Outerplanar and Forest Storyplans

Jiří Fiala **Oksana Firman** Alexander Wolff Johannes Zink Giuseppe Liotta

Univerzita Karlova, Prague, Czech Republic

Julius-Maximilians-Universität Würzburg, Germany

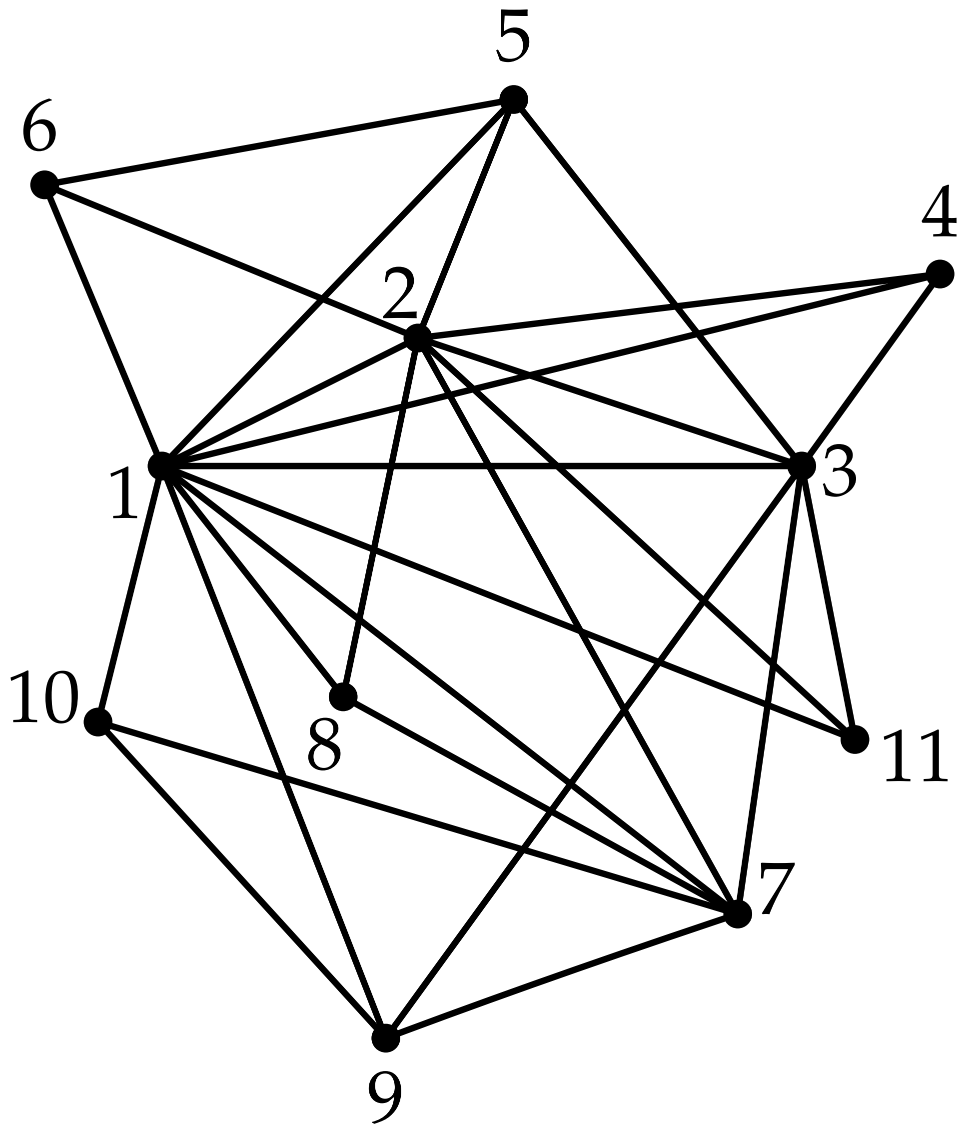
Università degli Studi di Perugia, Italy

CHARLES UNIVERSITY

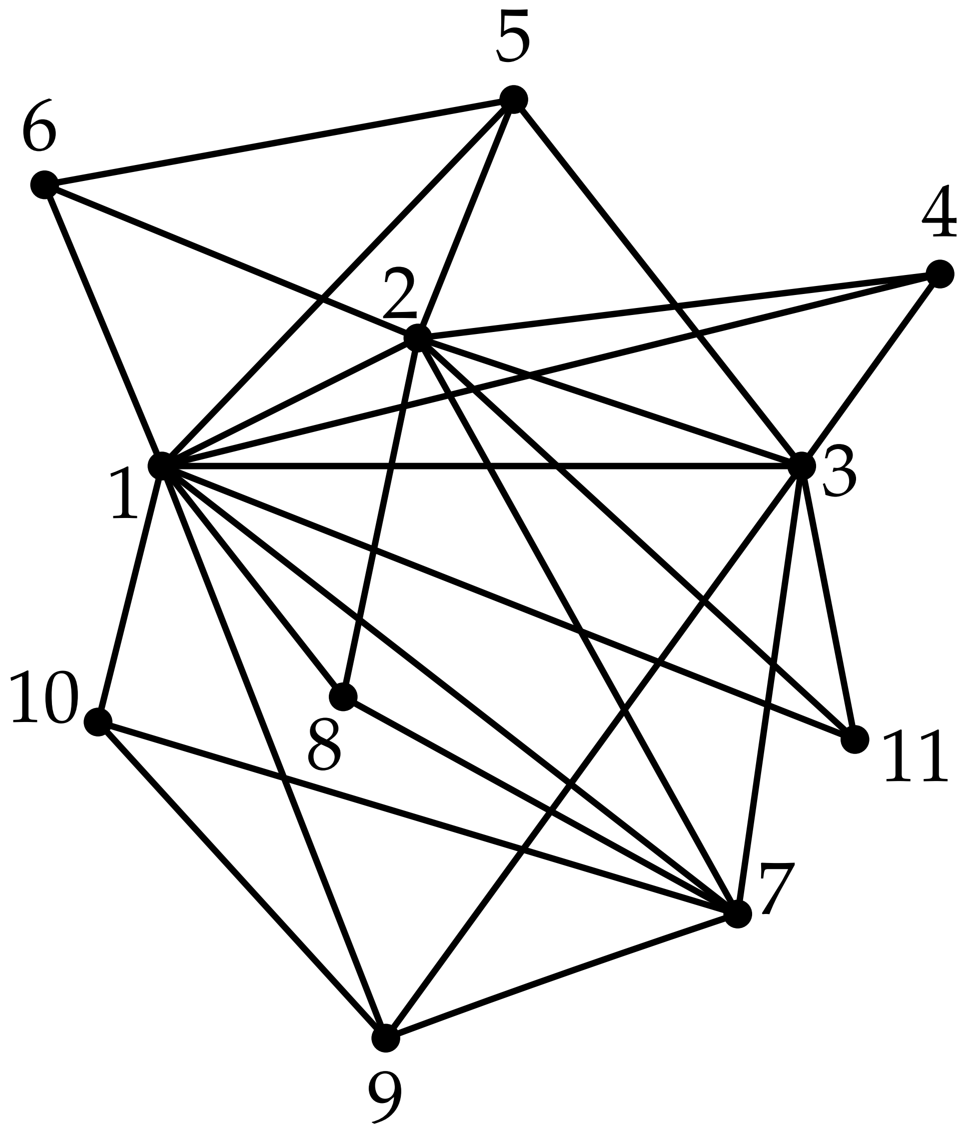
Julius-Maximilians-UNIVERSITÄT WÜRZBURG

A.D. 1308 unipg UNIVERSITÀ DEGLI STUDI DI PERUGIA

Motivation

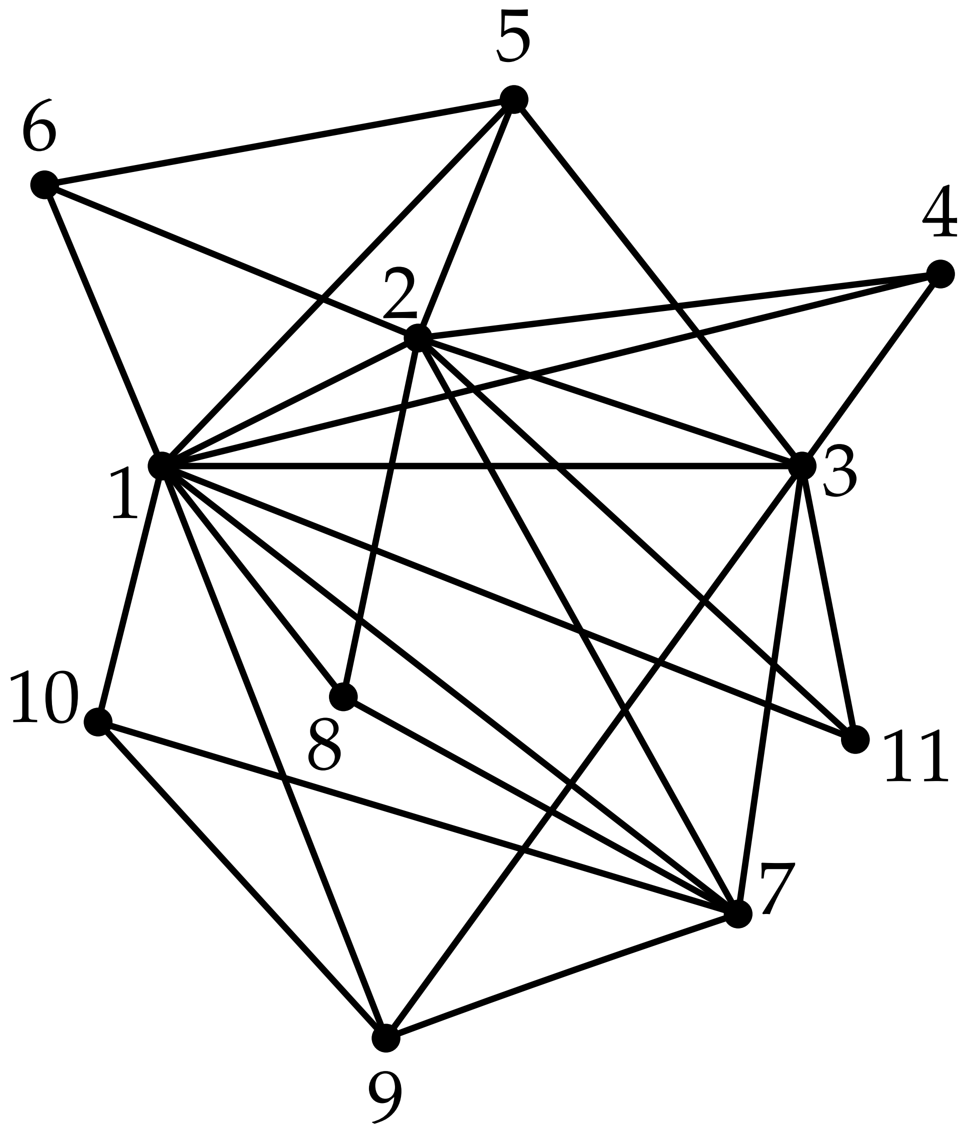


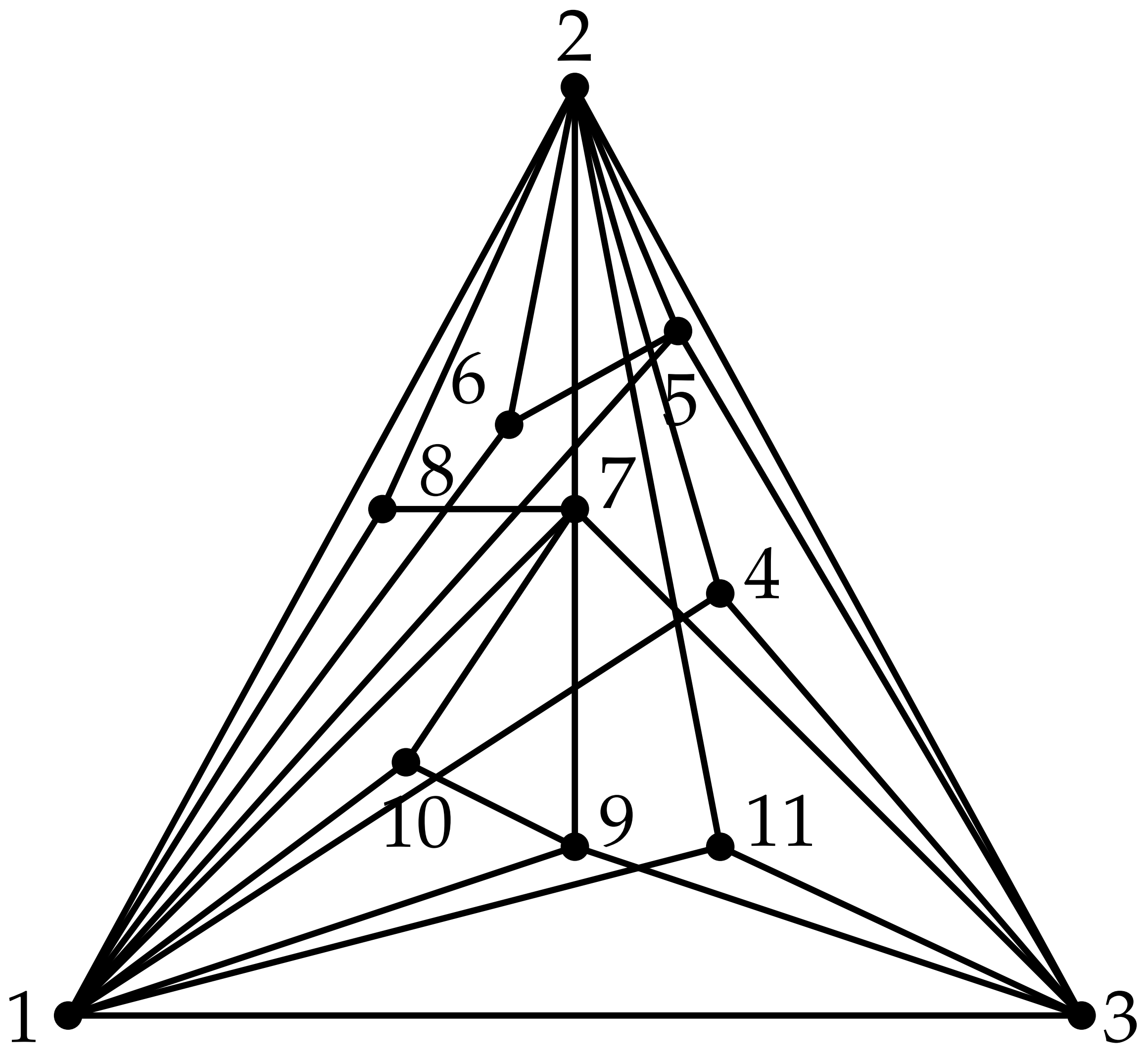
Motivation



Show a graph in a few steps s.t. in each step the visible part of the graph is better "readable", that is, there are no crossings.

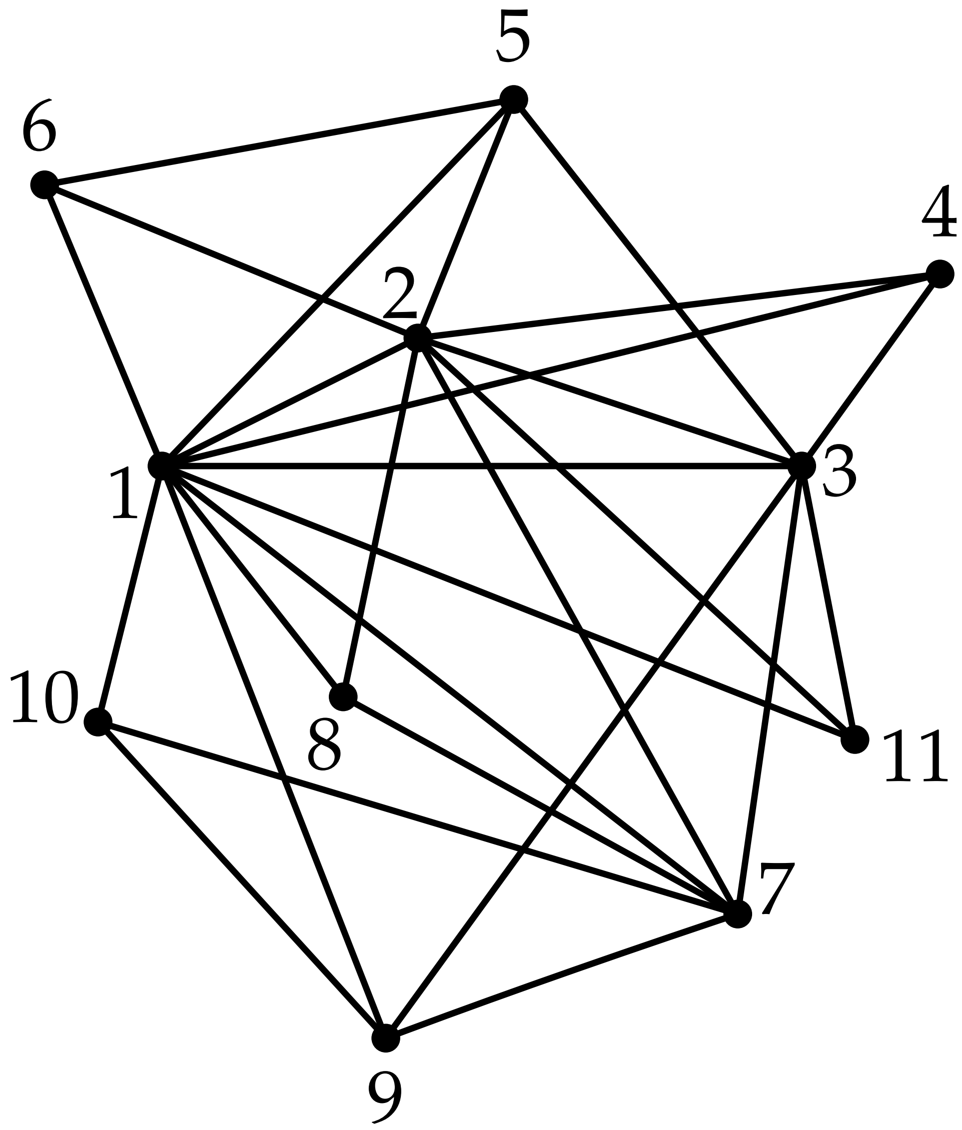
Motivation

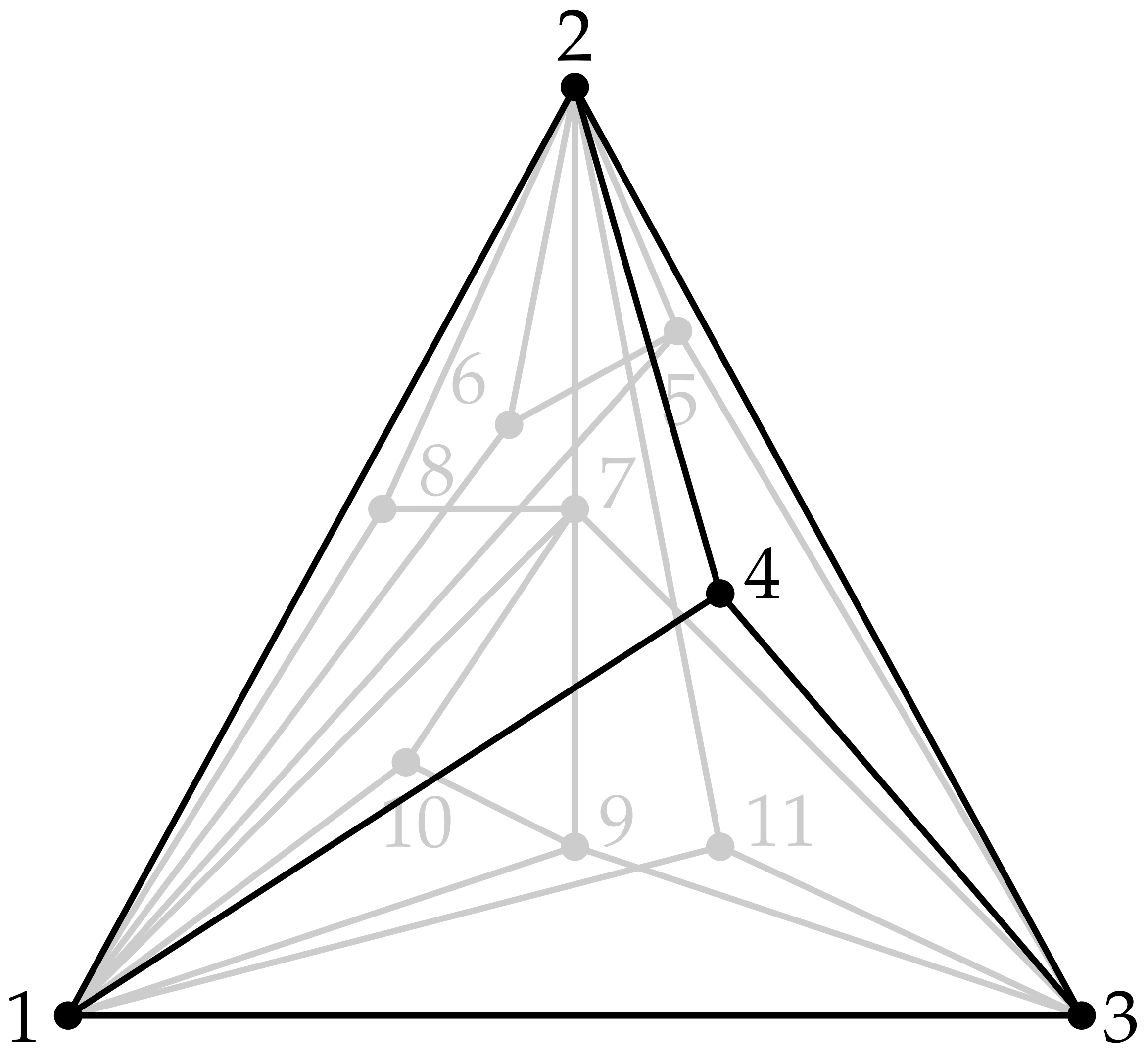




Show a graph in a few steps s.t. in each step the visible part of the graph is better "readable", that is, there are no crossings.

