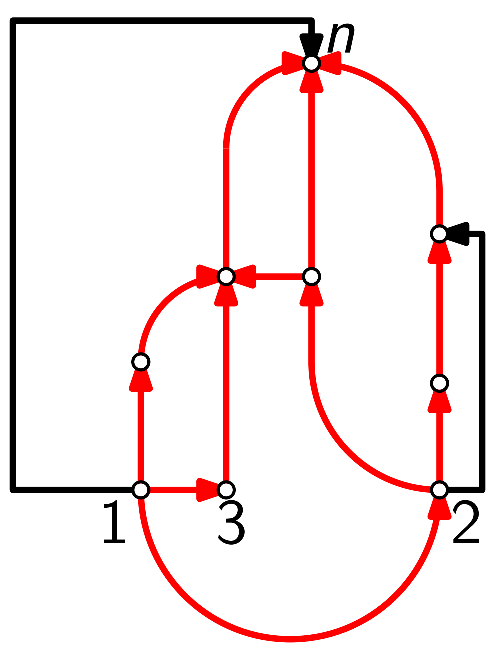
smooth



SC_2-Layouts

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-2 layout

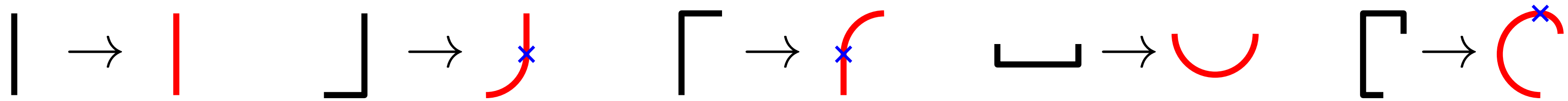
smooth

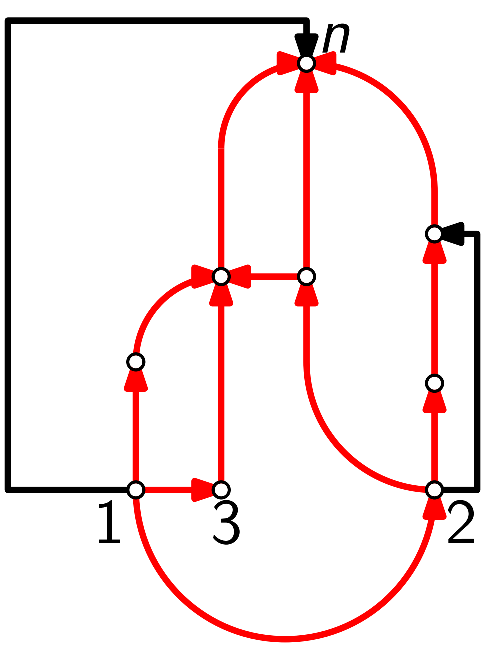


SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout

smooth

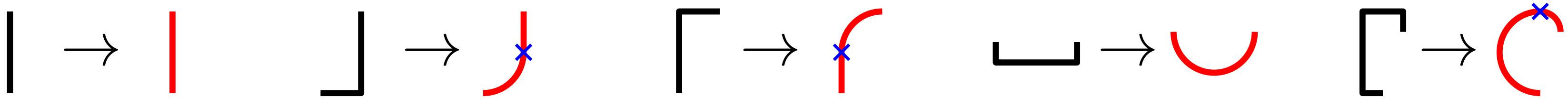


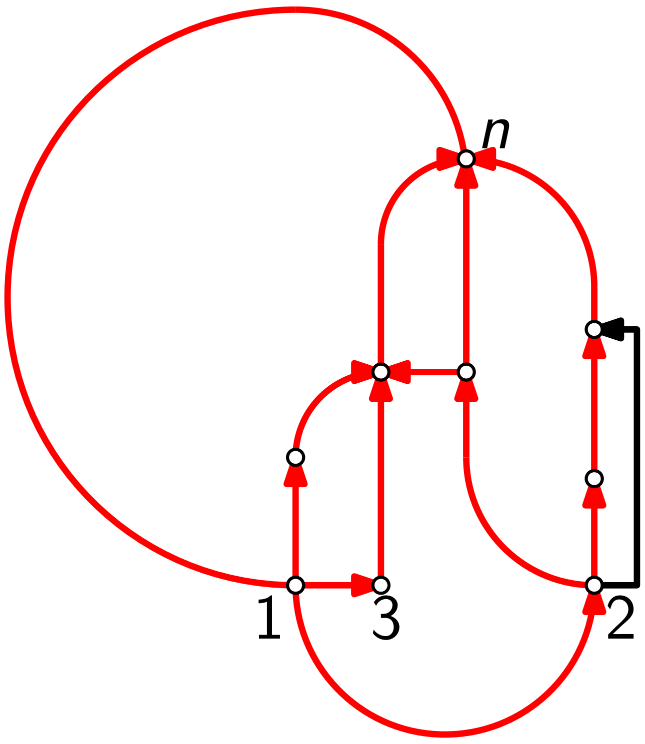


SC_2-Layouts

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-2 layout

smooth

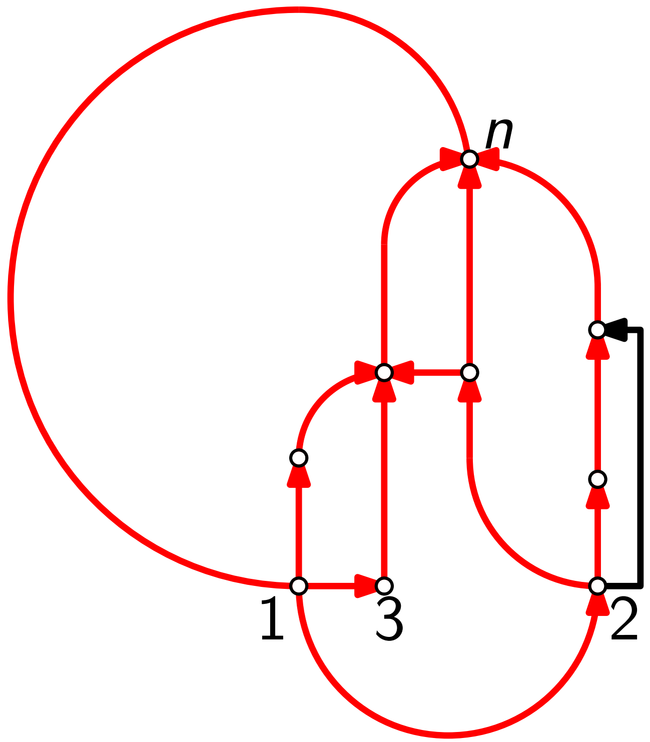




SC_2-Layouts

biconnected 4-planar graph → orthogonal complexity-2 layout

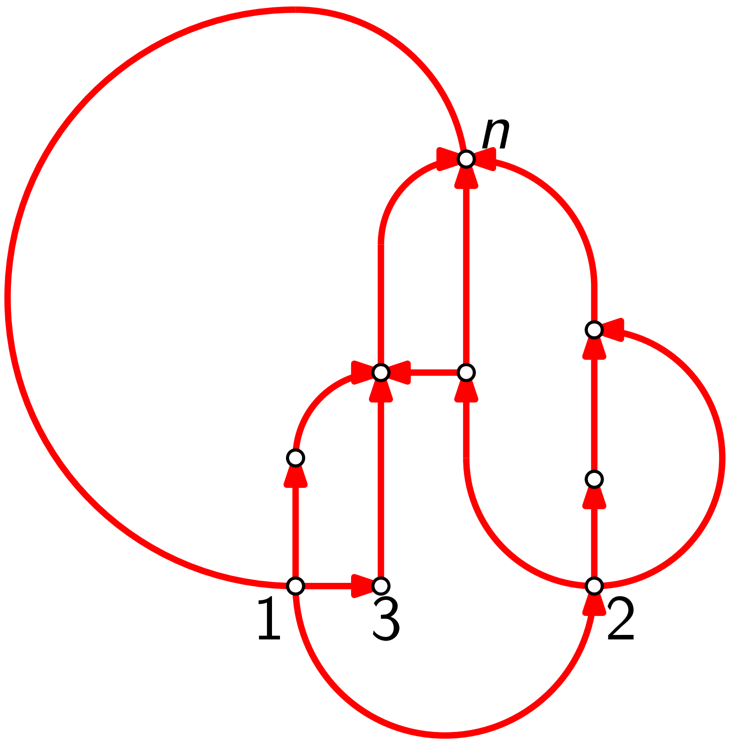
smooth



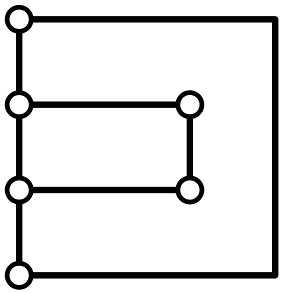
SC$_2$-Layouts

biconnected 4-planar graph $\rightarrow$ orthogonal complexity-2 layout

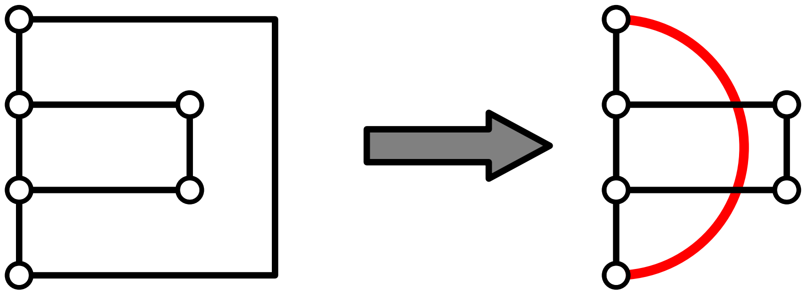
smooth



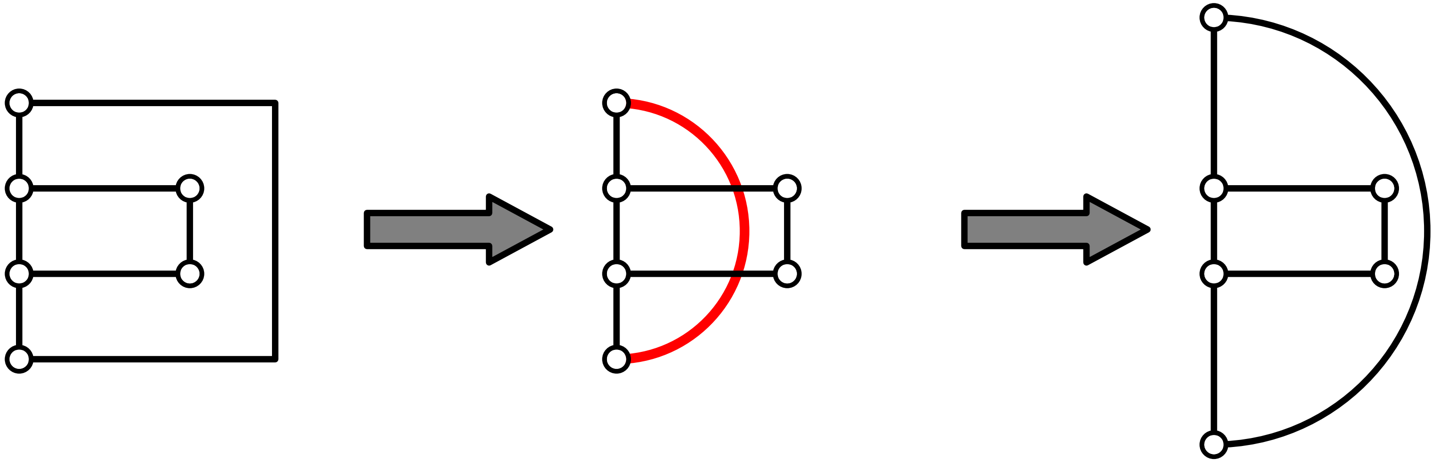
Crossings



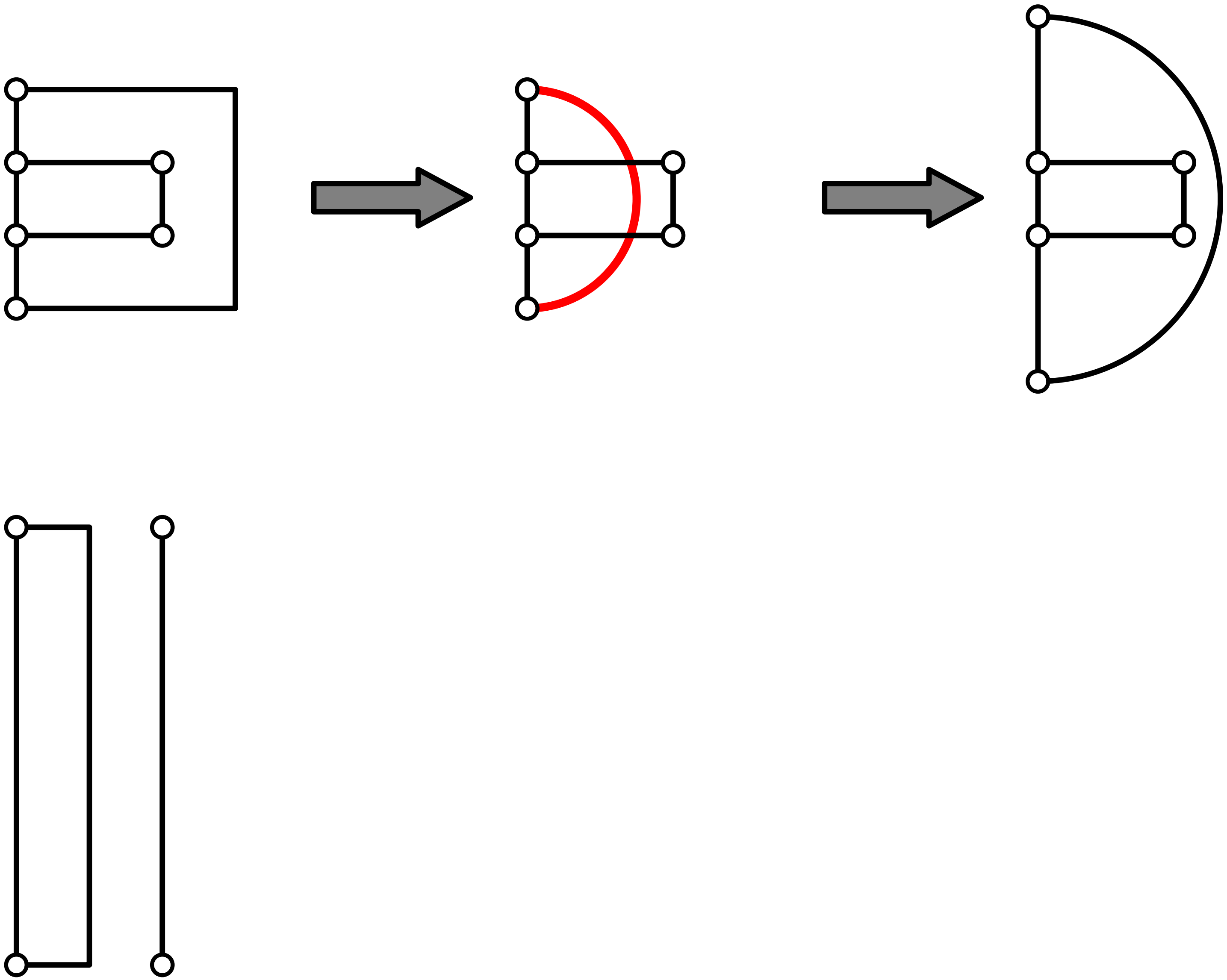
Crossings



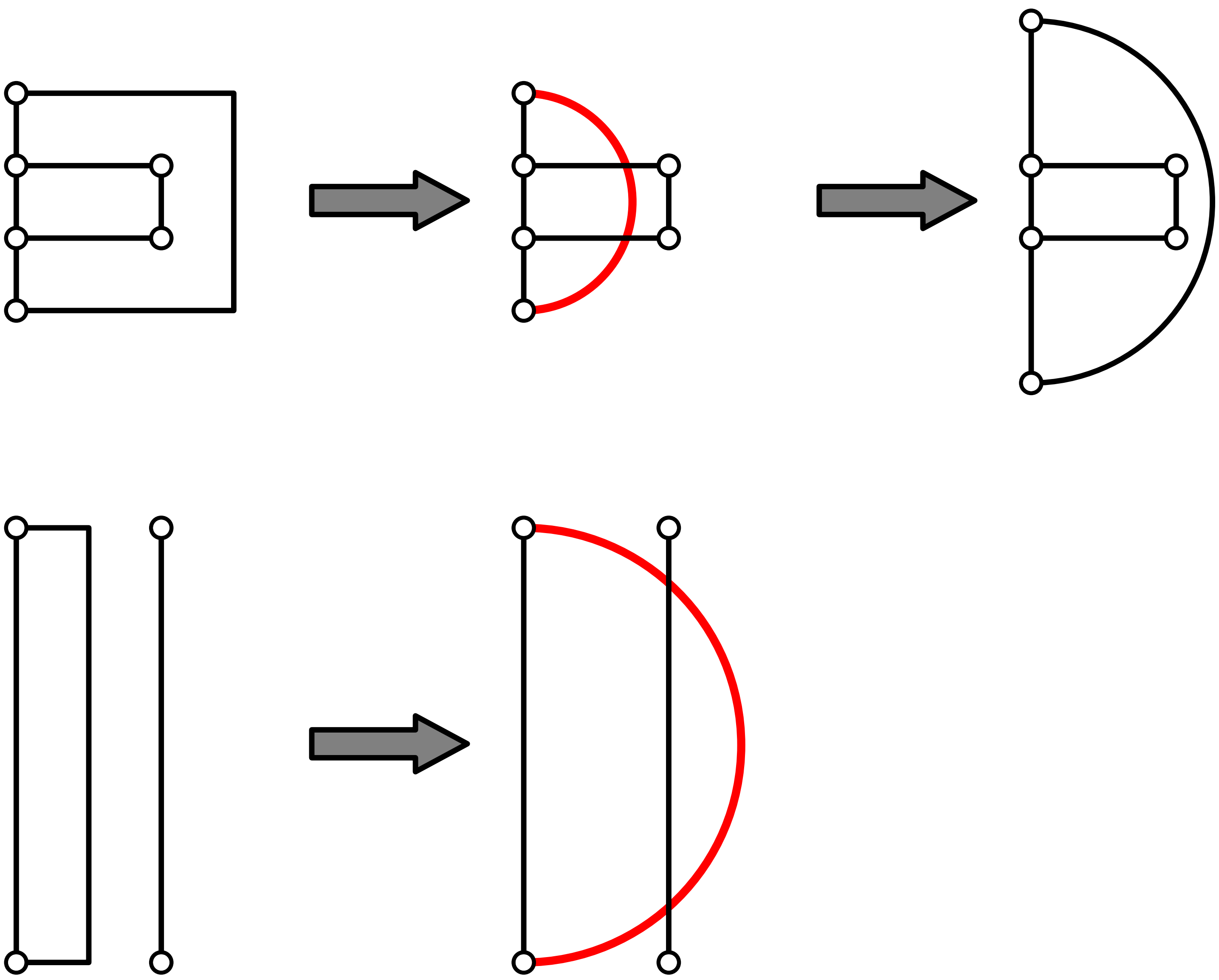
Crossings



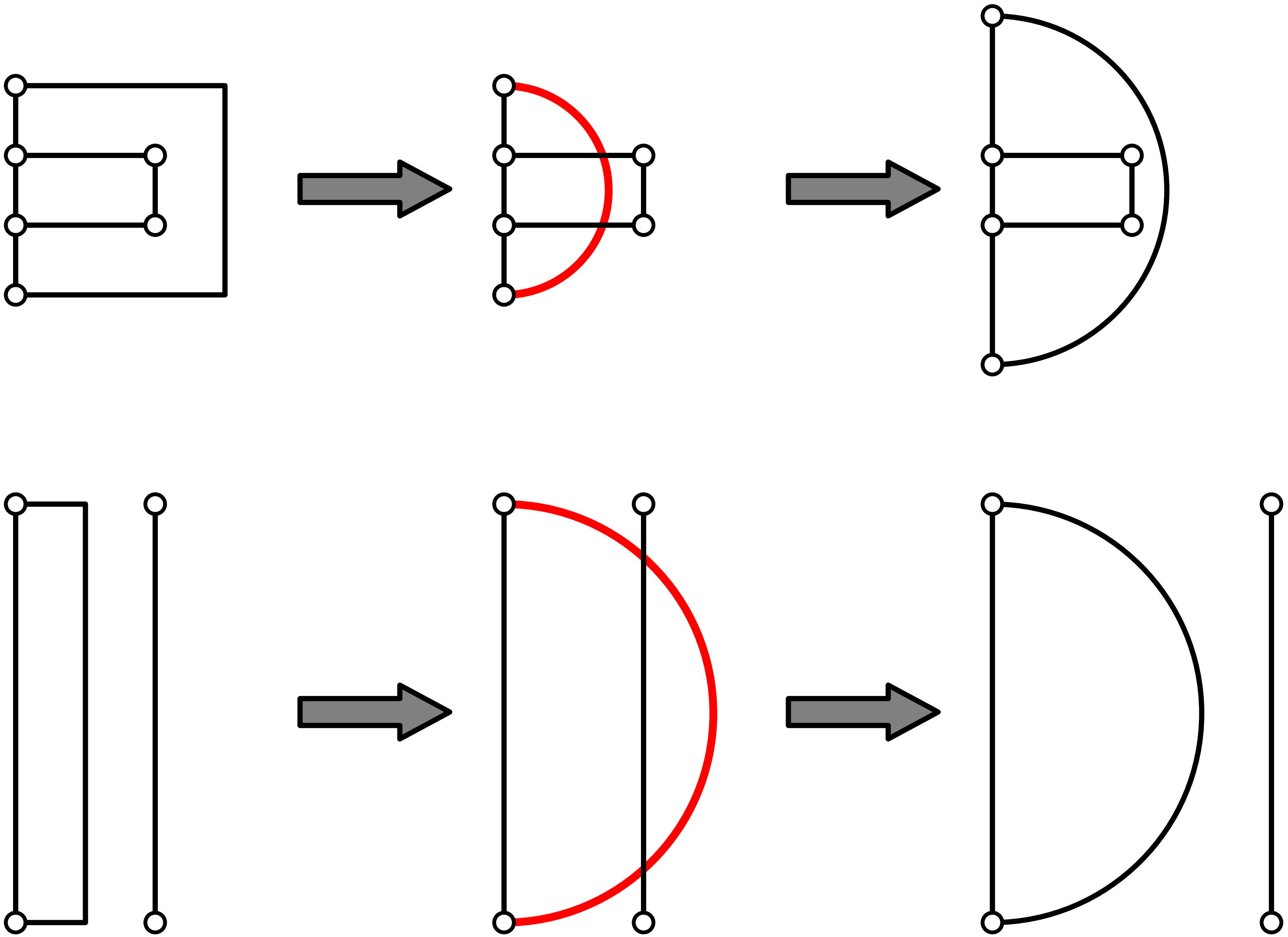
Crossings



Crossings



Crossings

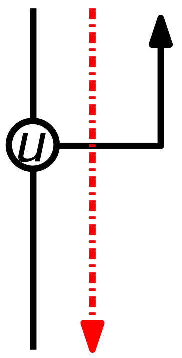


Cut

Def.

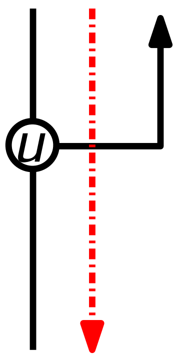
Cut

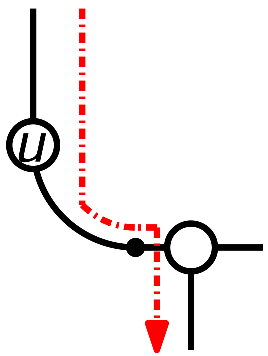
Def. • y-monotone curve



Cut

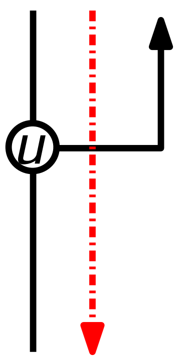
Def.
- y-monotone curve
- consists of horizontal, vertical and circular segments

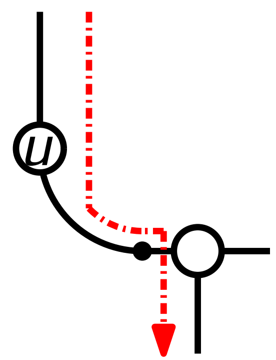




Cut

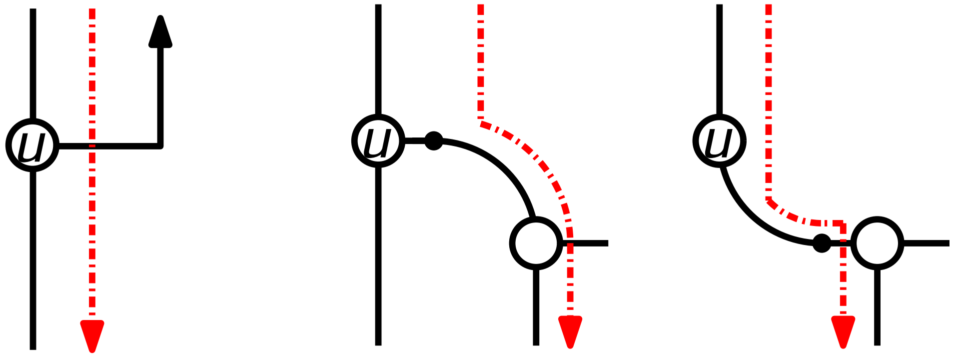
Def.
- *y*-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part





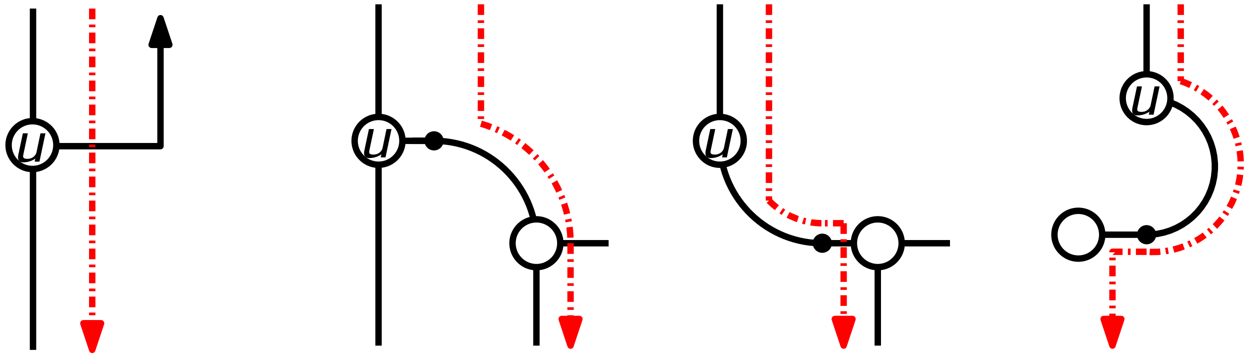
Cut

Def.
- y-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part
- intersects only horizontal segments



