Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Seminar:
Themen der Algorithmik

Wintersemester 2025

Einführungsveranstaltung am 14. Oktober 2025

Lehrstuhl für Informatik I

Alexander Wolff, Boris Klemz, Diana Sieper, Tim Hegemann, Samuel Wolf,
Antonio Lauerbach

Ziele und Inhalte

In diesem Seminar geht es teils um **aktuelle Forschungsthemen** und **neue Trends**, teils **klassische Resultate** aus dem Gebiet **Algorithmik**.

Ziele und Inhalte

In diesem Seminar geht es teils um **aktuelle Forschungsthemen** und **neue Trends**, teils **klassische Resultate** aus dem Gebiet **Algorithmik**.

JedeR TeilnehmerIn arbeitet sich in ein abgegrenztes Thema ein. Dieses ist didaktisch aufzubereiten und den anderen KursteilnehmerInnen in einem **Vortrag** zu vermitteln, sowie in einer **schriftliche Ausarbeitung** darzustellen.

Ablauf des Seminars

- Di, 14.10.2025: **Einführung**

Ablauf des Seminars

- Di, 14.10.2025: **Einführung**
- Di, 21.10.2025: **Kurzvorträge** zu jedem Thema
  (etwa 5 Min., ca. 3 Folien)

Ablauf des Seminars

- Di, 14.10.2025: **Einführung**

- Di, 21.10.2025: **Kurzvorträge** zu jedem Thema
  (etwa 5 Min., ca. 3 Folien)

Inhalte:

- Ausblick auf den eigentlichen Vortrag geben
- Problemstellung nennen & motivieren
- Wichtigste Resultate nennen & einordnen

Ablauf des Seminars

- Di, 14.10.2025: **Einführung**

- Di, 21.10.2025: **Kurzvorträge** zu jedem Thema
  (etwa 5 Min., ca. 3 Folien)

Inhalte:

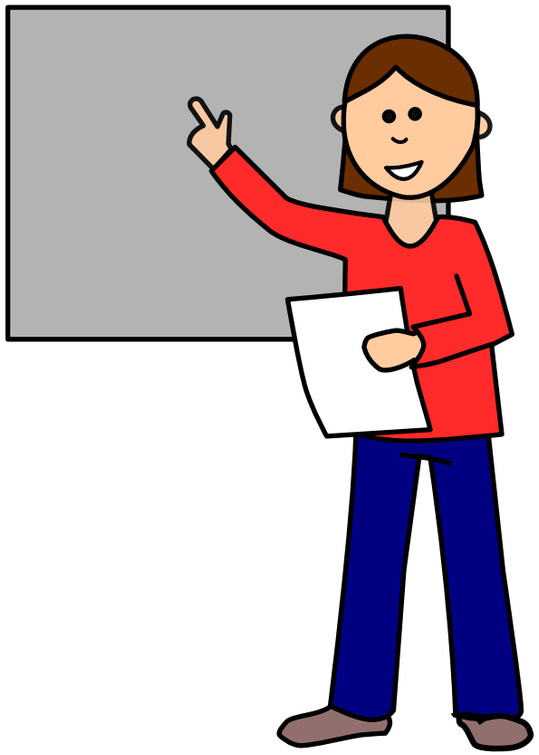
- Ausblick auf den eigentlichen Vortrag geben
- Problemstellung nennen & motivieren
- Wichtigste Resultate nennen & einordnen

Ziele:

- Zeitnah einarbeiten
- Themenauswahl prüfen
- Vortragen üben
- Feedback bekommen ohne Bewertung

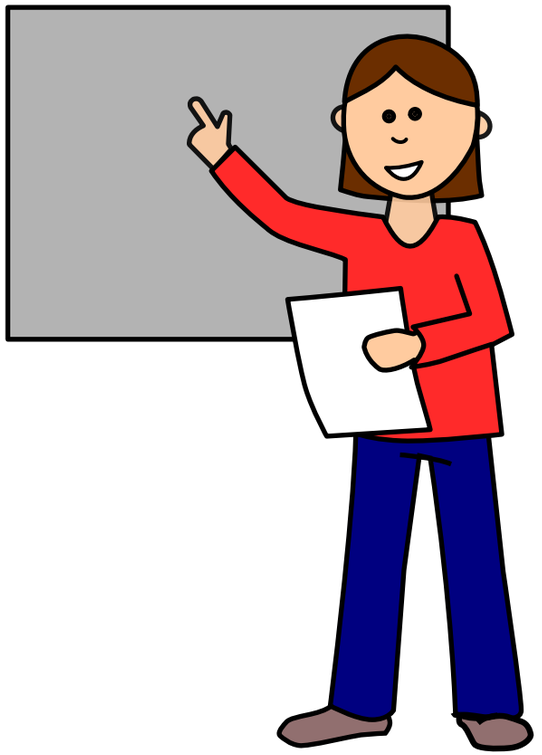
Ablauf des Seminars

- Di, 14.10.2025: **Einführung**
- Di, 21.10.2025: **Kurzvorträge** zu jedem Thema (etwa 5 Min., ca. 3 Folien)
- ab Di, 04.11.2025: **Hauptvorträge** (i.d.R. einer pro Woche)



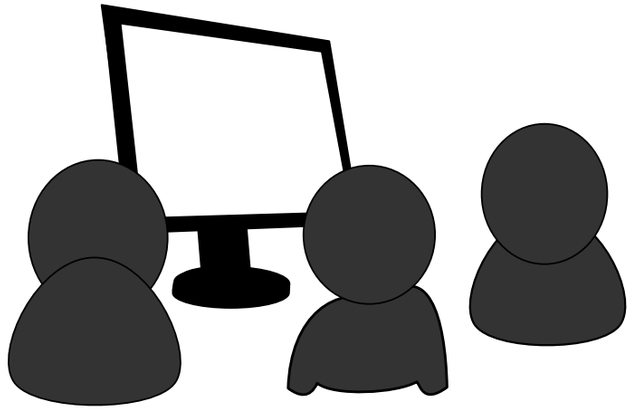
Ablauf des Seminars

- Di, 14.10.2025: **Einführung**
- Di, 21.10.2025: **Kurzvorträge** zu jedem Thema (etwa 5 Min., ca. 3 Folien)
- ab Di, 04.11.2025: **Hauptvorträge** (i.d.R. einer pro Woche)
- Mo, 12.02.2026: **Ausarbeitungen** abgeben



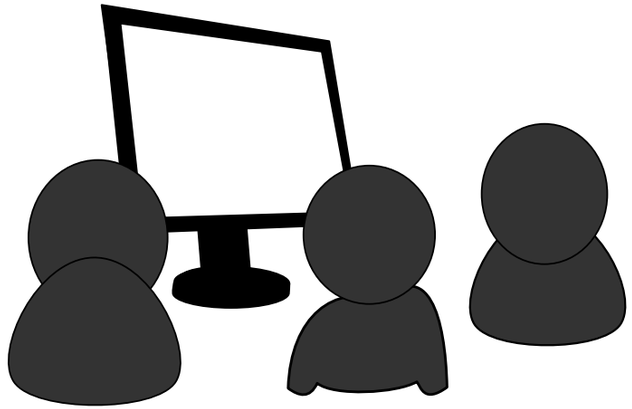
Vorträge

- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)



Vorträge

- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)

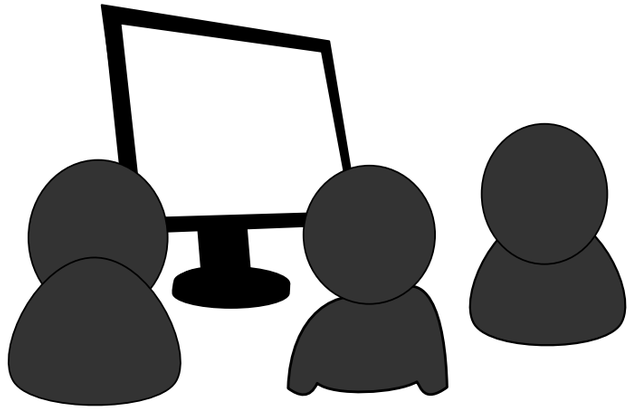


Das reicht i.d.R. nicht um alles im Detail zu besprechen!

$\rightarrow$ wesentliche Teile identifizieren und ausführlich behandeln, unwesentliche Teile skizzieren

Vorträge

- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)



Das reicht i.d.R. nicht um alles im Detail zu besprechen!

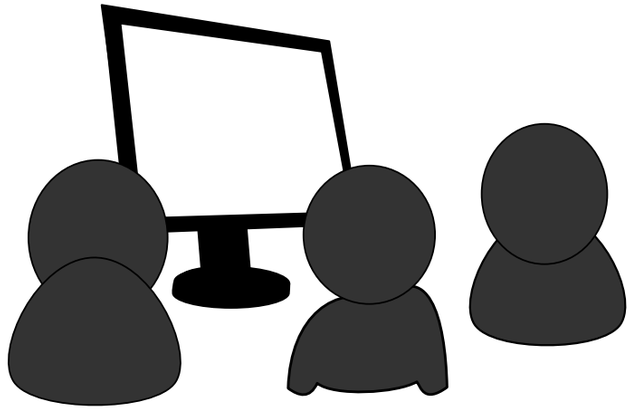
→ wesentliche Teile identifizieren und ausführlich behandeln, unwesentliche Teile skizzieren

Ausnahme: Einige Themen sind weniger umfangreich

→ verbleibende Zeit durch Inhalte angrenzender Literatur füllen (eigene Literaturrecherche!)

Vorträge

- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)



Das reicht i.d.R. nicht um alles im Detail zu besprechen!

→ wesentliche Teile identifizieren und ausführlich behandeln, unwesentliche Teile skizzieren

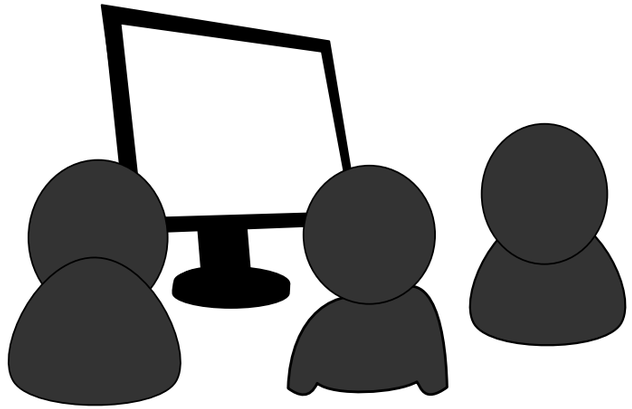
Ausnahme: Einige Themen sind weniger umfangreich

→ verbleibende Zeit durch Inhalte angrenzender Literatur füllen (eigene Literaturrecherche!)

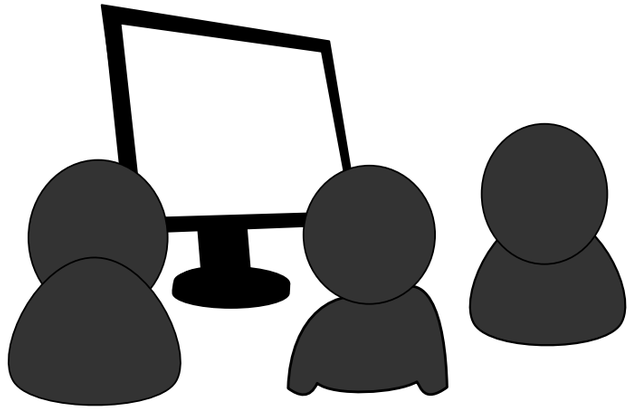
In jedem Fall sollen die 45 / 60 Minuten stimmig ausgefüllt werden.

Vorträge

- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)



Vorträge

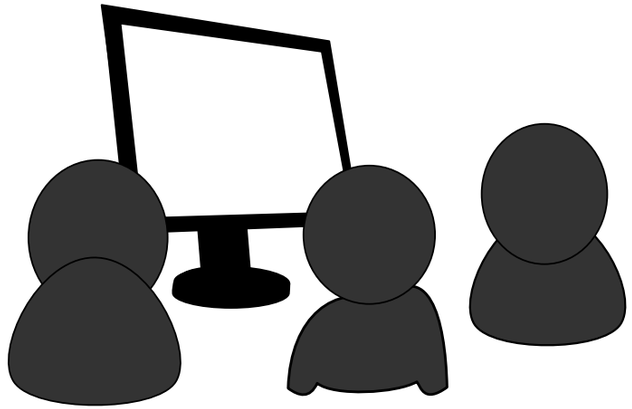


- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)

- anschließend/währenddessen **Diskussion / Interaktion**
  (Übungsaufgaben, interaktive Beispiele, Besprechung offener Probleme, etc.) (geht nicht in die Zeit ein)

  Ideen aus der Diskussion in die Ausarbeitung mitaufnehmen!

Vorträge



- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)

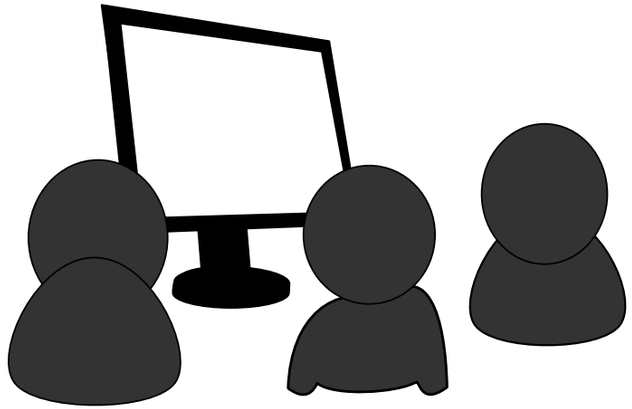
- anschließend/währenddessen **Diskussion / Interaktion**
  (Übungsaufgaben, interaktive Beispiele, Besprechung offener Probleme, etc.) (geht nicht in die Zeit ein)

  Ideen aus der Diskussion in die Ausarbeitung mitaufnehmen!

Vorbesprechungen (verpflichtend):

- **Drei** Wochen vor dem eigenen Vortrag:
  Besprechung der **Inhaltsübersicht** mit eurer BetreuerIn

Vorträge



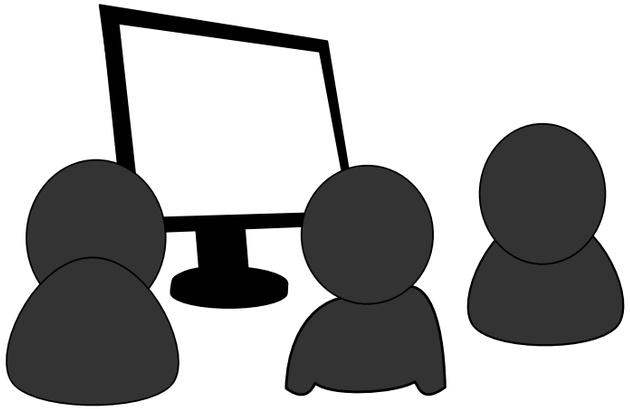
- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)
- anschließend/währenddessen **Diskussion / Interaktion**
  (Übungsaufgaben, interaktive Beispiele, Besprechung offener Probleme, etc.) (geht nicht in die Zeit ein)

  Ideen aus der Diskussion in die Ausarbeitung mitaufnehmen!

Vorbesprechungen (verpflichtend):

- **Drei** Wochen vor dem eigenen Vortrag:
  Besprechung der **Inhaltsübersicht** mit eurer BetreuerIn
- **Zwei** Wochen vor dem Vortrag:
  Besprechung eurer **Folien** mit eurer BetreuerIn

Vorträge



- etwa 45 Minuten **Vortrag**
  (zu zweit etwa 60 Minuten)

- anschließend/währenddessen **Diskussion / Interaktion**
  (Übungsaufgaben, interaktive Beispiele, Besprechung offener Probleme, etc.) (geht nicht in die Zeit ein)

  Ideen aus der Diskussion in die Ausarbeitung mitaufnehmen!