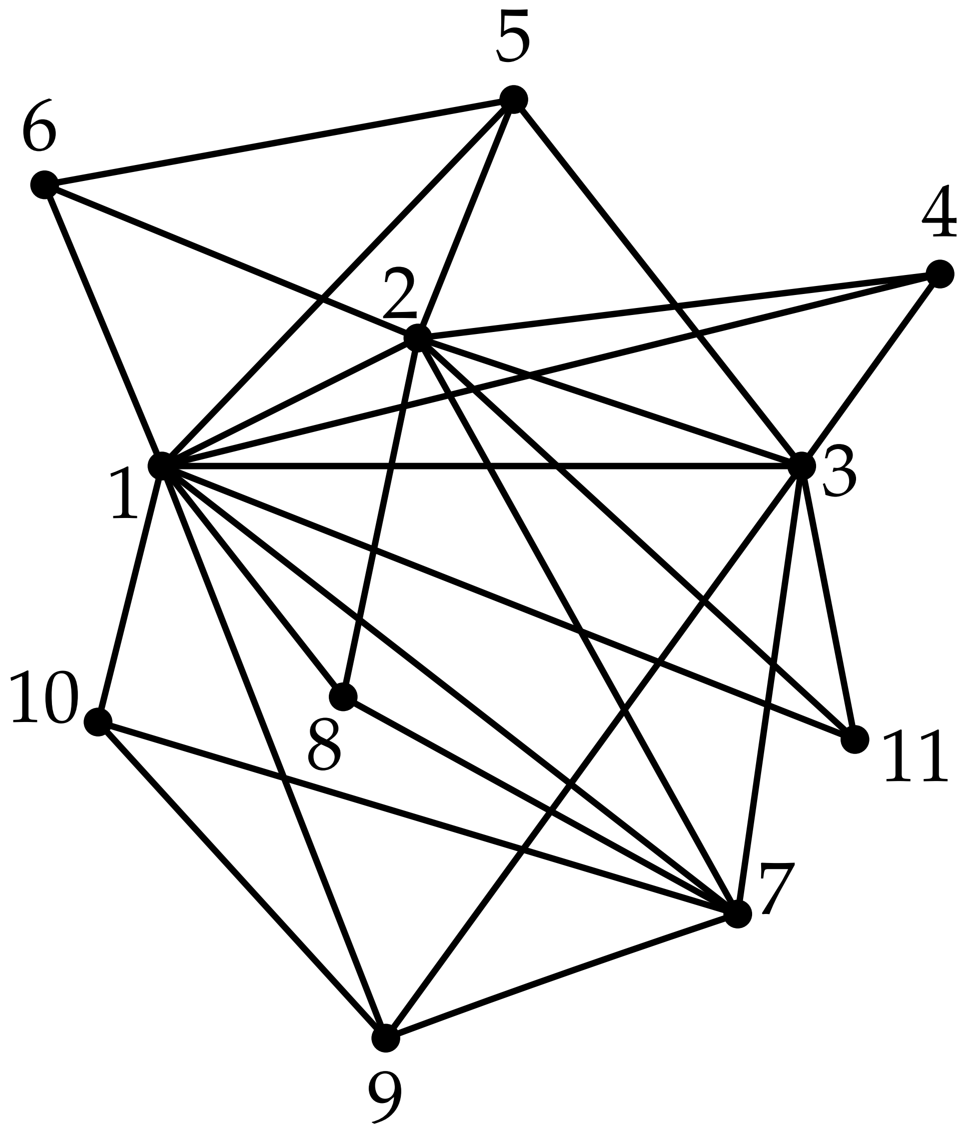
Motivation

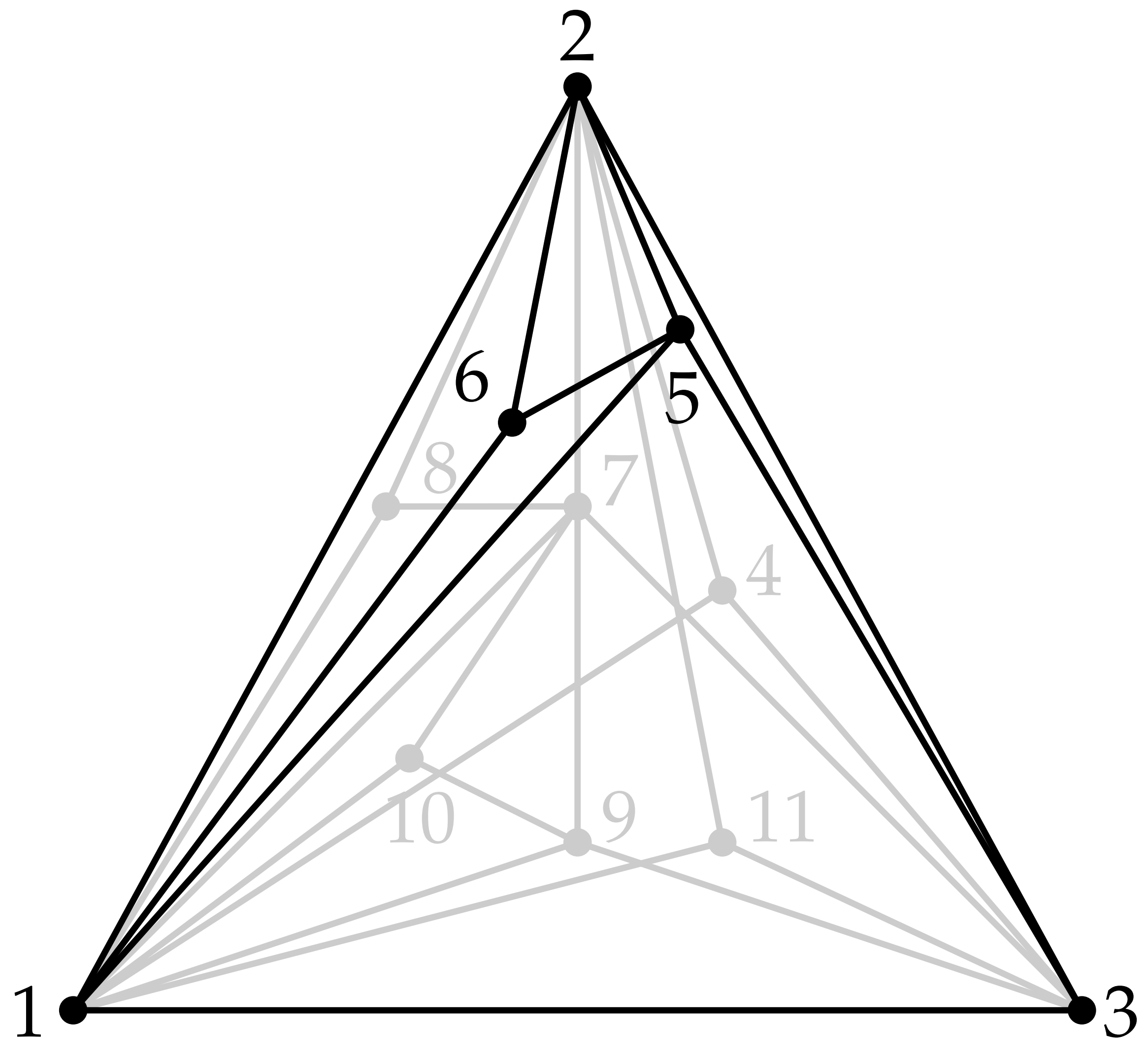




Show a graph in a few steps s.t. in each step the visible part of the graph is better "readable", that is, there are no crossings.

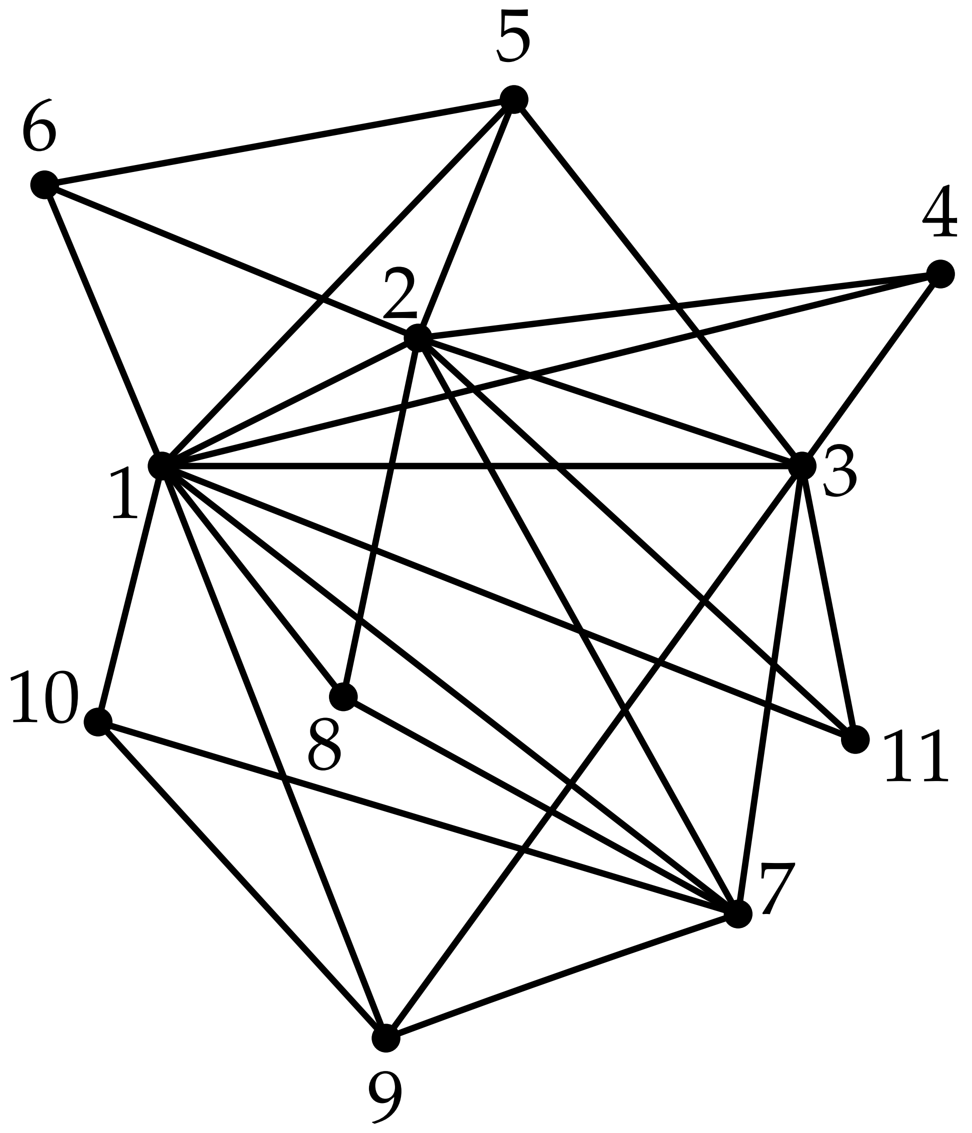
Motivation

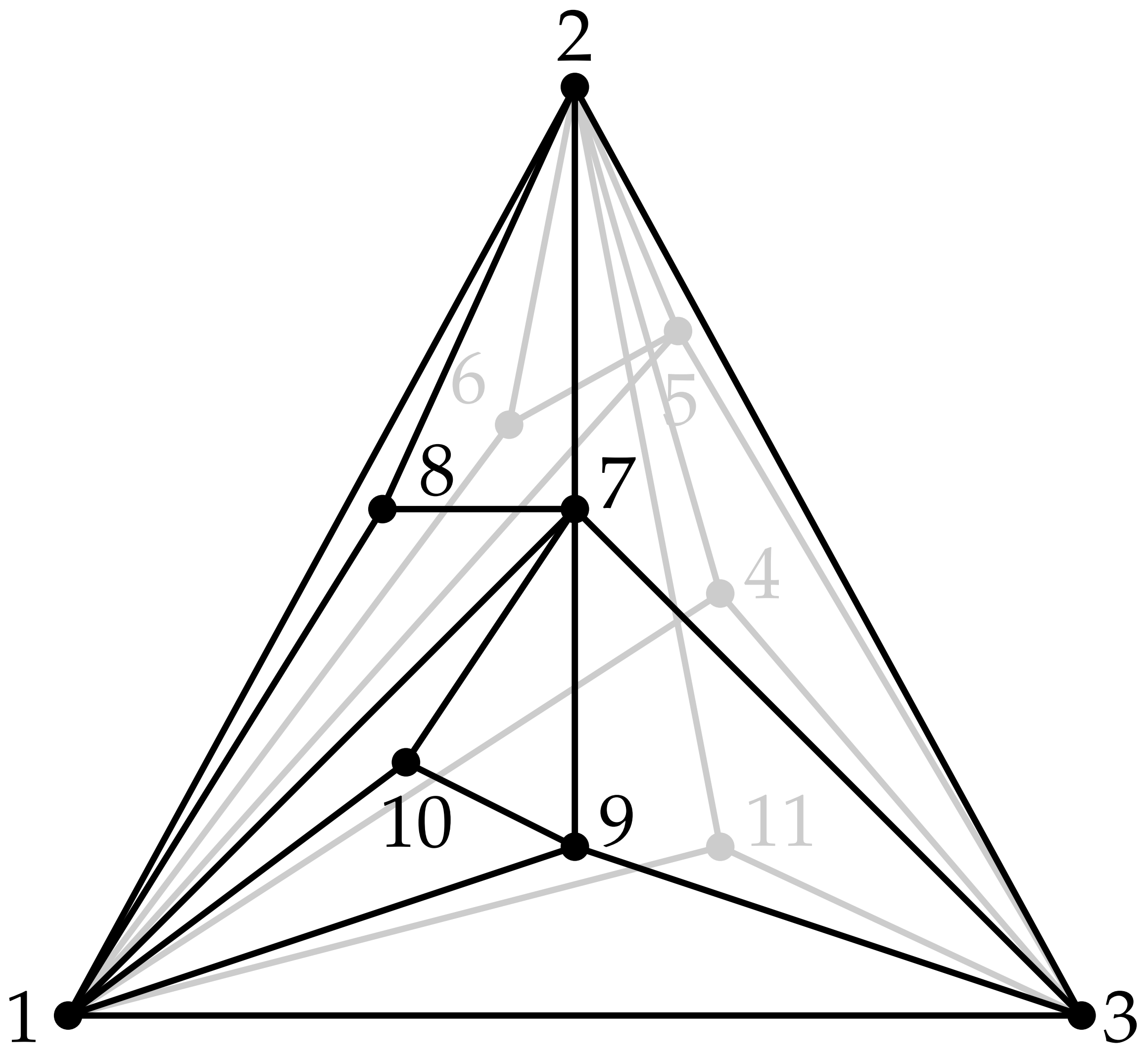




Show a graph in a few steps s.t. in each step the visible part of the graph is better "readable", that is, there are no crossings.

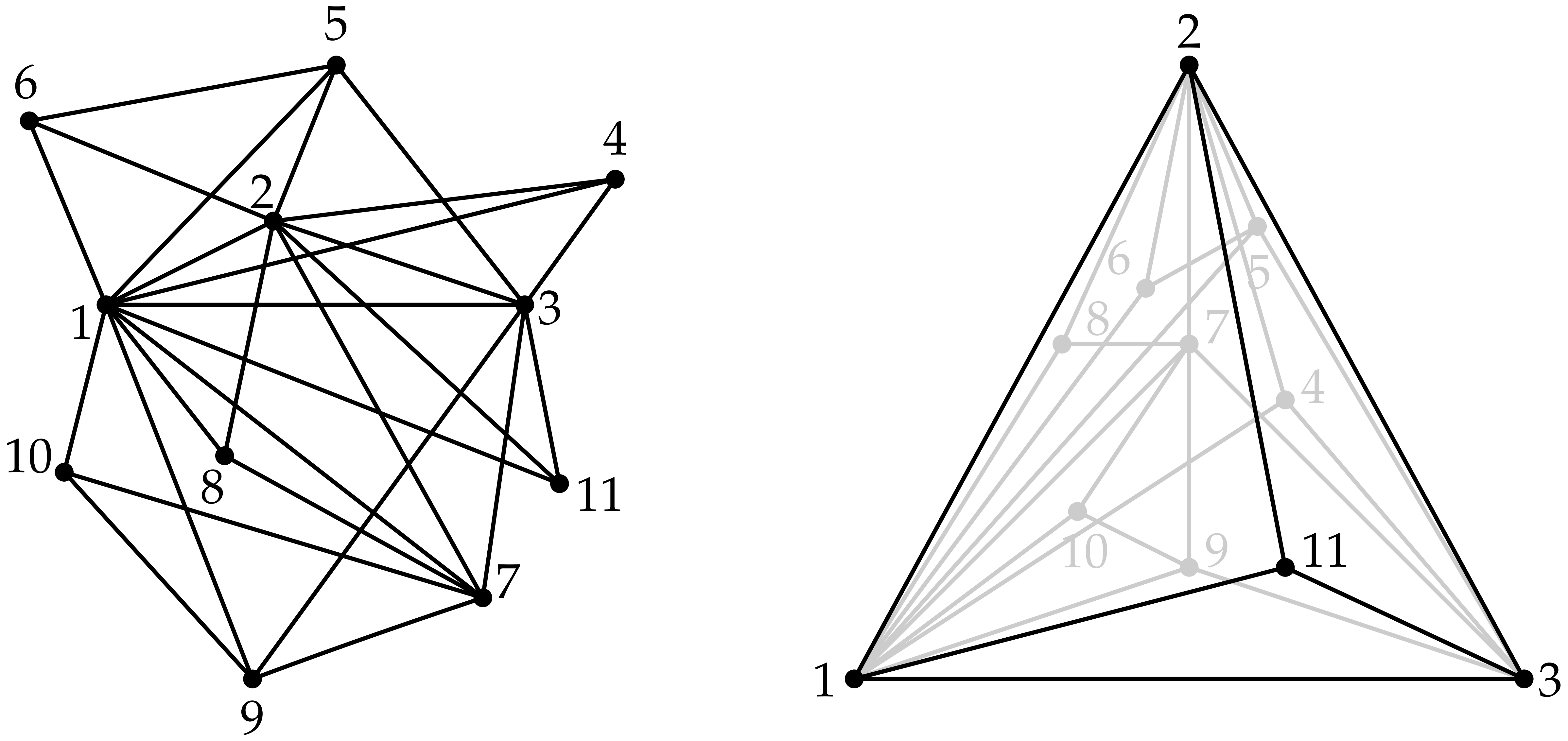
Motivation





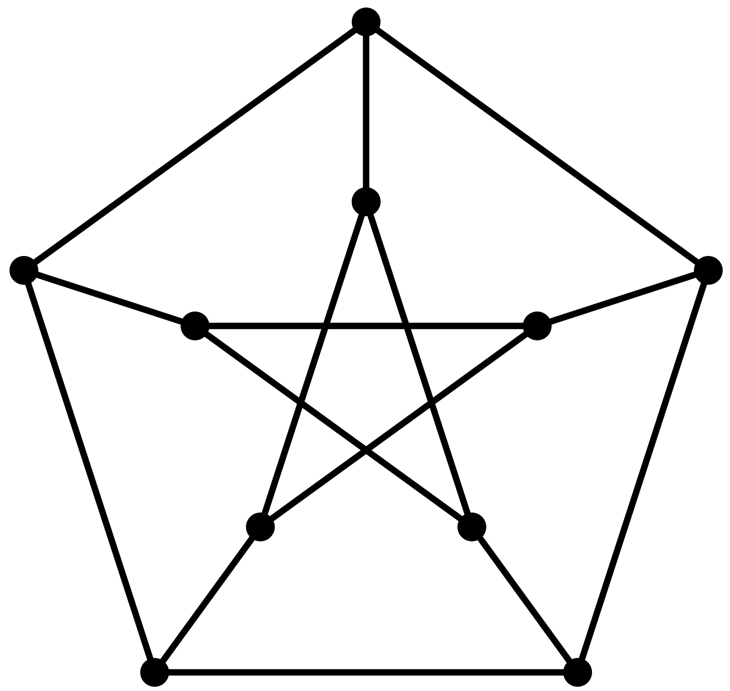
Show a graph in a few steps s.t. in each step the visible part of the graph is better "readable", that is, there are no crossings.