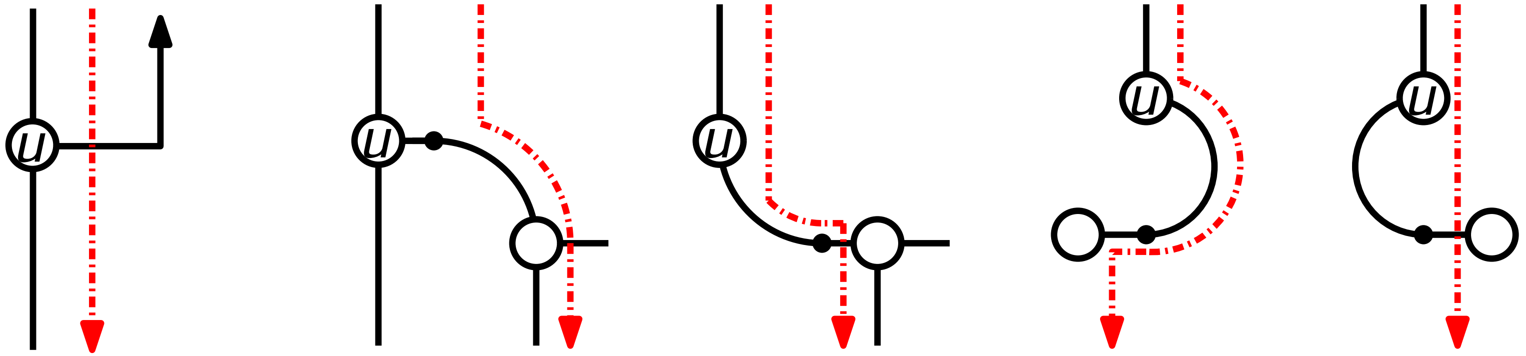
Cut

Def.
- y-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part
- intersects only horizontal segments



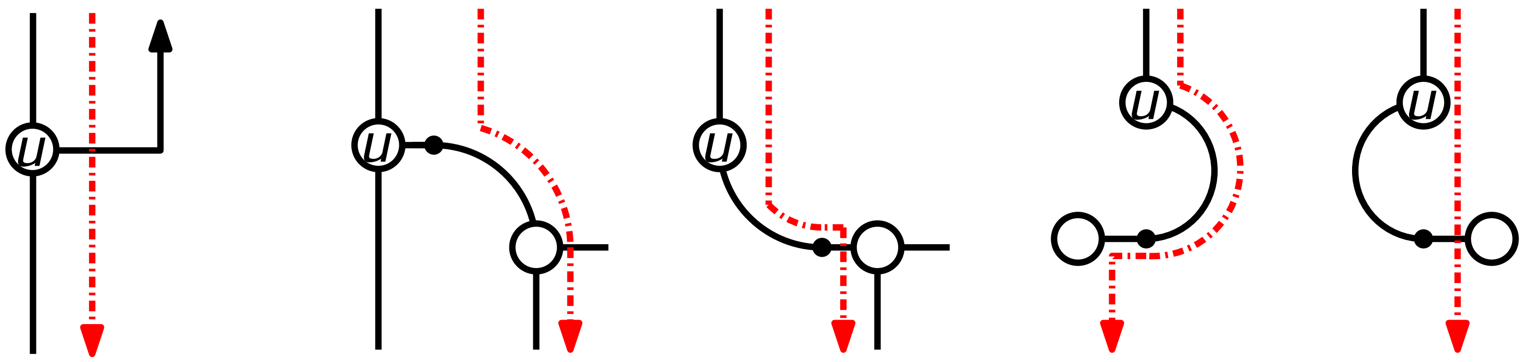
Cut

Def.
- *y*-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part
- intersects only horizontal segments



Cut

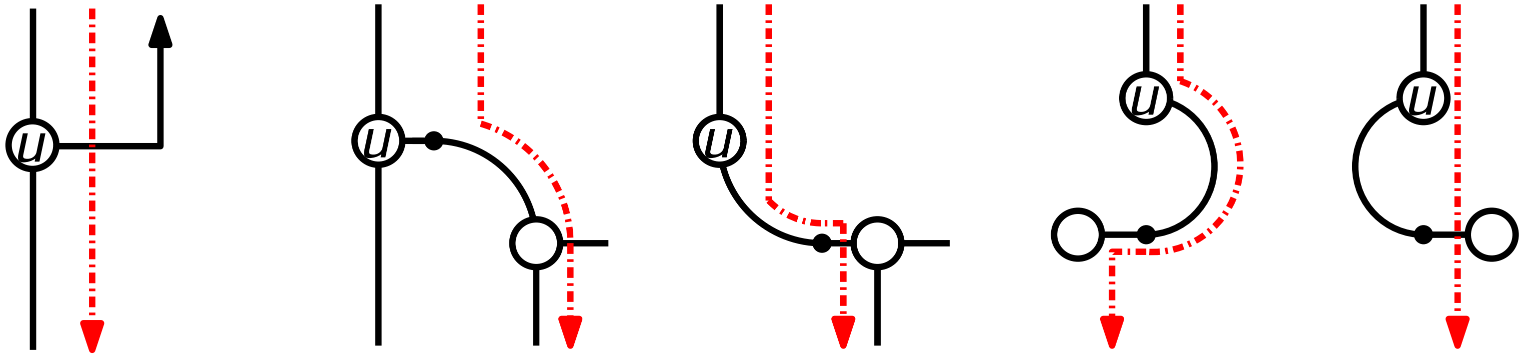
Def.
- y-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part
- intersects only horizontal segments



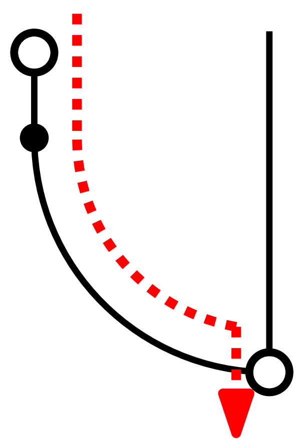
Problems:

Cut

Def.
- y-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part
- intersects only horizontal segments

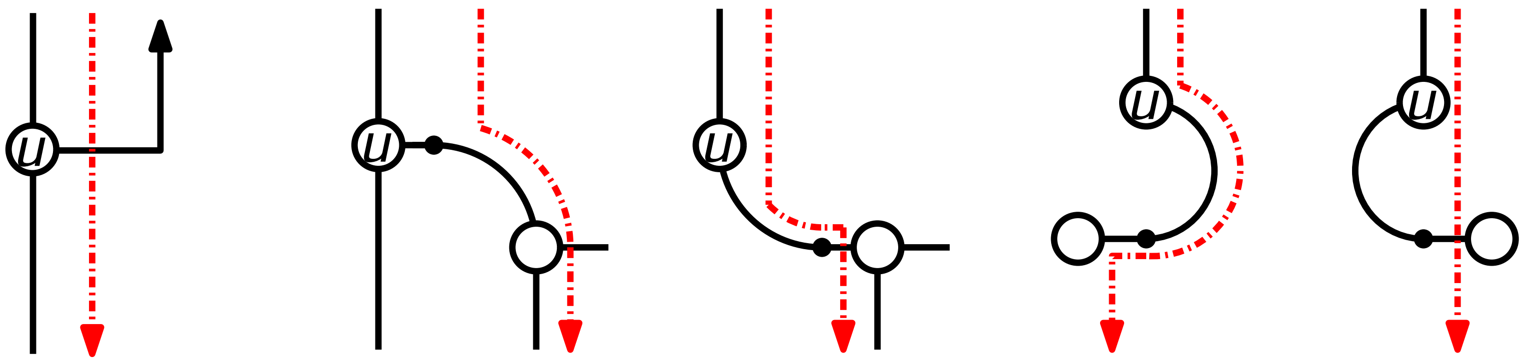


Problems:

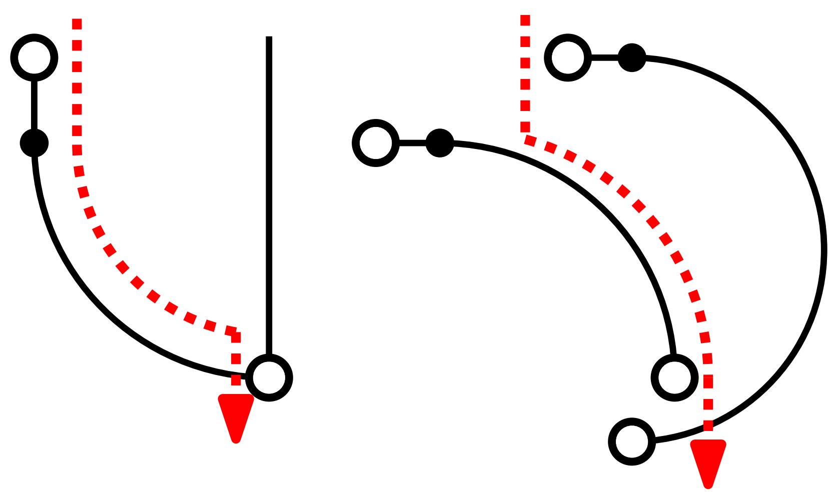


Cut

Def.
- y-monotone curve
- consists of horizontal, vertical and circular segments
- divides the current drawing into a left and a right part
- intersects only horizontal segments

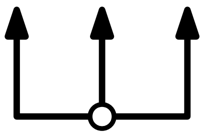


Problems:



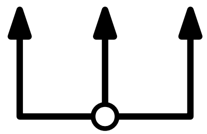
Invariants

(I_1) Every open edge is associated with a column

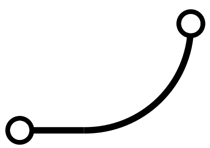


Invariants

(I_1) Every open edge is associated with a column

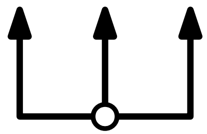


(I_2) An L-shape always contains a horizontal segment;
it never contains a vertical segment.

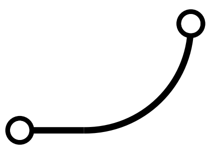


Invariants

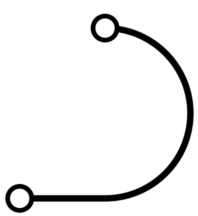
(I_1) Every open edge is associated with a column



(I_2) An L-shape always contains a horizontal segment; it never contains a vertical segment.

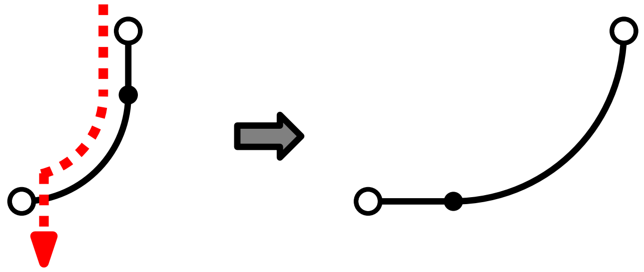


(I_3) A C-shape always has a horizontal segment incident to its bottom vertex.