Diese Termine sind **strikt** (außer für den 1. Vortrag)!

Vorbesprechungen (verpflichtend):

- **Drei** Wochen vor dem eigenen Vortrag:
  Besprechung der **Inhaltsübersicht** mit eurer BetreuerIn
- **Zwei** Wochen vor dem Vortrag:
  Besprechung eurer **Folien** mit eurer BetreuerIn

Ausarbeitung

- alleine 7–9, zu zweit 11–13 Seiten;

Ausarbeitung

- alleine 7–9, zu zweit 11–13 Seiten;

Wie schon beim Vortrag gilt auch hier:

Das reicht i.d.R. nicht um alles im Detail zu beschreiben!

$\rightarrow$ wesentliche Teile identifizieren und ausführlich behandeln, unwesentliche Teile skizzieren

Ausarbeitung

- alleine 7–9, zu zweit 11–13 Seiten;

Wie schon beim Vortrag gilt auch hier:

Das reicht i.d.R. nicht um alles im Detail zu beschreiben!

→ wesentliche Teile identifizieren und ausführlich behandeln, unwesentliche Teile skizzieren

Ausnahme: Einige Themen sind weniger umfangreich.

→ durch geeignete eigene Inhalte erweitern

(siehe nächste Folie)

Ausarbeitung

- alleine 7–9, zu zweit 11–13 Seiten;

Wie schon beim Vortrag gilt auch hier:

Das reicht i.d.R. nicht um alles im Detail zu beschreiben!

→ wesentliche Teile identifizieren und ausführlich behandeln, unwesentliche Teile skizzieren

Ausnahme: Einige Themen sind weniger umfangreich.

→ durch geeignete eigene Inhalte erweitern

(siehe nächste Folie)

In jedem Fall sollen die 7–9 / 11–13 Seiten stimmig ausgefüllt werden.

Ausarbeitung

- alleine 7–9, zu zweit 11–13 Seiten;
  Abbildungen sind hilfreich!

Ausarbeitung

- alleine 7–9, zu zweit 11–13 Seiten;
  Abbildungen sind hilfreich! (und gehen nicht in das Seitenlimit ein)

Ausarbeitung

Bitte Vektorgrafiken, keine Bitmaps!

- alleine 7–9, zu zweit 11–13 Seiten;
  Abbildungen sind hilfreich! (und gehen nicht in das Seitenlimit ein)

Ausarbeitung

Bitte Vektorgrafiken, keine Bitmaps!

- alleine 7–9, zu zweit 11–13 Seiten;
  Abbildungen sind hilfreich! (und gehen nicht in das Seitenlimit ein)
- **keine reine Zusammenfassung** des Artikels; wir erwarten einen **eigenen Beitrag**. Z.B. manche Resultate weglassen, andere Beweise ausführlicher, offene Probleme diskutieren, eigene Literaturrecherche & Material aus angrenzender Literatur, Verbindungen zu anderen Vortragsthemen etc.

Ausarbeitung

Bitte Vektorgrafiken, keine Bitmaps!

- alleine 7–9, zu zweit 11–13 Seiten;
  Abbildungen sind hilfreich! (und gehen nicht in das Seitenlimit ein)
- **keine reine Zusammenfassung** des Artikels; wir erwarten einen **eigenen Beitrag**. Z.B. manche Resultate weglassen, andere Beweise ausführlicher, offene Probleme diskutieren, eigene Literaturrecherche & Material aus angrenzender Literatur, Verbindungen zu anderen Vortragsthemen etc.
- LaTeX-Vorlage auf der WueCampus Seite!

Ausarbeitung

Bitte Vektorgrafiken, keine Bitmaps!

- alleine 7–9, zu zweit 11–13 Seiten;
  Abbildungen sind hilfreich! (und gehen nicht in das Seitenlimit ein)
- **keine reine Zusammenfassung** des Artikels; wir erwarten einen **eigenen Beitrag**. Z.B. manche Resultate weglassen, andere Beweise ausführlicher, offene Probleme diskutieren, eigene Literaturrecherche & Material aus angrenzender Literatur, Verbindungen zu anderen Vortragsthemen etc.
- LaTeX-Vorlage auf der WueCampus Seite!
- **Vorabversion** der Ausarbeitung bis spätestens 2 Wochen nach dem eigenen Vortrag abgeben, um Feedback zu erhalten (freiwillig).

Bestehen & Bewertung

Voraussetzungen für das Bestehen des Seminars

- Halten einer Präsentation zum gewählten Thema
- Anfertigen einer Ausarbeitung
- Teilnahme an den Diskussionen

Bestehen & Bewertung

Voraussetzungen für das Bestehen des Seminars

- Halten einer Präsentation zum gewählten Thema
- Anfertigen einer Ausarbeitung
- Teilnahme an den Diskussionen

Bewertung

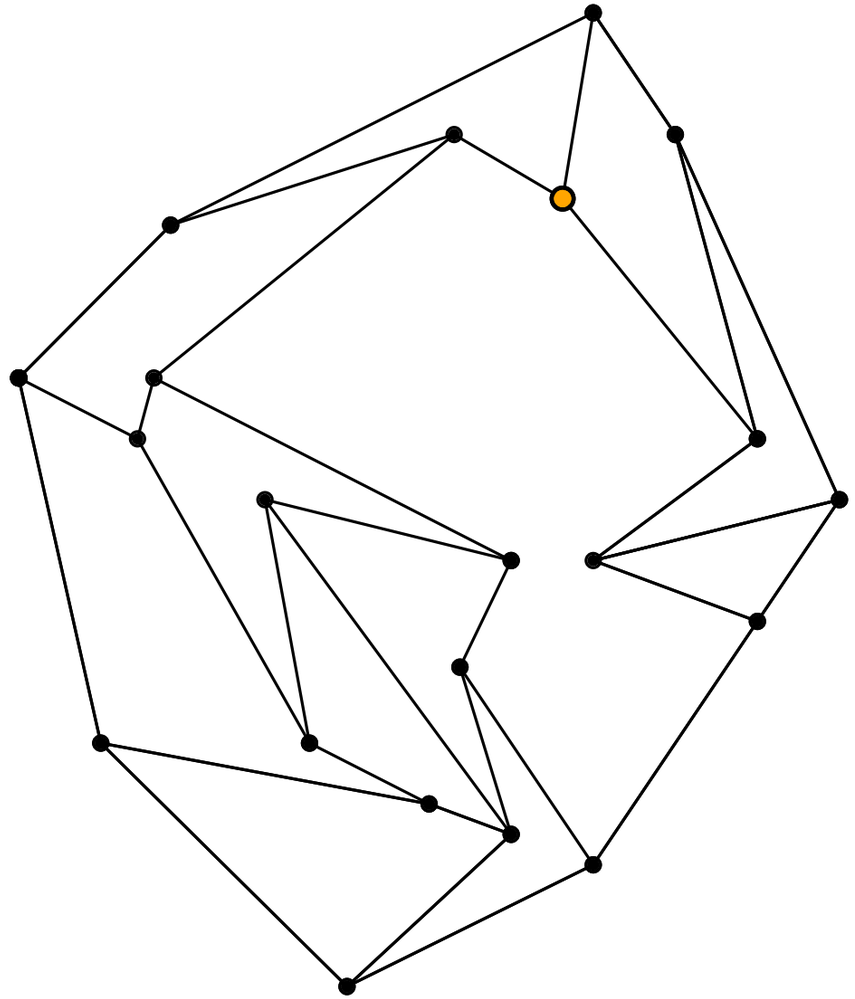
- Vortrag (Inhalte, Gestaltung der Folien, Verständlichkeit, Interaktivität)
- Ausarbeitung (Inhalte, roter Faden, sprachliche Darstellung, Rechtschreibung, eigener Beitrag)
- 50 : 50

Themenübersicht

1. How to Morph Planar Graph Drawings
2. Sliding Squares in Parallel
3. Geometric Spanners of Bounded Tree-width
4. Kuratowski's Theorem
5. Obtaining Kernels with Linear Programming
6. Bidimensonality
7. Structural Parameterization of k-Planarity
8. Universally Optimal Dijkstra's
9. Grid-Drawings of Graphs in 3D
10. Heuristics for Exact 1-Planarity
11. A Shape-First Methodology for Orthogonal Drawings

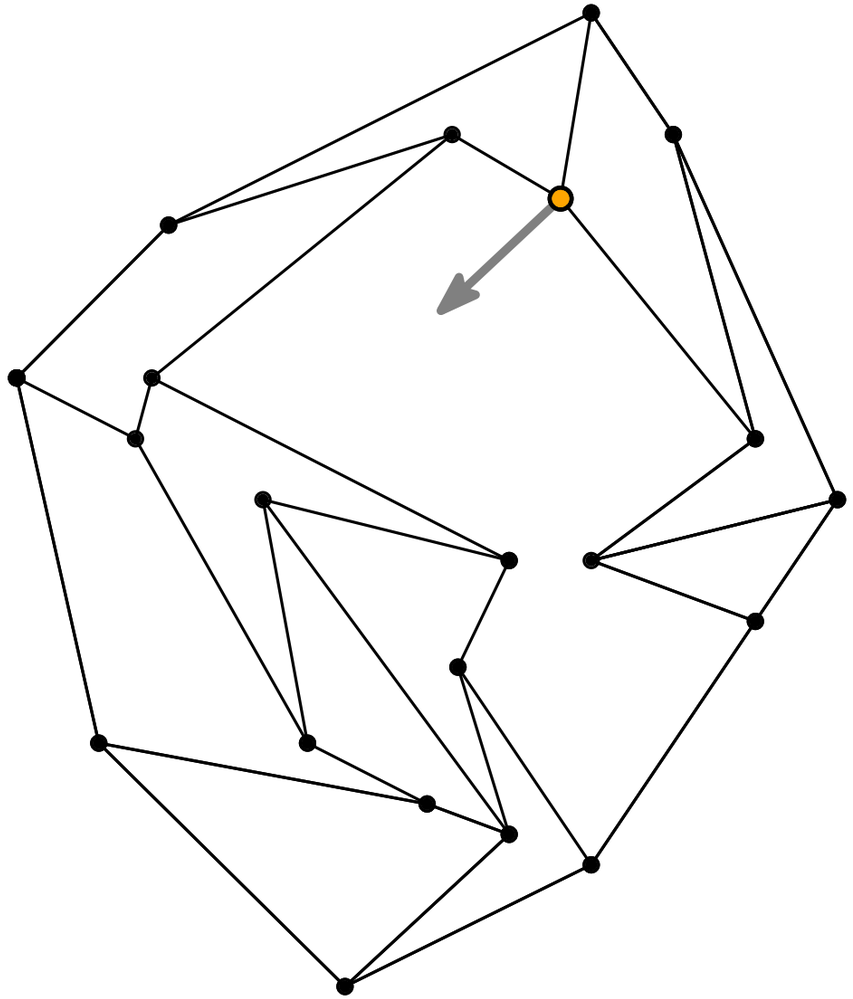
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



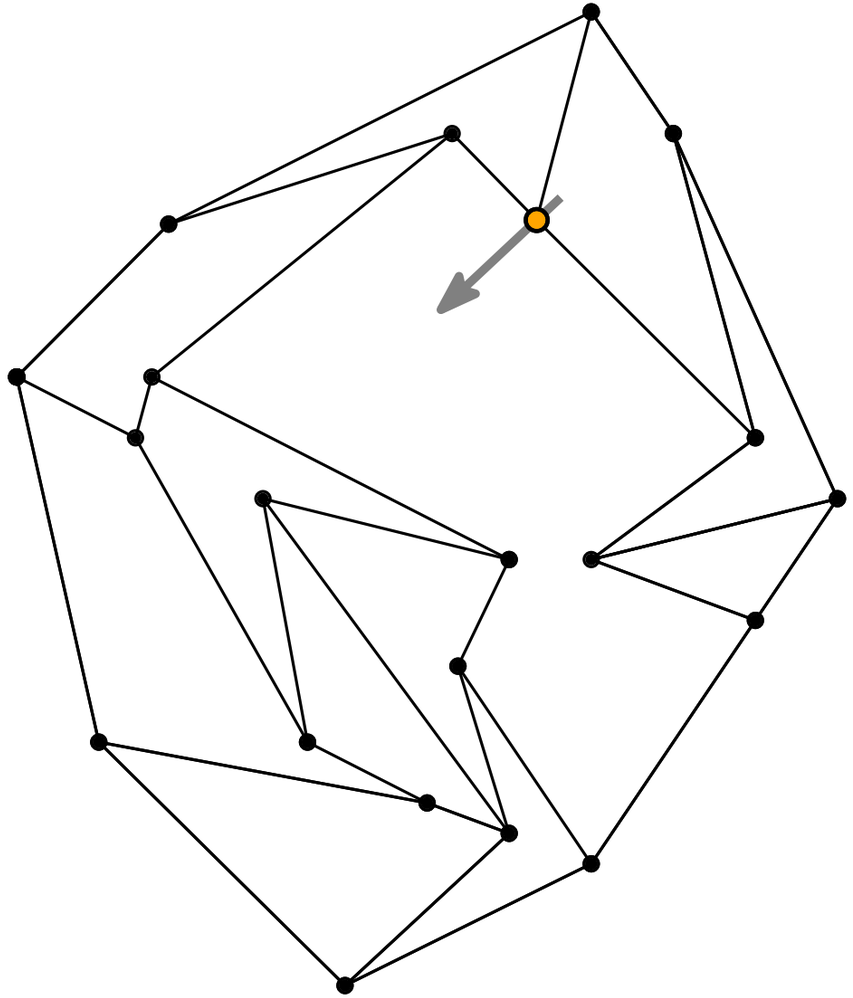
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



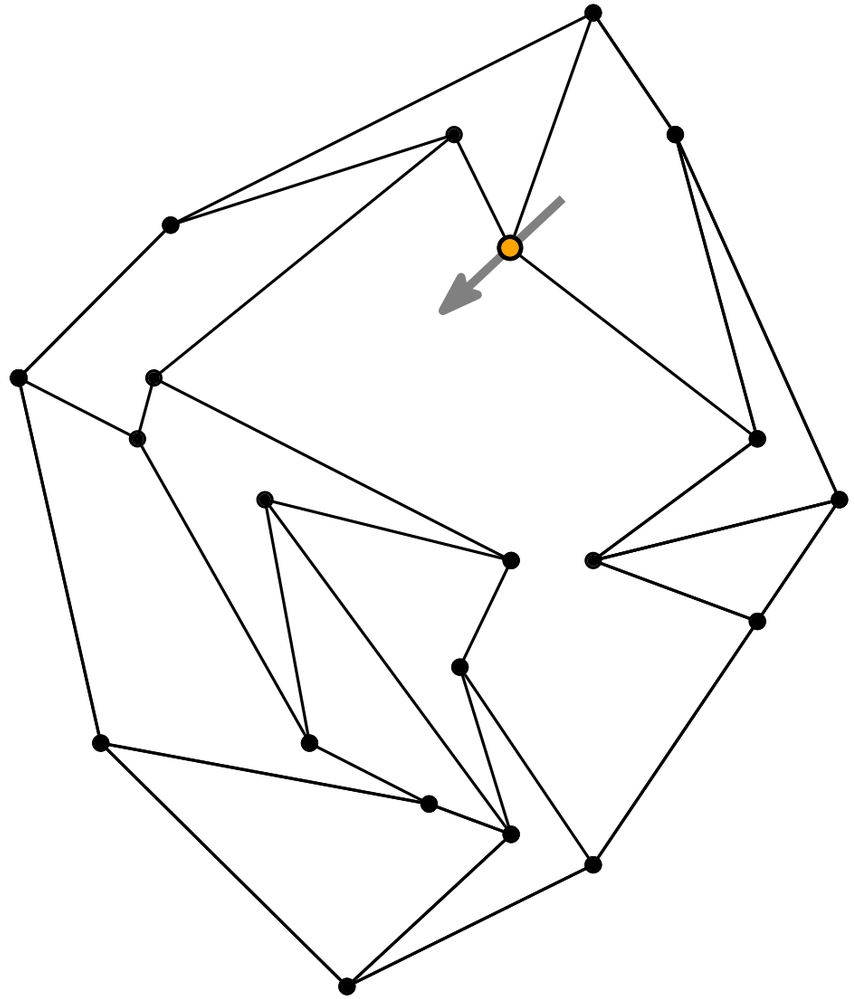
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