Motivation



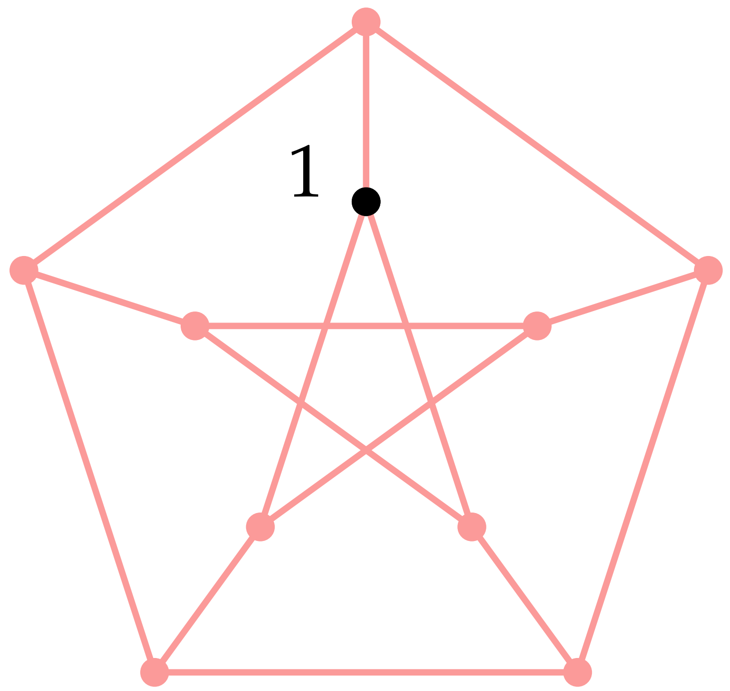
Show a graph in a few steps s.t. in each step the visible part of the graph is better "readable", that is, there are no crossings.

Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

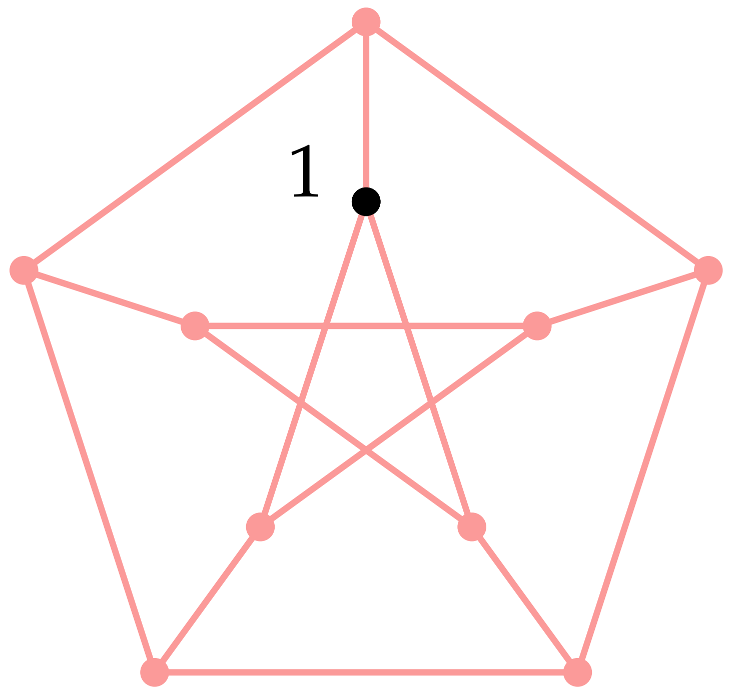
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.

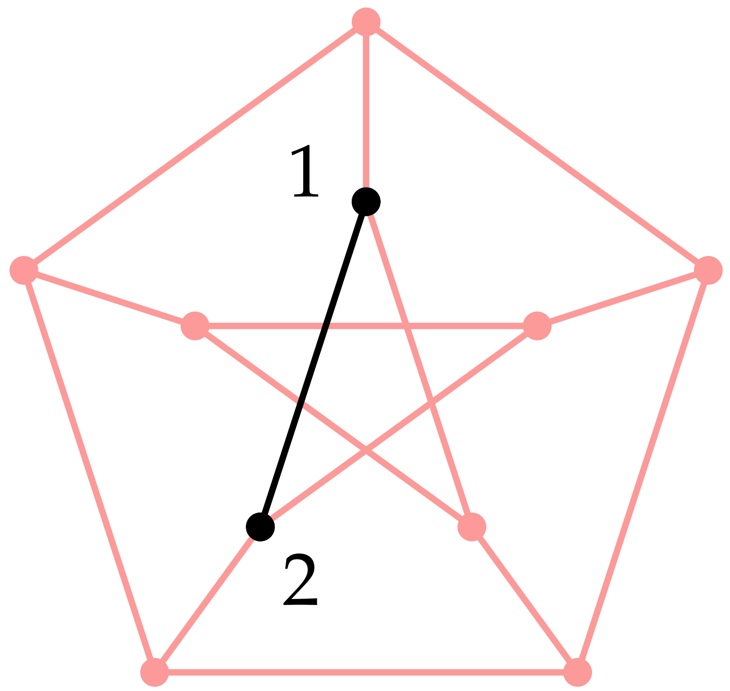
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented.

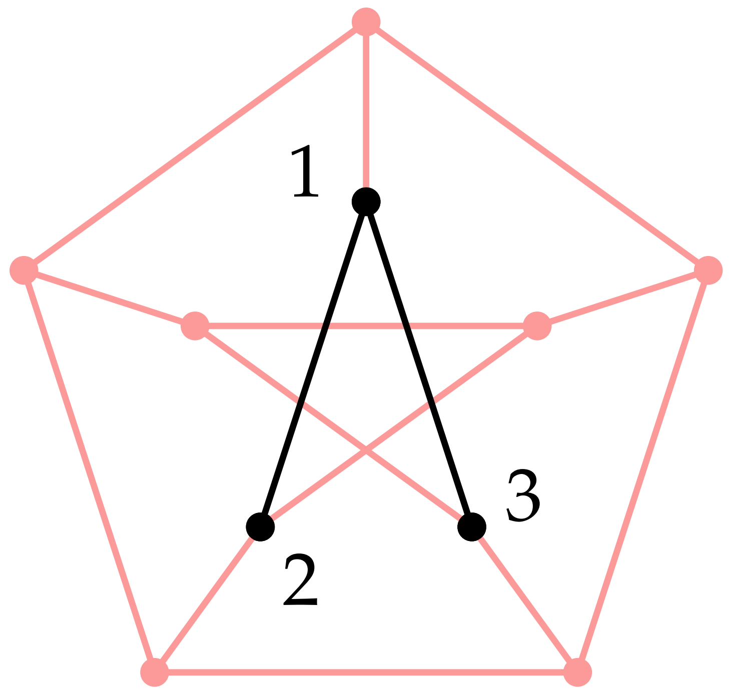
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented.

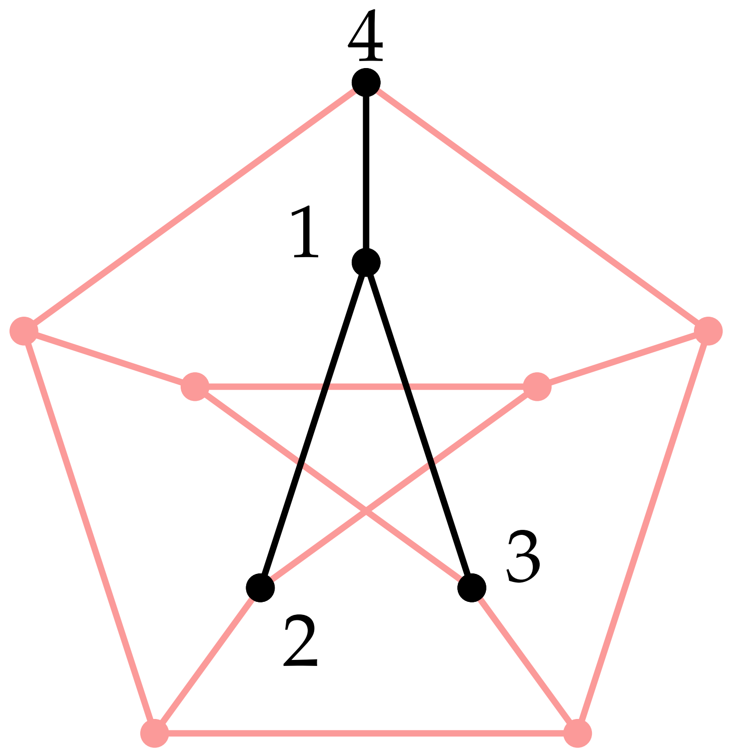
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented.

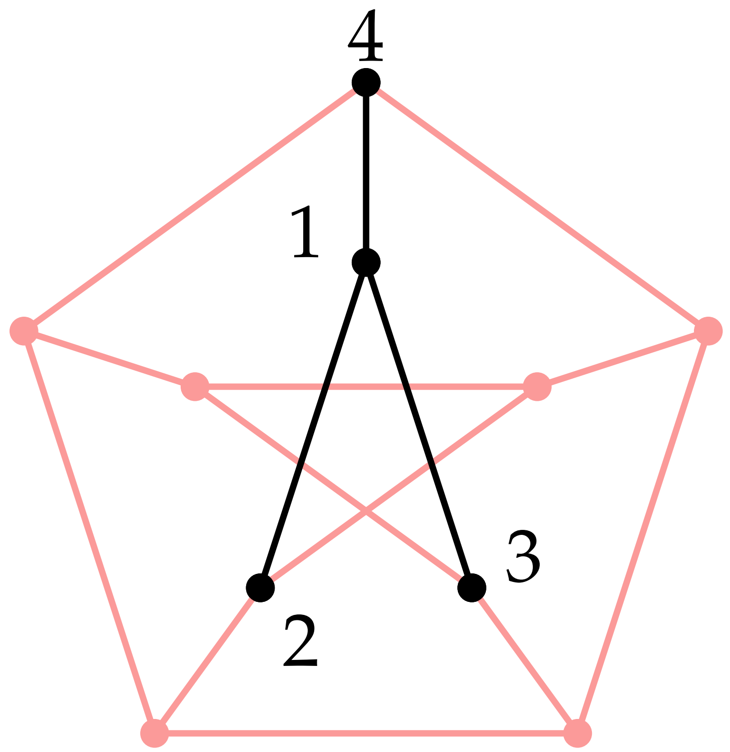
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented.

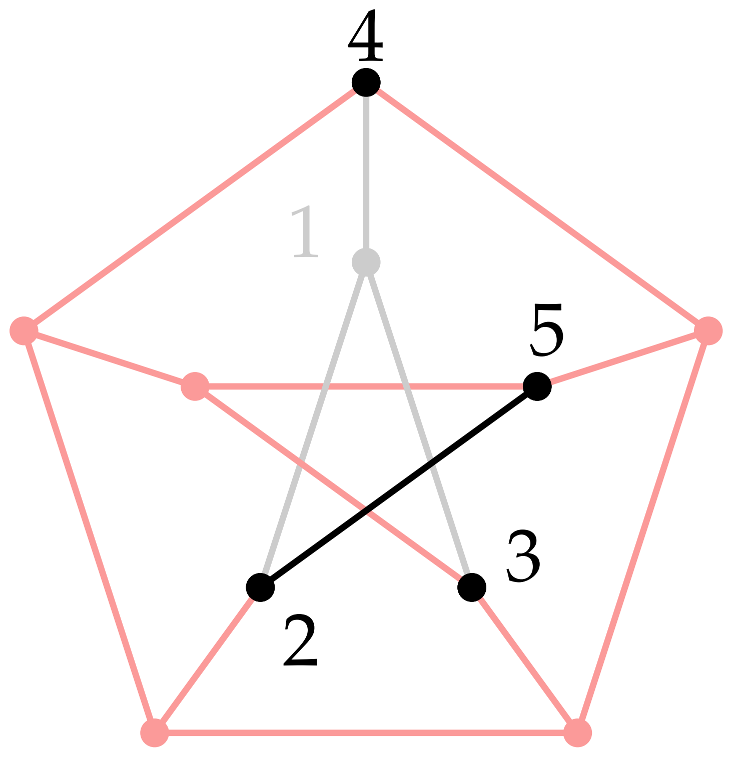
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

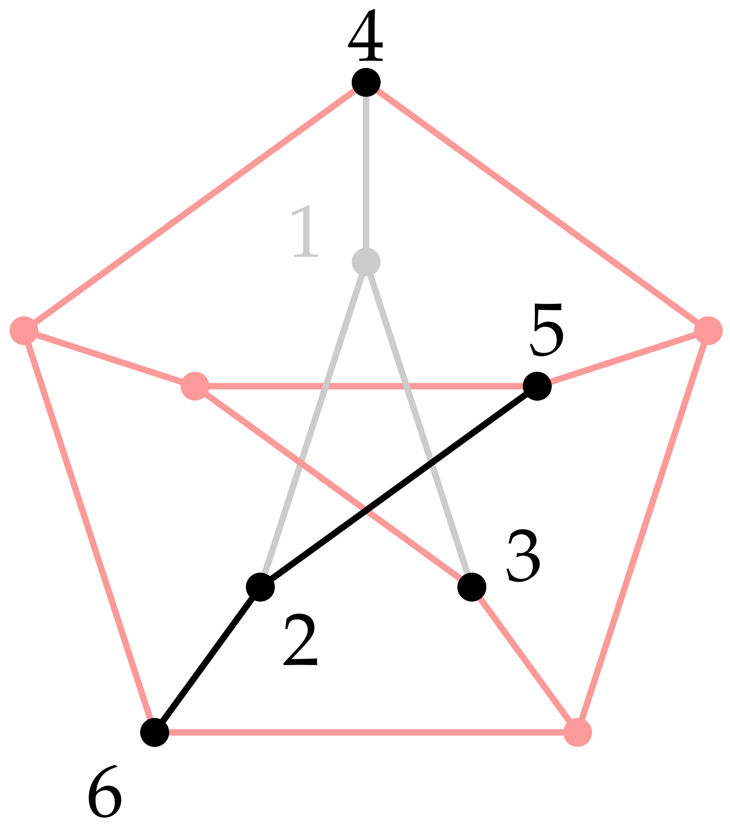
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

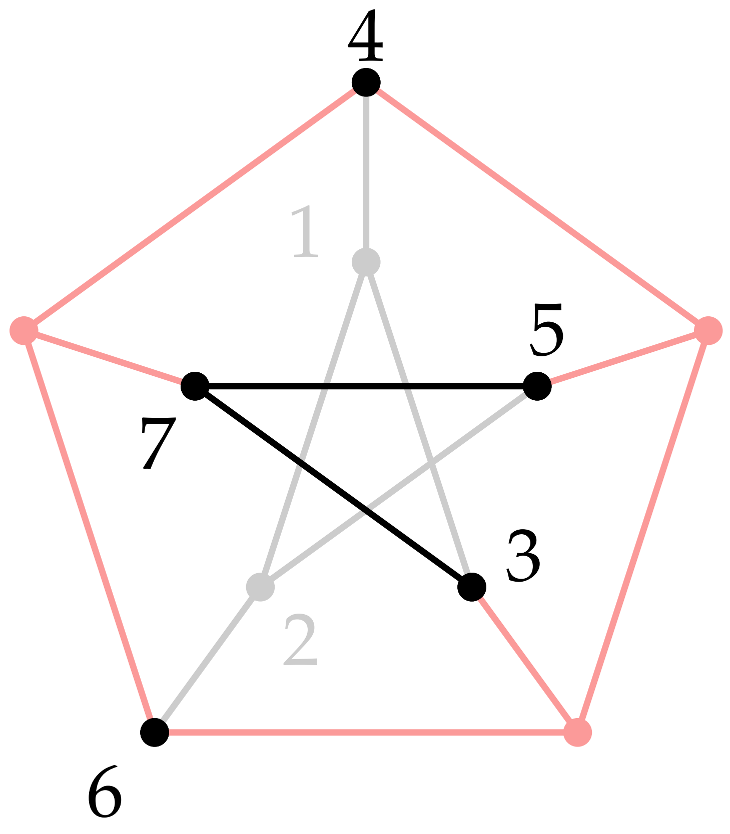
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

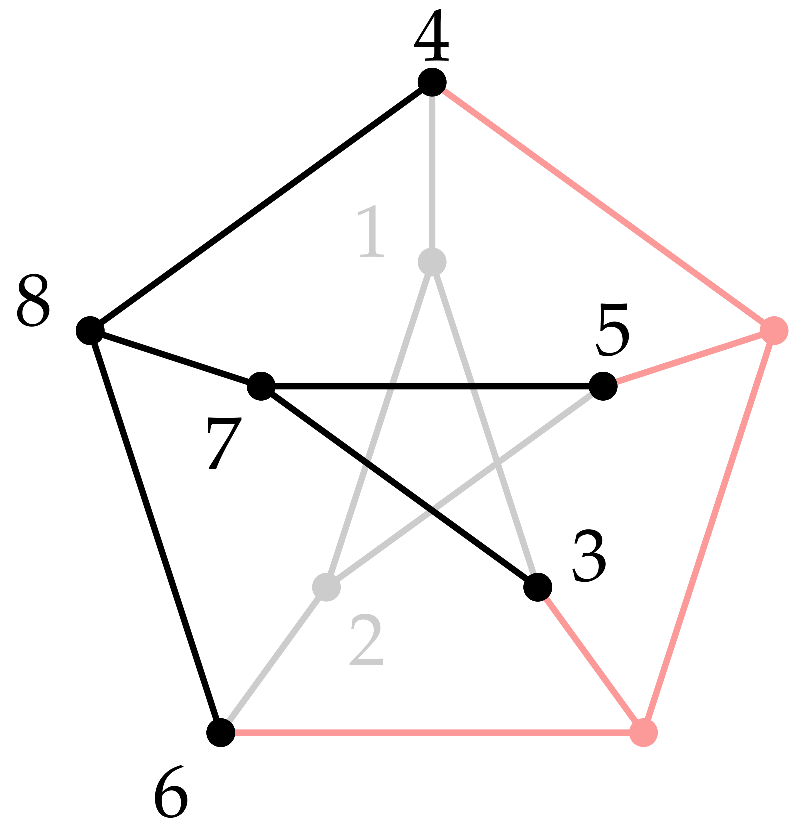
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