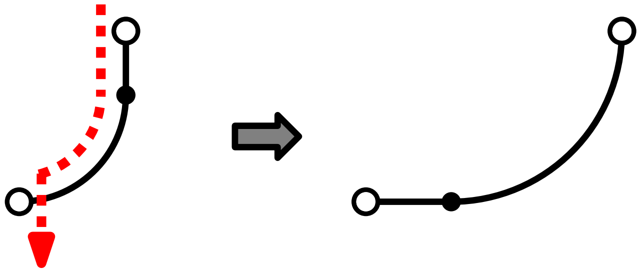
Maintain invariants

L-shape

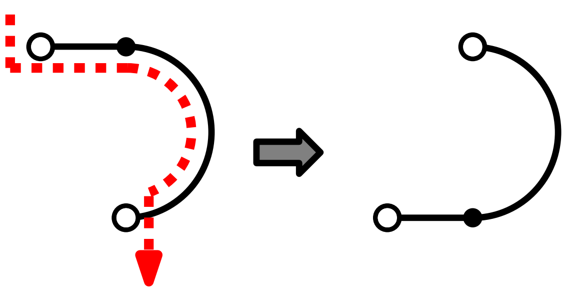


Maintain invariants

L-shape

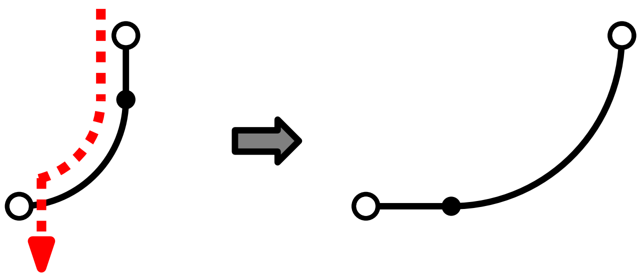


C-shape

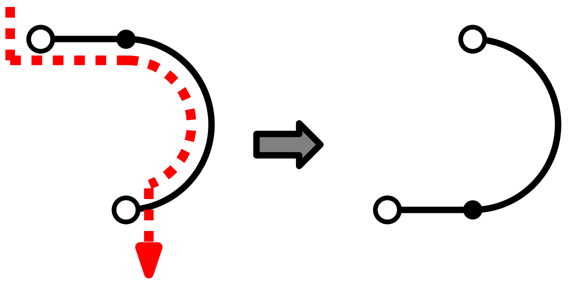


Maintain invariants

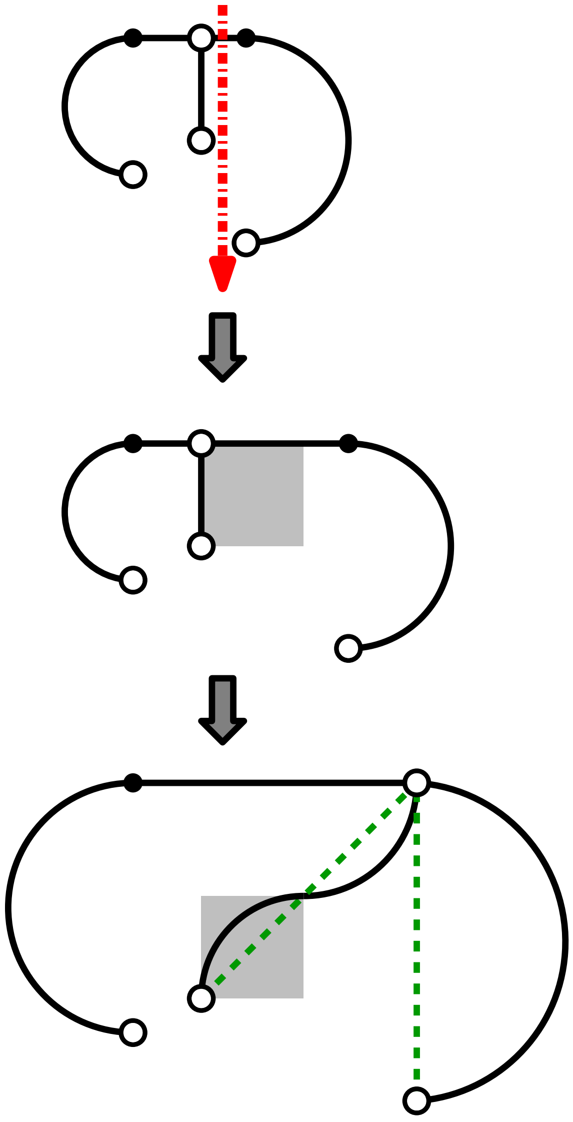
L-shape



C-shape

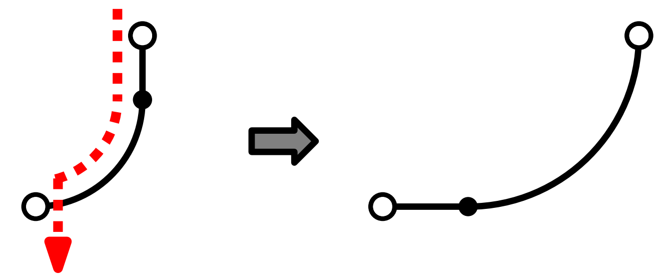


Double C-shape

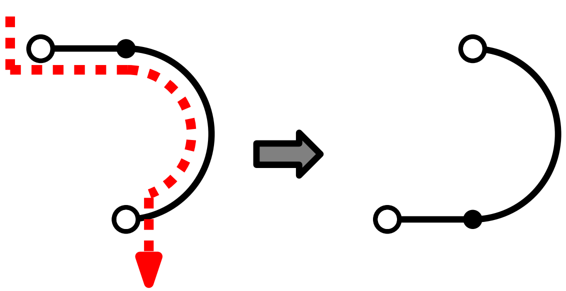


Maintain invariants

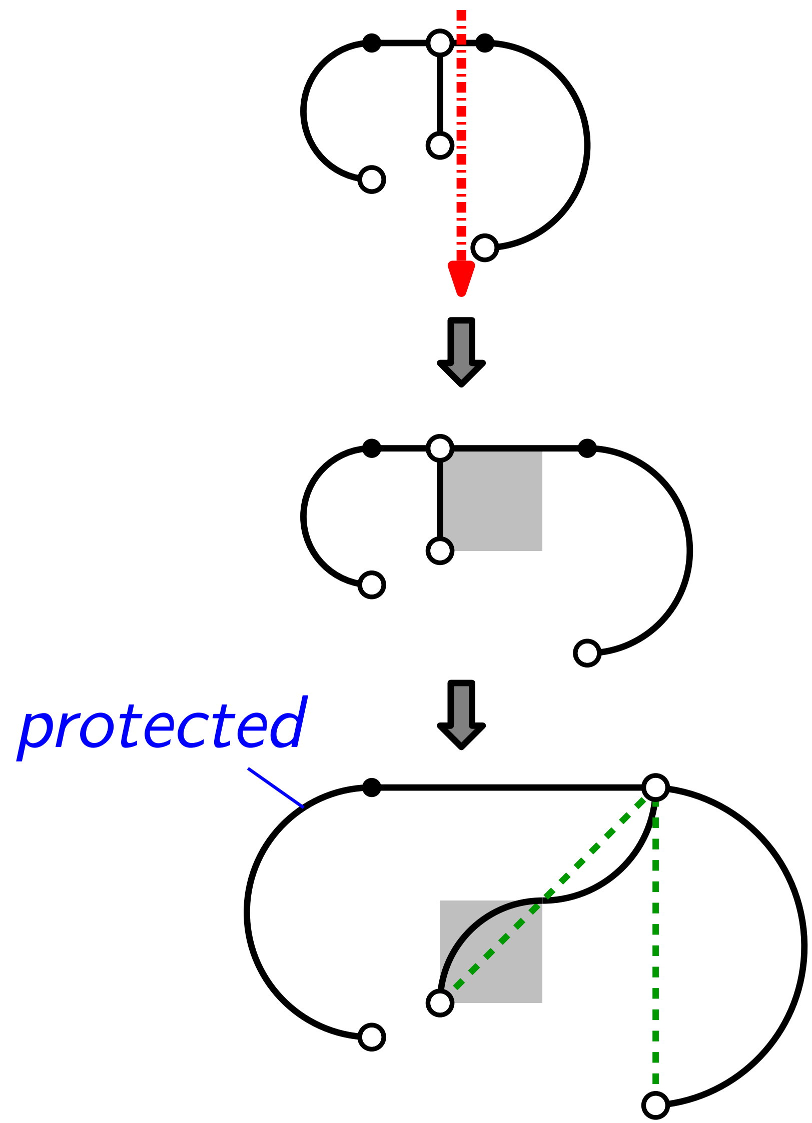
L-shape



C-shape

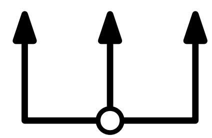


Double C-shape

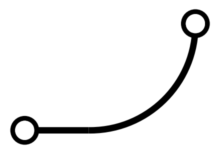


Invariants, updated

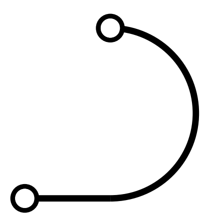
(I_1) Every open edge is associated with a column



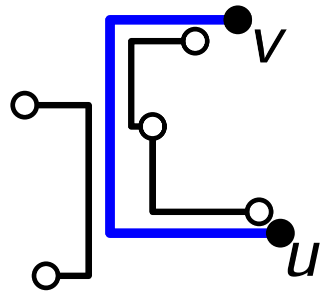
(I_2) An L-shape always contains a horizontal segment;
it never contains a vertical segment.

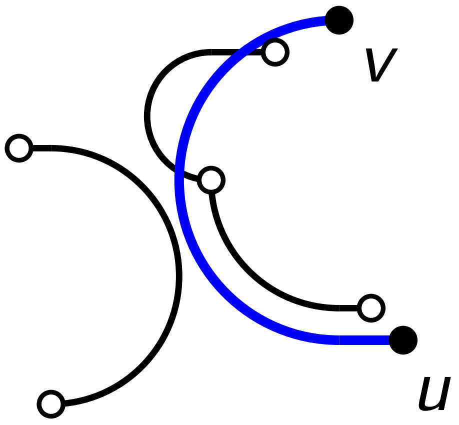


(I_3) An *unprotected* C-shape always has a horizontal segment incident to its bottom vertex.



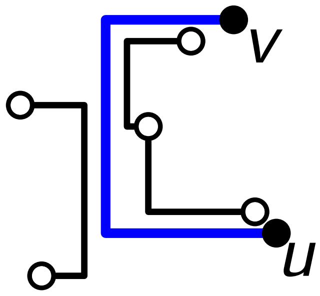
Eliminate crossings

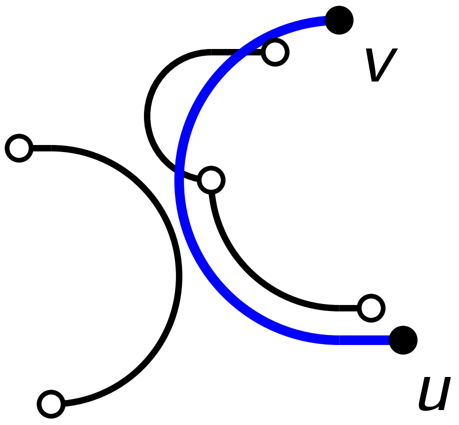




Eliminate crossings

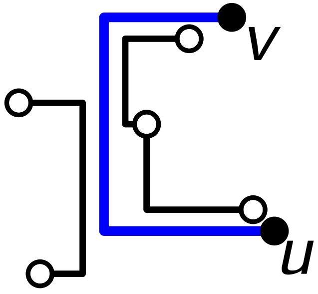
1. move v up

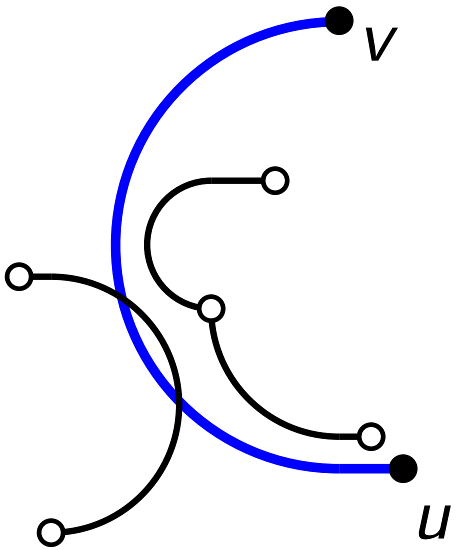




Eliminate crossings

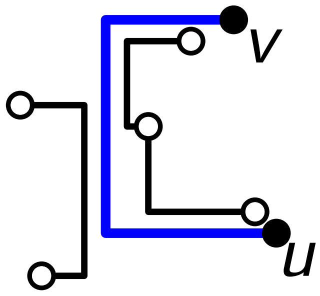
1. move v up

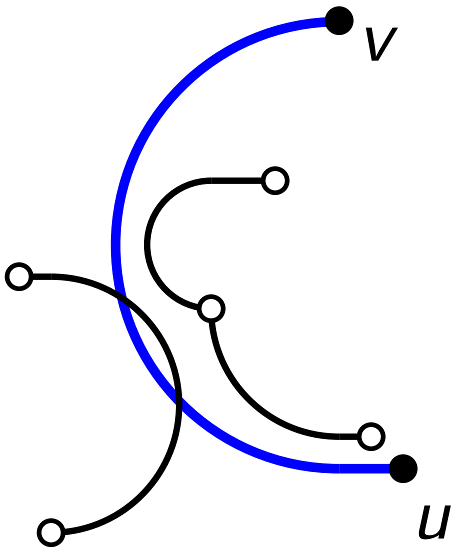




Eliminate crossings

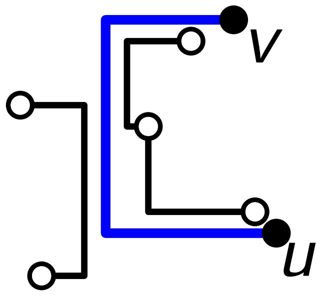
1. move v up
2. find a cut

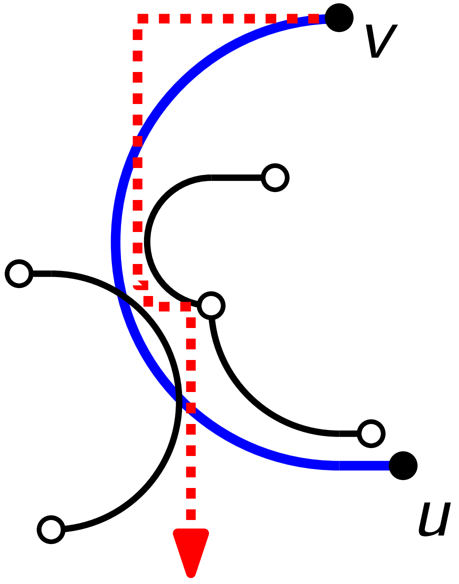




Eliminate crossings

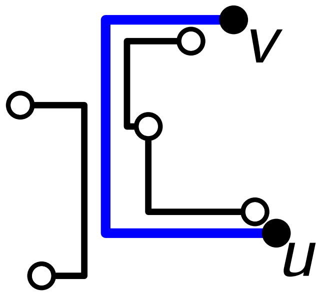
1. move v up
2. find a cut

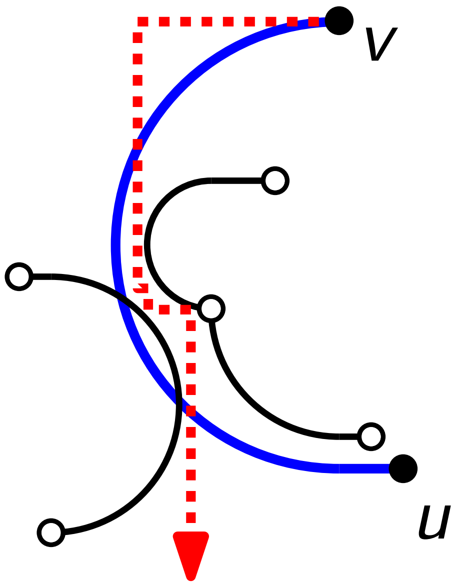




Eliminate crossings

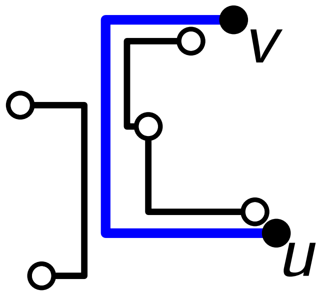
1. move v up
2. find a cut
3. move vertices to the left

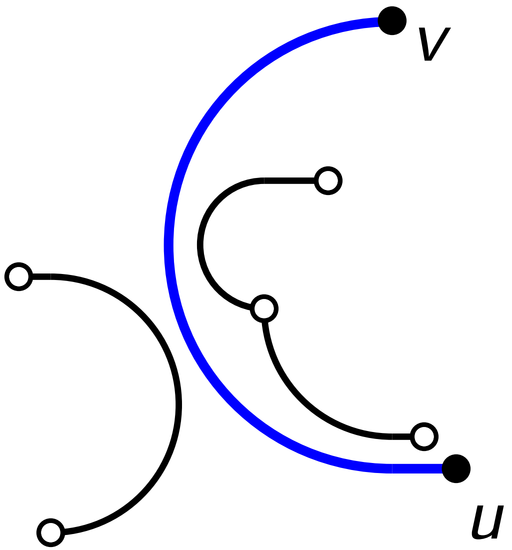




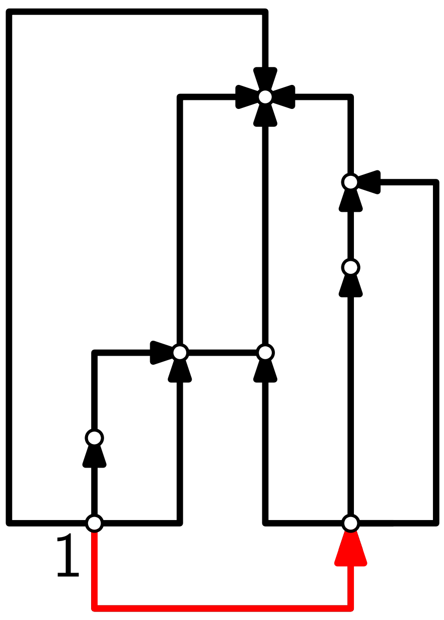
Eliminate crossings

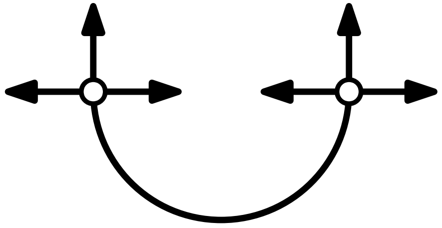
1. move v up
2. find a cut
3. move vertices to the left