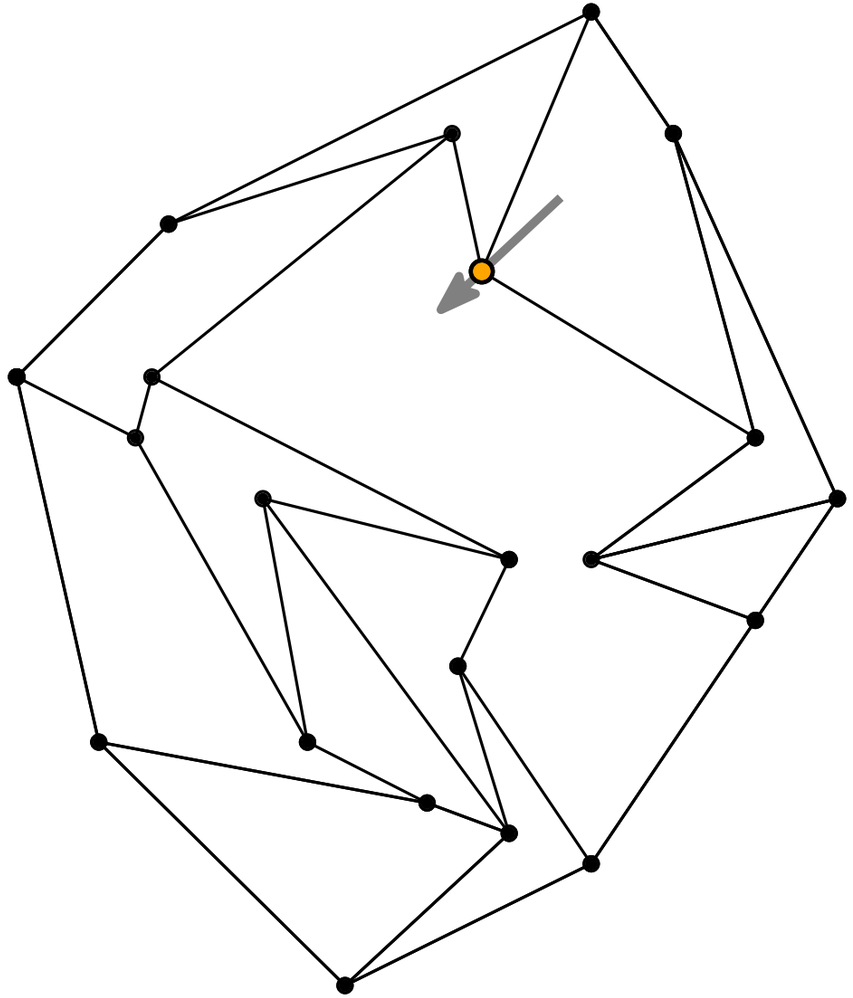
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



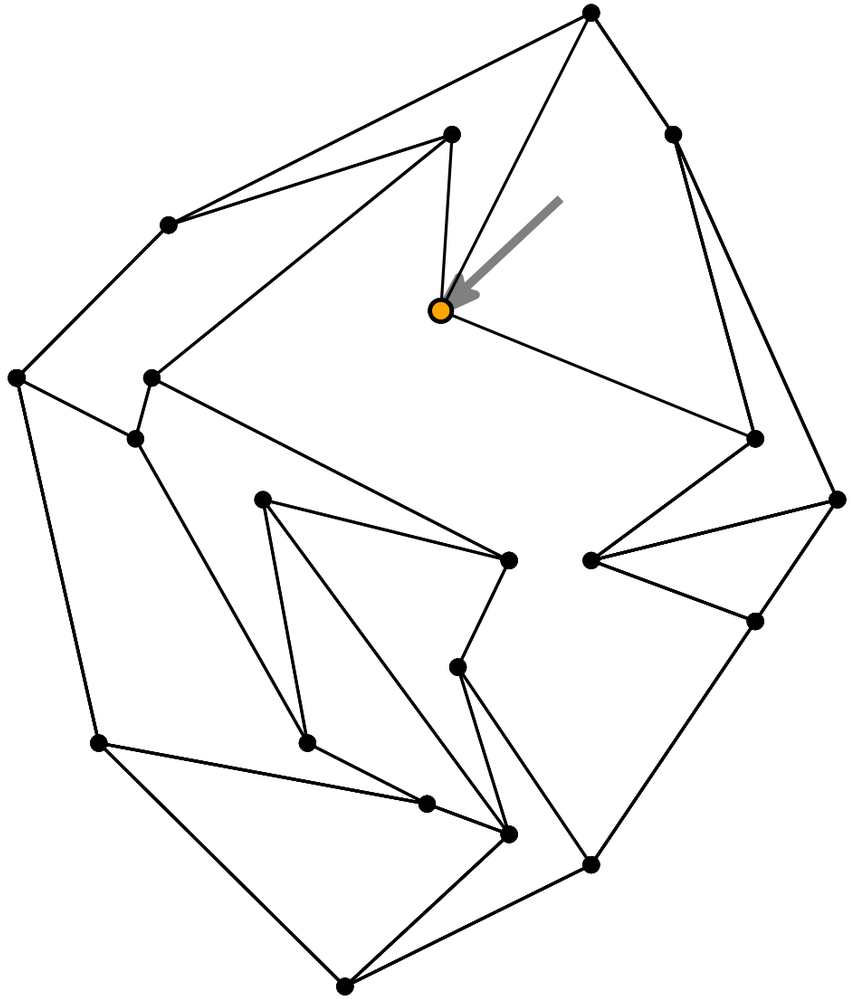
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



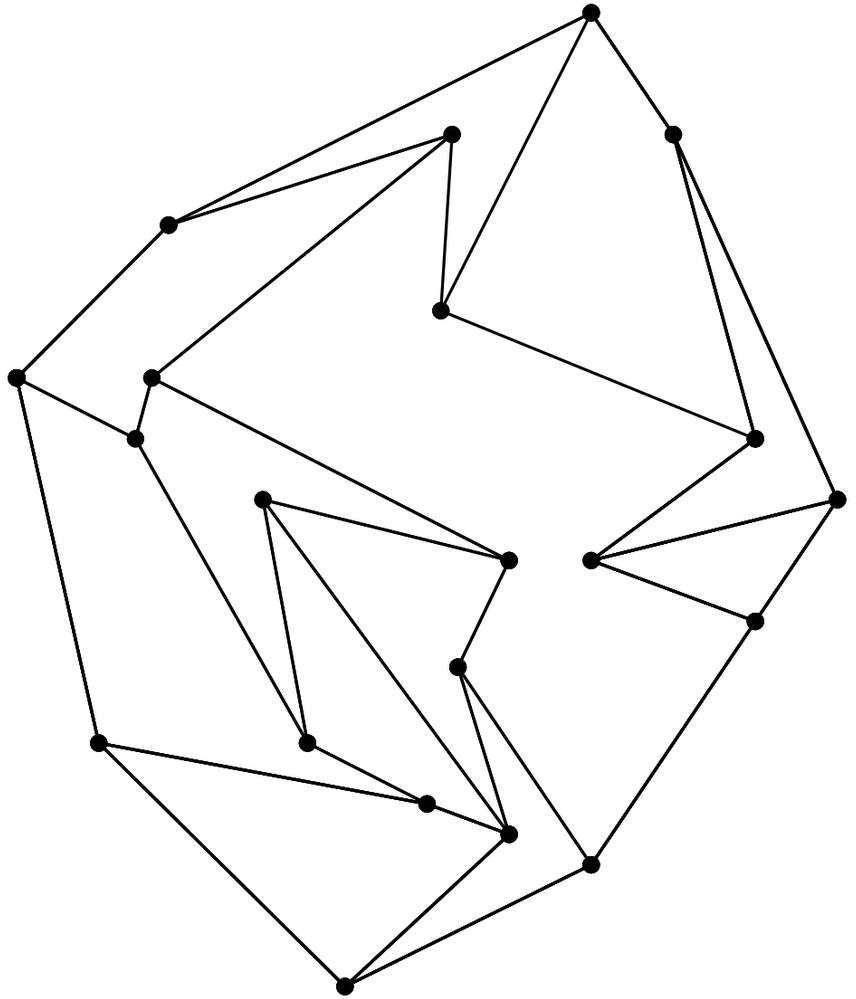
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



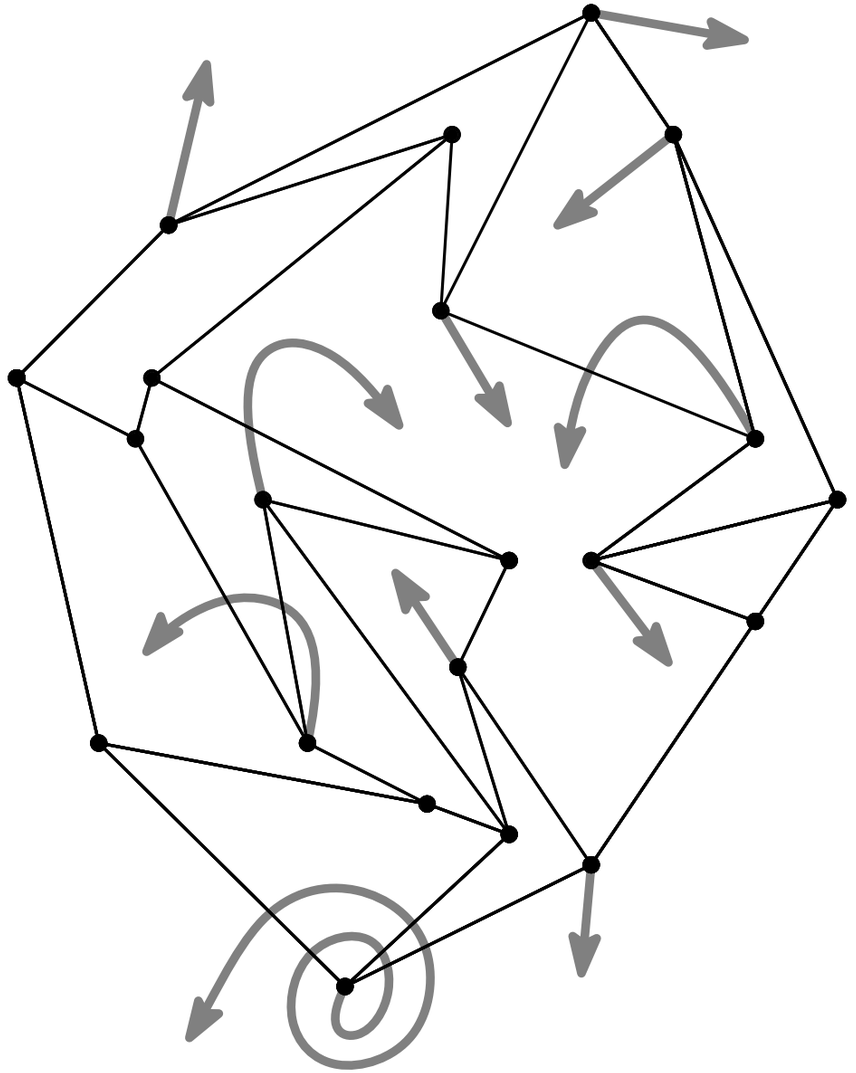
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



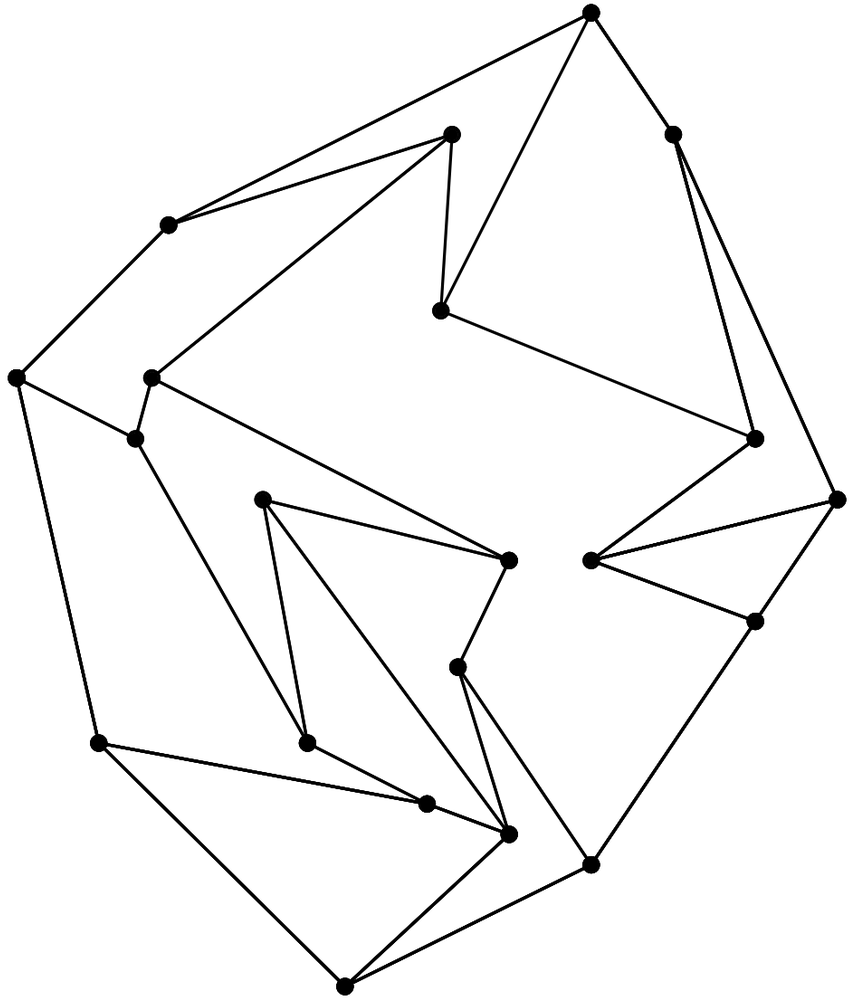
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



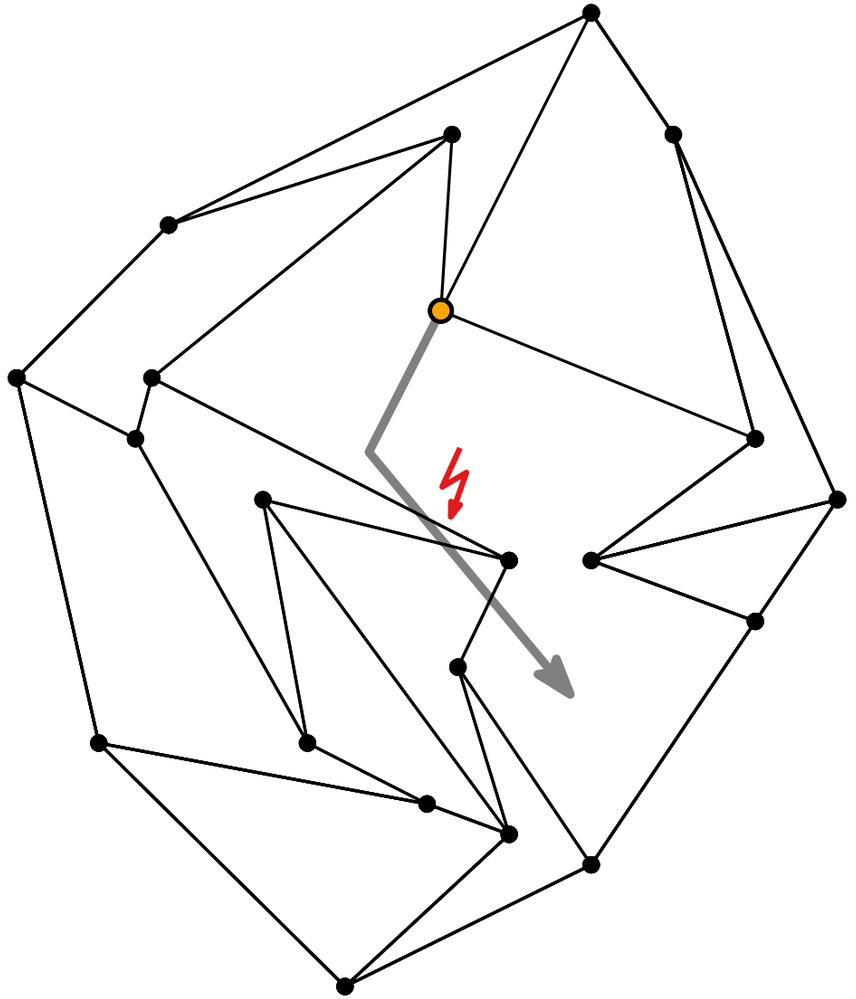
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