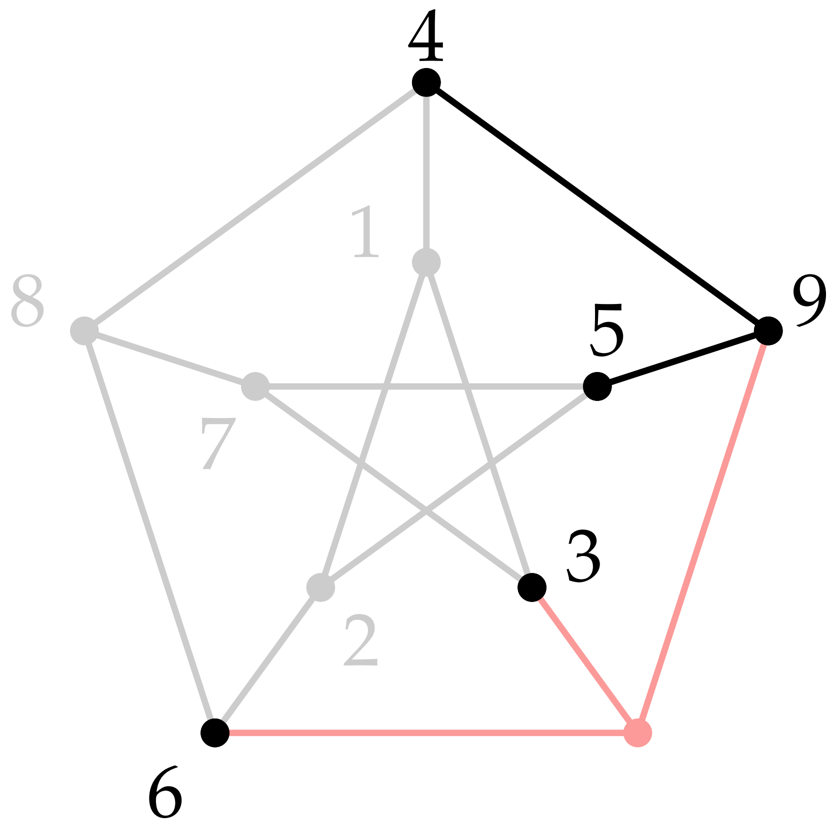
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

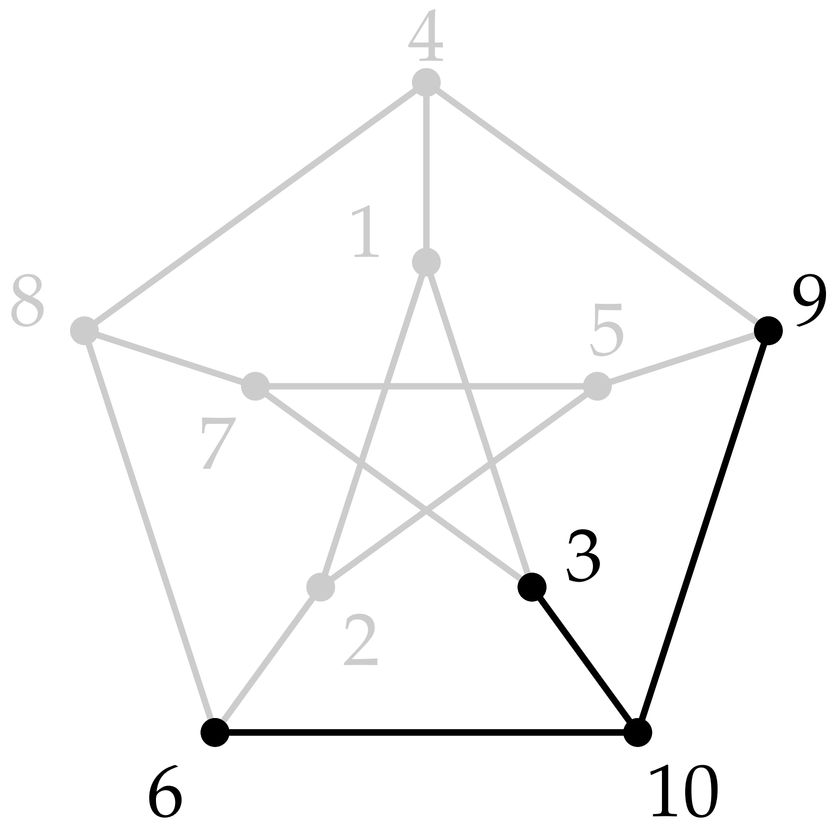
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

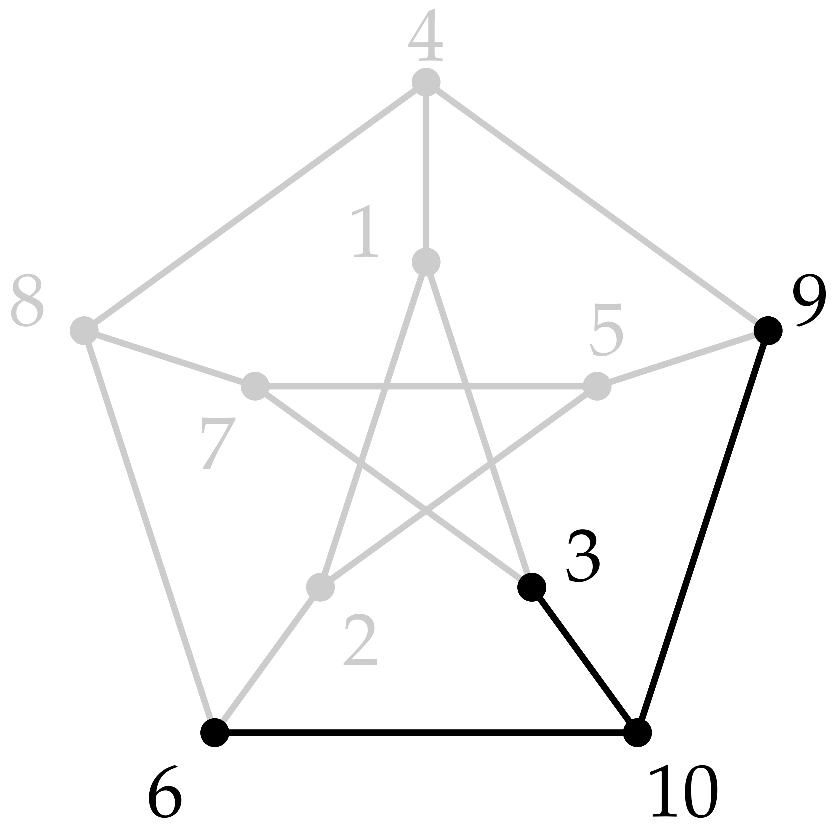
Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

Planar Storyplan – Definition, Part I



In each step the drawing of the subgraph that is shown has no crossings.

- In each step, we add *one* vertex.
- This vertex stays visible until all its neighbors have been presented. Then the vertex dissapears and will not appear any more.

For a given (non-planar) graph, find an order of the vertices that yields a *planar storyplan* (if such an order exists).

Planar Storyplan – Definition, Part II

total order
of vertices

$$\langle \tau, \{D_i\}_{i \in [n]} \rangle$$

sequence of drawings
(we also call them *frames*)

Planar Storyplan – Definition, Part II

total order of vertices

$$\langle \tau, \boxed{\{D_i\}_{i \in [n]}} \rangle$$

sequence of drawings (we also call them *frames*)

Planar Storyplan – Definition, Part II

total order of vertices

$\langle \tau, \boxed{\{D_i\}_{i \in [n]}} \rangle$

sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

Planar Storyplan – Definition, Part II

total order of vertices

$$\langle \tau, \boxed{\{D_i\}_{i \in [n]}} \rangle$$

sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is planar;

Planar Storyplan – Definition, Part II

total order of vertices

$$\langle \tau, \boxed{\{D_i\}_{i \in [n]}} \rangle$$

sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is planar;

(III) the point representing a vertex v does not change during the steps where v is visible;

Planar Storyplan – Definition, Part II

total order of vertices

$$\langle \tau, \boxed{\{D_i\}_{i \in [n]}} \rangle$$

sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is planar;

(III) the point representing a vertex v does not change during the steps where v is visible;

(IV) the curve representing an edge e is the same over all drawings that contain e.

~~Planar~~ Storyplan – Definition, Part II
Outerplanar

total order of vertices

$\langle \tau, \{D_i\}_{i \in [n]} \rangle$

sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is ~~planar~~; outerplanar

(III) the point representing a vertex v does not change during the steps where v is visible;

(IV) the curve representing an edge e is the same over all drawings that contain e.

~~Planar~~ Storyplan – Definition, Part II
~~Outerplanar~~
Forest

total order of vertices

$$\langle \tau, \{D_i\}_{i \in [n]} \rangle$$

sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is ~~planar;~~ ~~outerplanar~~ a crossing-free drawing of a forest

(III) the point representing a vertex v does not change during the steps where v is visible;

(IV) the curve representing an edge e is the same over all drawings that contain e.

~~Planar~~ Storyplan – Definition, Part II
~~Outerplanar~~
Forest

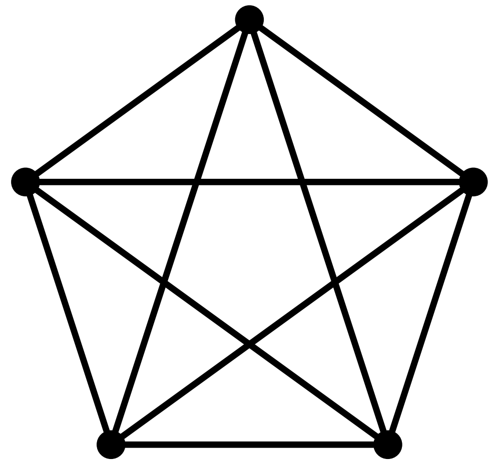
total order of vertices $\langle \tau, \{D_i\}_{i\in[n]} \rangle$ sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is ~~planar~~; ~~outerplanar~~ a crossing-free drawing of a forest

(III) the point representing a vertex v does not change during the steps where v is visible;

(IV) the curve representing an edge e is the same over all drawings that contain e.



K_5 has no planar storyplan :-(

~~Planar~~ Storyplan – Definition, Part II
~~Outerplanar~~
Forest

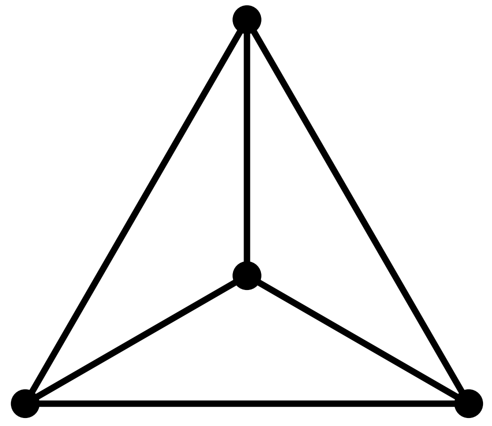
total order of vertices $\langle \tau, \{D_i\}_{i\in[n]} \rangle$ sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is ~~planar;~~ ~~outerplanar~~ a crossing-free drawing of a forest

(III) the point representing a vertex v does not change during the steps where v is visible;

(IV) the curve representing an edge e is the same over all drawings that contain e.



K_4 has no outerplanar storyplan :-(

~~Planar~~ Storyplan – Definition, Part II
~~Outerplanar~~
Forest

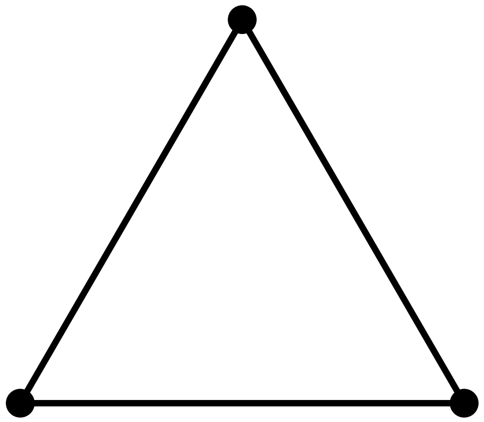
total order of vertices $\langle \tau, \{D_i\}_{i \in [n]} \rangle$ sequence of drawings (we also call them *frames*)

(I) each D_i contains all vertices visible at step i;

(II) each D_i is ~~planar~~; ~~outerplanar~~ a crossing-free drawing of a forest

(III) the point representing a vertex v does not change during the steps where v is visible;

(IV) the curve representing an edge e is the same over all drawings that contain e.



K_3 has no forest storyplan :-(

Related Work

	Storyplan	Graph Stories	Streamed Graphs
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:			
Condition:			
Aim:			

Related Work

	Storyplan	Graph Stories	Streamed Graphs
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph	• a graph
Condition:			
Aim:			

Related Work

	Storyplan	Graph Stories	Streamed Graphs
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph • vertex order	• a graph • edge order
Condition:			
Aim:			

Related Work

	Storyplan	**Graph Stories**	**Streamed Graphs**
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph • vertex order • fixed amount of steps for vertices	• a graph • edge order • fixed amount of steps for edges
Condition:			
Aim:			

Related Work

	Storyplan	**Graph Stories**	**Streamed Graphs**
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph • vertex order • fixed amount of steps for vertices	• a graph • edge order • fixed amount of steps for edges

Condition: Positions of vertices and edges do not change once they have been placed.

Aim:

Related Work

	Storyplan	**Graph Stories**	**Streamed Graphs**
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph • vertex order • fixed amount of steps for vertices	• a graph • edge order • fixed amount of steps for edges
Condition:	Positions of vertices and edges do not change once they have been placed.		
Aim:	Find a vertex order and drawings s.t. …		

…in each step the drawings are planar (or are crossing-free drawings of trees).

Related Work

	Storyplan	Graph Stories	Streamed Graphs
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph • vertex order • fixed amount of steps for vertices	• a graph • edge order • fixed amount of steps for edges

Condition: Positions of vertices and edges do not change once they have been placed.

Aim:	Find a vertex order and drawings s.t. …	Place vertices on a grid of small size s.t. …	

…in each step the drawings are planar (or are crossing-free drawings of trees).

Related Work

	Storyplan	**Graph Stories**	**Streamed Graphs**
	Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis [GD'22]	Borrazzo, Da Lozzo, Di Battista, Frati, and Patrignani [JGAA'20] Di Battista, Didimo, Grilli, Grosso, Ortali, Patrignani, and Tappini [GD'22]	Binucci, Brandes, Di Battista, Didimo, Gaertler, Palladino, Patrignani, Symvonis, and Zweig [IPL'12] Da Lozzo and Rutter [TCS'19]
Given:	• a graph	• a graph • vertex order • fixed amount of steps for vertices	• a graph • edge order • fixed amount of steps for edges

Condition: Positions of vertices and edges do not change once they have been placed.

Aim:	Find a vertex order and drawings s.t. …	Place vertices on a grid of small size s.t. …	Place edges s.t. …

…in each step the drawings are planar (or are crossing-free drawings of trees).

Related Work on Storyplans

Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis
On the complexity of the storyplan problem, GD'22

Related Work on Storyplans

Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis
On the complexity of the storyplan problem, GD'22

- It is NP-complete to decide whether a given graph admits a planar storyplan.

Related Work on Storyplans

Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis
On the complexity of the storyplan problem, GD'22

- It is NP-complete to decide whether a given graph admits a planar storyplan.

- Two parametrized algorithms:
  - w.r.t. the vertex cover number
  - w.r.t. the feedback edge set number

Related Work on Storyplans

Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis
On the complexity of the storyplan problem, GD'22

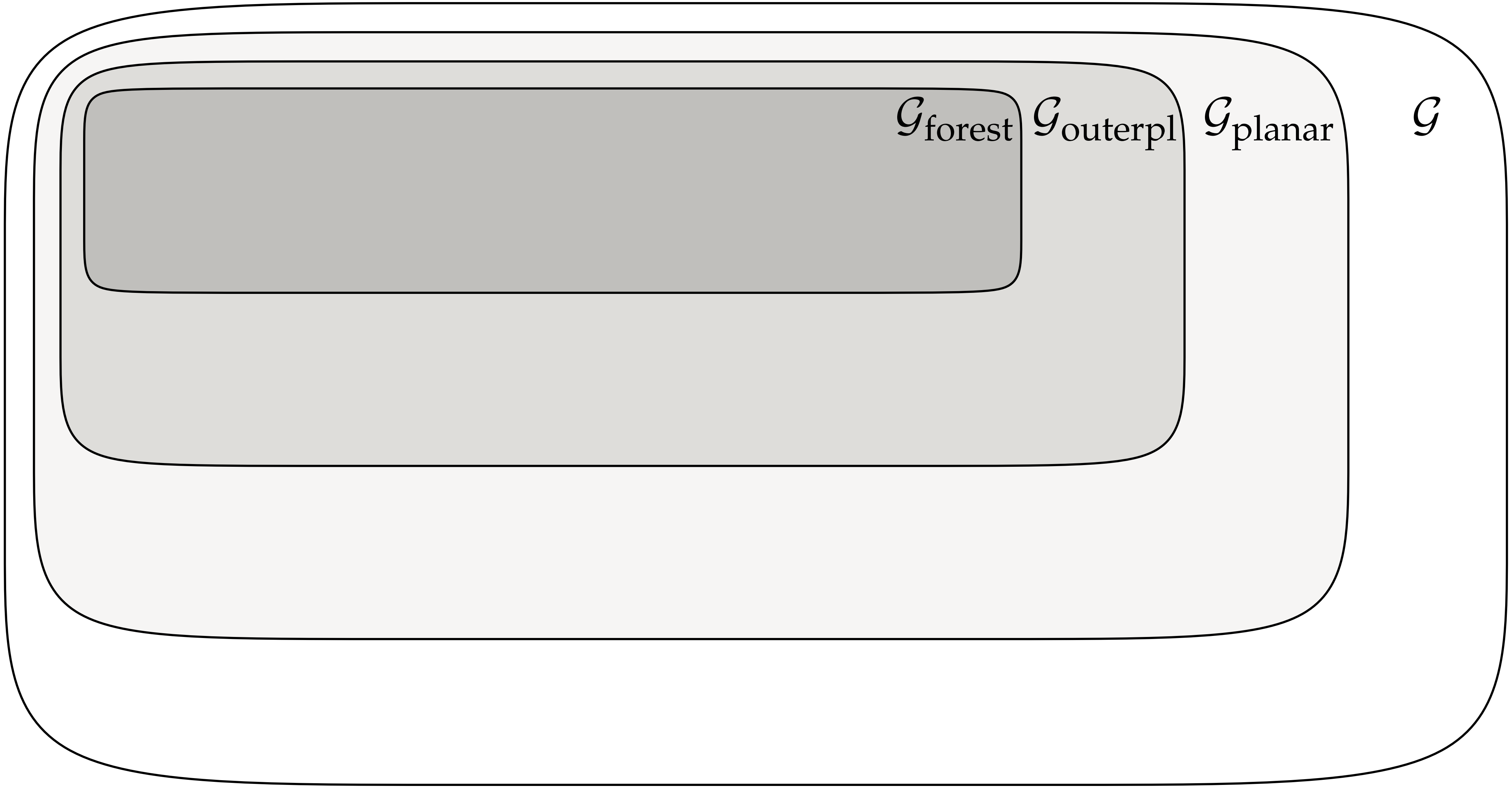
- It is NP-complete to decide whether a given graph admits a planar storyplan.
- Two parametrized algorithms:
  - w.r.t. the vertex cover number
  - w.r.t. the feedback edge set number
- Partial 3-trees always admit a planar storyplan.