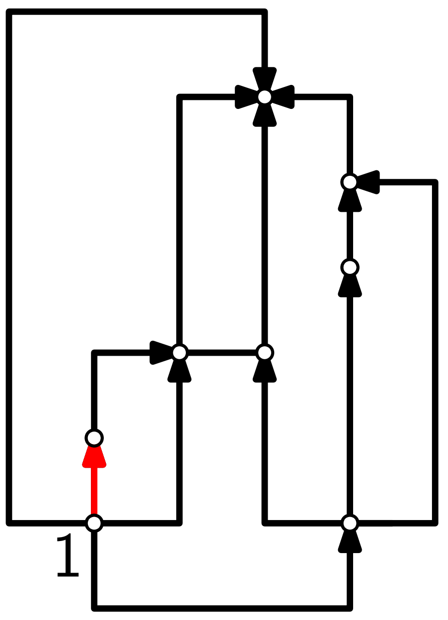


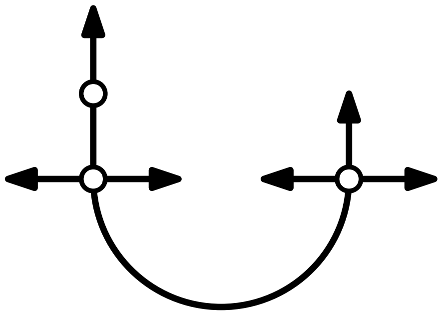
Example Run



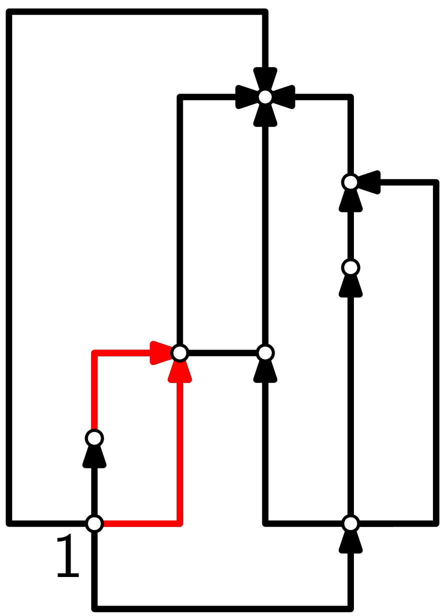


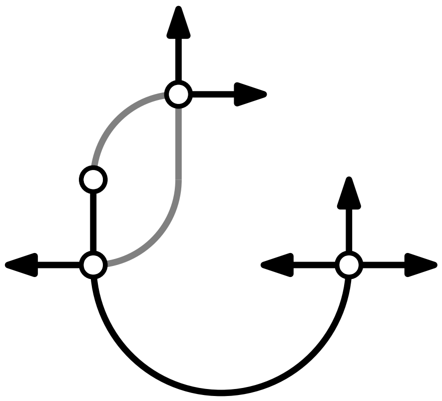
Example Run



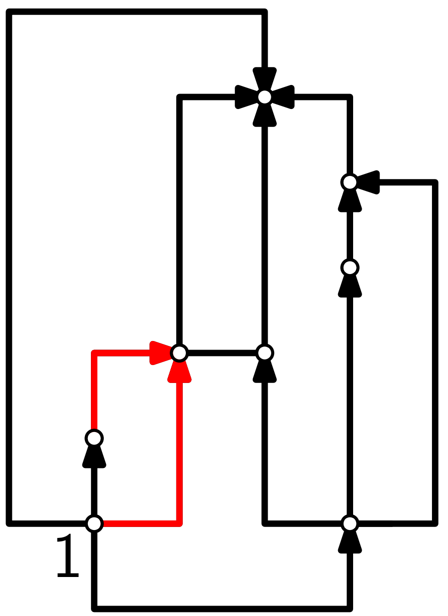


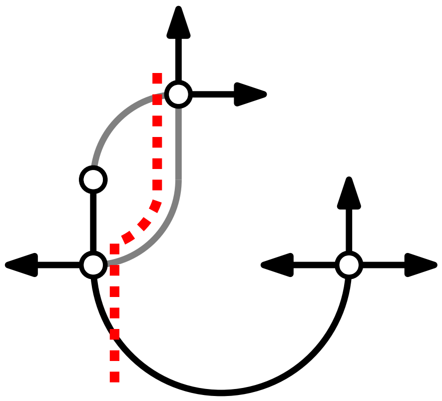
Example Run



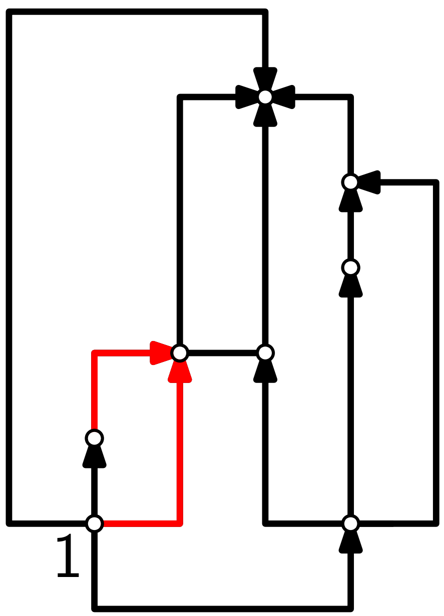


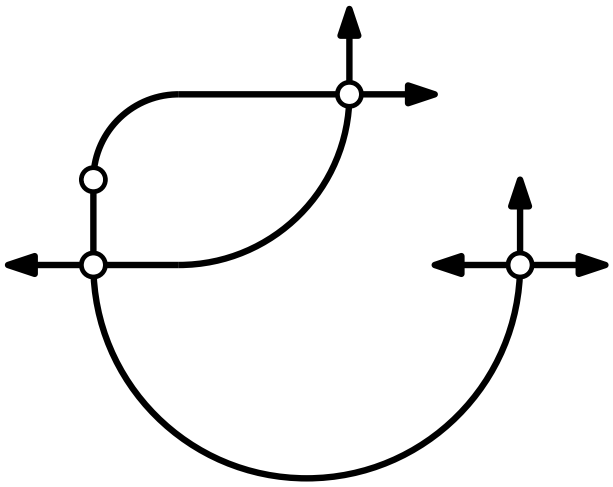
Example Run



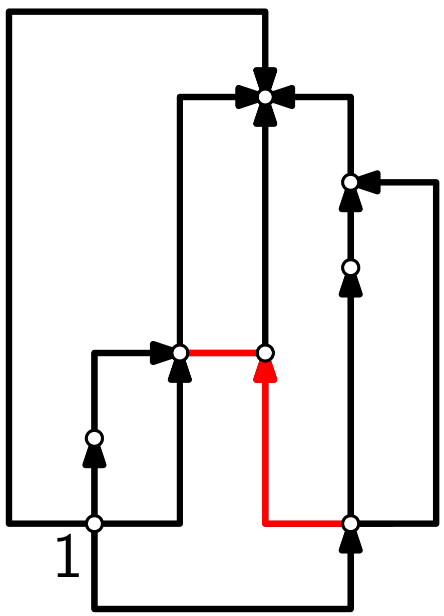


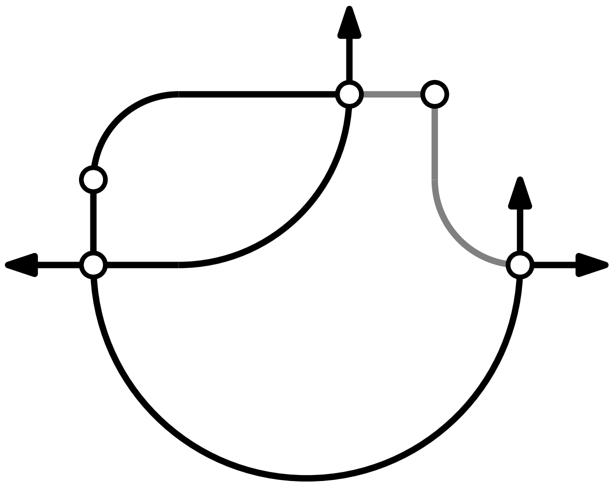
Example Run



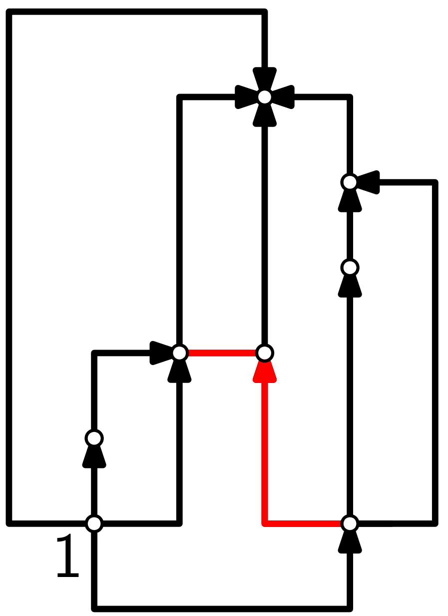


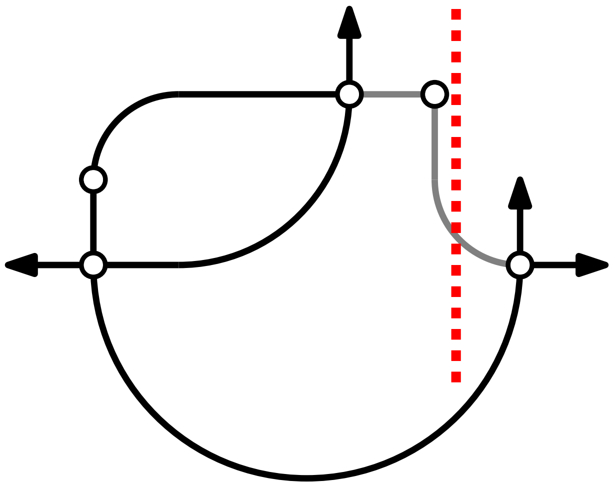
Example Run



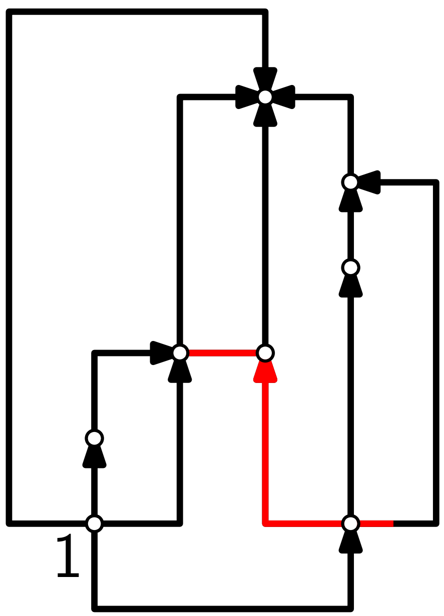


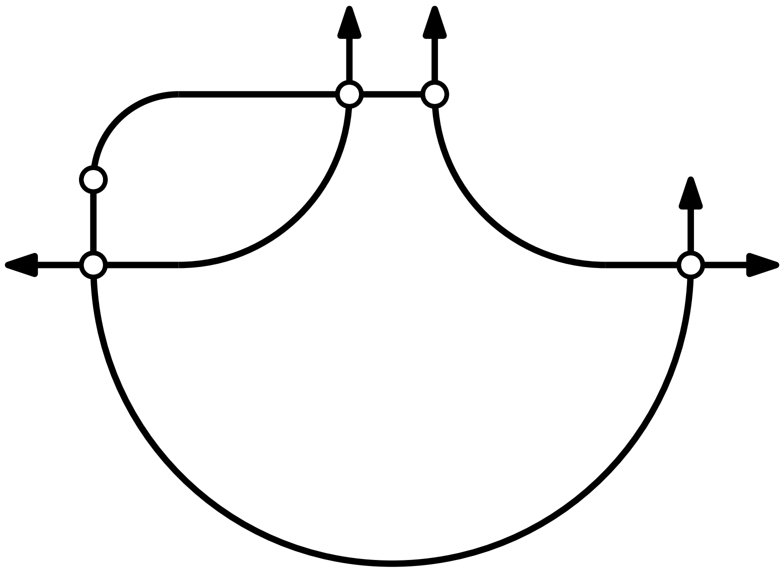
Example Run



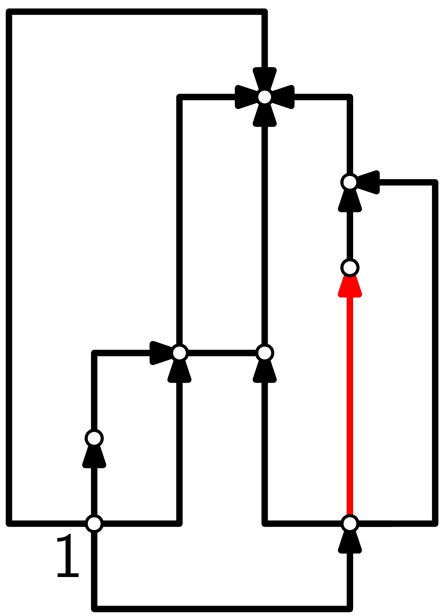


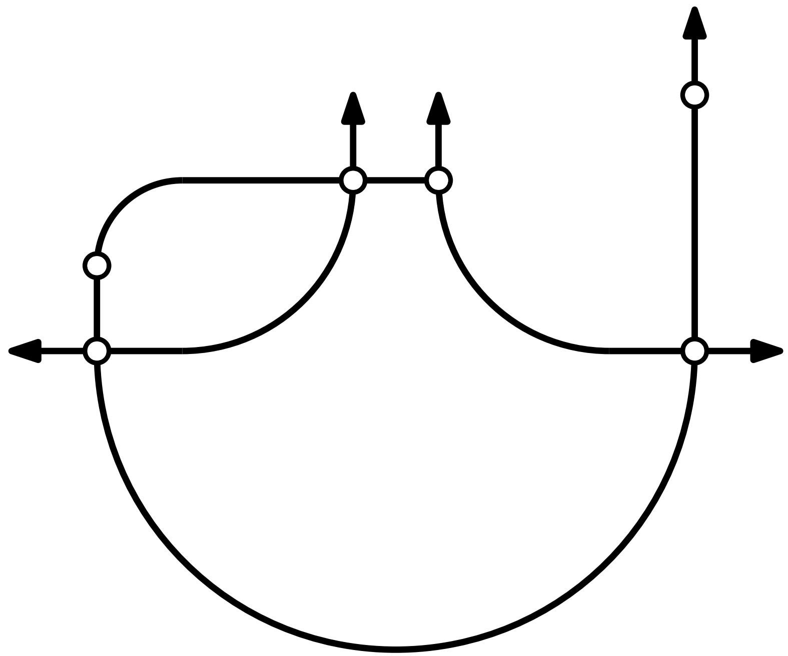
Example Run



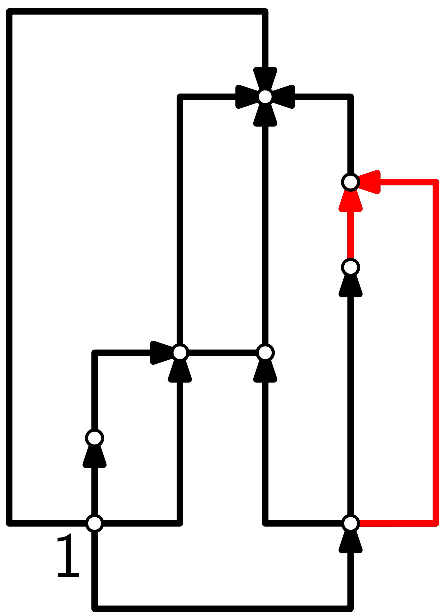


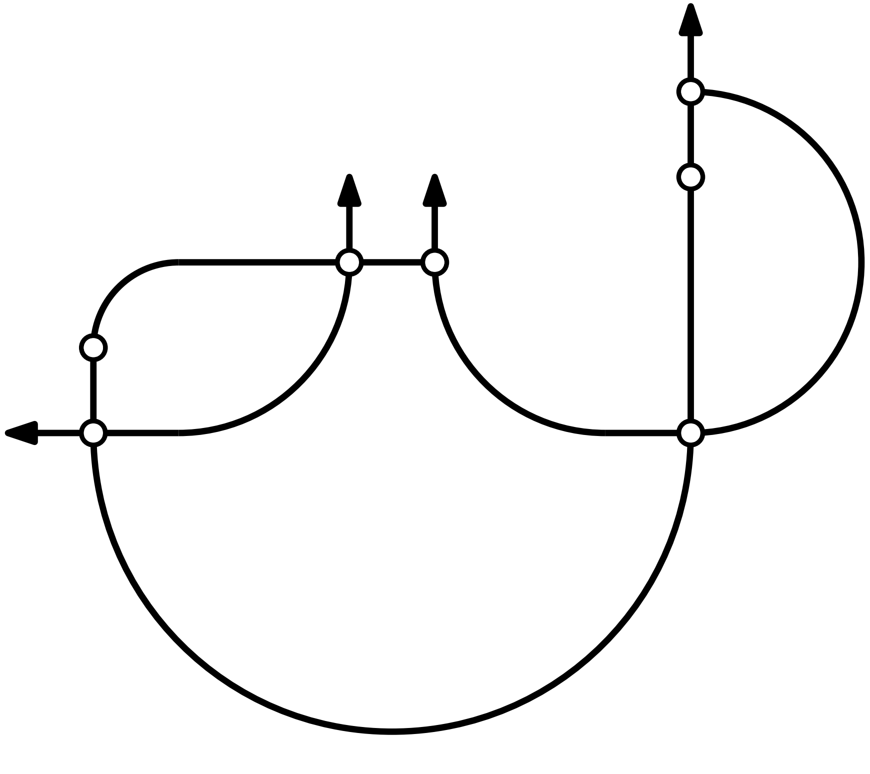
Example Run



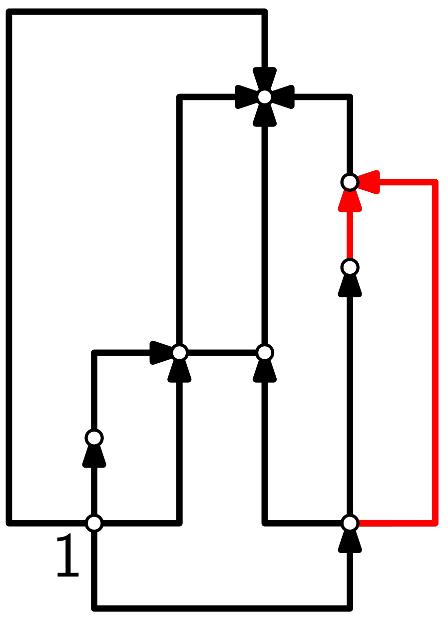


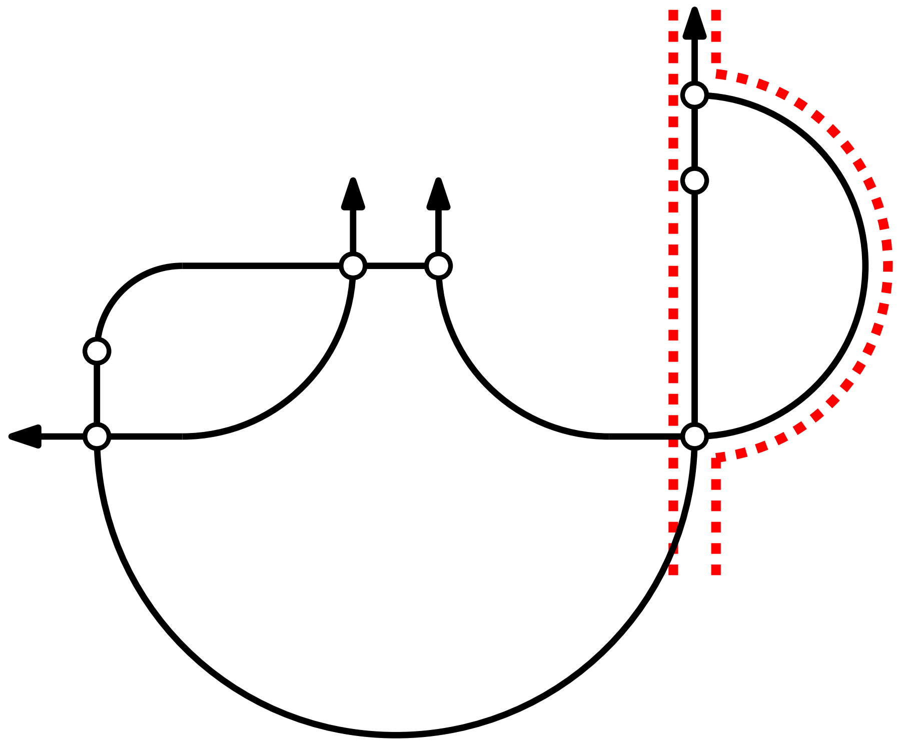
Example Run



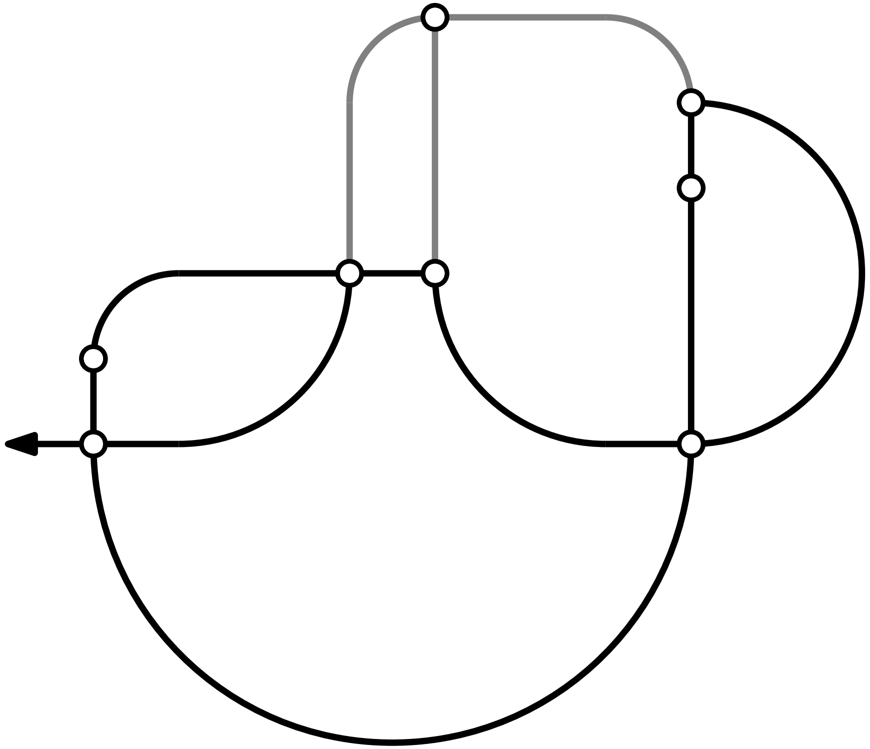


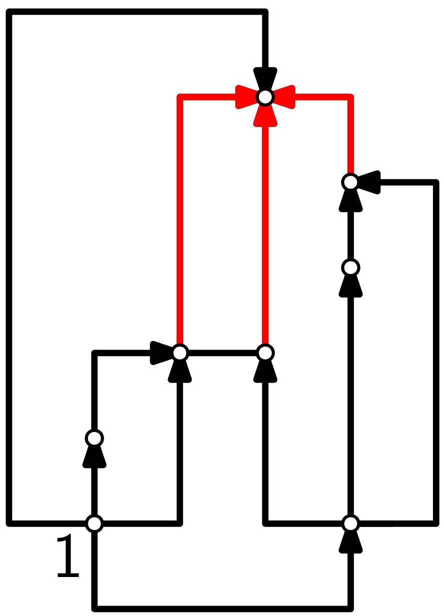
Example Run



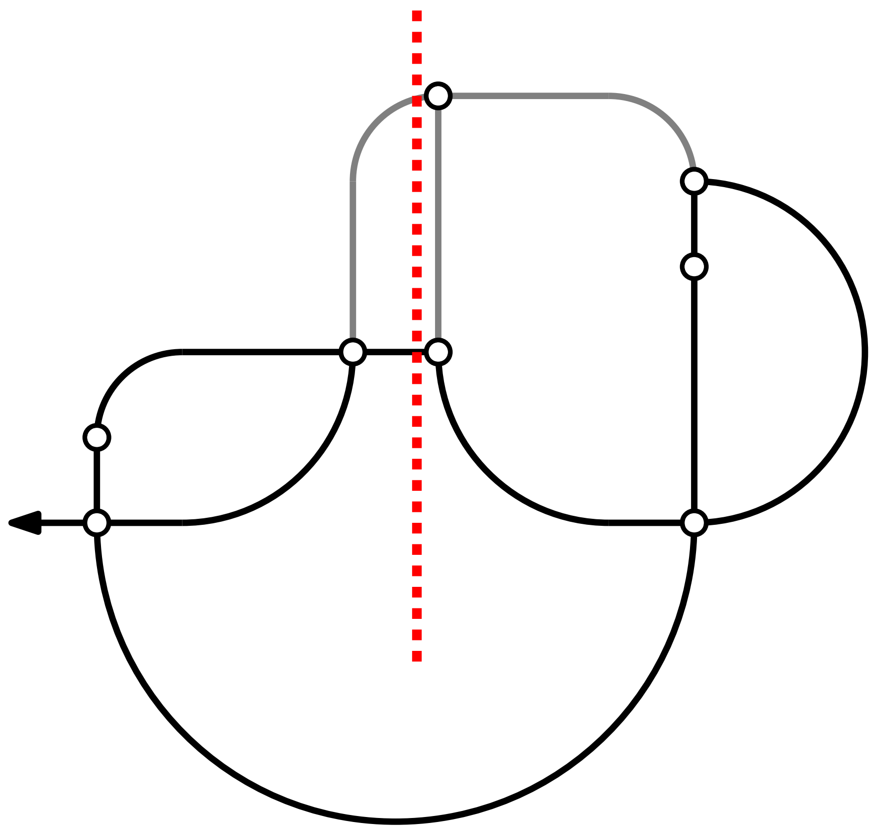