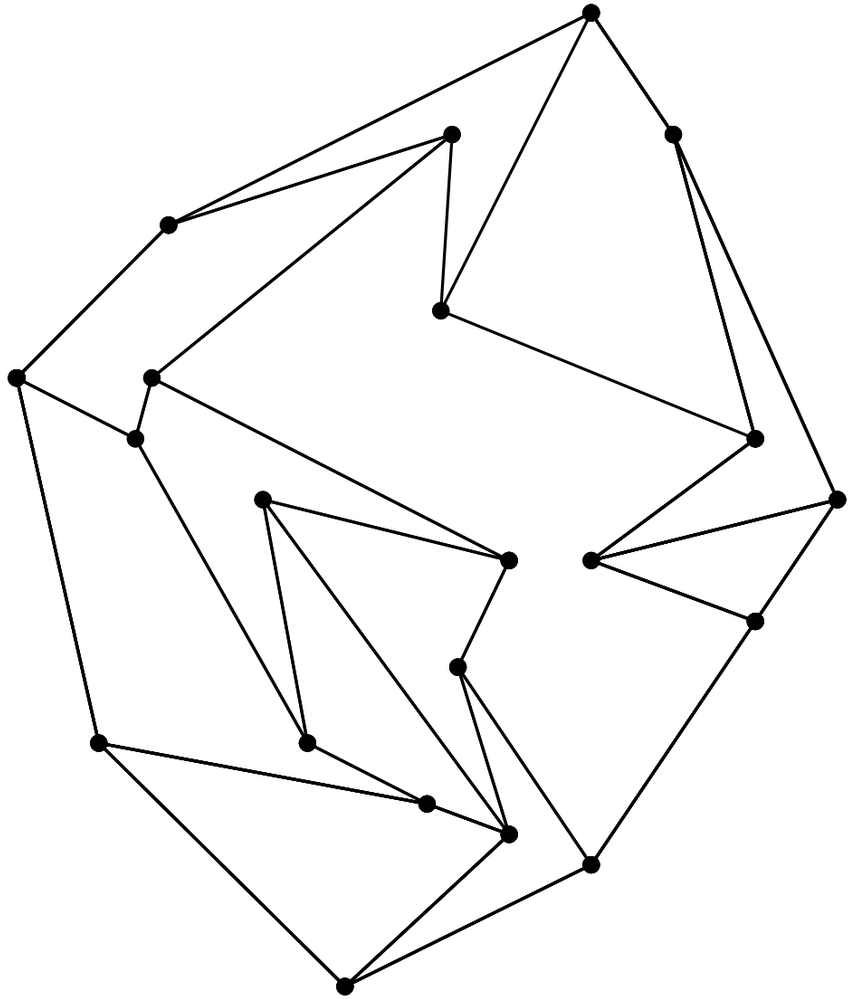
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



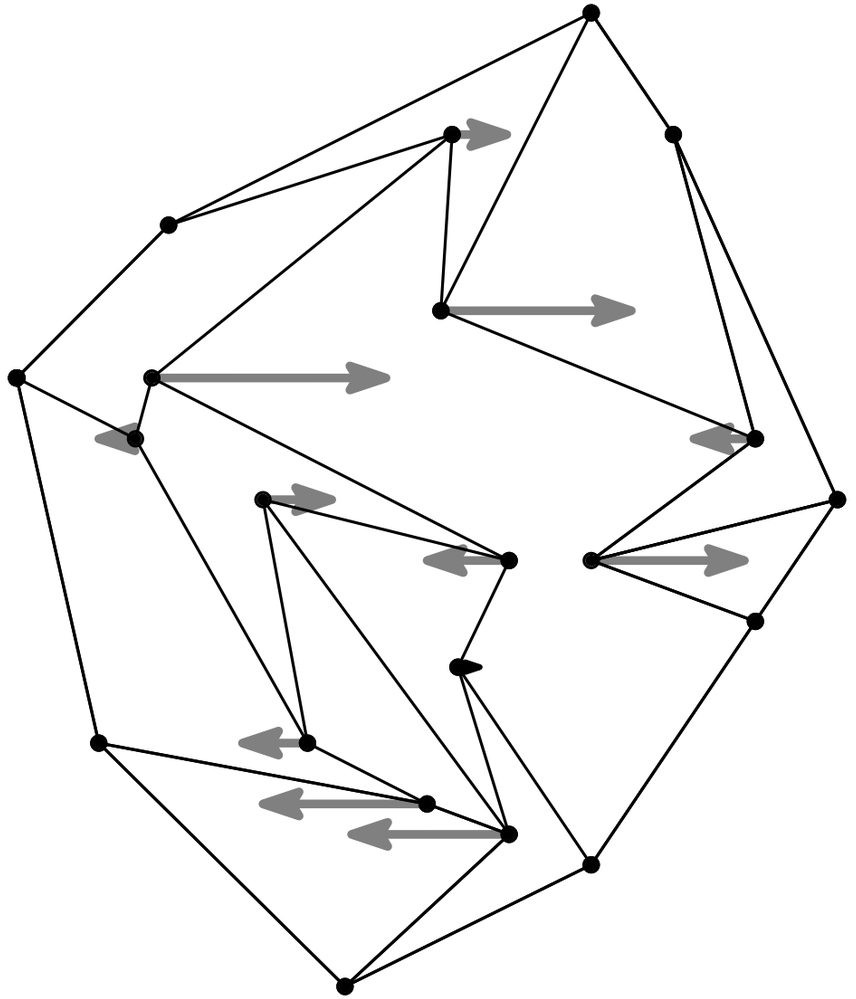
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



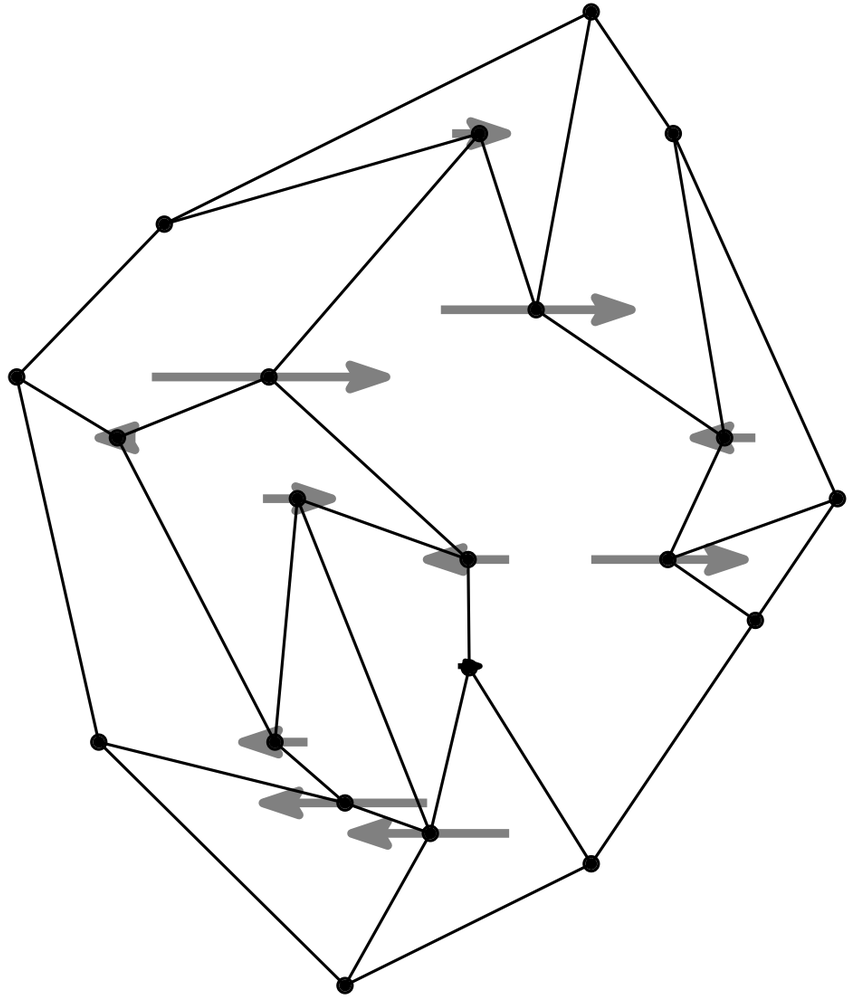
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



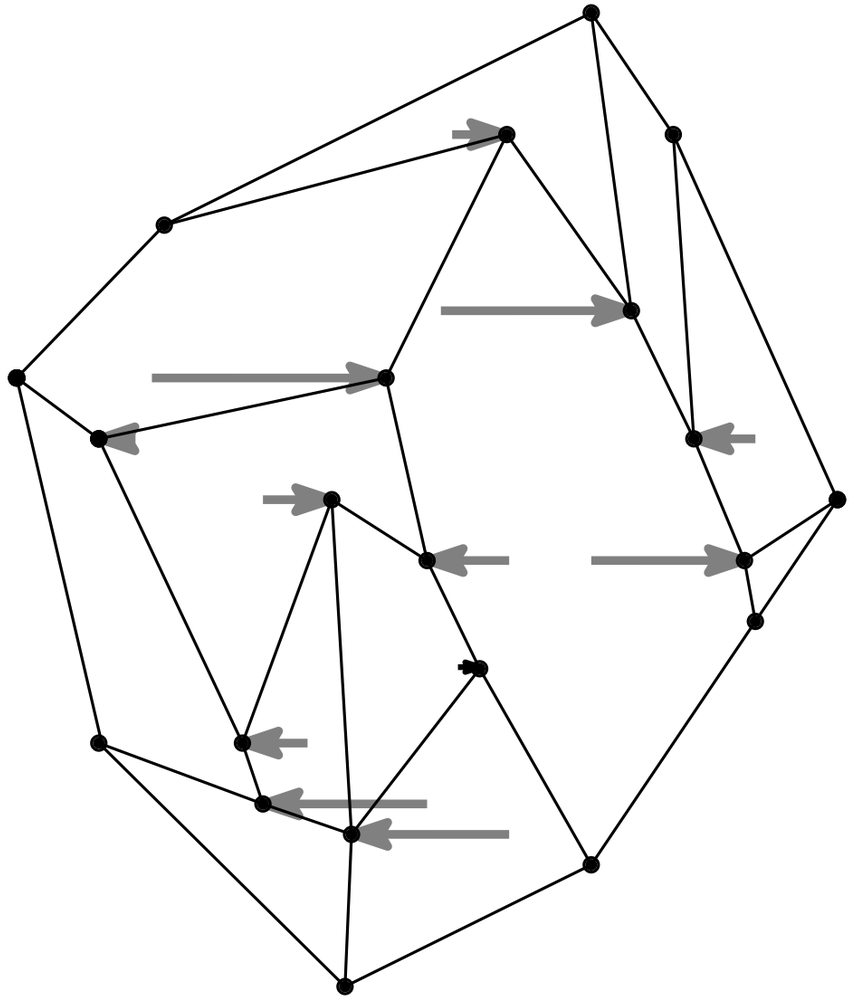
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



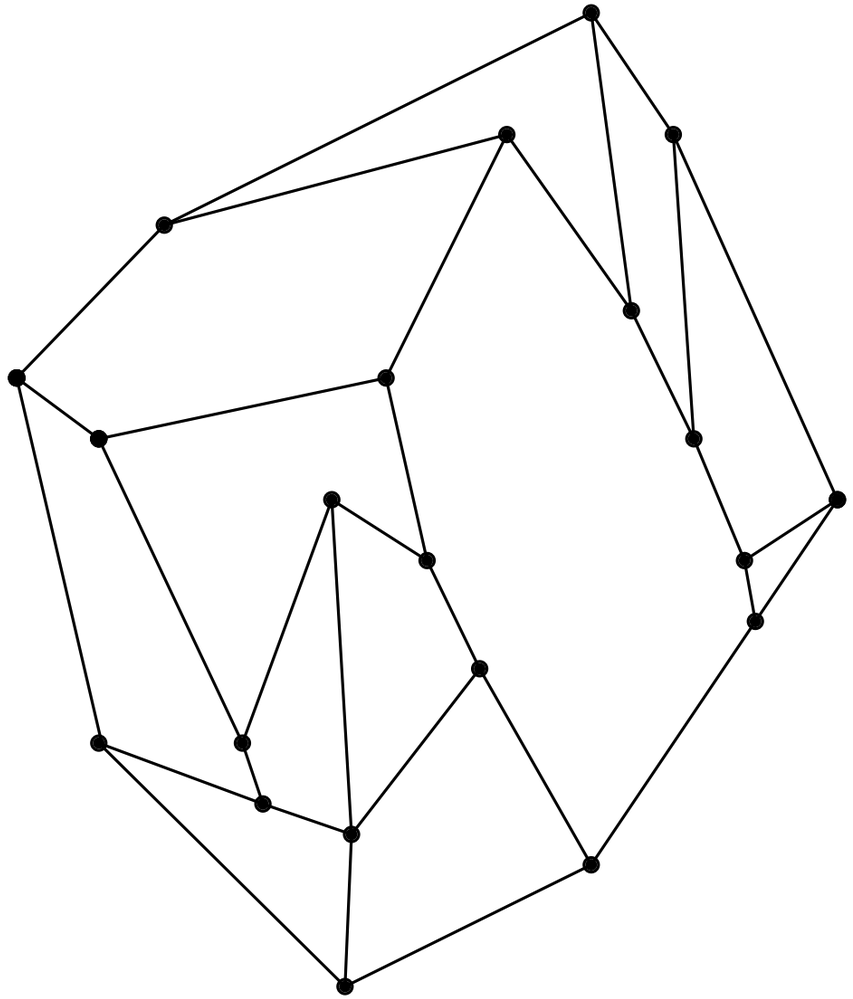
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