Related Work on Storyplans

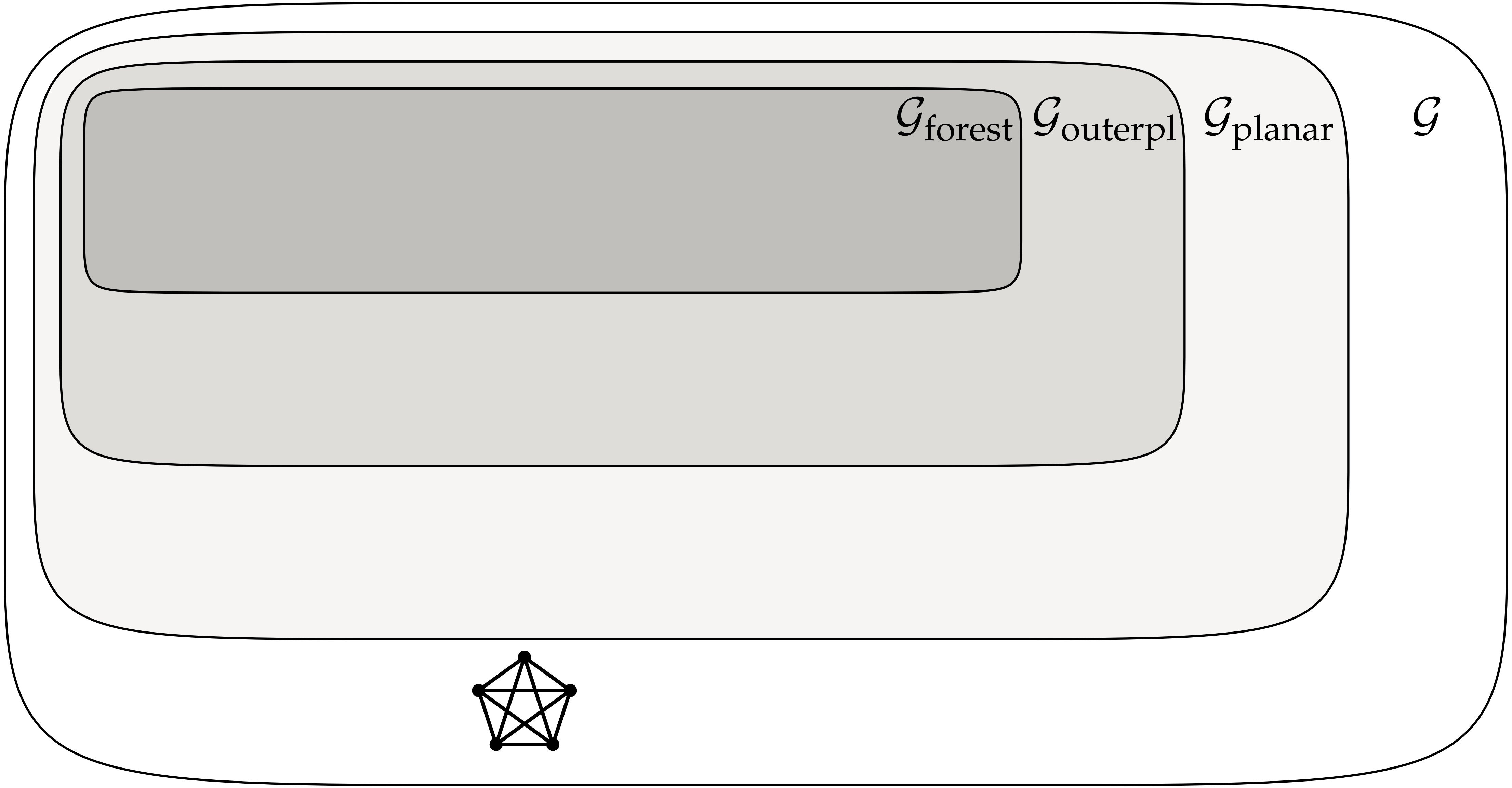
Binucci, Di Giacomo, Lenhart, Liotta, Montecchiani, Nöllenburg, and Symvonis
On the complexity of the storyplan problem, GD'22

- It is NP-complete to decide whether a given graph admits a planar storyplan.
- Two parametrized algorithms:
  - w.r.t. the vertex cover number
  - w.r.t. the feedback edge set number
- Partial 3-trees always admit a planar storyplan.
- Even if the total vertex order is given, the problem is still NP-complete.