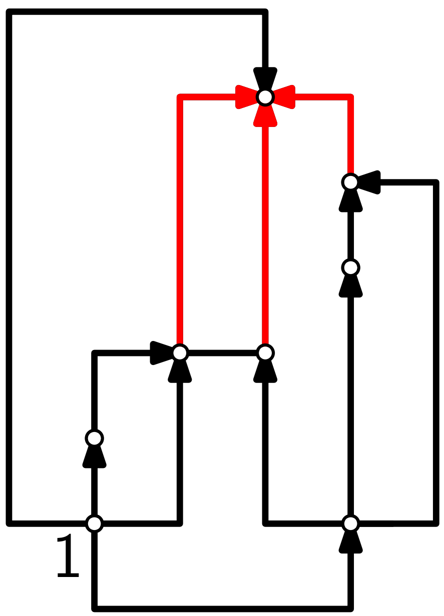
Example Run



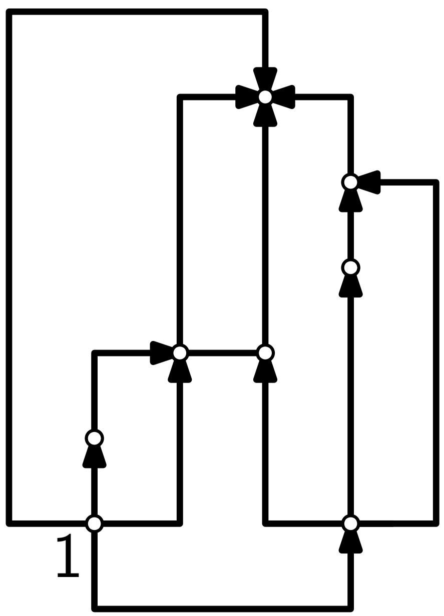


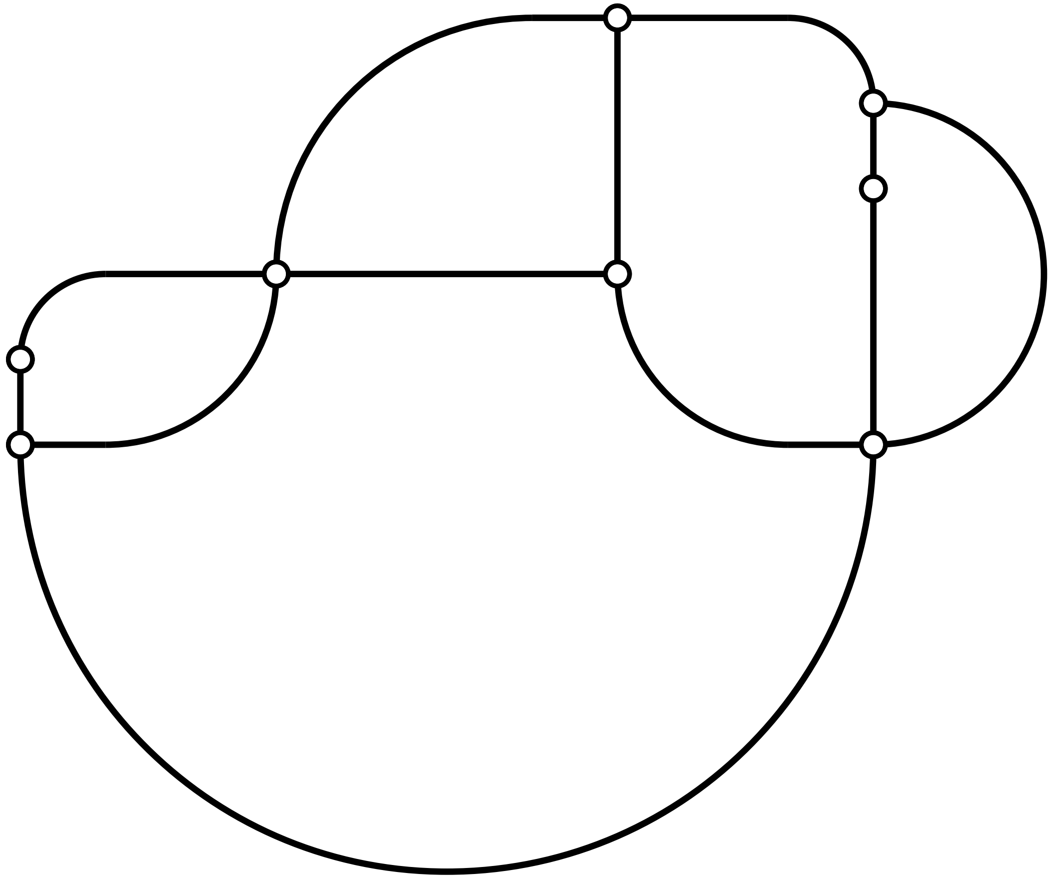
Example Run



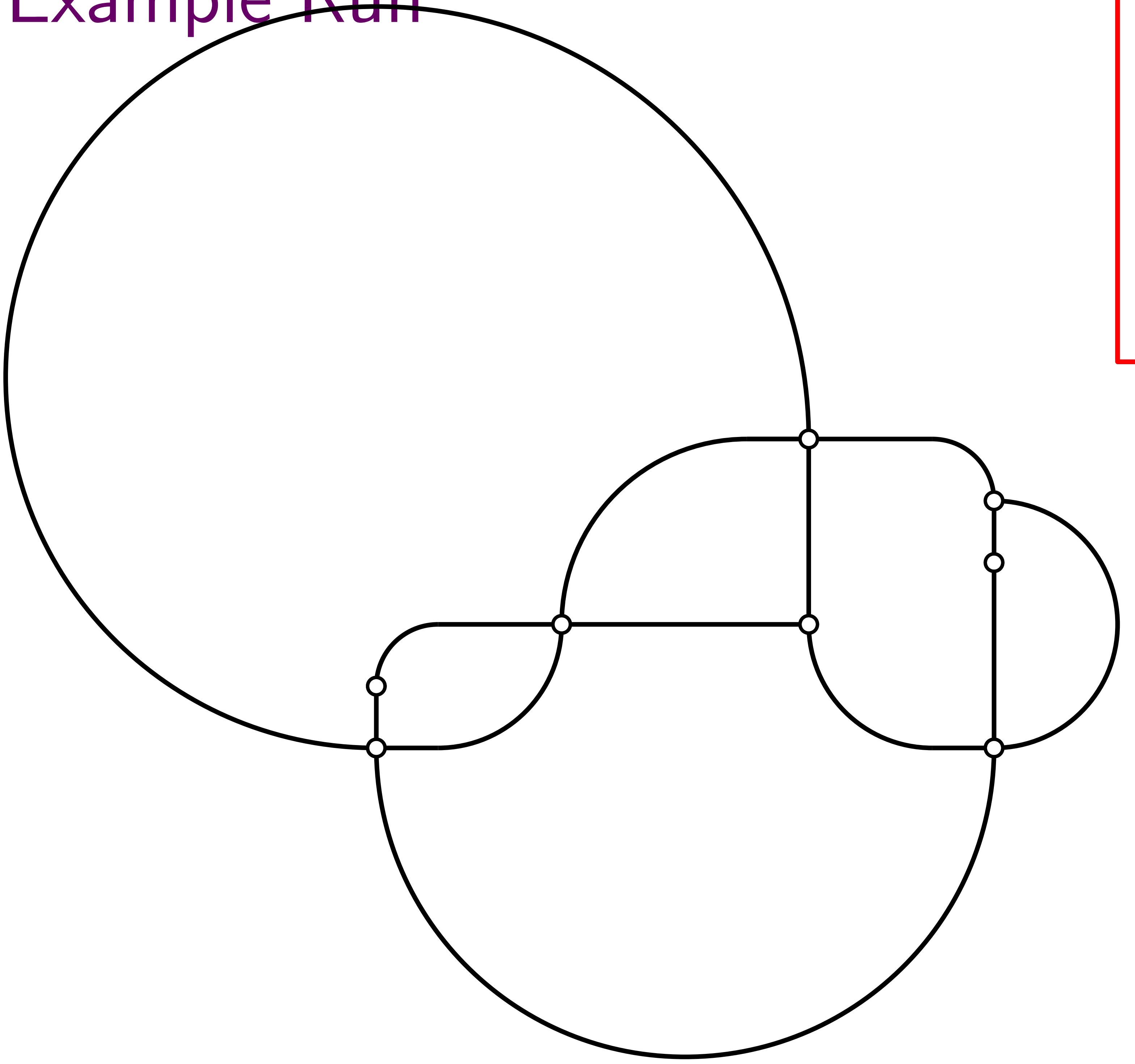


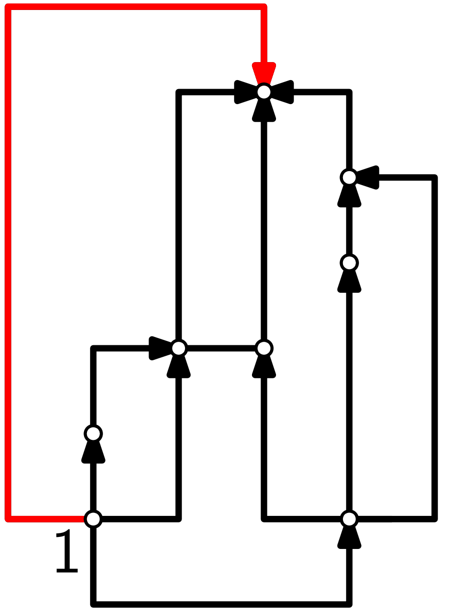
Example Run



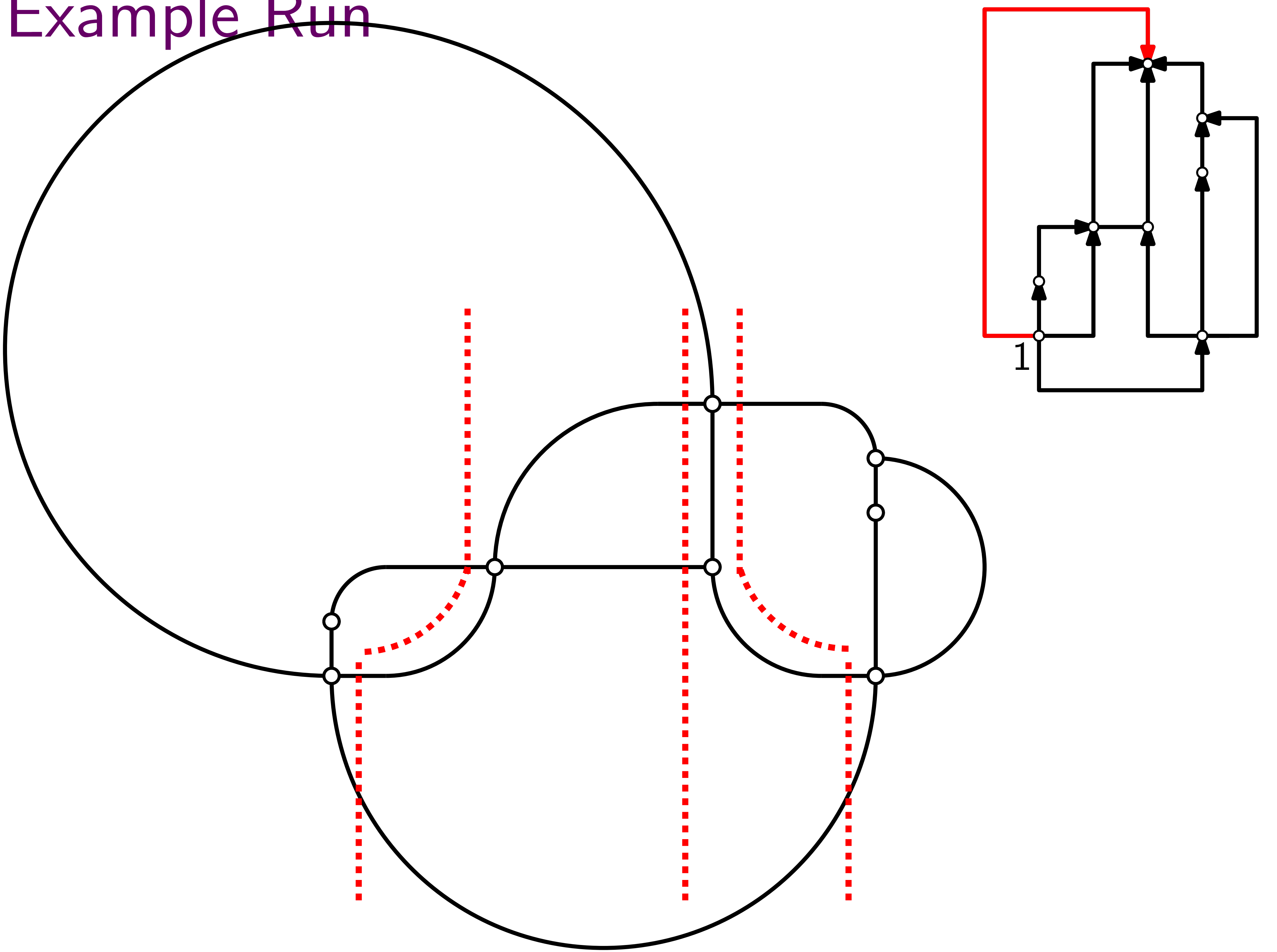


Example Run

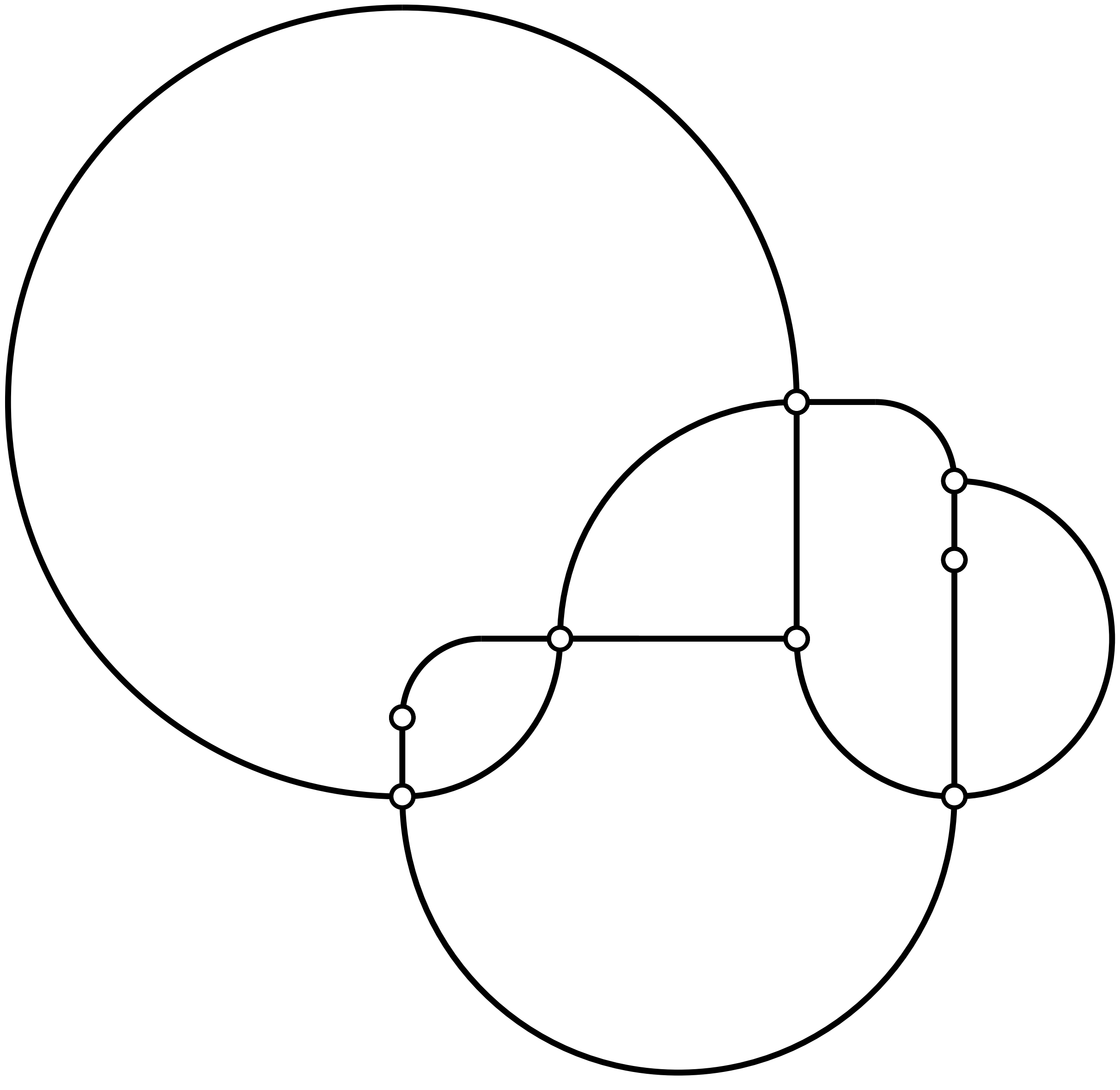


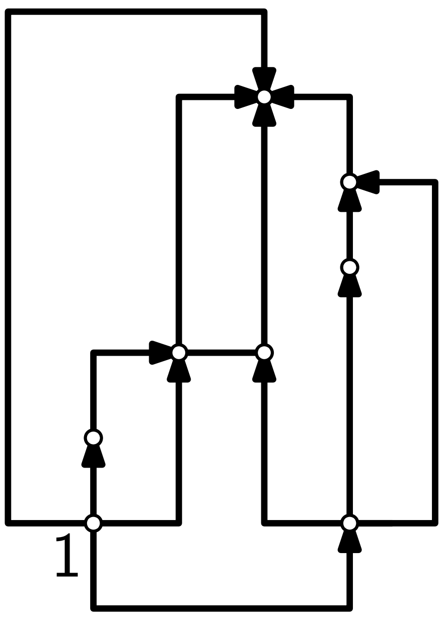


Example Run

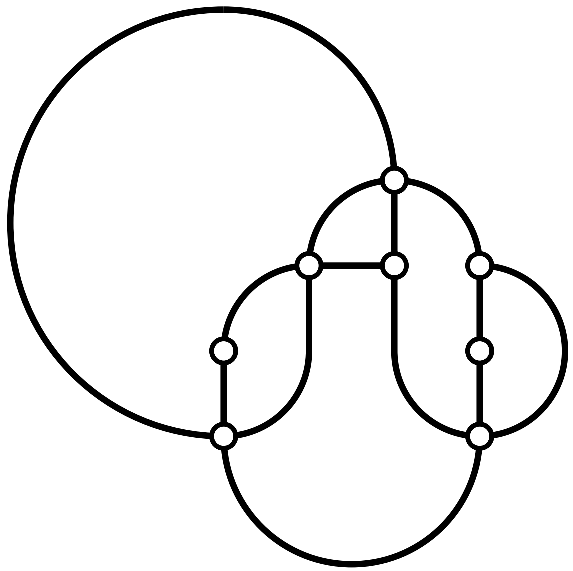


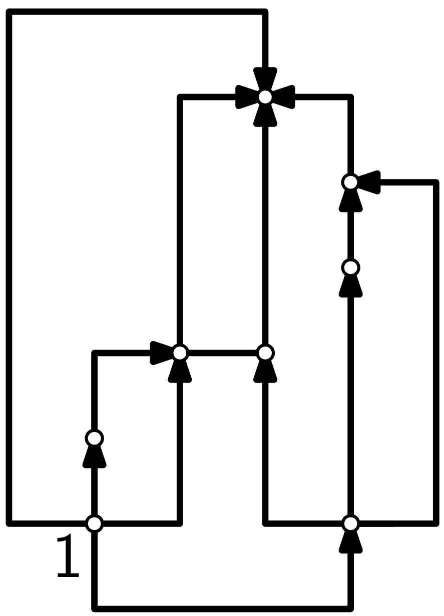
Example Run





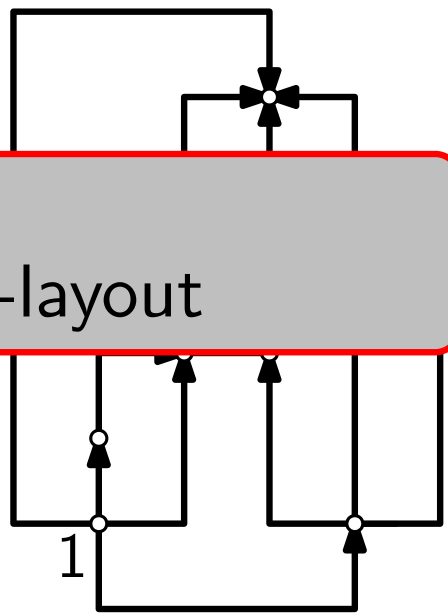
Example Run

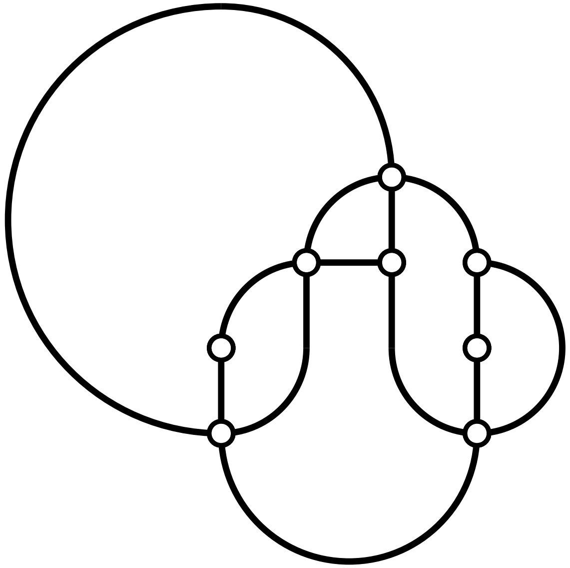




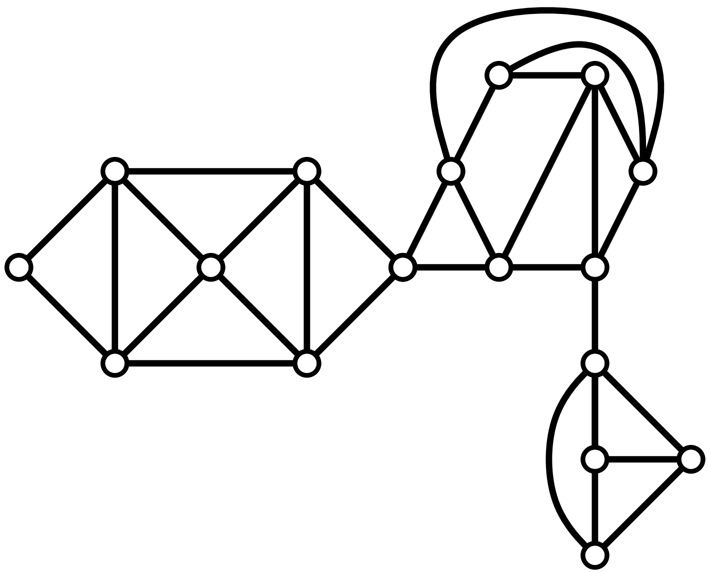
Example Run

[Theorem]
Every biconnected 4-planar graph admits an SC_2-layout



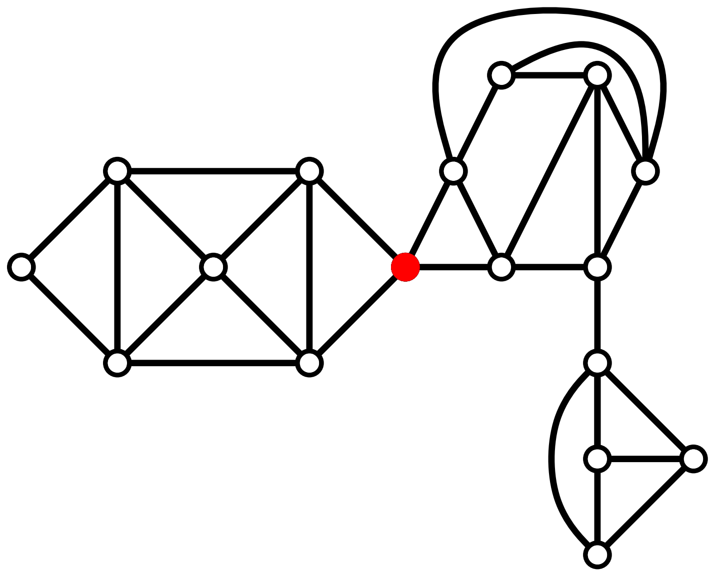


Extension to Arbitrary Graphs



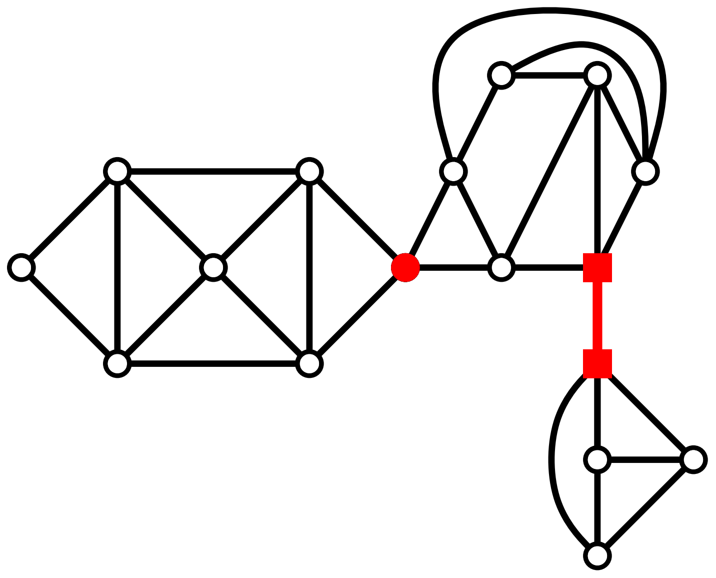
Extension to Arbitrary Graphs

- cutvertices



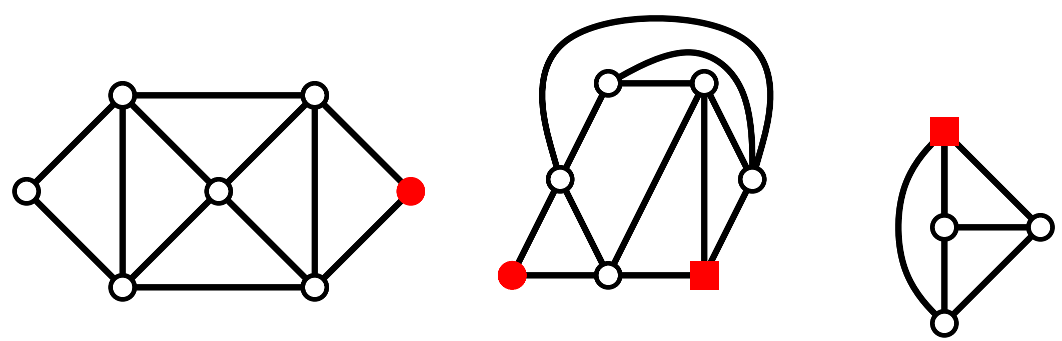
Extension to Arbitrary Graphs

- cutvertices
- bridges



Extension to Arbitrary Graphs

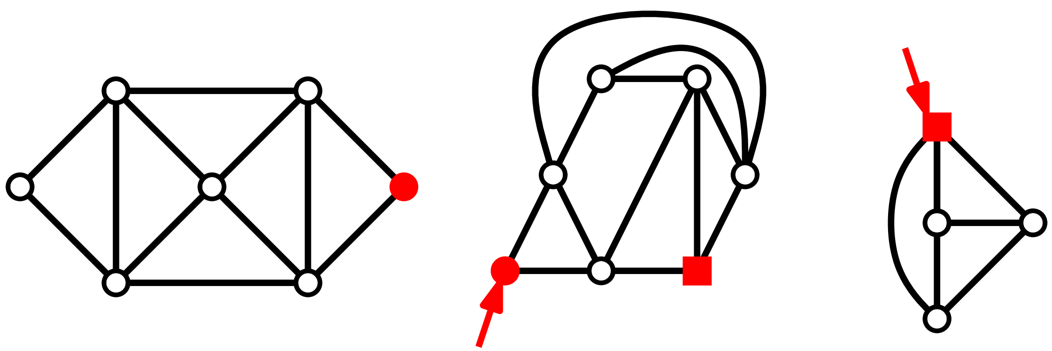
- ● cutvertices
- ■—■ bridges



Extension to Arbitrary Graphs