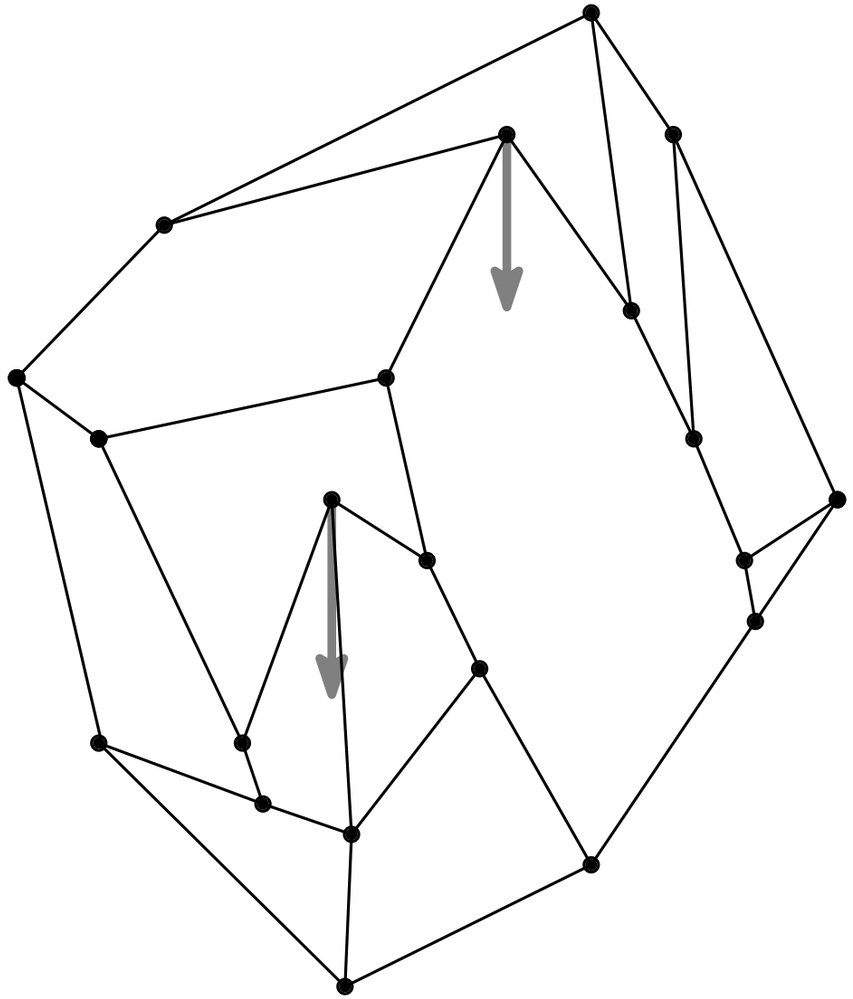
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



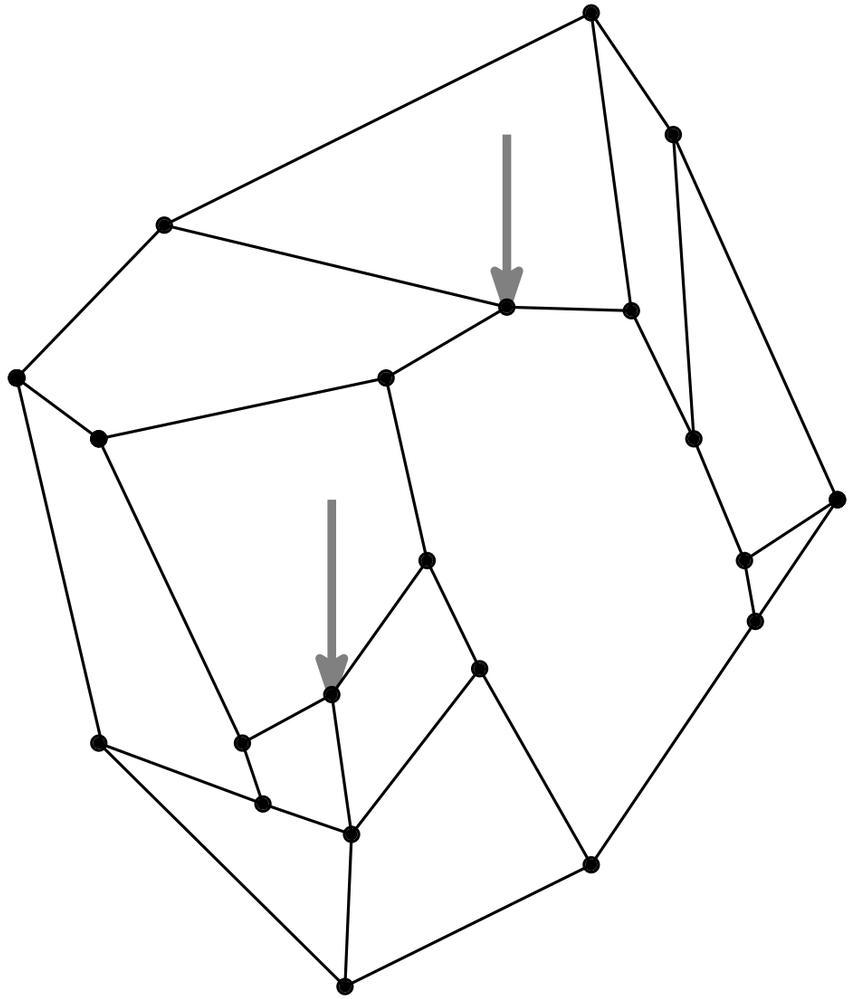
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



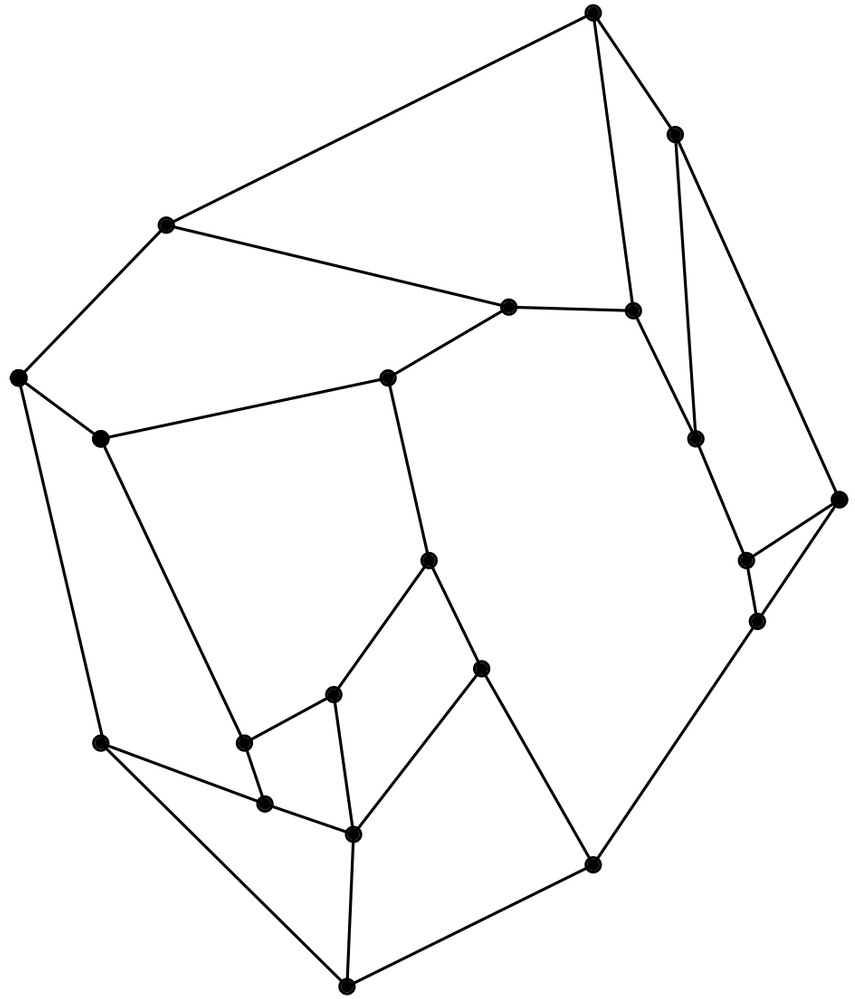
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



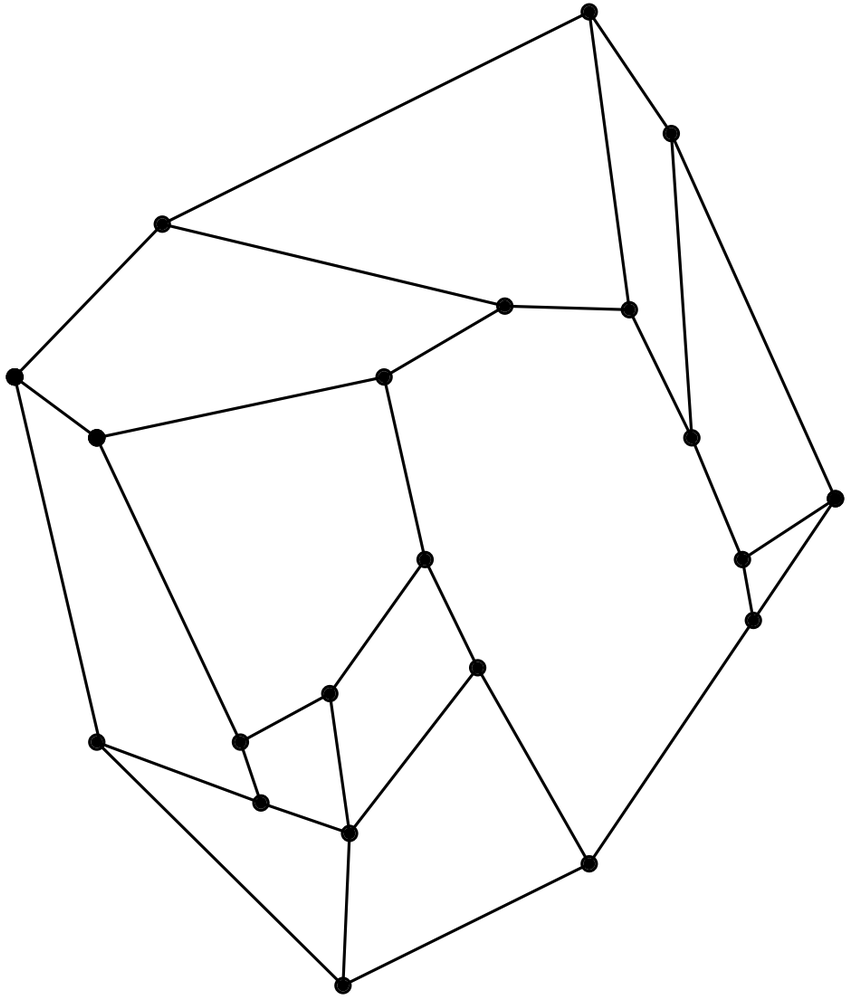
1. How to Morph Planar Graph Drawings

Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



1. How to Morph Planar Graph Drawings

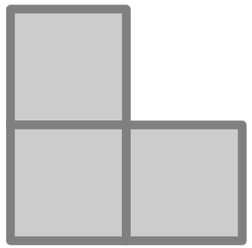
Morph: continuous deformation of a graph drawing that preserves straight-line crossing-free edges.



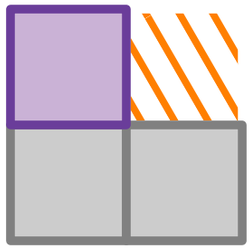
Questions:

- Does there always exist a morph between two crossing-free drawings of the same graph?
- How can it be computed and encoded?

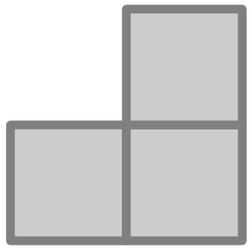
2. Sliding Squares in Parallel



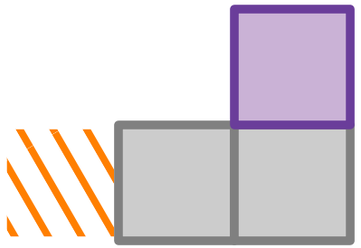
2. Sliding Squares in Parallel



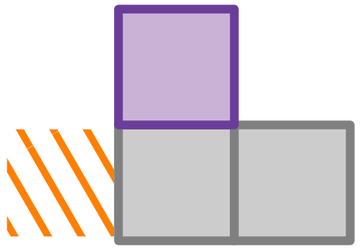
2. Sliding Squares in Parallel



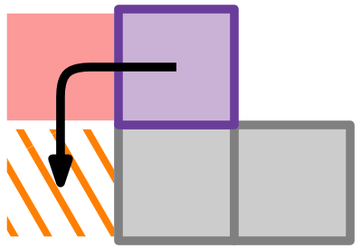
2. Sliding Squares in Parallel



2. Sliding Squares in Parallel

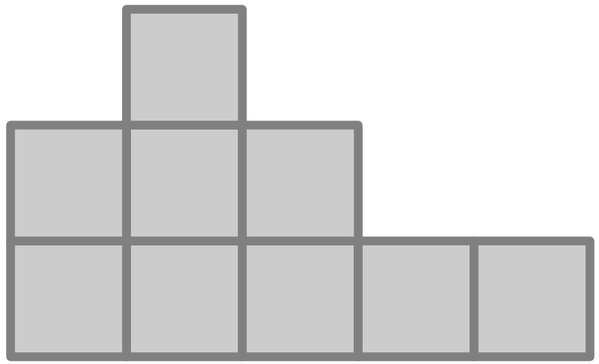


2. Sliding Squares in Parallel

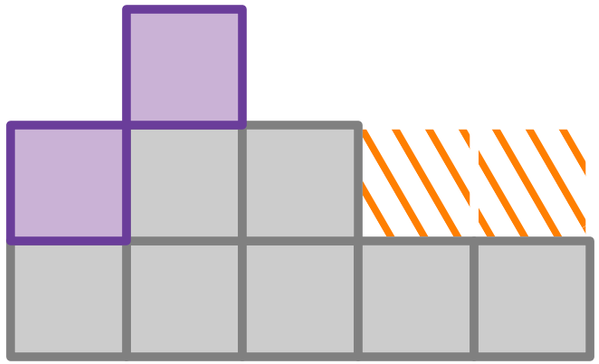


2. Sliding Squares in Parallel

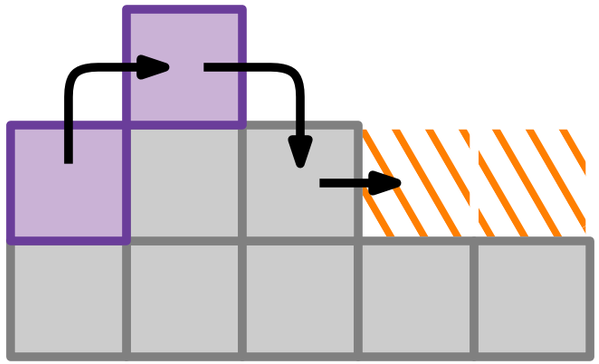
2. Sliding Squares in Parallel



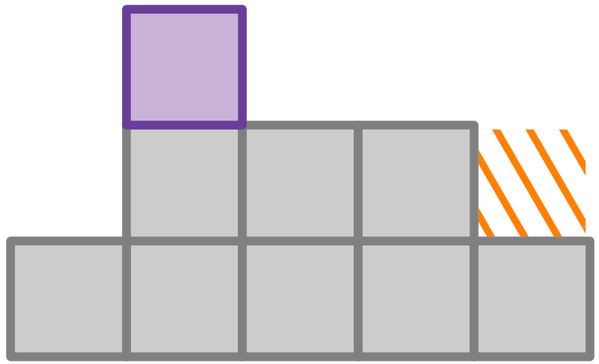
2. Sliding Squares in Parallel



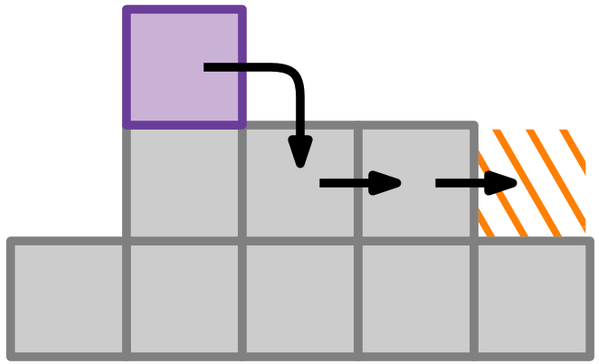
2. Sliding Squares in Parallel



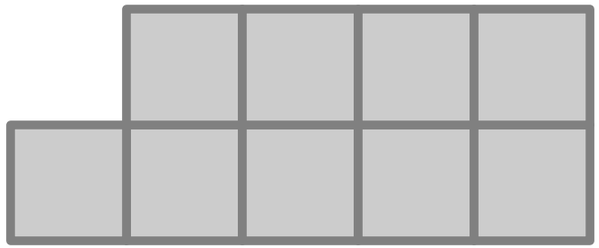
2. Sliding Squares in Parallel



2. Sliding Squares in Parallel



2. Sliding Squares in Parallel



3. Geometric Spanners of Bounded Tree-width

- Tree-width: Measure for how tree-like a graph is.