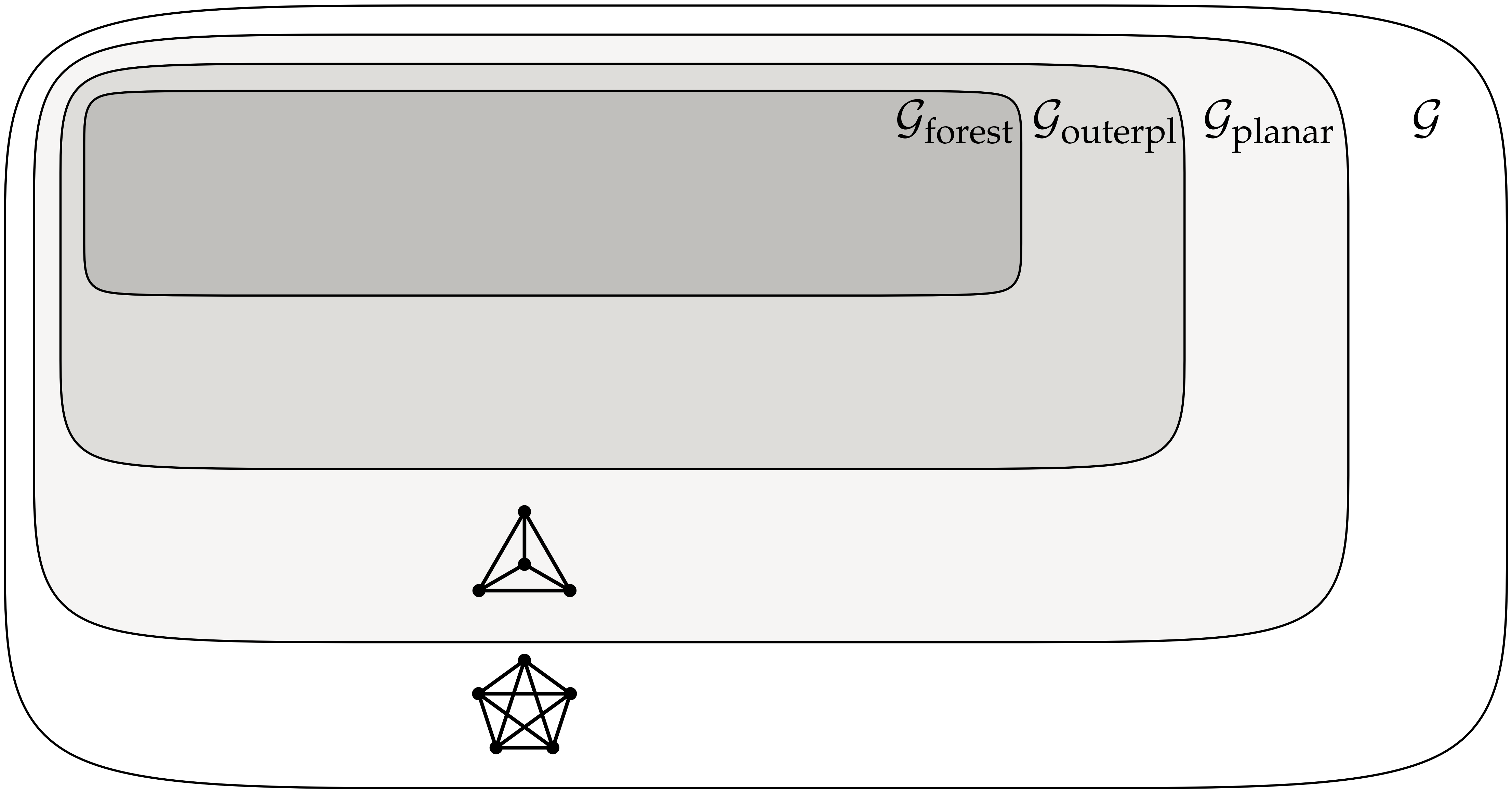
Our Contribution



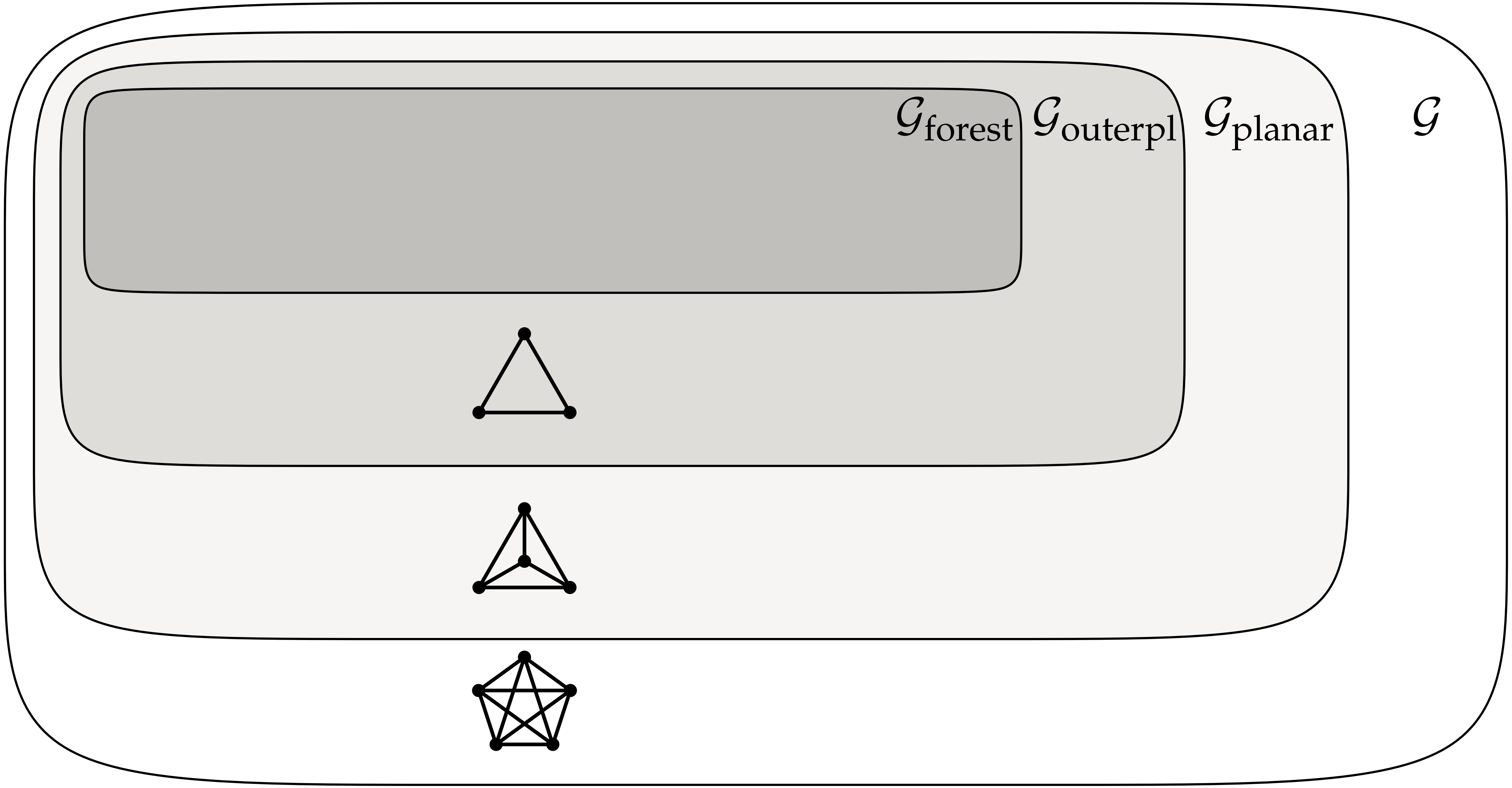
Our Contribution



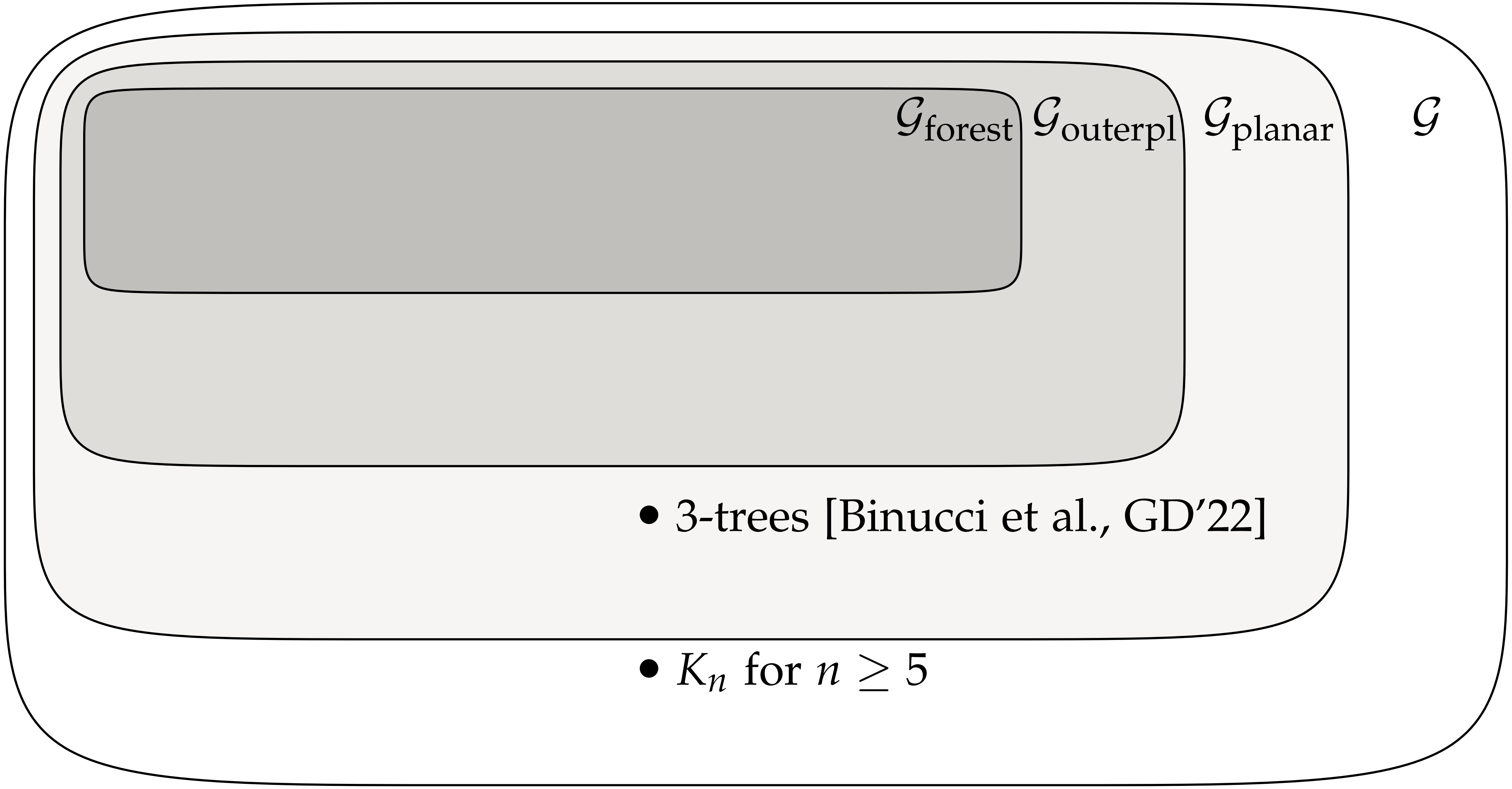
Our Contribution



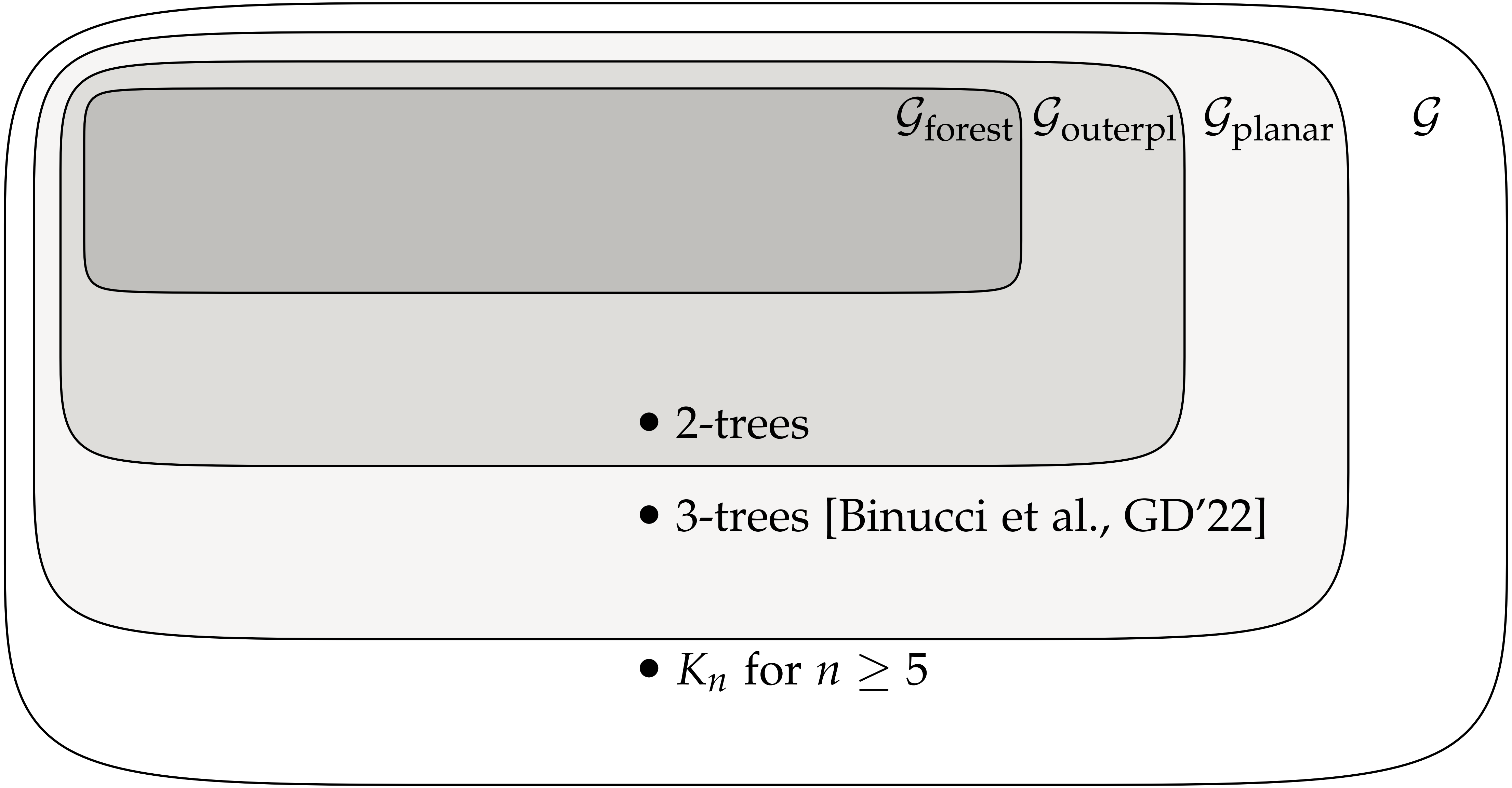
Our Contribution



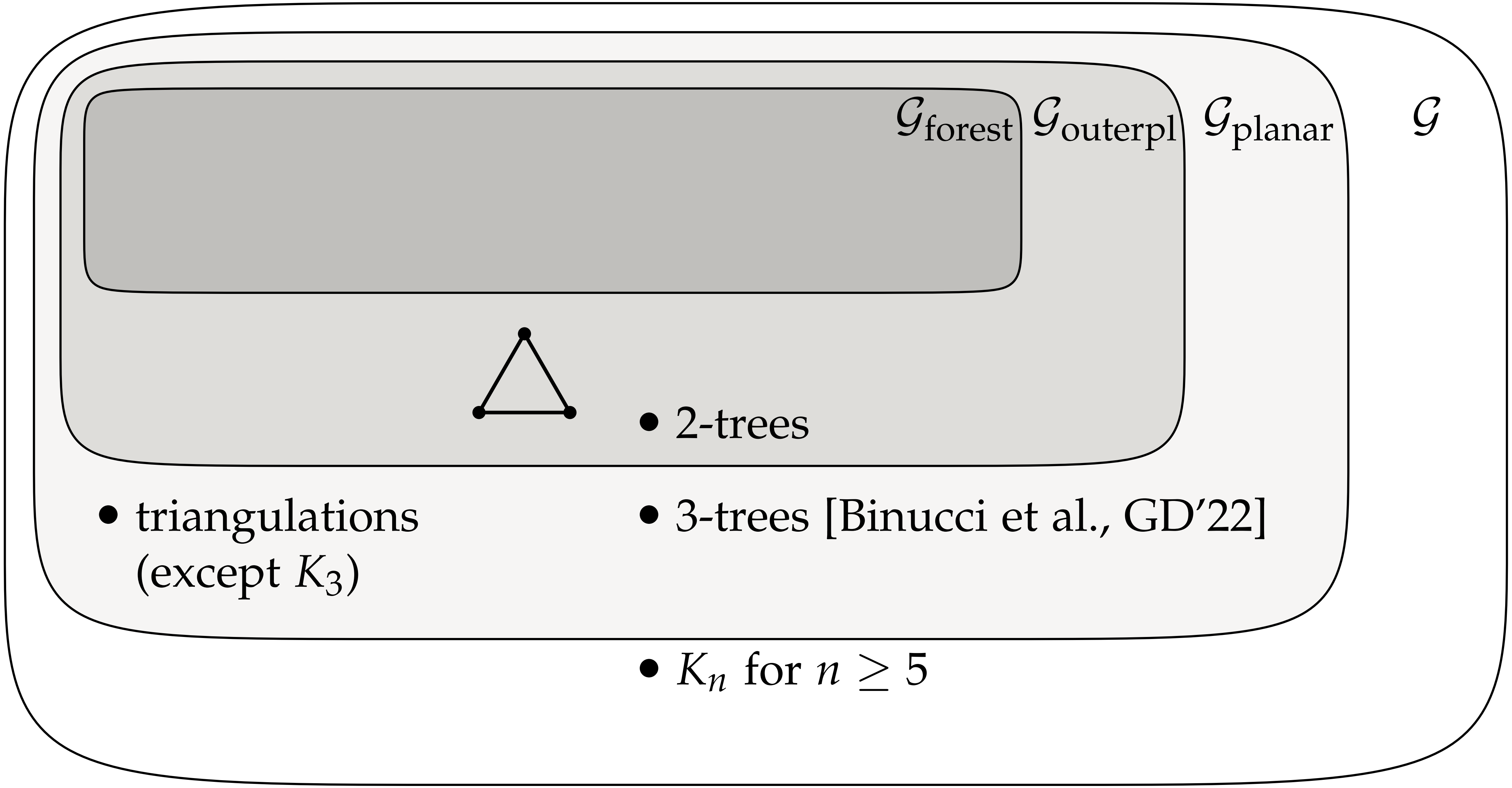
Our Contribution



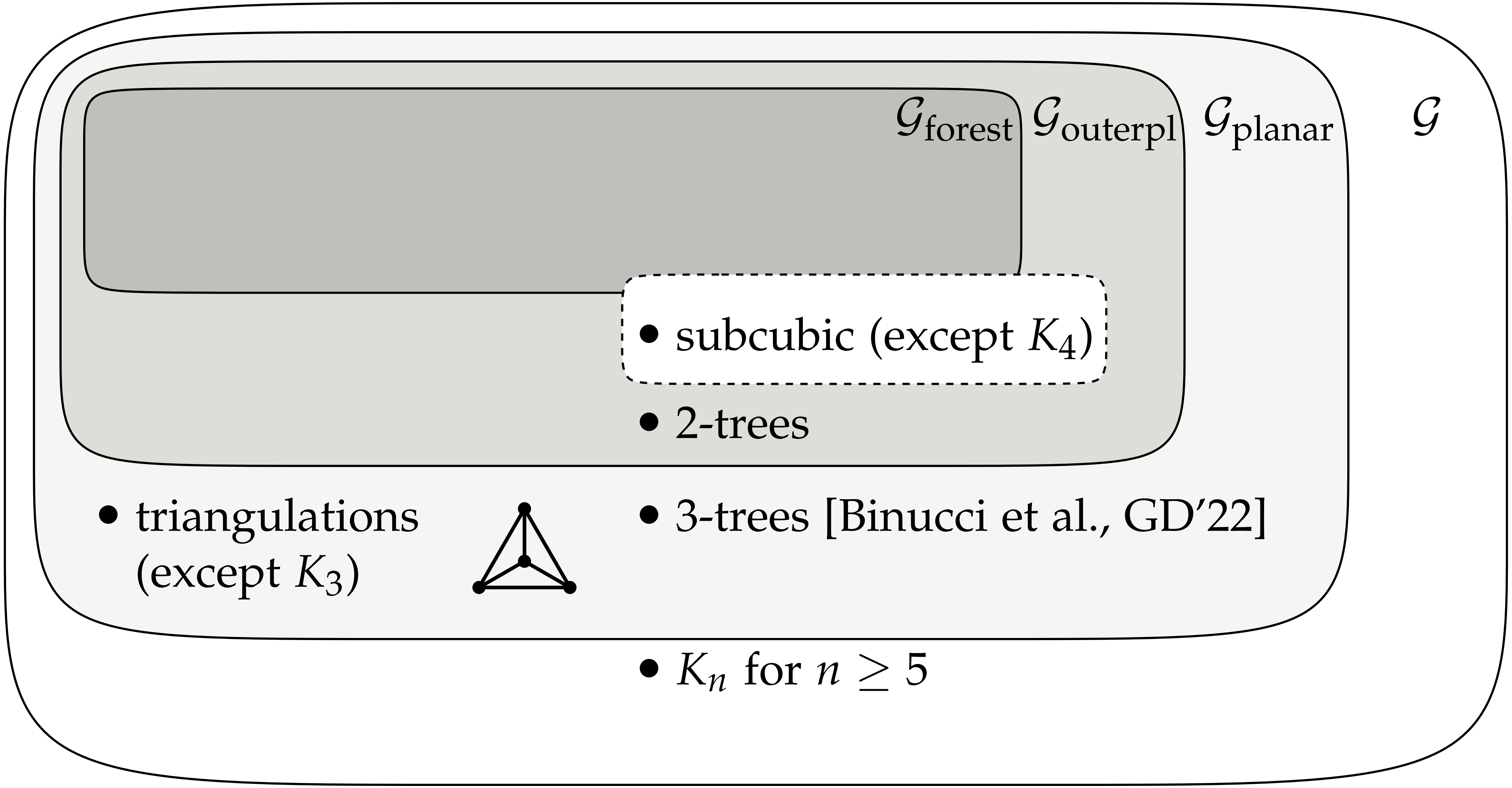
Our Contribution



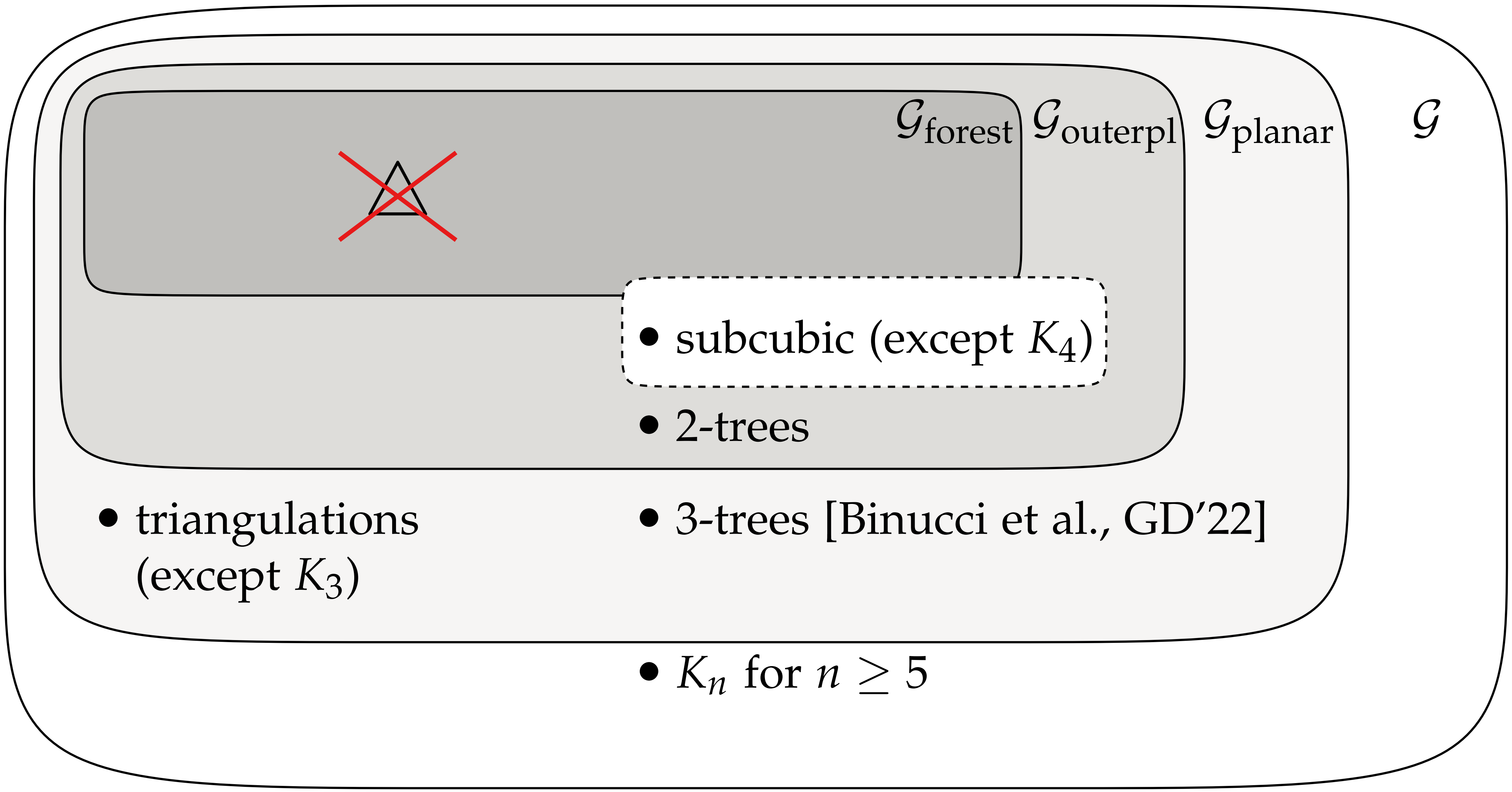
Our Contribution



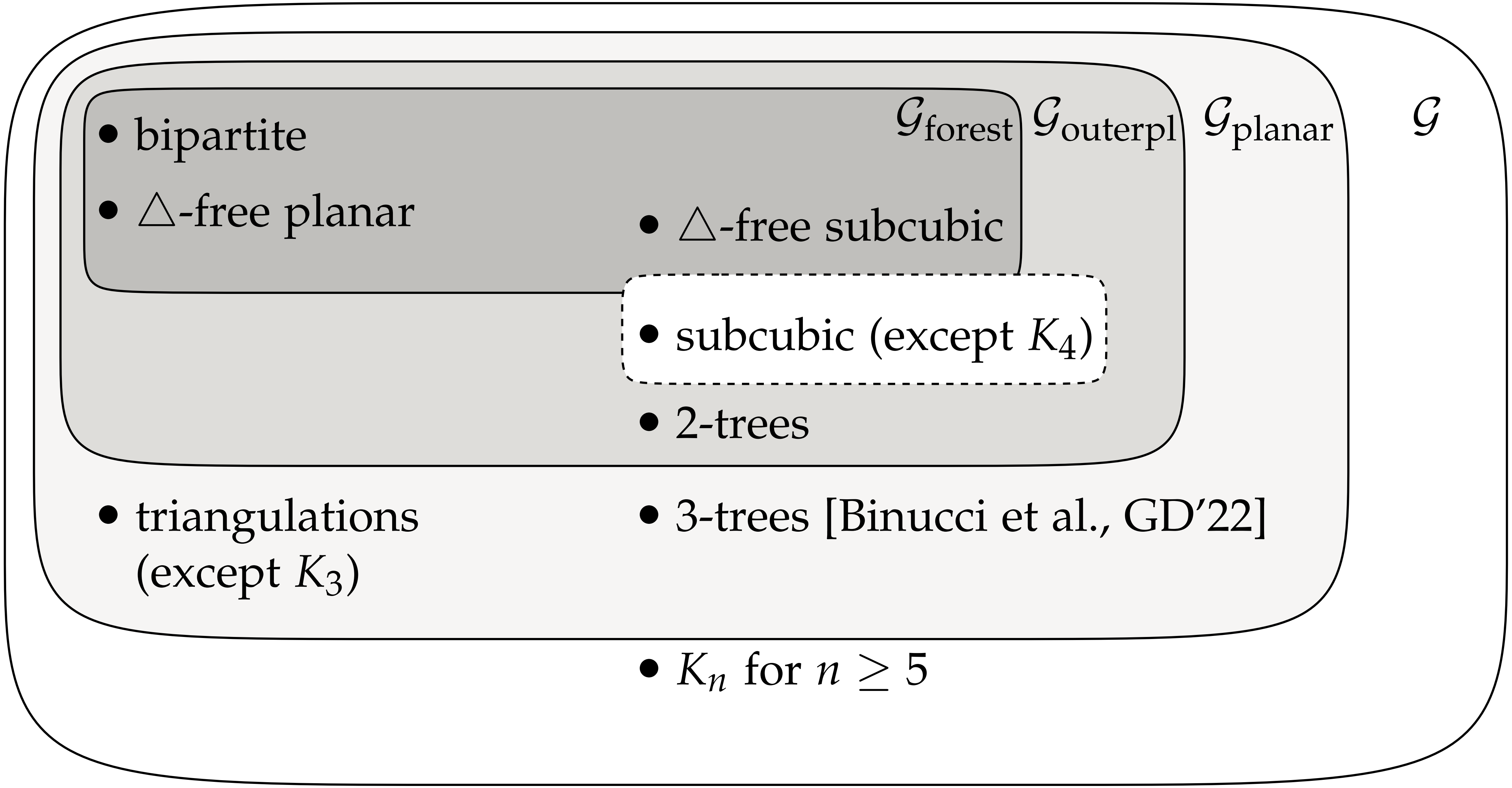
Our Contribution



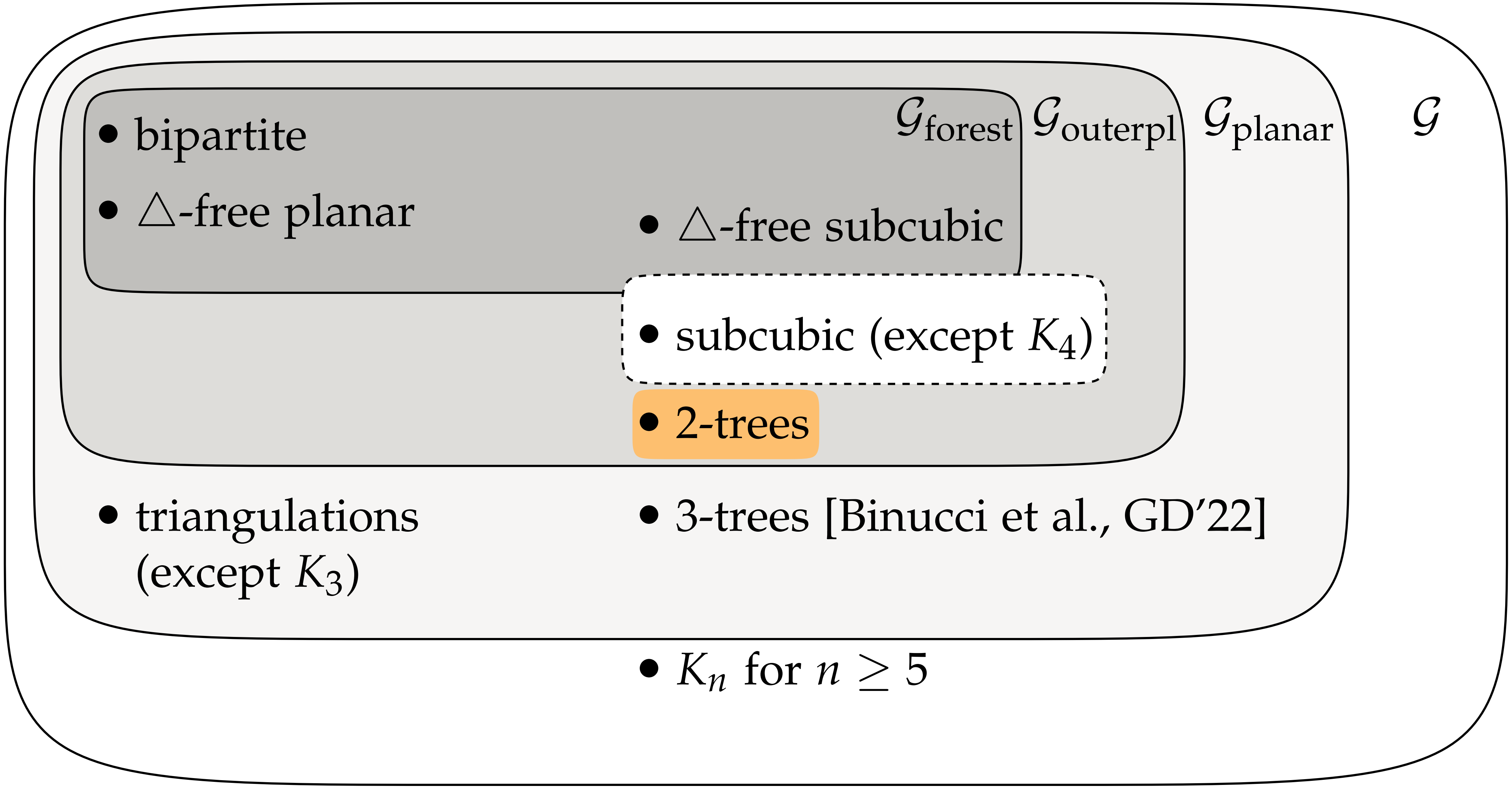
Our Contribution



Our Contribution



Our Contribution



Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

If a graph admits a storyplan,
its subgraph also admits a storyplan.

Outerplanar Storyplans of 2-Trees

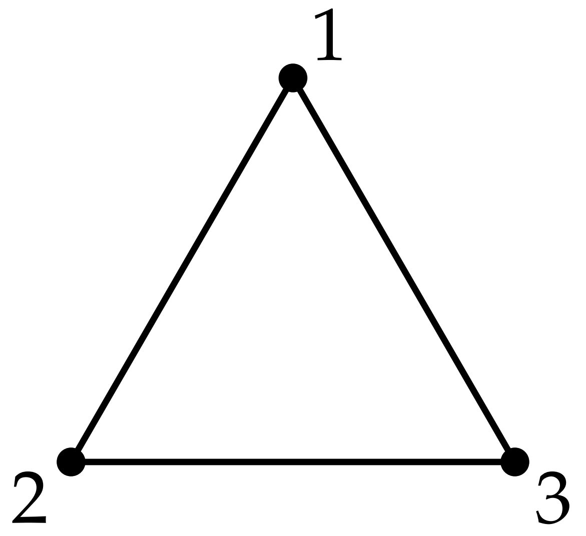
Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

If a graph admits a storyplan,
its subgraph also admits a storyplan.

Thus, it is enough to consider 2-trees.

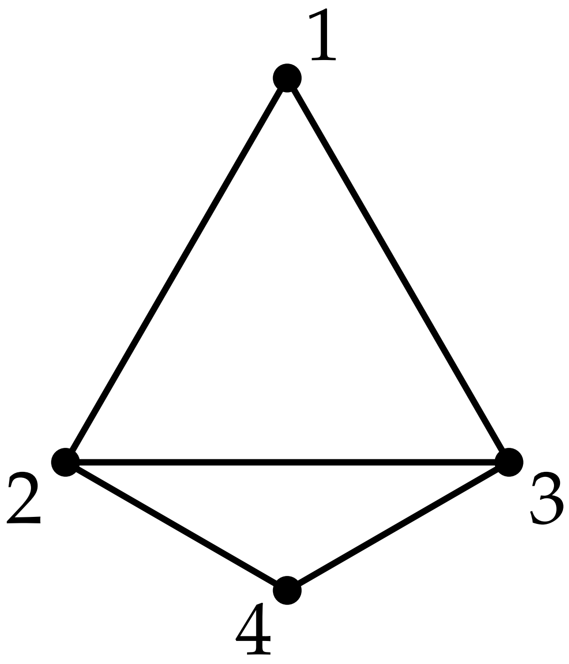
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.



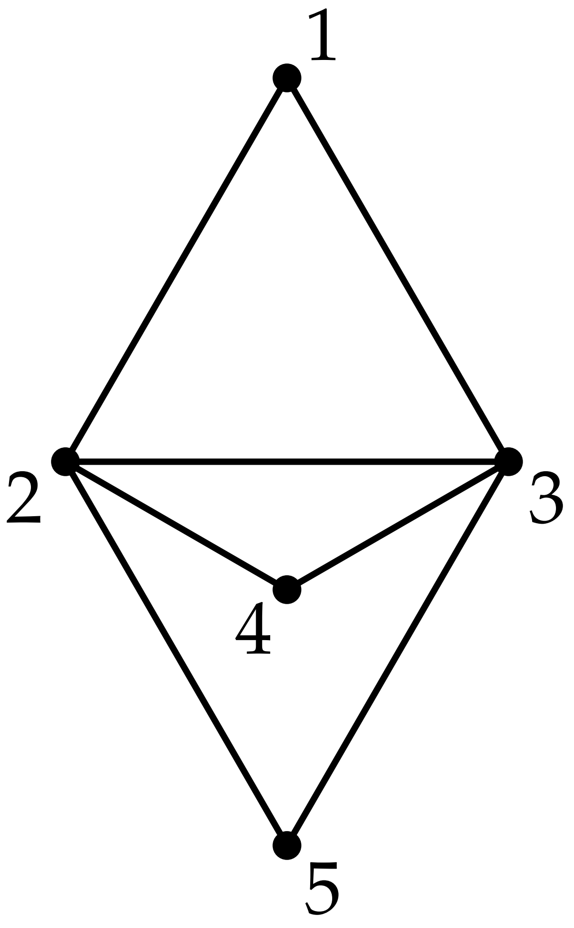
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.