- ● cutvertices
- ■—■ bridges

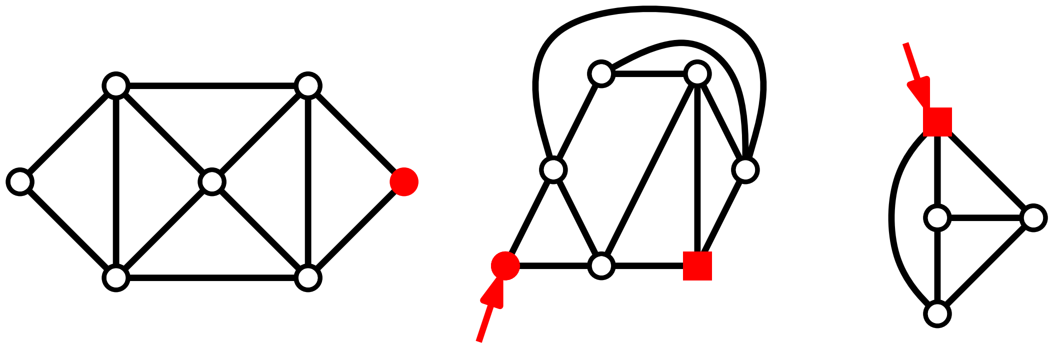
Draw specific vertex on outer face



Extension to Arbitrary Graphs

- cutvertices
- bridges

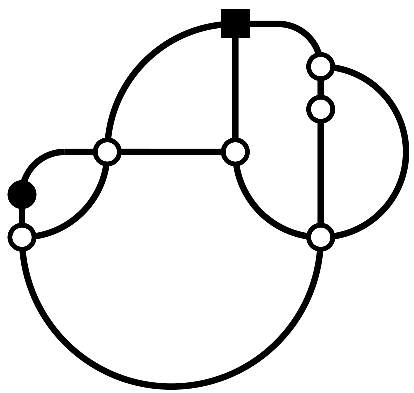
Draw specific vertex on outer face
Draw cut vertices with right angles



Extension to Arbitrary Graphs

- cutvertices
- bridges

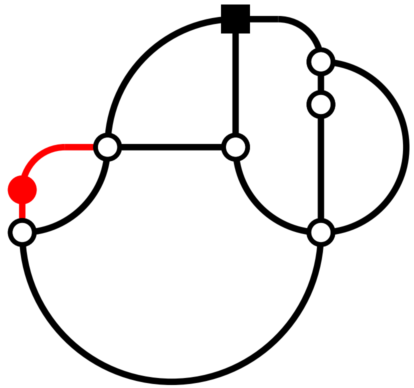
Draw specific vertex on outer face
Draw cut vertices with right angles



Extension to Arbitrary Graphs

- cutvertices
- bridges

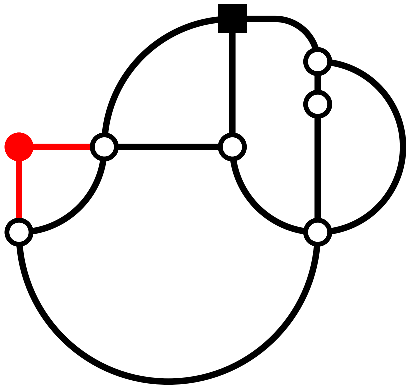
Draw specific vertex on outer face
Draw cut vertices with right angles



Extension to Arbitrary Graphs

- cutvertices
- bridges

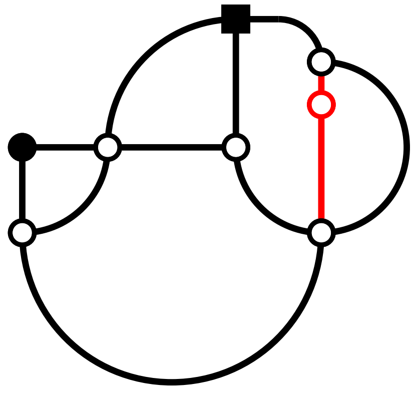
Draw specific vertex on outer face
Draw cut vertices with right angles



Extension to Arbitrary Graphs

- cutvertices
- bridges

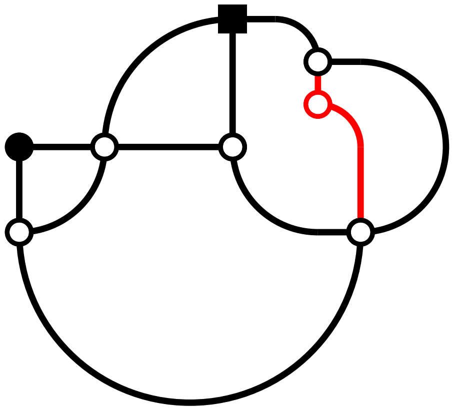
Draw specific vertex on outer face
Draw cut vertices with right angles



Extension to Arbitrary Graphs

- cutvertices
- bridges

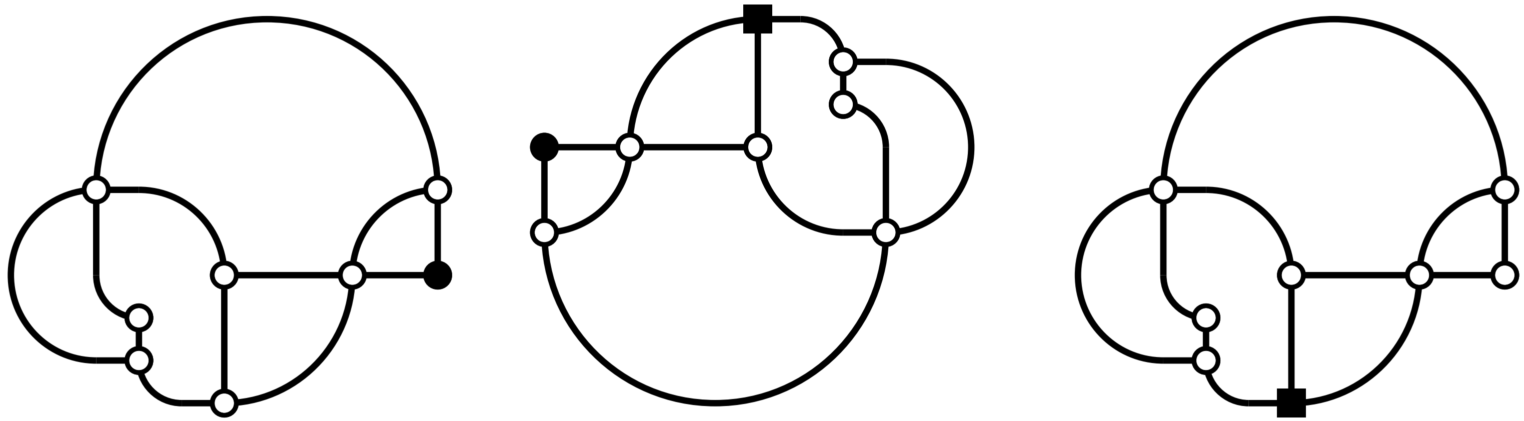
Draw specific vertex on outer face
Draw cut vertices with right angles