3. Geometric Spanners of Bounded Tree-width

- Tree-width: Measure for how tree-like a graph is.
  Trees have tree-width 1; the complete graph K_n has tree-width $n - 1$.

3. Geometric Spanners of Bounded Tree-width

- Tree-width: Measure for how tree-like a graph is.
  Trees have tree-width 1; the complete graph K_n has tree-width $n-1$.

- Given a set P of points in $\mathbb{R}^d$, a geometric t-spanner graph for P is a graph G with vertex set P and, for every two points u and v in P,
  $d_G(u,v) \leq t \cdot \|u-v\|$.

3. Geometric Spanners of Bounded Tree-width

- Tree-width: Measure for how tree-like a graph is.
  Trees have tree-width 1; the complete graph K_n has tree-width $n-1$.

- Given a set P of points in $\mathbb{R}^d$, a geometric t-spanner graph for P is a graph G with vertex set P and, for every two points u and v in P,
  $d_G(u,v) \leq t \cdot \|u-v\|$. The number t is called the *dilation* of G.

3. Geometric Spanners of Bounded Tree-width

- Tree-width: Measure for how tree-like a graph is.
  Trees have tree-width 1; the complete graph K_n has tree-width $n-1$.

- Given a set P of points in $\mathbb{R}^d$, a geometric t-spanner graph for P is a graph G with vertex set P and, for every two points u and v in P,
  $d_G(u,v) \leq t \cdot \|u-v\|$. The number t is called the *dilation* of G.

- The authors show how to compute, for any fixed dimension d, a t-spanner with tree-width k and $t \in O(n/k^{d(d-1)})$.

3. Geometric Spanners of Bounded Tree-width

- Tree-width: Measure for how tree-like a graph is.
  Trees have tree-width 1; the complete graph K_n has tree-width $n-1$.

- Given a set P of points in $\mathbb{R}^d$, a geometric t-spanner graph for P is a graph G with vertex set P and, for every two points u and v in P,
  $d_G(u,v) \leq t \cdot \|u-v\|$. The number t is called the *dilation* of G.

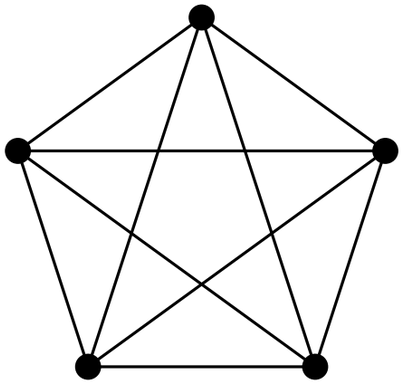
- The authors show how to compute, for any fixed dimension d, a t-spanner with tree-width k and $t \in O(n/k^{d(d-1)})$.
  The result is asymptotically worst-case optimal.

4. Kuratowski's Theorem

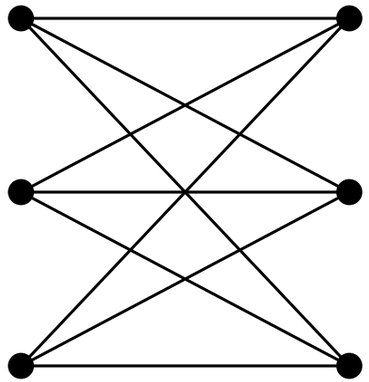
When is a graph planar?

4. Kuratowski's Theorem

When is a graph planar?



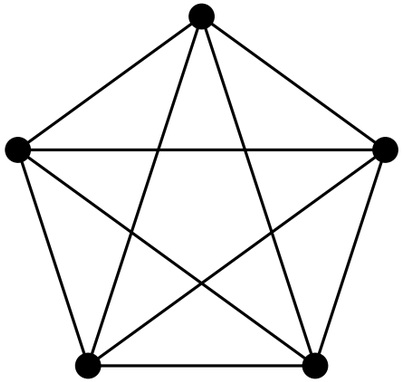
K_5



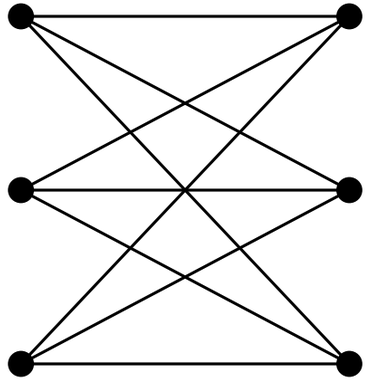
$K_{3,3}$

4. Kuratowski's Theorem

When is a graph planar?



K_5

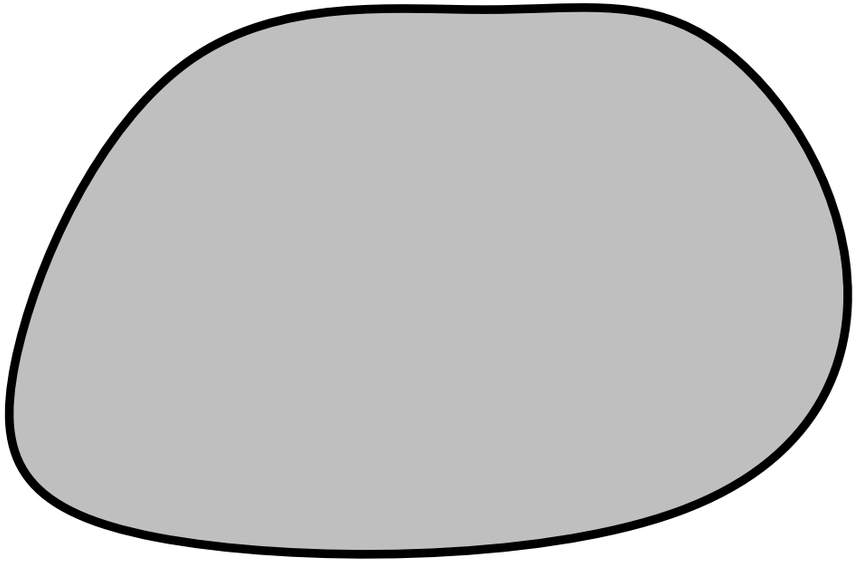


$K_{3,3}$

Kuratowski's Theorem:
A graph is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor.

5. Obtaining Kernels with Linear Programming

Recommended Knowledge: AGT



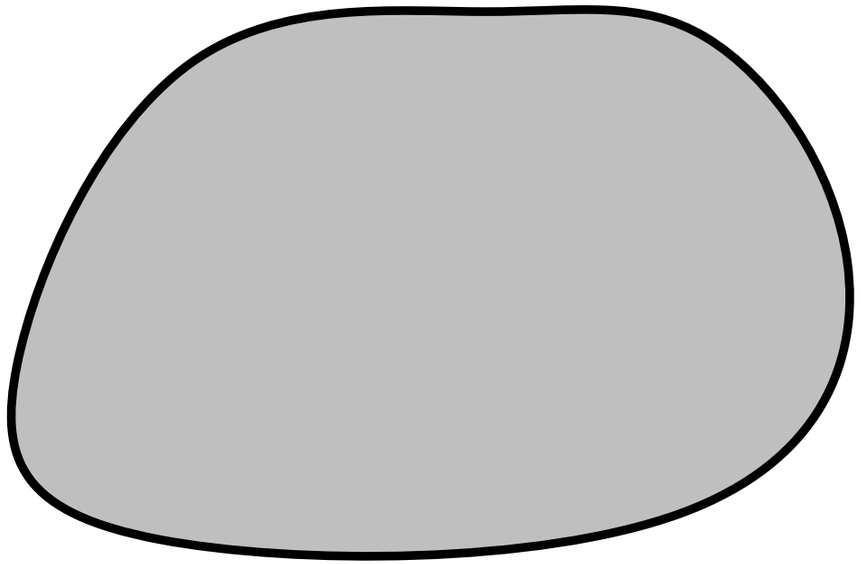
Graph G

Size: $|V| + |E|$

5. Obtaining Kernels with Linear Programming

Recommended Knowledge: AGT

Goal: find a much smaller graph G'
that is equivalent to the orignial



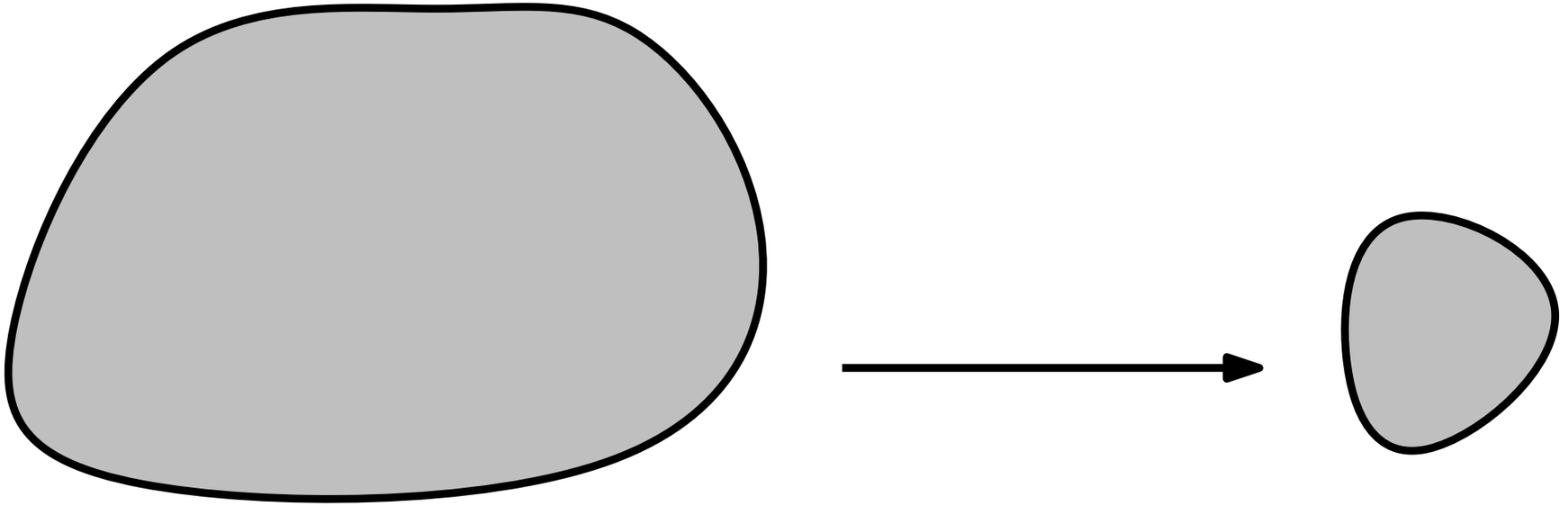
Graph G

Size: $|V| + |E|$

5. Obtaining Kernels with Linear Programming

Recommended Knowledge: AGT

Goal: find a much smaller graph G'
that is equivalent to the orignial



Graph G

Size: $|V| + |E|$

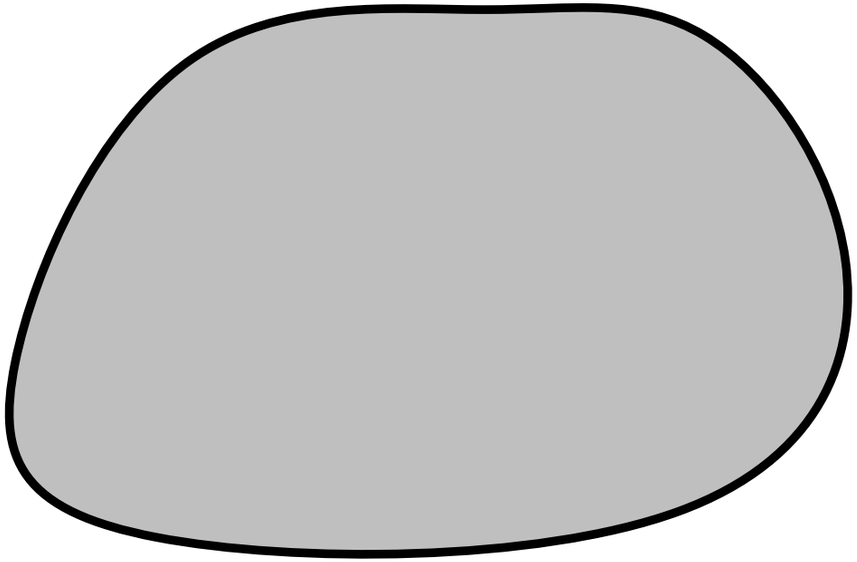
Graph G'

Size: $f(k)$ for some parameter k

5. Obtaining Kernels with Linear Programming

Recommended Knowledge: AGT

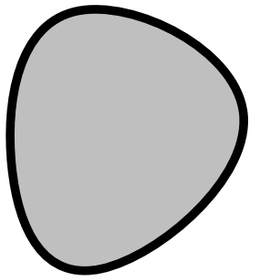
Goal: find a much smaller graph G'
that is equivalent to the orignial



Graph G

Size: $|V| + |E|$

Kernel of G



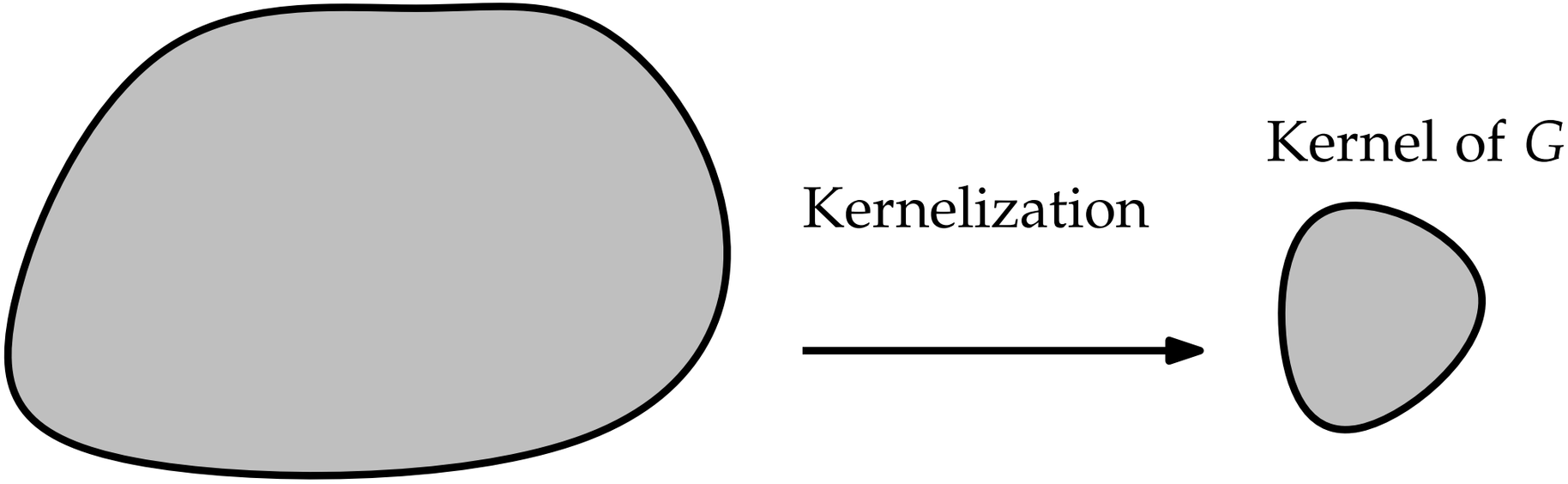
Graph G'