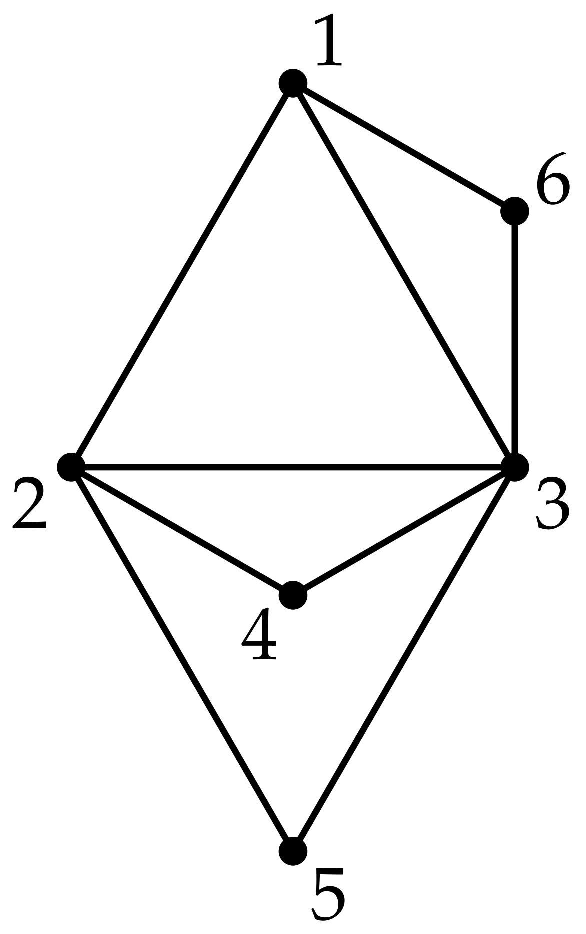
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.



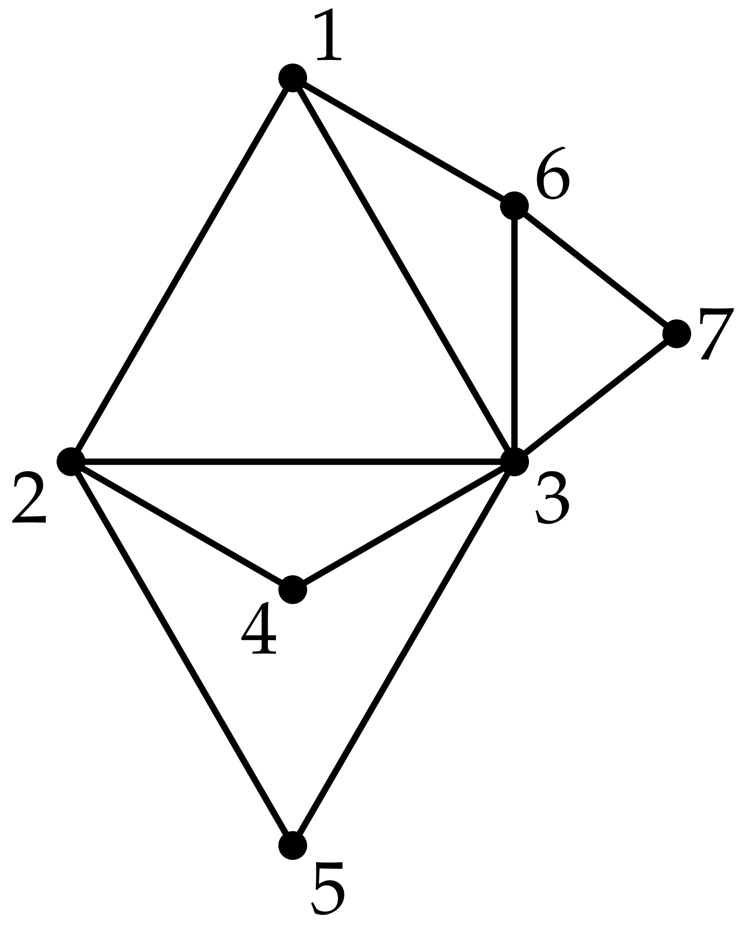
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.



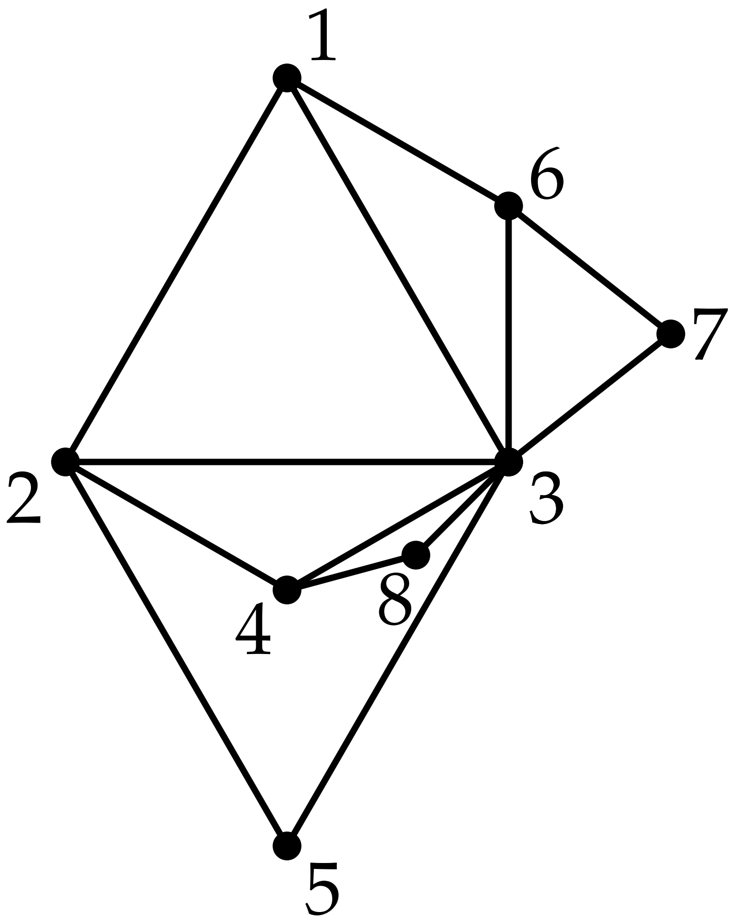
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.



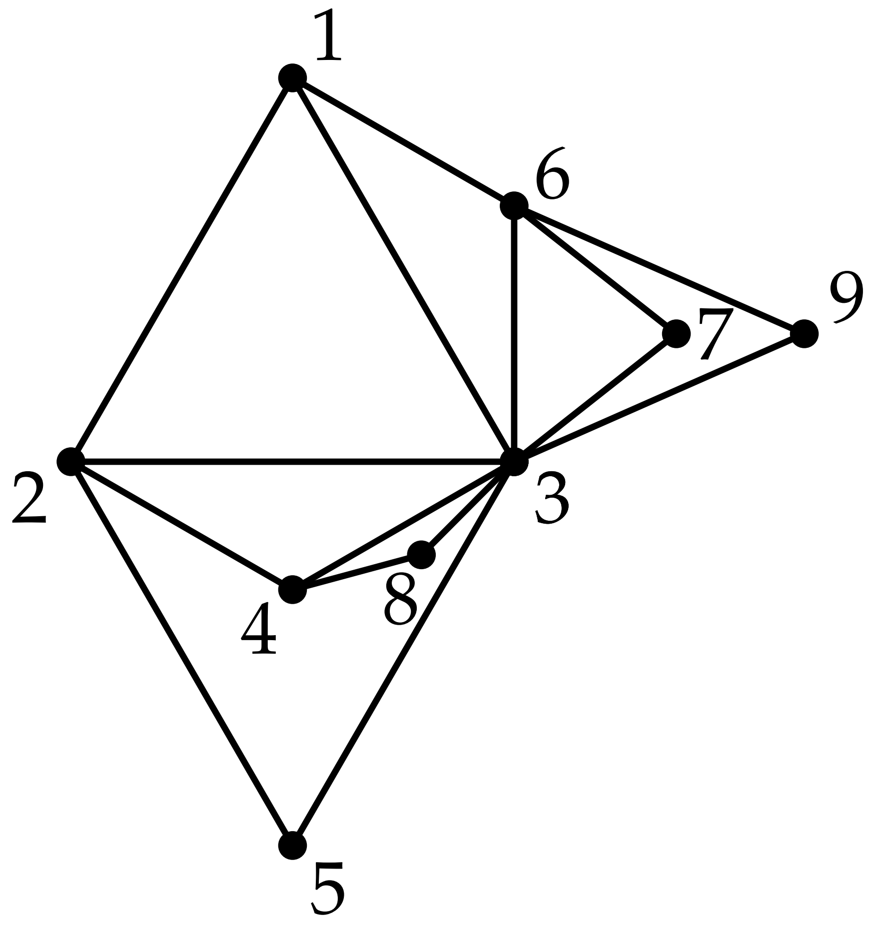
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.



Outerplanar Storyplans of 2-Trees

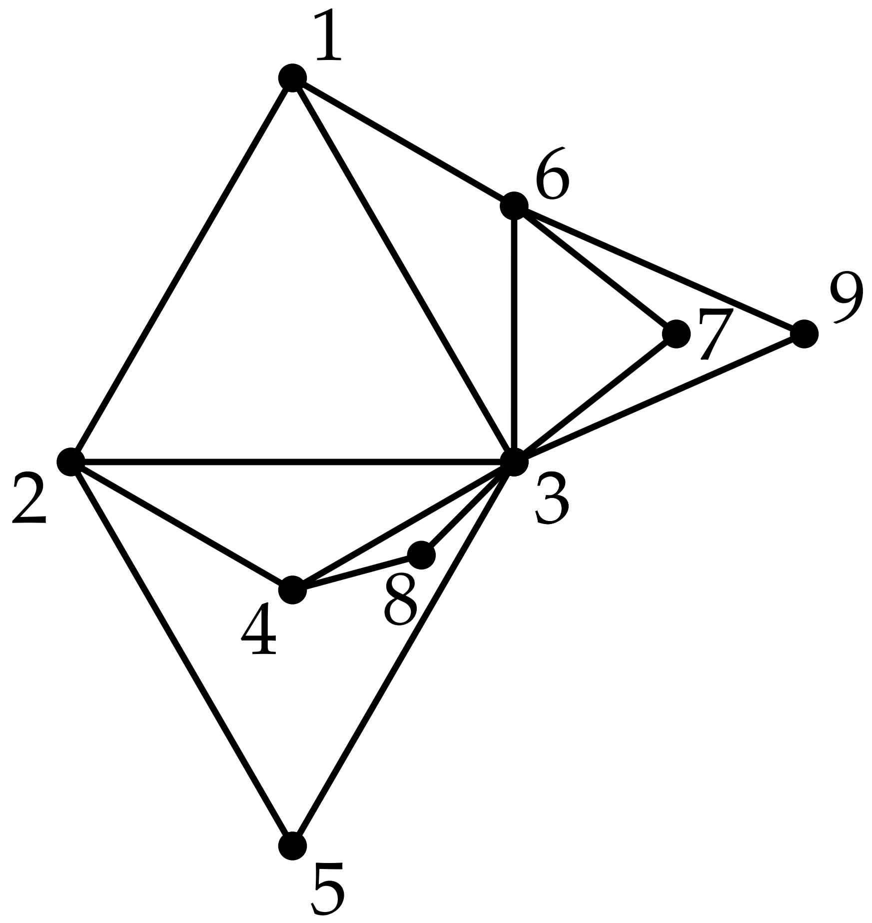
Theorem.
Every (partial) **2-tree** admits a straight-line outerplanar storyplan.



Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

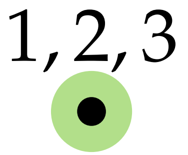
- Using the stacking order of G construct a *tree decomposition* of G.