Extension to Arbitrary Graphs

- cutvertices
- bridges

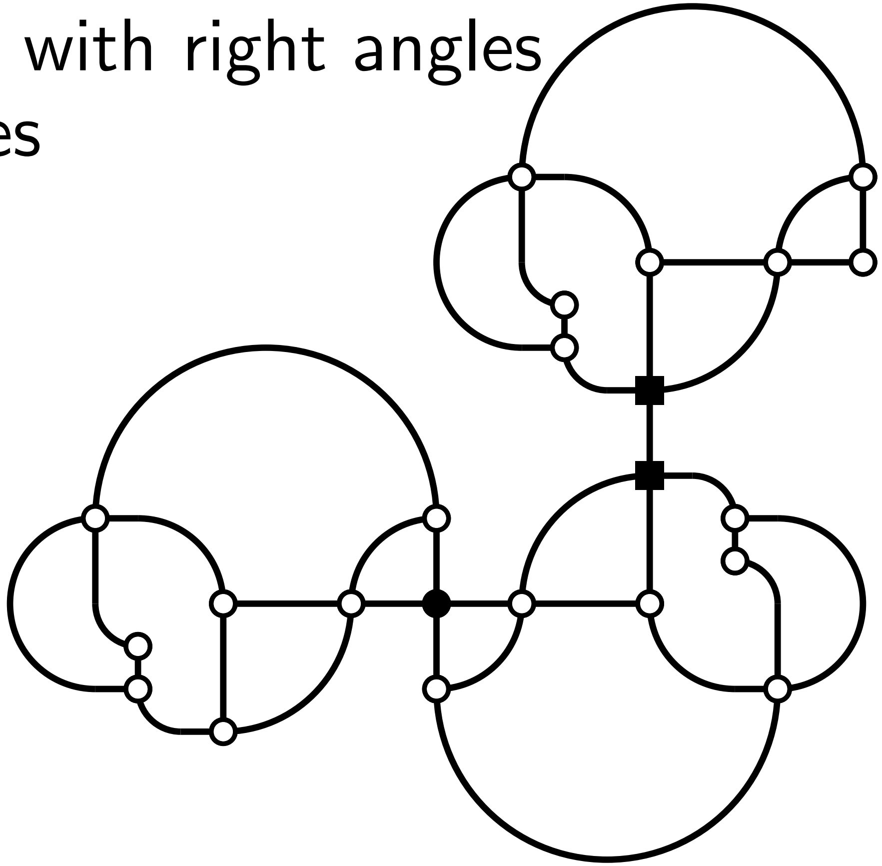
Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces



Extension to Arbitrary Graphs

- cutvertices
- bridges

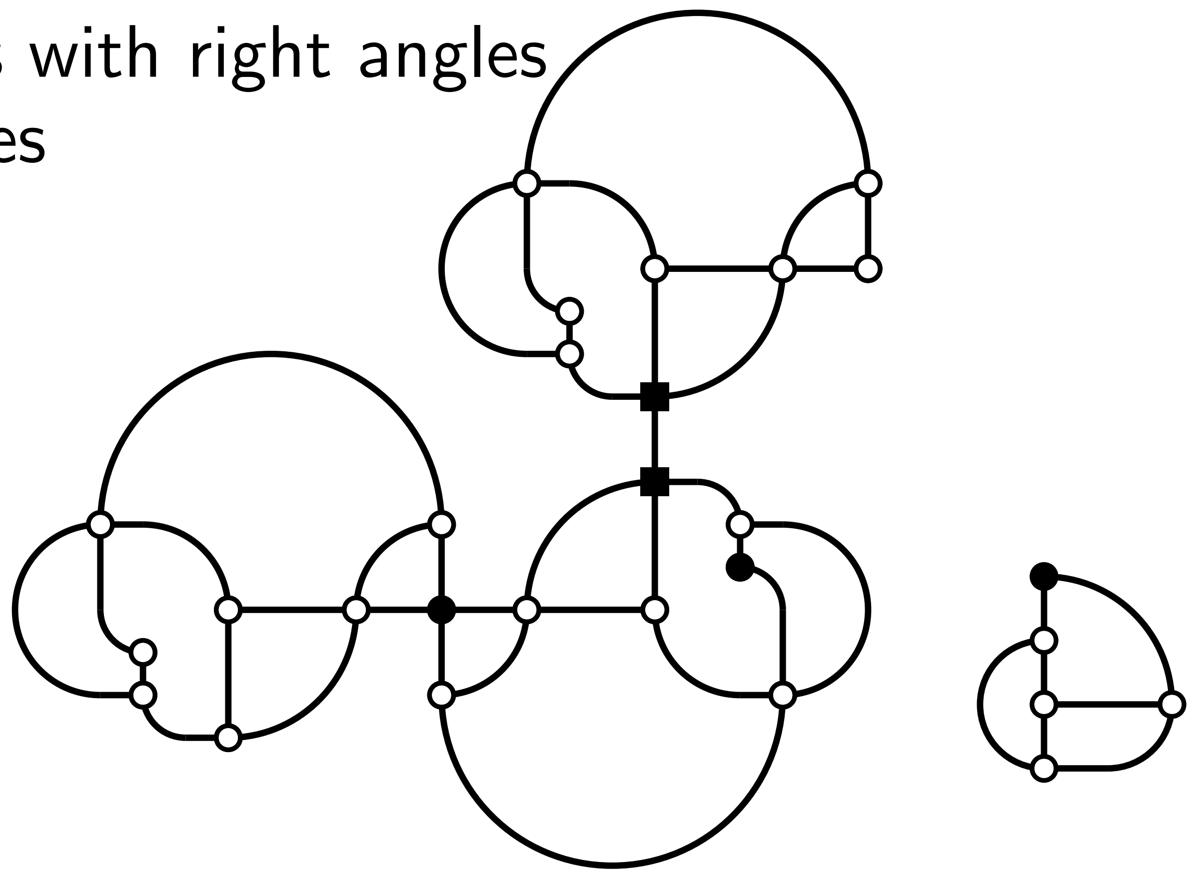
Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces



Extension to Arbitrary Graphs

- cutvertices
- bridges

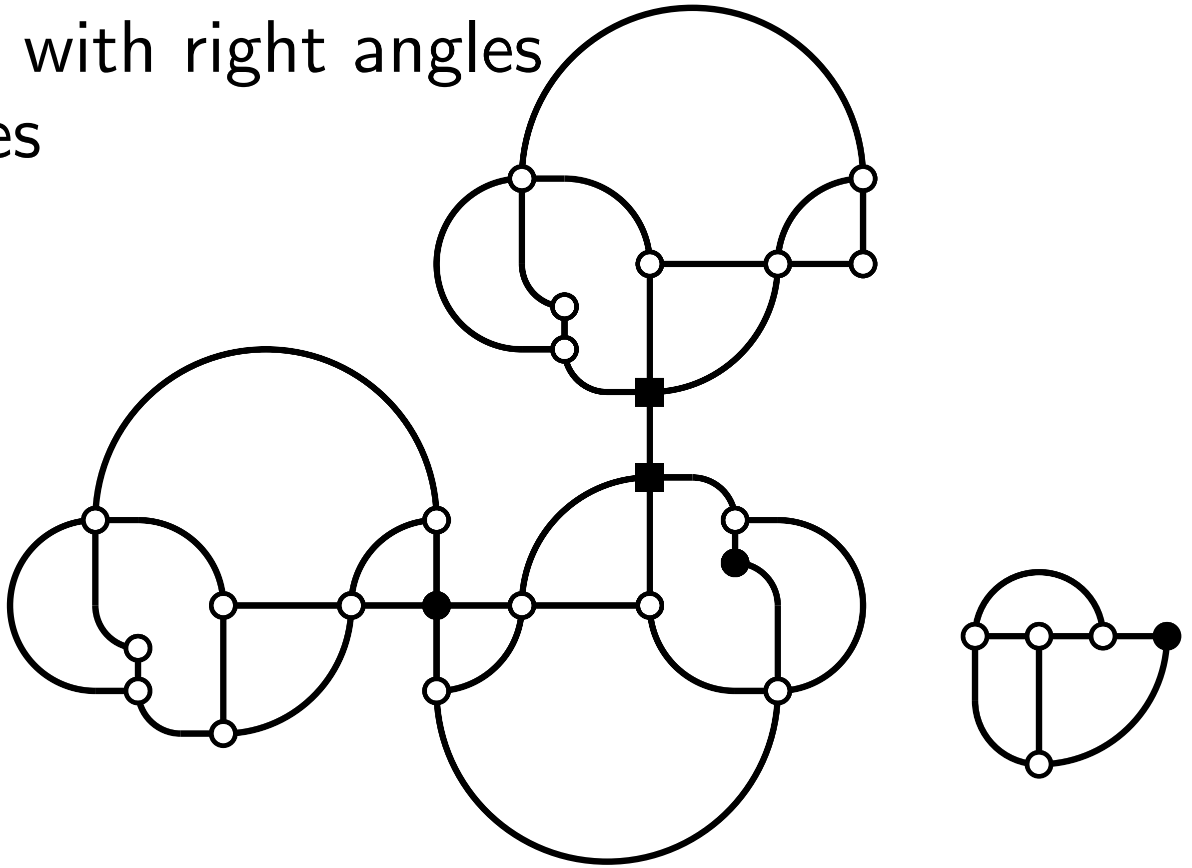
Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces



Extension to Arbitrary Graphs

- cutvertices
- bridges

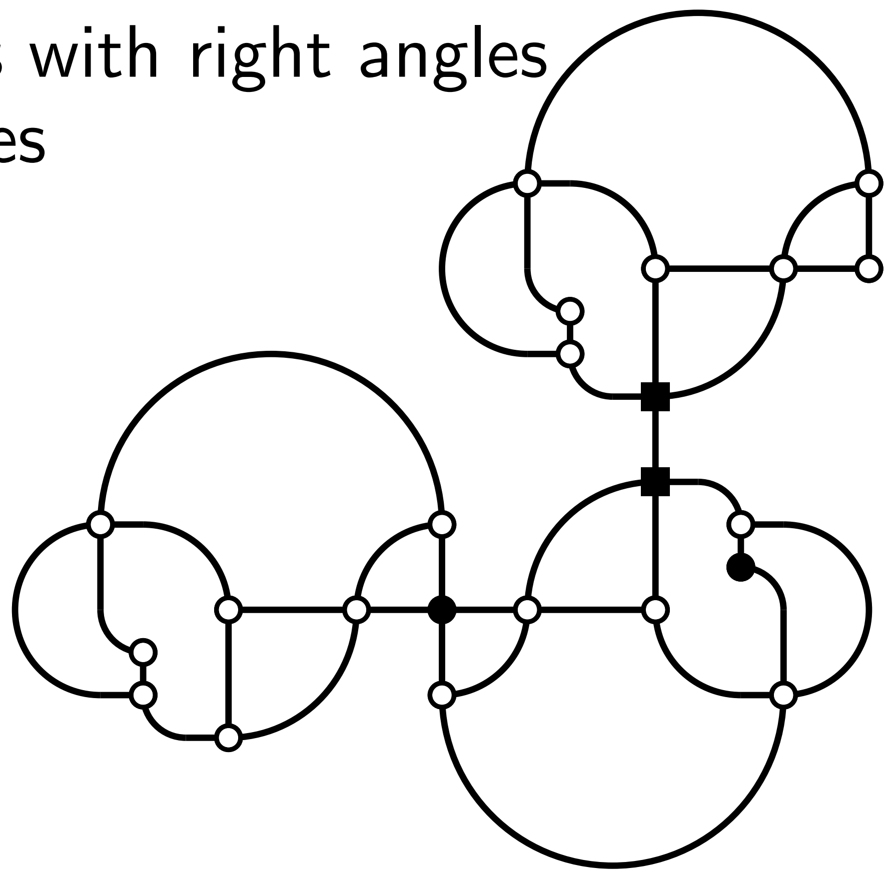
Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces

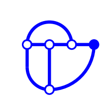


Extension to Arbitrary Graphs

- cutvertices
- bridges

Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces

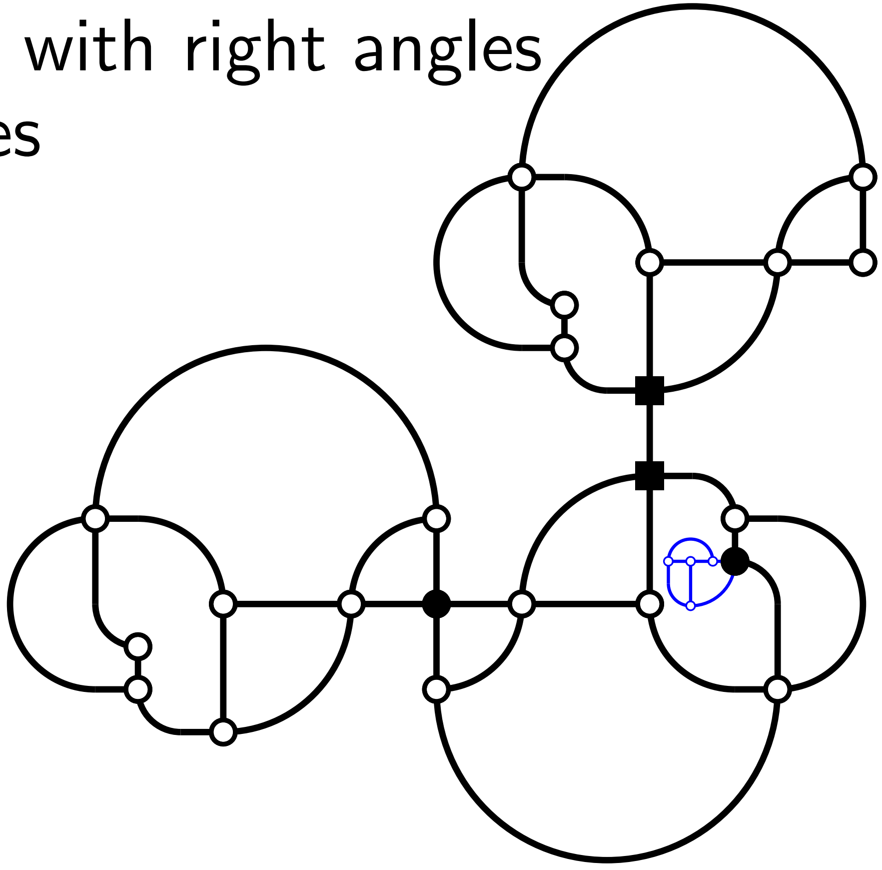




Extension to Arbitrary Graphs

- cutvertices
- bridges

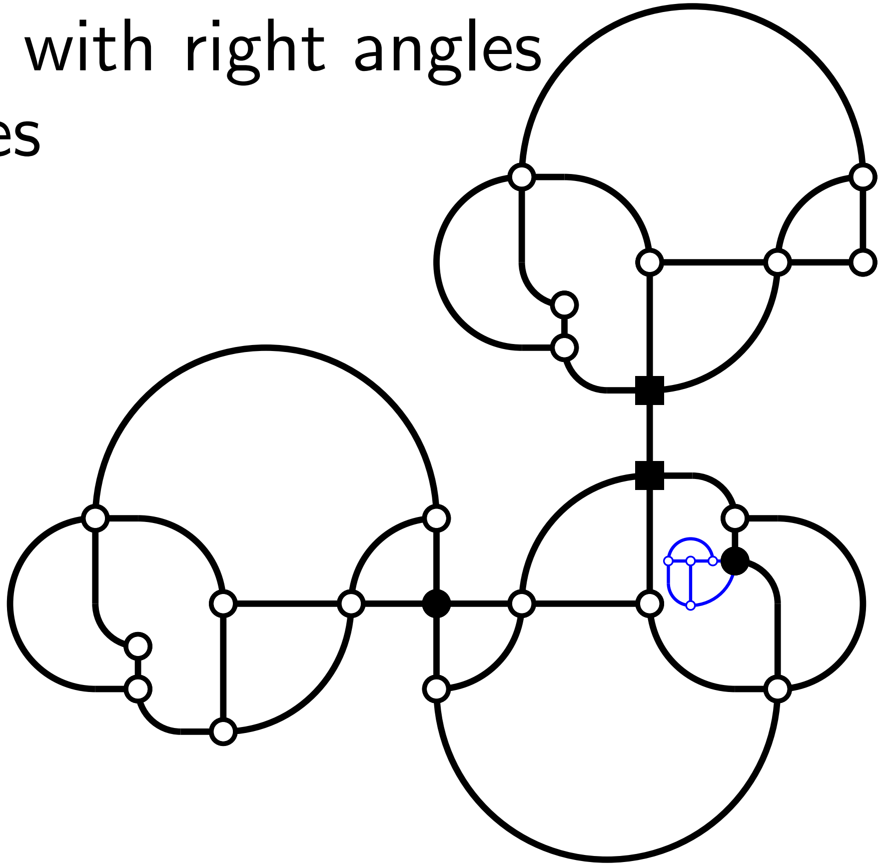
Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces



Extension to Arbitrary Graphs

[Theorem]
Every 4-planar graph admits an SC_2-layout

Draw specific vertex on outer face
Draw cut vertices with right angles
Connect the pieces



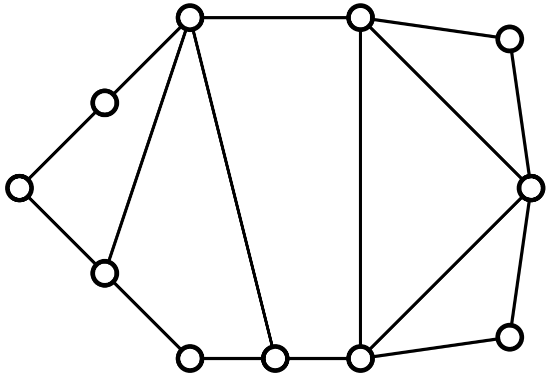
SC_1-Layouts

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

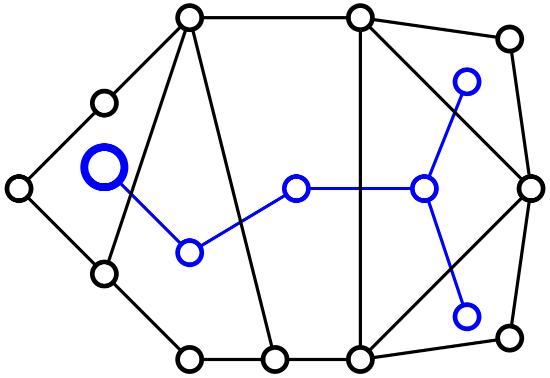
SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout



SC_1-Layouts

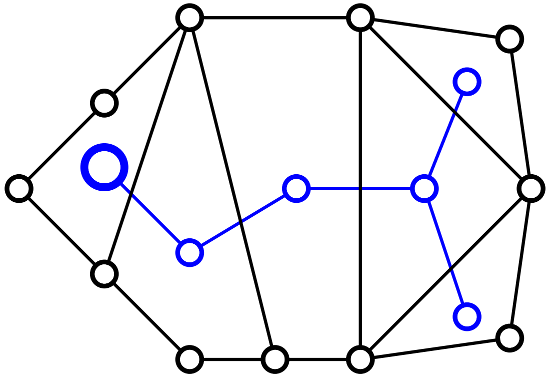
[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout



Consider the dual tree

SC_1-Layouts

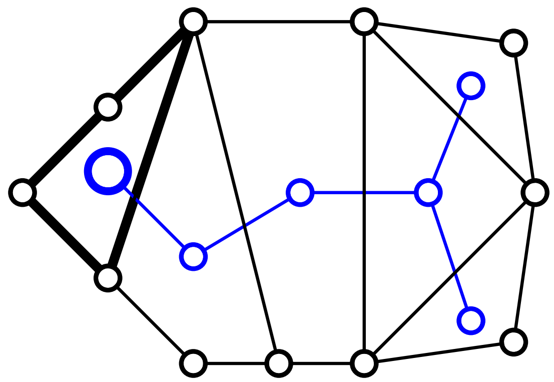
[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

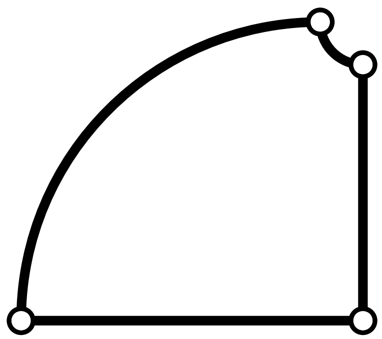


Consider the dual tree
Pick a root, traverse the tree

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout





Consider the dual tree
Pick a root, traverse the tree

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

Consider the dual tree
Pick a root, traverse the tree

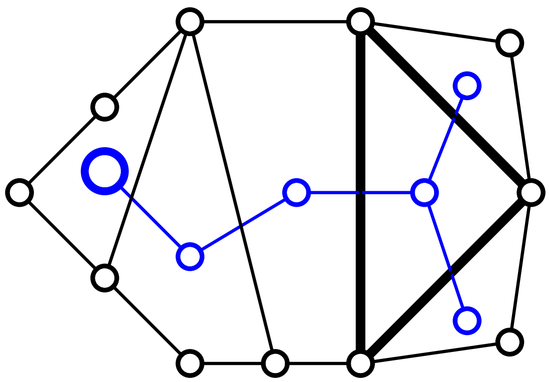
SC_1-Layouts

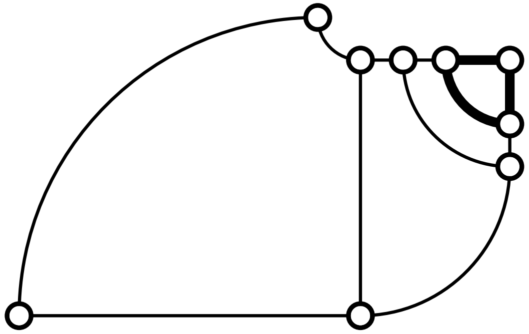
[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

Consider the dual tree
Pick a root, traverse the tree

SC$_1$-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC$_1$-layout

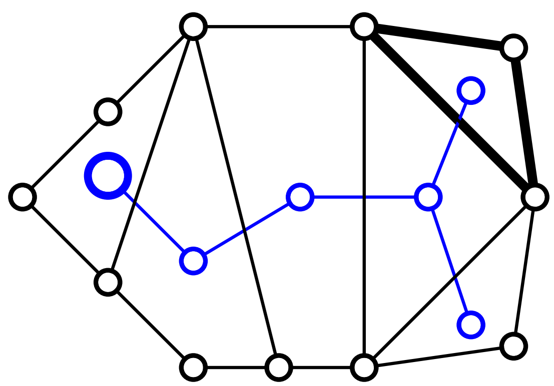


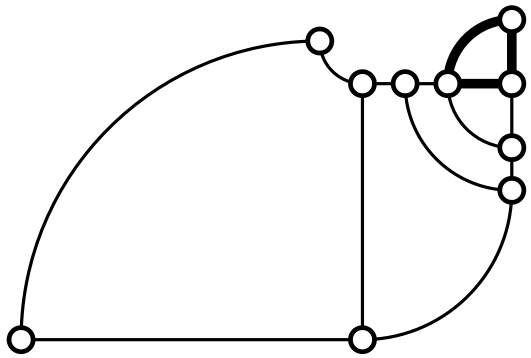


Consider the dual tree
Pick a root, traverse the tree

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

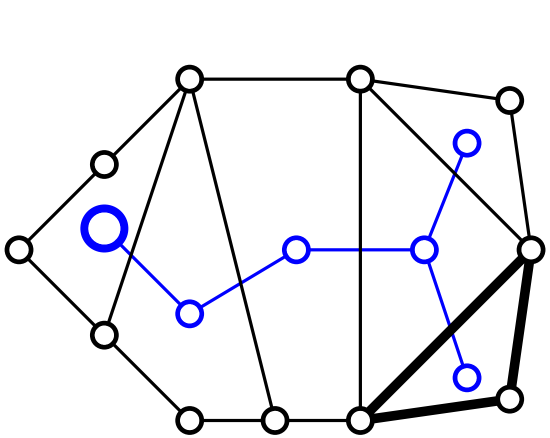


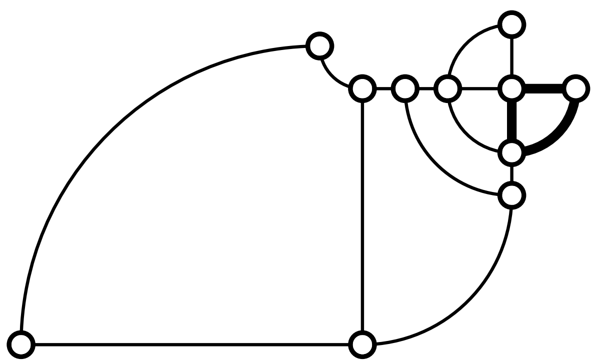


Consider the dual tree
Pick a root, traverse the tree

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

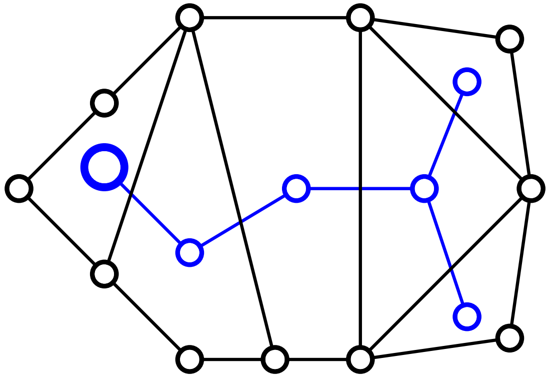


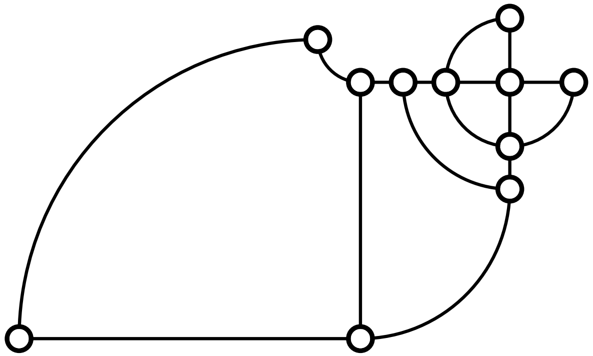


Consider the dual tree
Pick a root, traverse the tree

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout





Consider the dual tree
Pick a root, traverse the tree

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

[Theorem]
Any triconnected 3-planar graph admits an SC_1-layout.

SC_1-Layouts

[Theorem]
Every biconnected 4-outerplanar graph admits an SC_1-layout

[Theorem]
Any triconnected 3-planar graph admits an SC_1-layout.

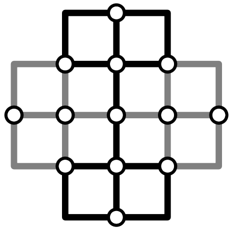
[Theorem]
Any Hamiltonian 3-planar graph admits an SC_1-layout.

Area Requirement of SC_1-Layouts

[Theorem]
There is an infinite class of graphs that require exponential area if they are drawn with SC_1.

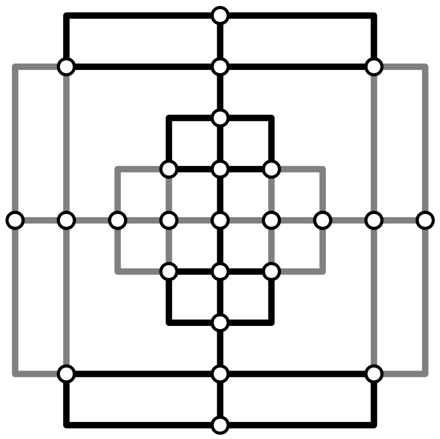
Area Requirement of SC_1-Layouts

[Theorem]
There is an infinite class of graphs that require exponential area if they are drawn with SC_1.



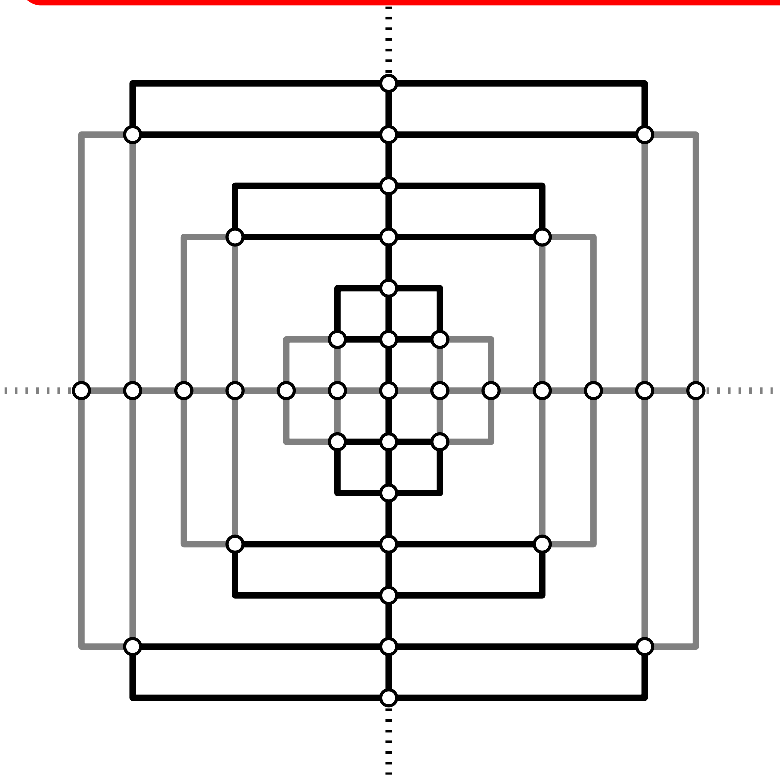
Area Requirement of SC_1-Layouts

[Theorem]
There is an infinite class of graphs that require exponential area if they are drawn with SC_1.



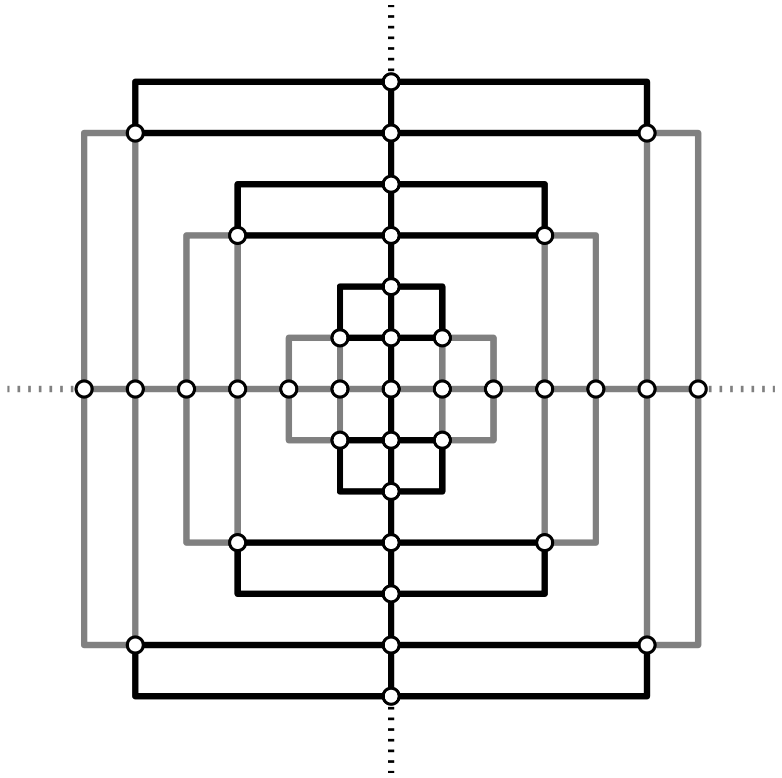
Area Requirement of SC_1-Layouts

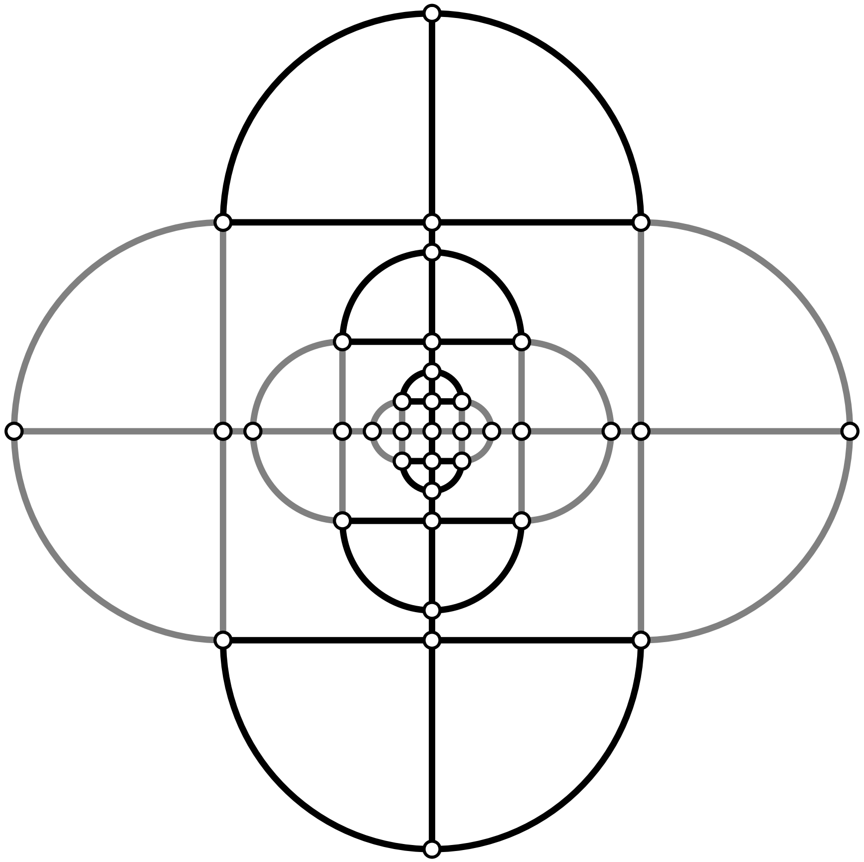
[Theorem]
There is an infinite class of graphs that require exponential area if they are drawn with SC_1.



Area Requirement of SC_1-Layouts

[Theorem]
There is an infinite class of graphs that require exponential area if they are drawn with SC_1.





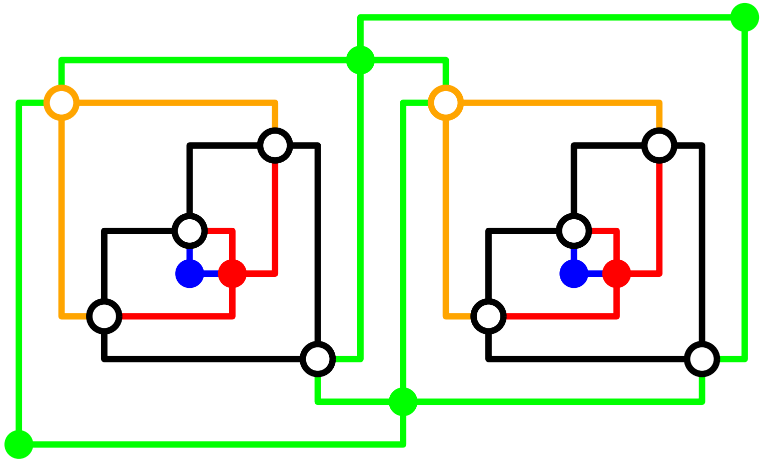
Biconnected Graphs without SC_1-Layout

[Theorem]

There exists a biconnected 4-planar graph that admits an OC_2-layout, but does not admit an SC_1-layout.

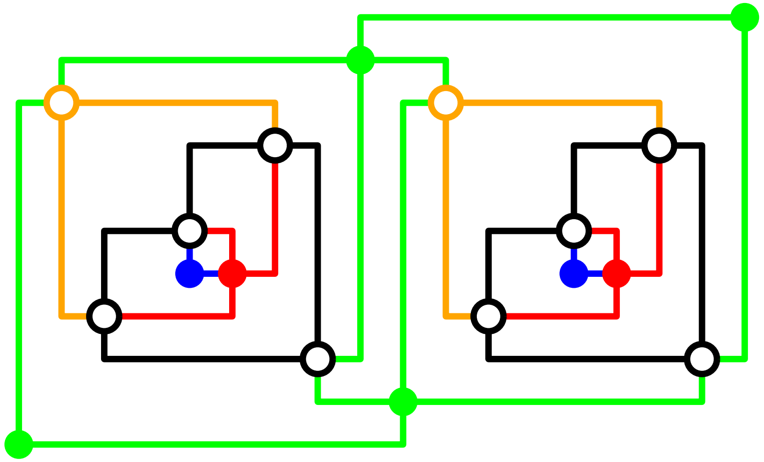
Biconnected Graphs without SC_1-Layout

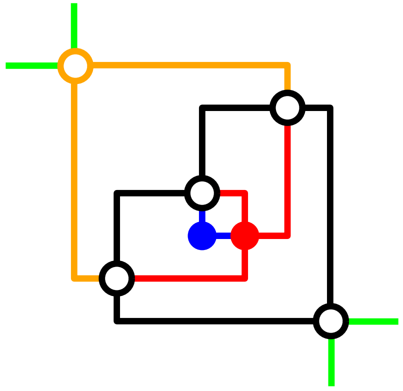
[Theorem]
There exists a biconnected 4-planar graph that admits an OC_2-layout, but does not admit an SC_1-layout.



Biconnected Graphs without SC_1-Layout

[Theorem]
There exists a biconnected 4-planar graph that admits an OC_2-layout, but does not admit an SC_1-layout.





Open Problems

- Do all 4-planar graphs admit an SC_2-layout *in polynomial area*?

Open Problems

- Do all 4-planar graphs admit an SC_2-layout *in polynomial area*?
- Do all 4-outerplanar graphs admit an SC_1-layout?

Open Problems

- Do all 4-planar graphs admit an SC_2-layout *in polynomial area*?

- Do all 4-outerplanar graphs admit an SC_1-layout?

- Do all 3-planar graphs admit an SC_1-layout?

Open Problems

- Do all 4-planar graphs admit an SC_2-layout *in polynomial area*?
- Do all 4-outerplanar graphs admit an SC_1-layout?
- Do all 3-planar graphs admit an SC_1-layout?
- Is it NP-hard to decide whether a 4-planar graph admits an SC_1-layout?