Size: $f(k)$ for some parameter k

5. Obtaining Kernels with Linear Programming

Recommended Knowledge: AGT

Goal: find a much smaller graph G'
that is equivalent to the orignial



Graph G

Size: $|V| + |E|$

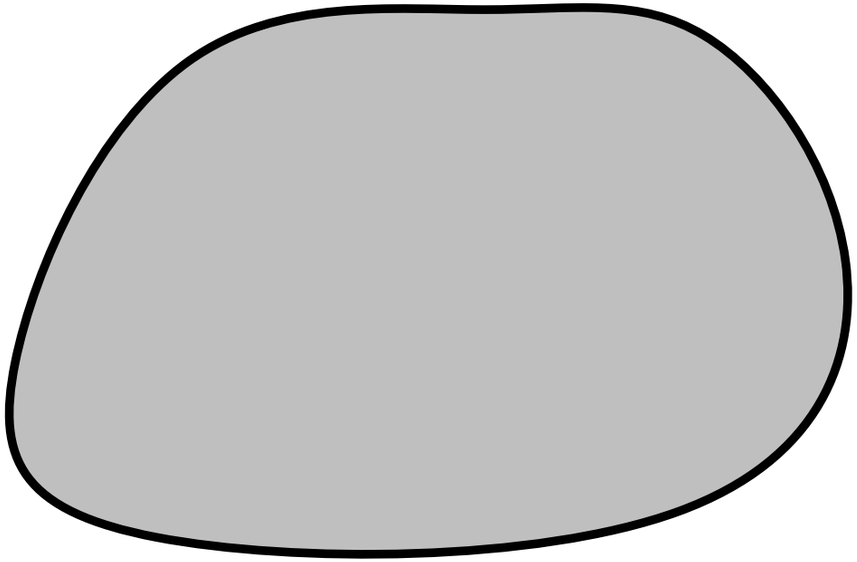
Graph G'

Size: $f(k)$ for some parameter k

5. Obtaining Kernels with Linear Programming

Recommended Knowledge: AGT

Goal: find a much smaller graph G'
that is equivalent to the orignial

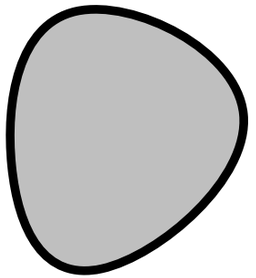


Graph G

Size: $|V| + |E|$

Kernelization
with ILPs

Kernel of G

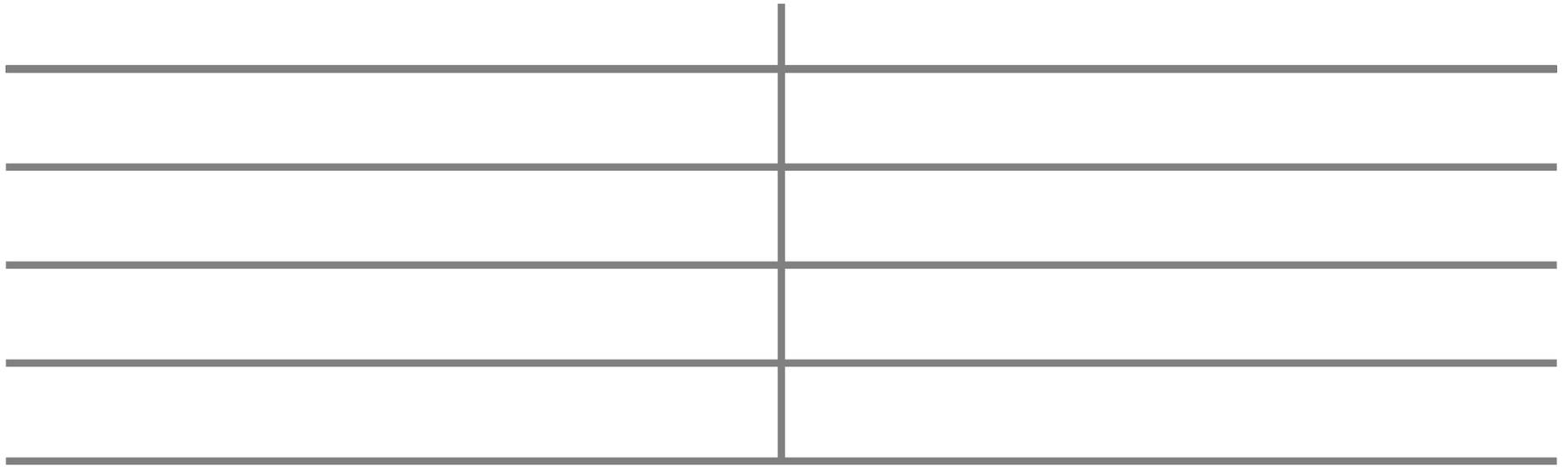


Graph G'

Size: $f(k)$ for some parameter k

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)



6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	
NP-hard	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	
NP-hard	
W[2]-hard	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	
NP-hard	
W[2]-hard	
no $o(\log n)$ approx.	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	
NP-hard	
W[2]-hard	
no $o(\log n)$ approx.	
no $2^{o(n)}$ algorithm	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	
W[2]-hard	
no $o(\log n)$ approx.	
no $2^{o(n)}$ algorithm	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	still NP-hard, but...
W[2]-hard	
no $o(\log n)$ approx.	
no $2^{o(n)}$ algorithm	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	still NP-hard, but...
W[2]-hard	FPT/linear kernel
no $o(\log n)$ approx.	
no $2^{o(n)}$ algorithm	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	still NP-hard, but...
W[2]-hard	FPT/linear kernel
no $o(\log n)$ approx.	$(1+\varepsilon)$-approx.
no $2^{o(n)}$ algorithm	

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	still NP-hard, but...
W[2]-hard	FPT/linear kernel
no $o(\log n)$ approx.	$(1+\varepsilon)$-approx.
no $2^{o(n)}$ algorithm	$2^{\mathcal{O}(\sqrt{k})} + \mathcal{O}(n)$ algorithm

k: solution size

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	still NP-hard, but...
W[2]-hard	FPT/linear kernel
no $o(\log n)$ approx.	$(1+\varepsilon)$-approx.
no $2^{o(n)}$ algorithm	$2^{\mathcal{O}(\sqrt{k})} + \mathcal{O}(n)$ algorithm

k: solution size

Why?

6. Bidimensionality

Recommended Knowledge: Exact Algorithms (FPT/Treewidth)

Dominating Set	Planar Dominating Set
NP-hard	still NP-hard, but...
W[2]-hard	FPT/linear kernel
no $o(\log n)$ approx.	$(1+\varepsilon)$-approx.
no $2^{o(n)}$ algorithm	$2^{\mathcal{O}(\sqrt{k})} + \mathcal{O}(n)$ algorithm

k: solution size

Why?

Theory of Bidimensionality

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

7. Structural Parameterizations of k-Planarity

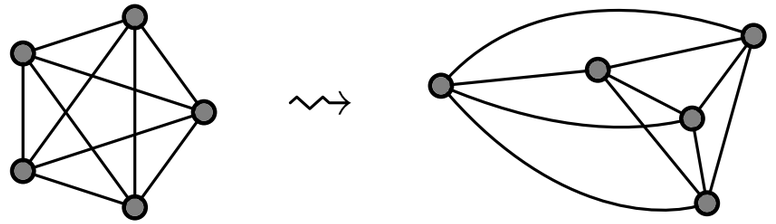
A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

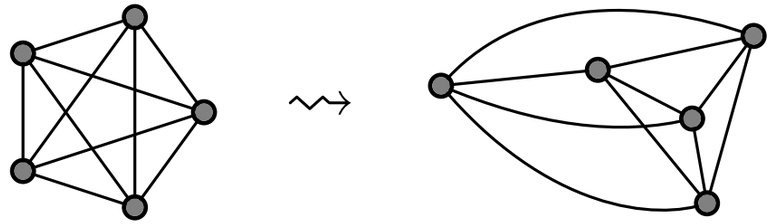
- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar



7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar

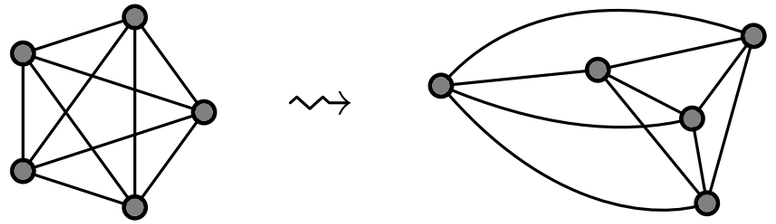


Testing whether a graph is k-planar for some $k \geq 1$ is NP-hard (even if $k = 1$).

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar



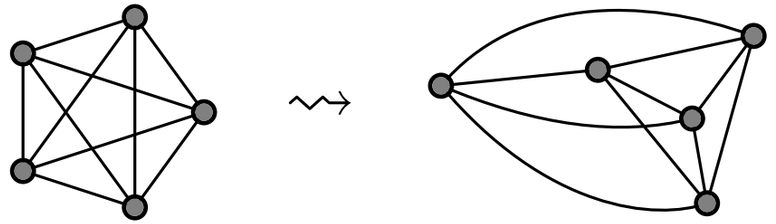
Testing whether a graph is k-planar for some $k \geq 1$ is NP-hard (even if $k = 1$).

There are many parameters that measure the structure of a graph, e.g.,

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar



Testing whether a graph is k-planar for some $k \geq 1$ is NP-hard (even if $k = 1$).

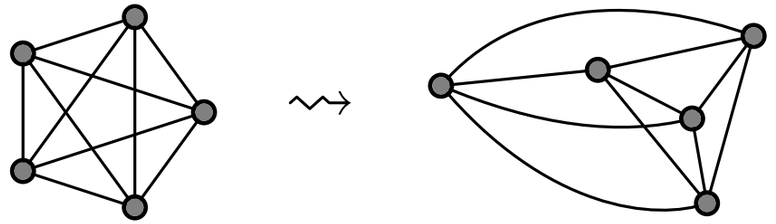
There are many parameters that measure the structure of a graph, e.g.,

- the **treewidth** of a graph measures how "tree-like" a graph is

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar



Testing whether a graph is k-planar for some $k \geq 1$ is NP-hard (even if $k = 1$).

There are many parameters that measure the structure of a graph, e.g.,

- the **treewidth** of a graph measures how "tree-like" a graph is

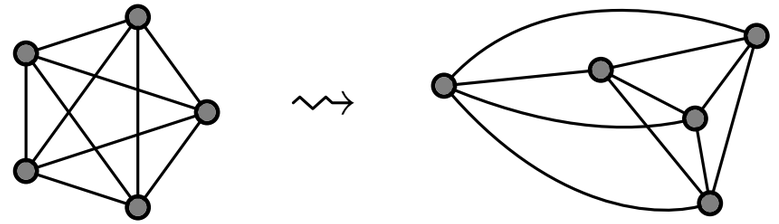
small number = very tree-like

large number = not at all tree-like

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar

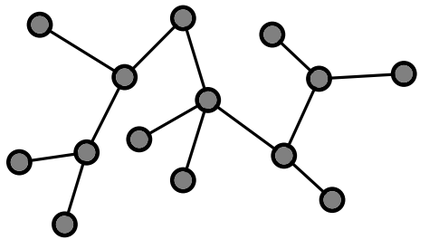


Testing whether a graph is k-planar for some $k \geq 1$ is NP-hard (even if $k = 1$).

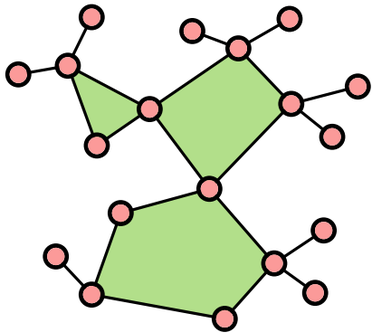
There are many parameters that measure the structure of a graph, e.g.,

- the **treewidth** of a graph measures how "tree-like" a graph is

small number = very tree-like

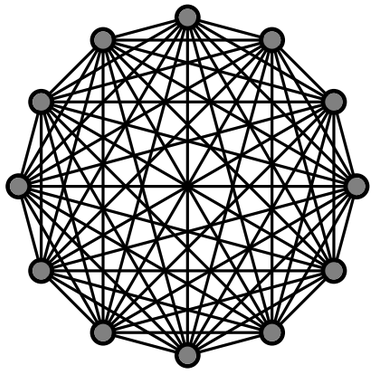


1



2

large number = not at all tree-like

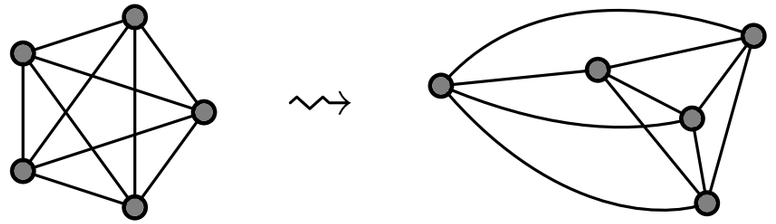


$n - 1$

7. Structural Parameterizations of k-Planarity

A graph is **k-planar** if it can be drawn with at most k crossings per edge, e.g.,

- planar graphs are 0-planar
- the complete graph K_5 on 5 vertices is 1-planar

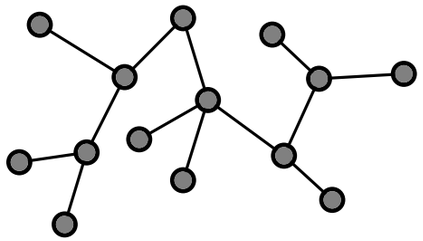


Testing whether a graph is k-planar for some $k \geq 1$ is NP-hard (even if $k = 1$).

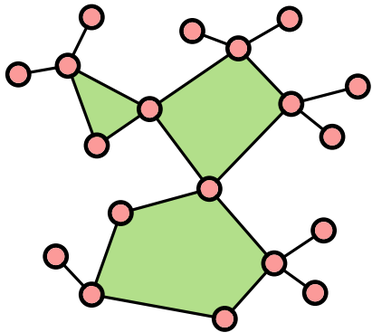
There are many parameters that measure the structure of a graph, e.g.,

- the **treewidth** of a graph measures how "tree-like" a graph is

small number = very tree-like

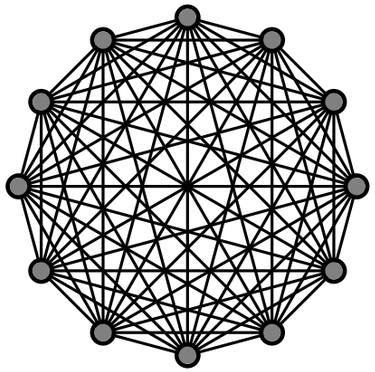


1



2

large number = not at all tree-like



$n - 1$

Can k-planarity be tested efficiently for very structured (e.g. tree-like) graphs?

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

Dijkstra's worst-case runtime: $\mathcal{O}(m + n \log n)$ (Fibonacci heap)

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

Dijkstra's worst-case runtime: $\mathcal{O}(m + n \log n)$ (Fibonacci heap)

Beyond worst-case analysis (for graph problems)

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

Dijkstra's worst-case runtime: $\mathcal{O}(m + n \log n)$ (Fibonacci heap)

Beyond worst-case analysis (for graph problems)

Instance optimality: An *instance-optimal* algorithm is at least as efficient as any correct algorithm, on every single input.

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

Dijkstra's worst-case runtime: $\mathcal{O}(m + n \log n)$ (Fibonacci heap)

Beyond worst-case analysis (for graph problems)

Instance optimality: An *instance-optimal* algorithm is at least as efficient as any correct algorithm, on every single input. very difficult :(

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

Dijkstra's worst-case runtime: $\mathcal{O}(m + n \log n)$ (Fibonacci heap)

Beyond worst-case analysis (for graph problems)

Instance optimality: An *instance-optimal* algorithm is at least as efficient as any correct algorithm, on every single input. very difficult :(

Universal optimality: A *universally optimal* algorithm is at least as efficient as any correct algorithm on any graph, for a worst-case choice of edge weights.

8. Universally Optimal Dijkstra's

Given: weighted graph G, a vertex s in G.

Want: a list of vertices of G ordered by their distance to s.

Dijkstra's worst-case runtime: $\mathcal{O}(m + n \log n)$ (Fibonacci heap)

Beyond worst-case analysis (for graph problems)

Instance optimality: An *instance-optimal* algorithm is at least as efficient as any correct algorithm, on every single input. very difficult :(

Universal optimality: A *universally optimal* algorithm is at least as efficient as any correct algorithm on any graph, for a worst-case choice of edge weights.

This paper: Dijkstra's is universally optimal when implemented with a *sufficiently efficient heap*.

9. Grid-Drawings of Graphs in 3D

10. Heuristics for Exact 1-Planarity

11. A Shape-First Methodology for Orthogonal Drawings

Themenverteilung

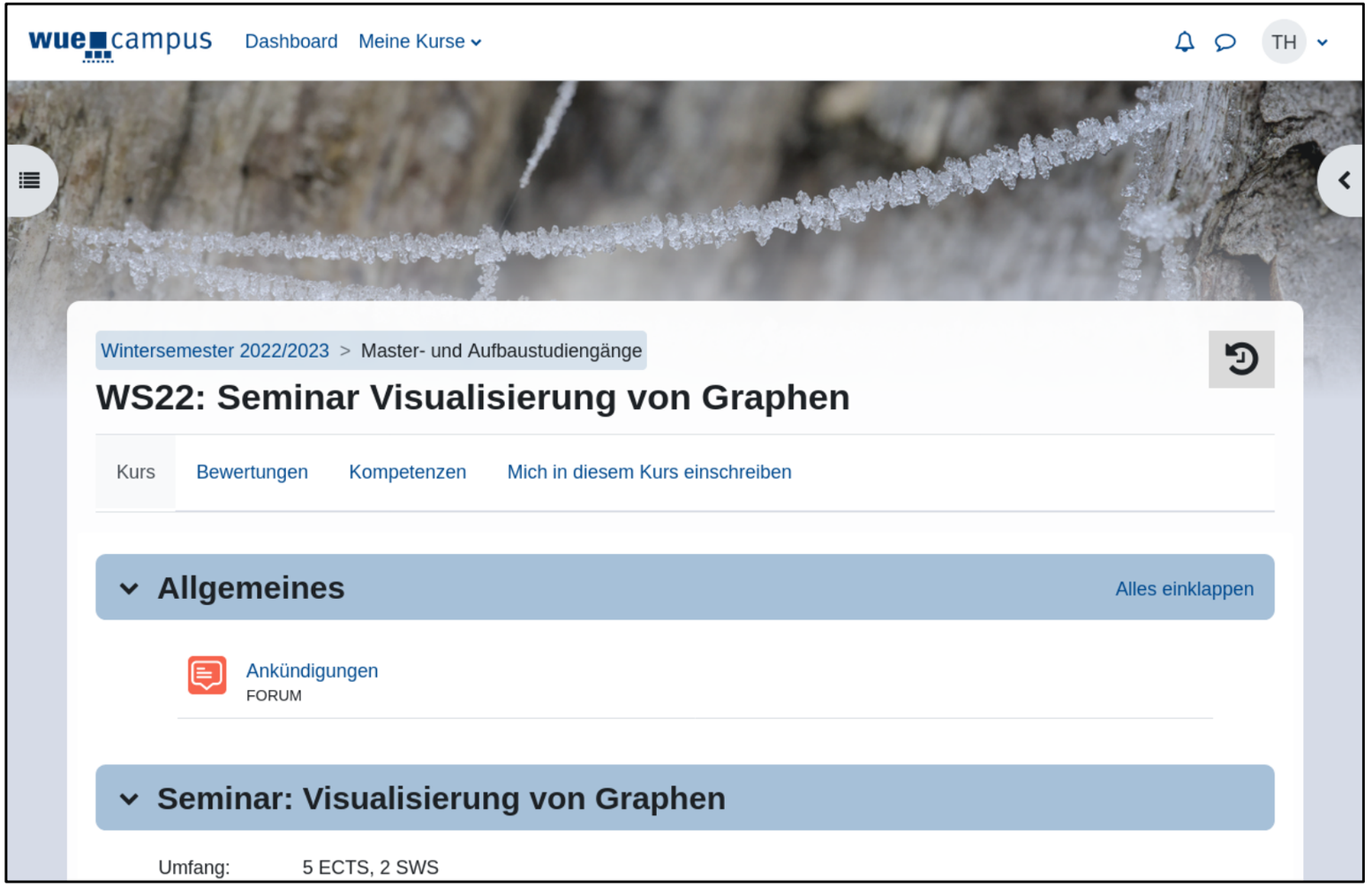
1. How to Morph Planar Graph Drawings
2. Sliding Squares in Parallel
3. Geometric Spanners of Bounded Tree-width
4. Kuratowski's Theorem
5. Obtaining Kernels with Linear Programming
6. Bidimensonality
7. Structural Parameterization of k-Planarity
8. Universally Optimal Dijkstra's
9. Grid-Drawings of Graphs in 3D
10. Heuristics for Exact 1-Planarity
11. A Shape-First Methodology for Orthogonal Drawings

Nächste Schritte

- In WueCampus anmelden

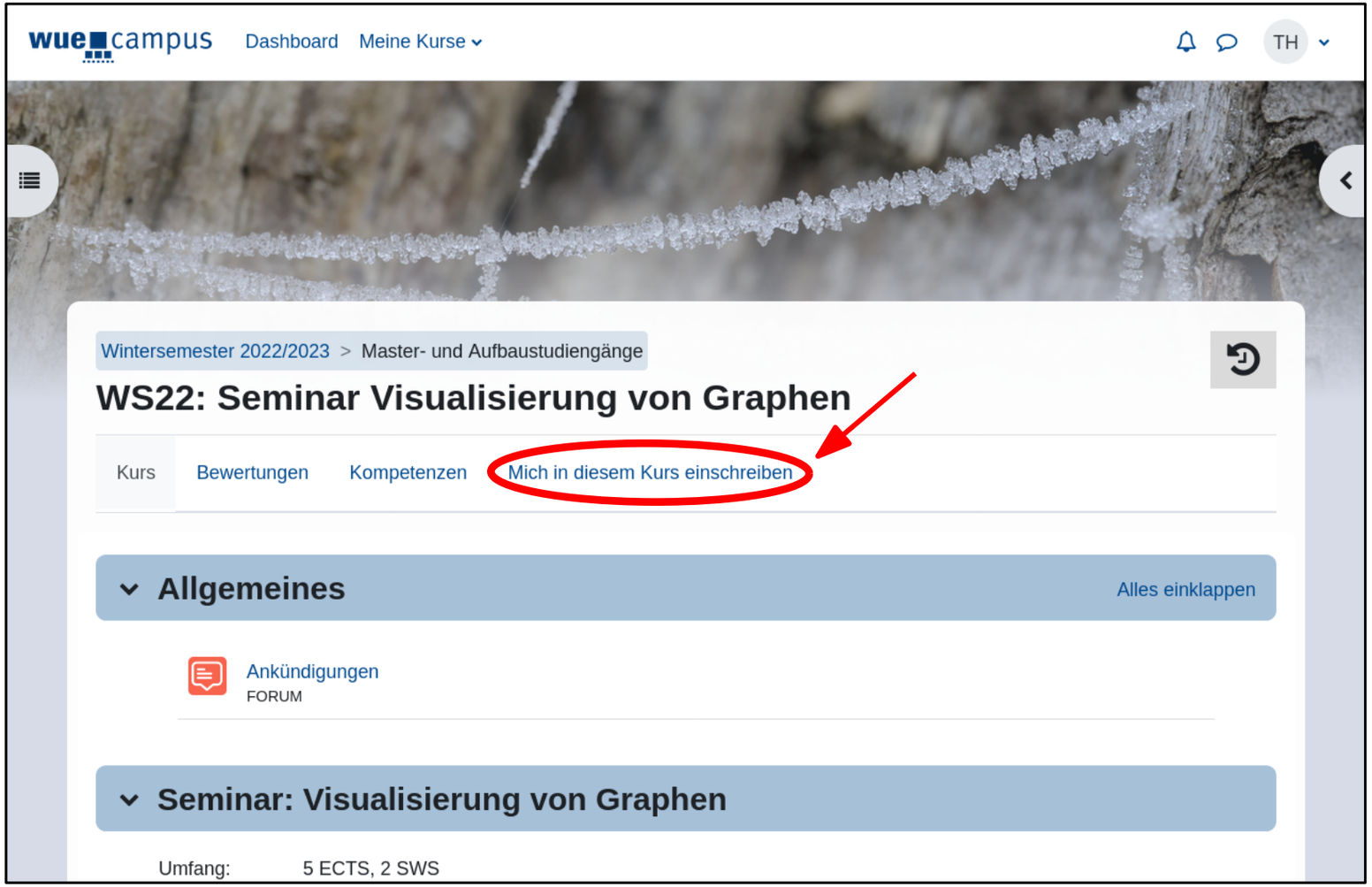
Nächste Schritte

- In WueCampus anmelden



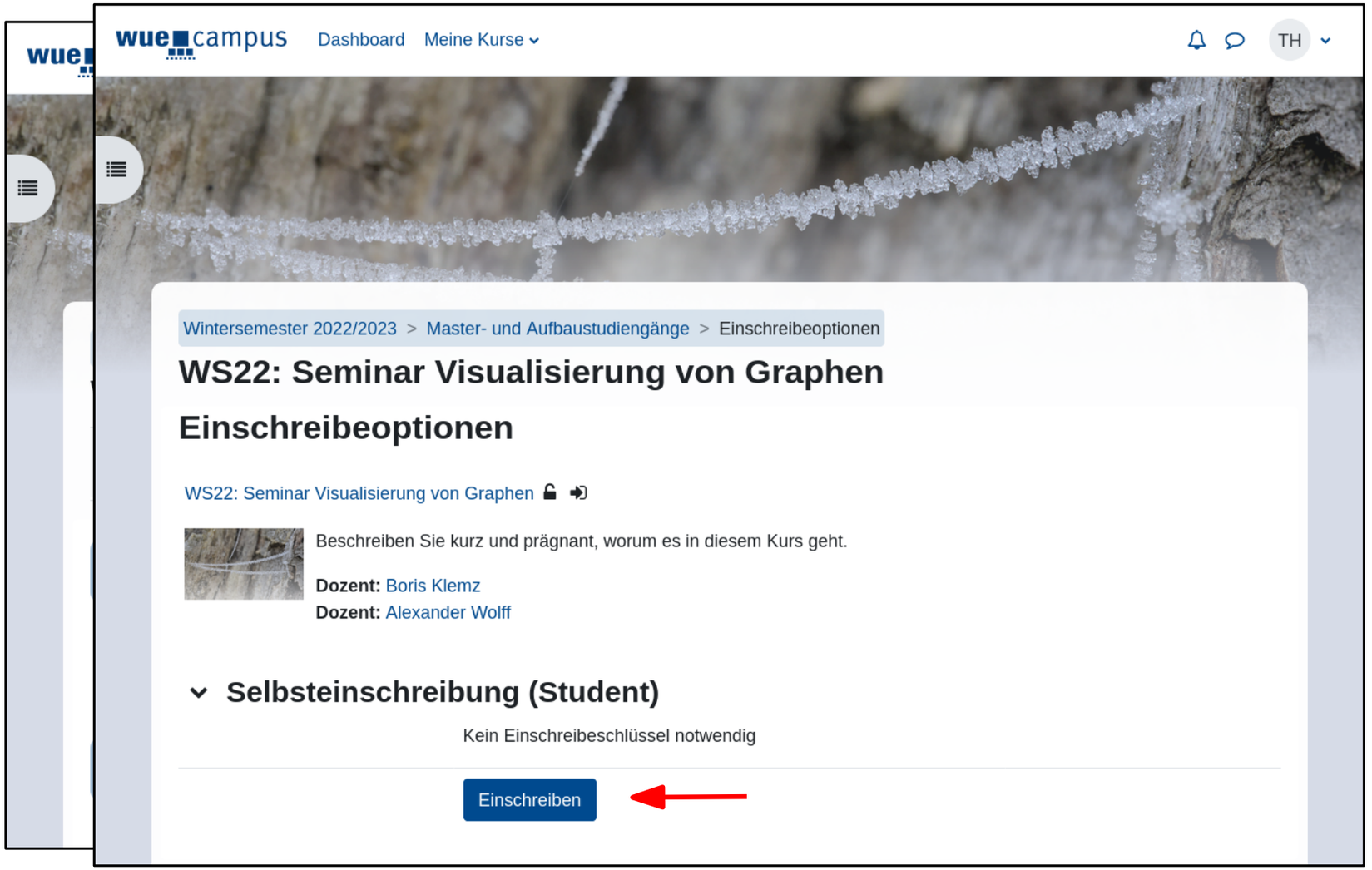
Nächste Schritte

- In WueCampus anmelden



Nächste Schritte

- In WueCampus anmelden



Nächste Schritte

- In WueCampus anmelden

Nächste Schritte

- In WueCampus anmelden
- In WueStudy anmelden

Nächste Schritte

- In WueCampus anmelden
- In WueStudy anmelden
- Überblick verschaffen und Kurzvortrag vorbereiten

Nächste Schritte

- In WueCampus anmelden
- In WueStudy anmelden
- Überblick verschaffen und Kurzvortrag vorbereiten
- Bei Fragen (oder *spätestens drei Wochen vor dem eigenen Vortrag*) an die BetreuerIn wenden

Nächste Schritte

Bei allgemeinen Fragen kann gerne das **Diskussionsforum** im WueCampus genutzt werden!

- In WueCampus anmelden
- In WueStudy anmelden
- Überblick verschaffen und Kurzvortrag vorbereiten
- Bei Fragen (oder *spätestens drei Wochen vor dem eigenen Vortrag*) an die BetreuerIn wenden

Nächste Schritte

Bei allgemeinen Fragen kann gerne das **Diskussionsforum** im WueCampus genutzt werden!

- In WueCampus anmelden
- In WueStudy anmelden
- Überblick verschaffen und Kurzvortrag vorbereiten
- Bei Fragen (oder *spätestens drei Wochen vor dem eigenen Vortrag*) an die BetreuerIn wenden

Zum Abschluss:

Demonstration des Programms IPE
zum Erstellen von Abbildungen und Folien

http://ipe.otfried.org/

Nächste Schritte

Bei allgemeinen Fragen kann gerne das **Diskussionsforum** im WueCampus genutzt werden!

- In WueCampus anmelden
- In WueStudy anmelden
- Überblick verschaffen und Kurzvortrag vorbereiten
- Bei Fragen (oder *spätestens drei Wochen vor dem eigenen Vortrag*) an die BetreuerIn wenden

Zum Abschluss: Demonstration des Programms IPE zum Erstellen von Abbildungen und Folien

http://ipe.otfried.org/

Übrigens: ein gemeinsames git-Verzeichnis eignet sich hervorragend zum gemeinsamen Bearbeiten von .tex, aber auch .ipe Dateien!

Nächste Schritte

Bei allgemeinen Fragen kann gerne das **Diskussionsforum** im WueCampus genutzt werden!

- In WueCampus anmelden
- In WueStudy anmelden
- Überblick verschaffen und Kurzvortrag vorbereiten
- Bei Fragen (oder *spätestens drei Wochen vor dem eigenen Vortrag*) an die BetreuerIn wenden

Zum Abschluss:

Demonstration des Programms IPE
zum Erstellen von Abbildungen und Folien

http://ipe.otfried.org/

https://gitlab2.informatik.uni-wuerzburg.de/

Übrigens: ein gemeinsames git-Verzeichnis eignet sich hervorragend zum gemeinsamen Bearbeiten von .tex, aber auch .ipe Dateien!