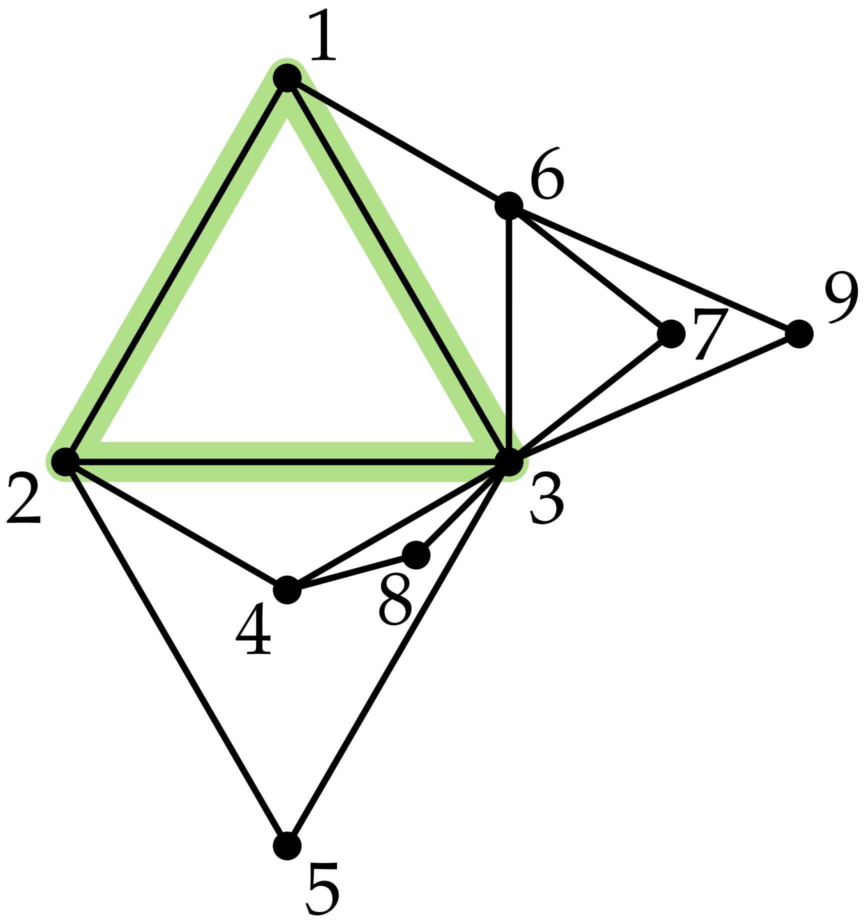


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

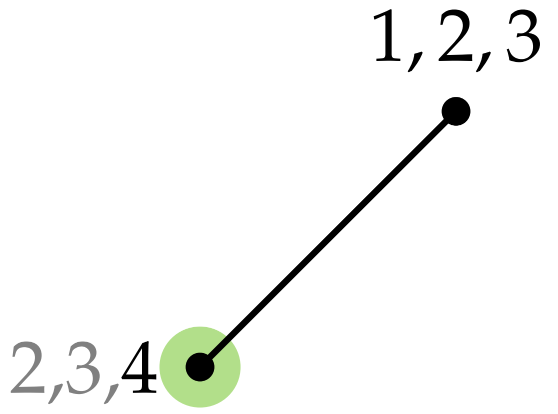


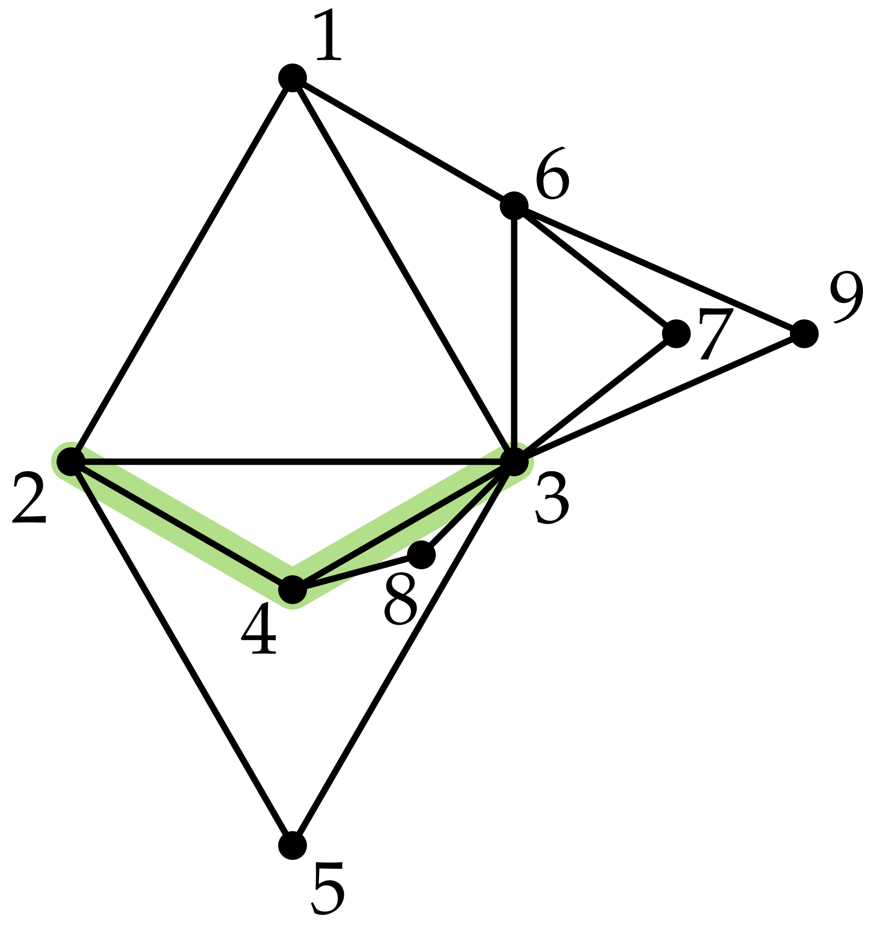


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

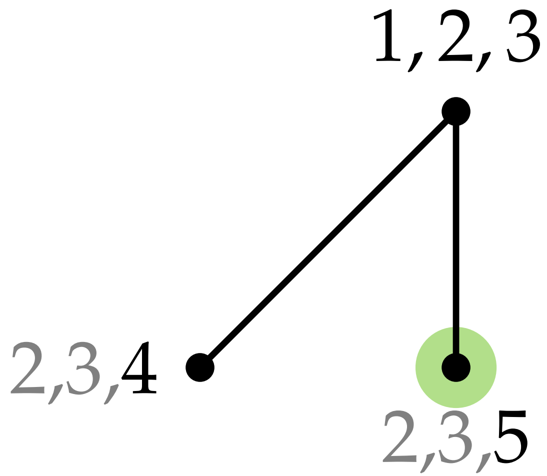


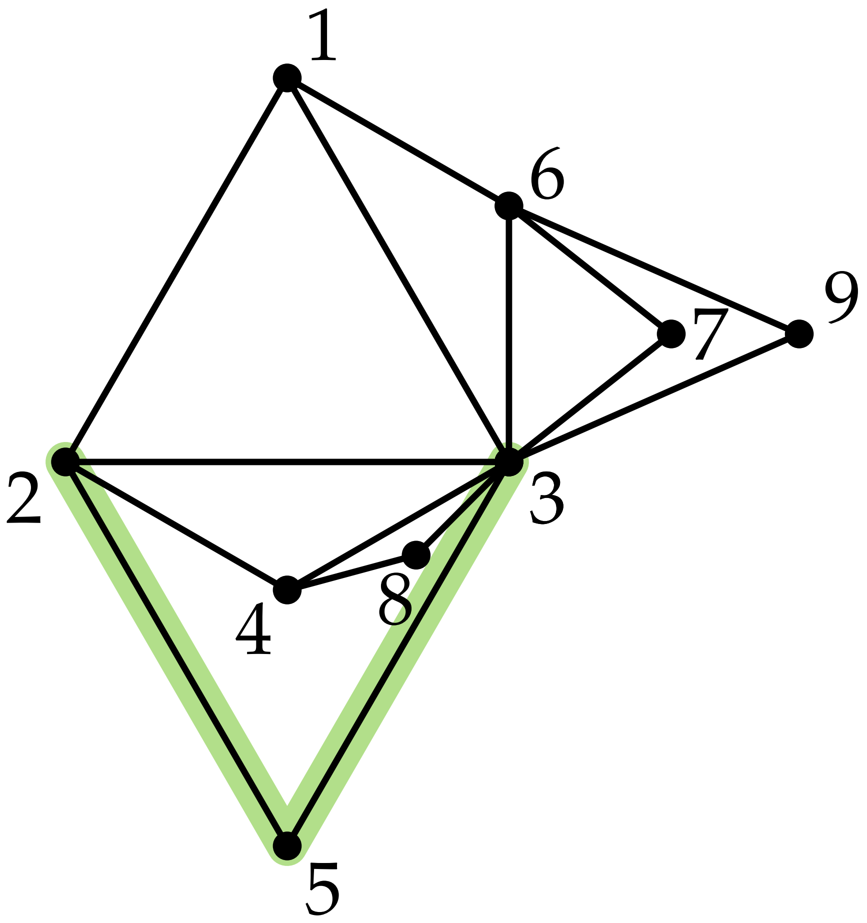


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

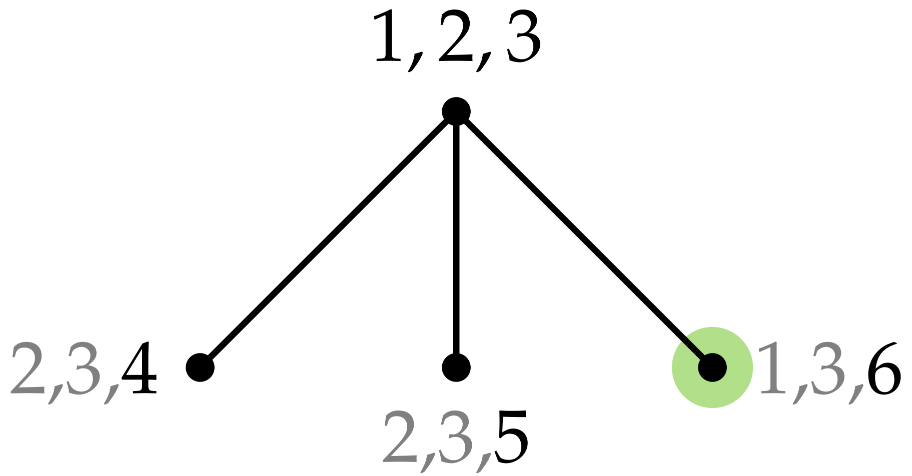


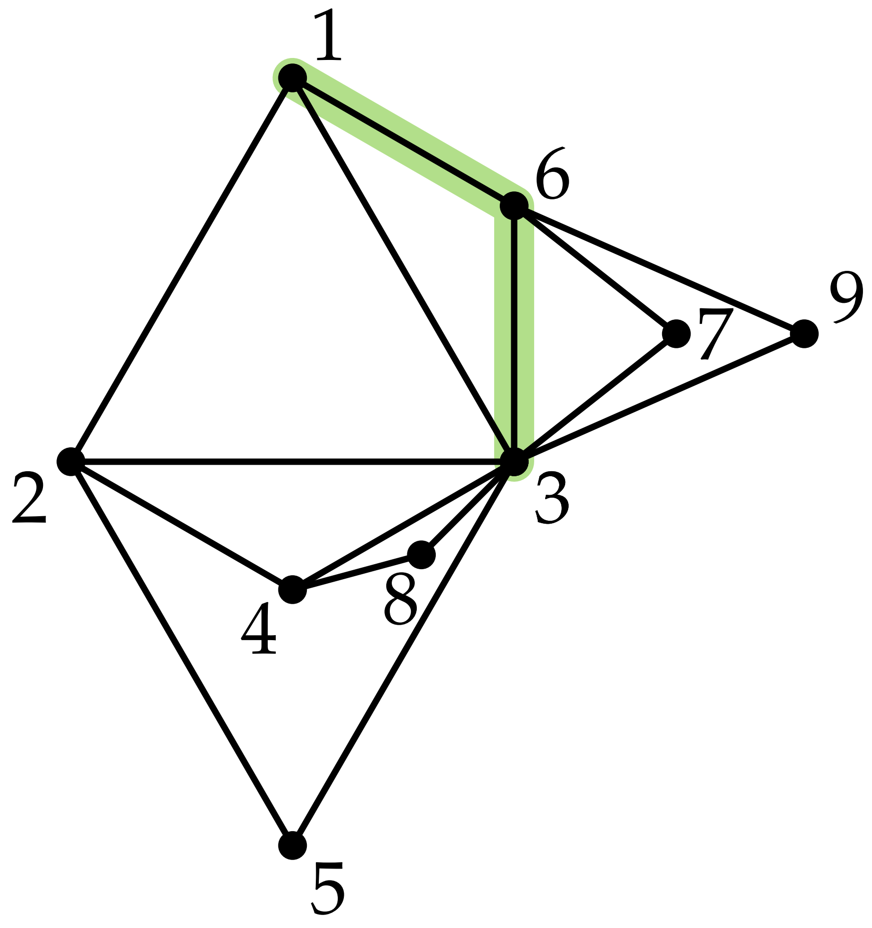


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

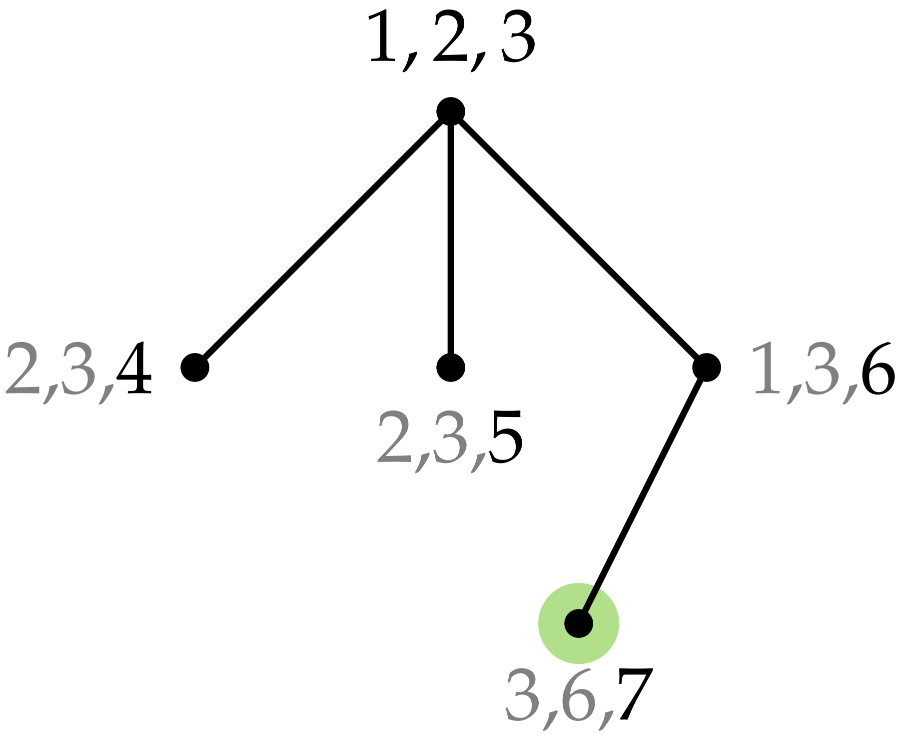


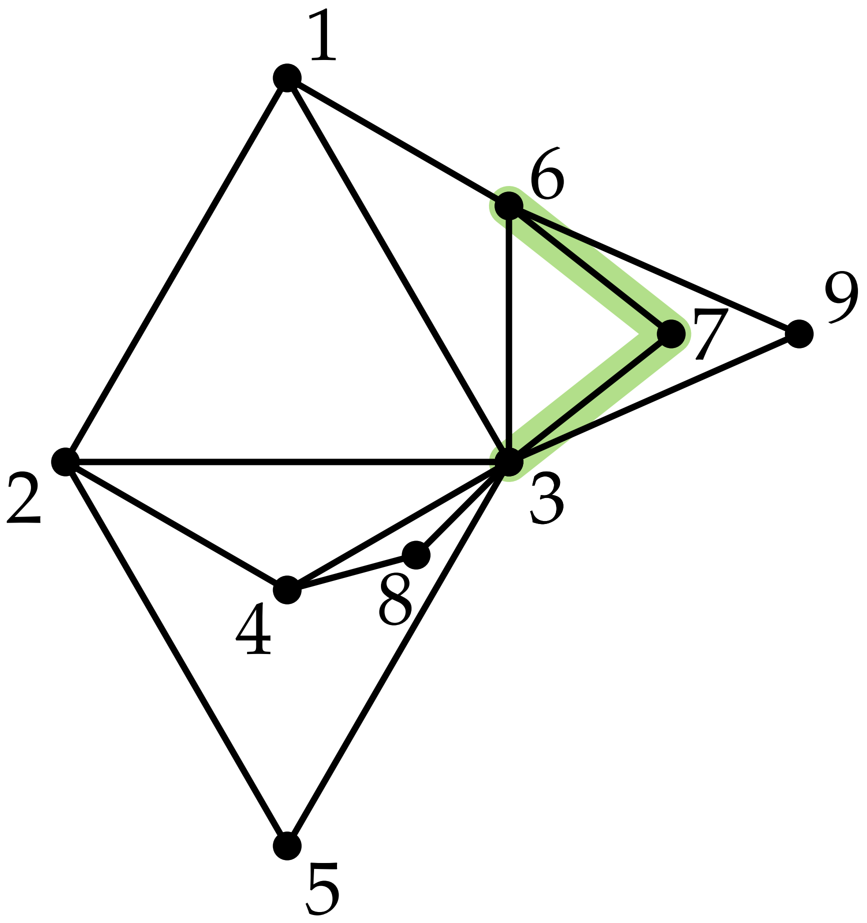


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

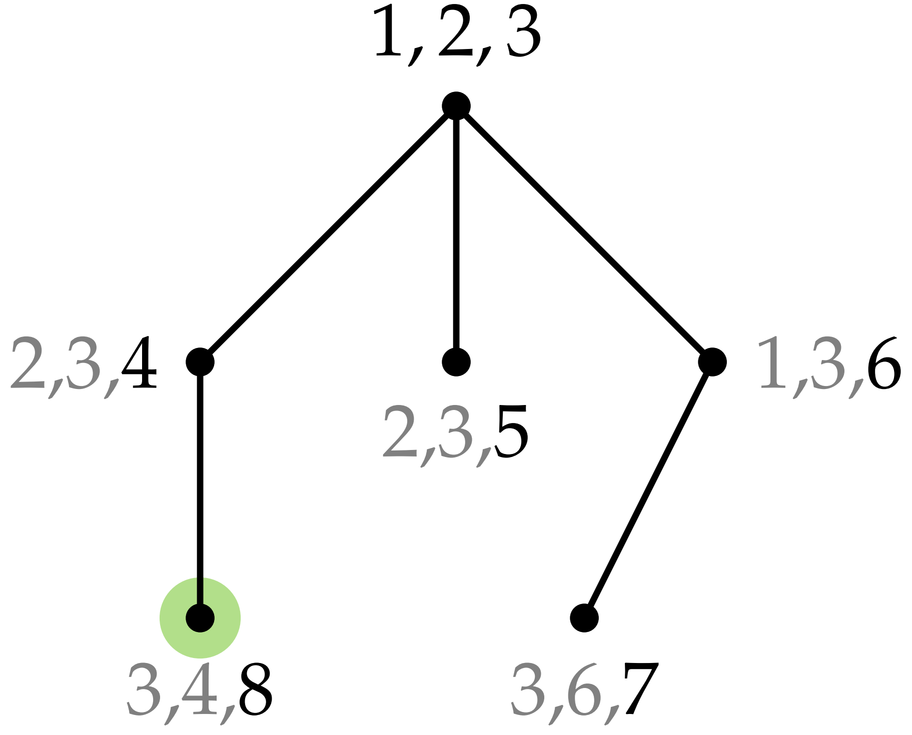


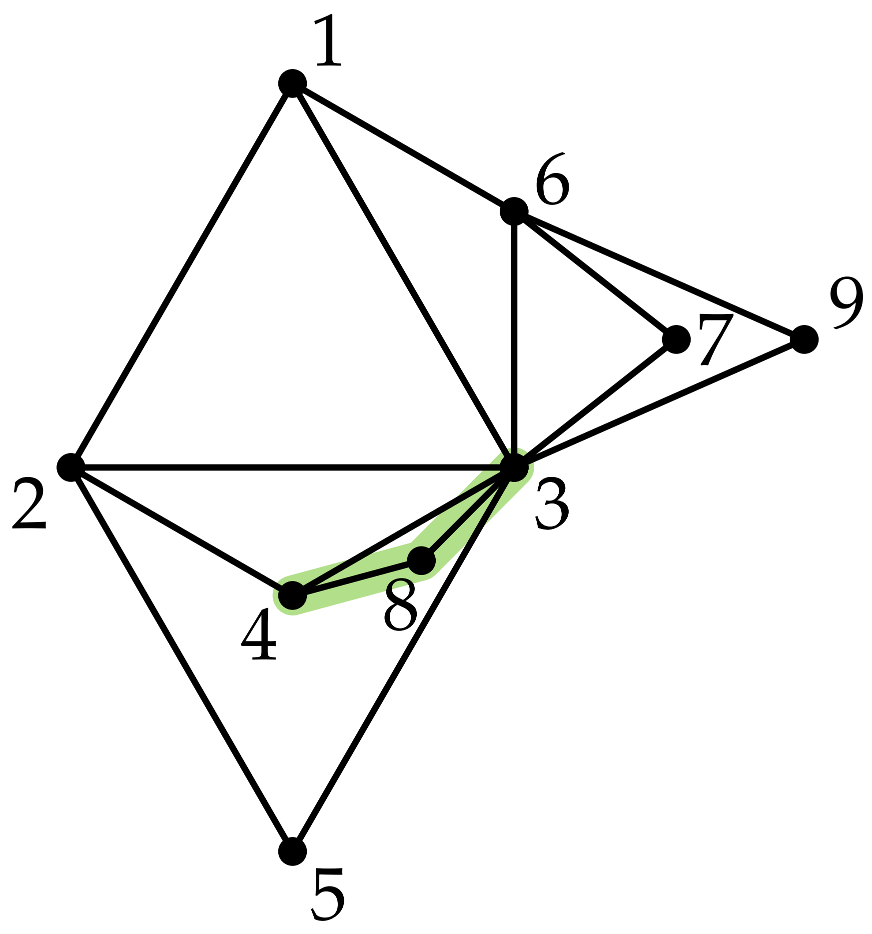


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

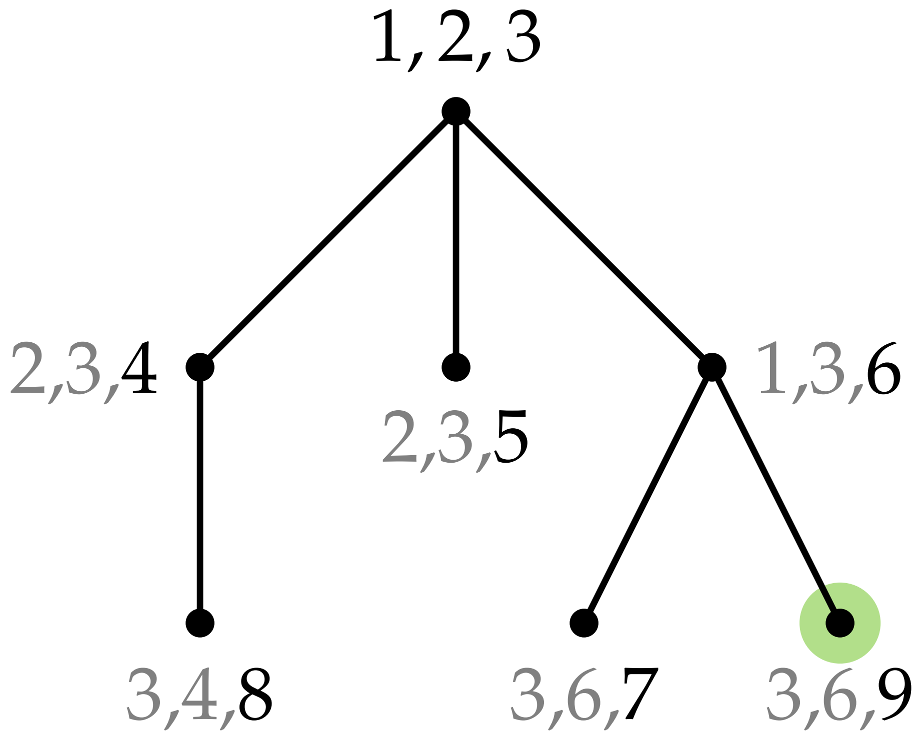


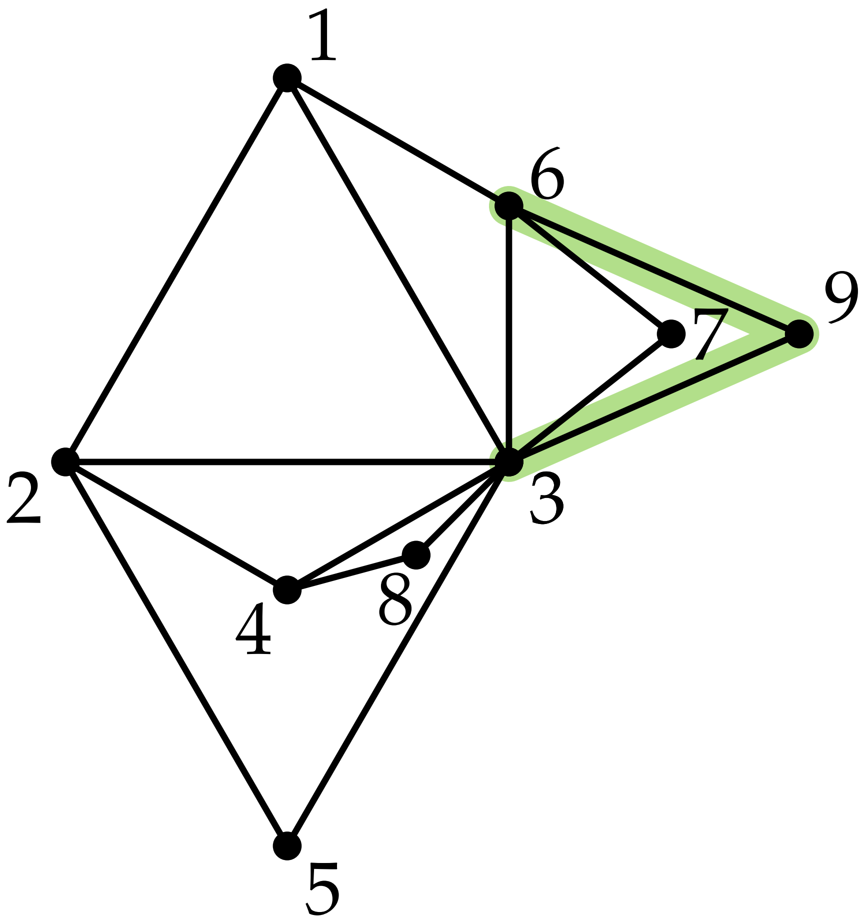


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

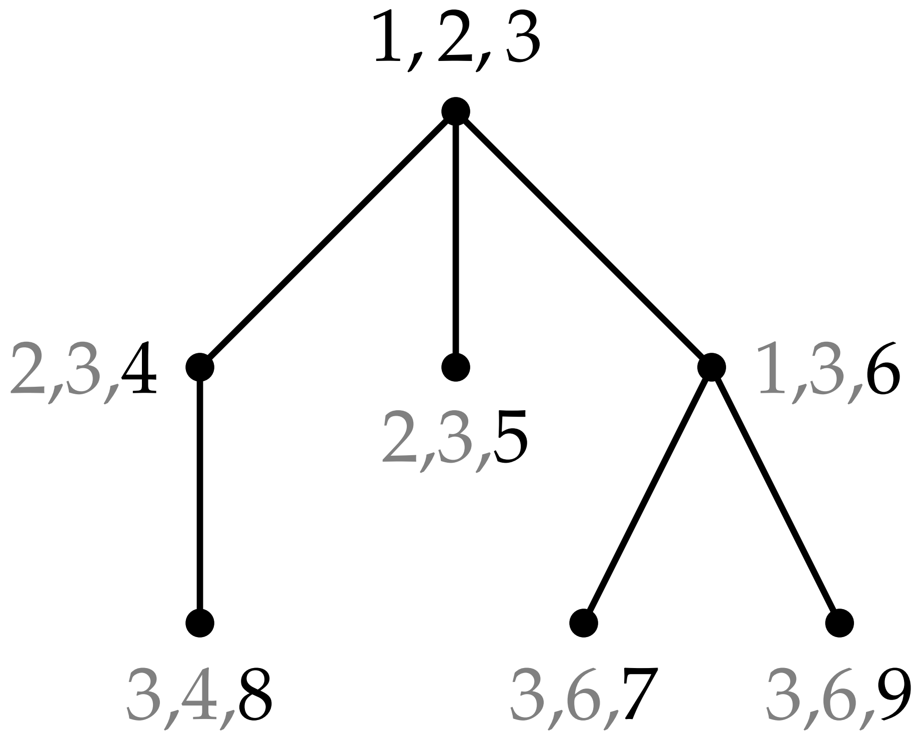


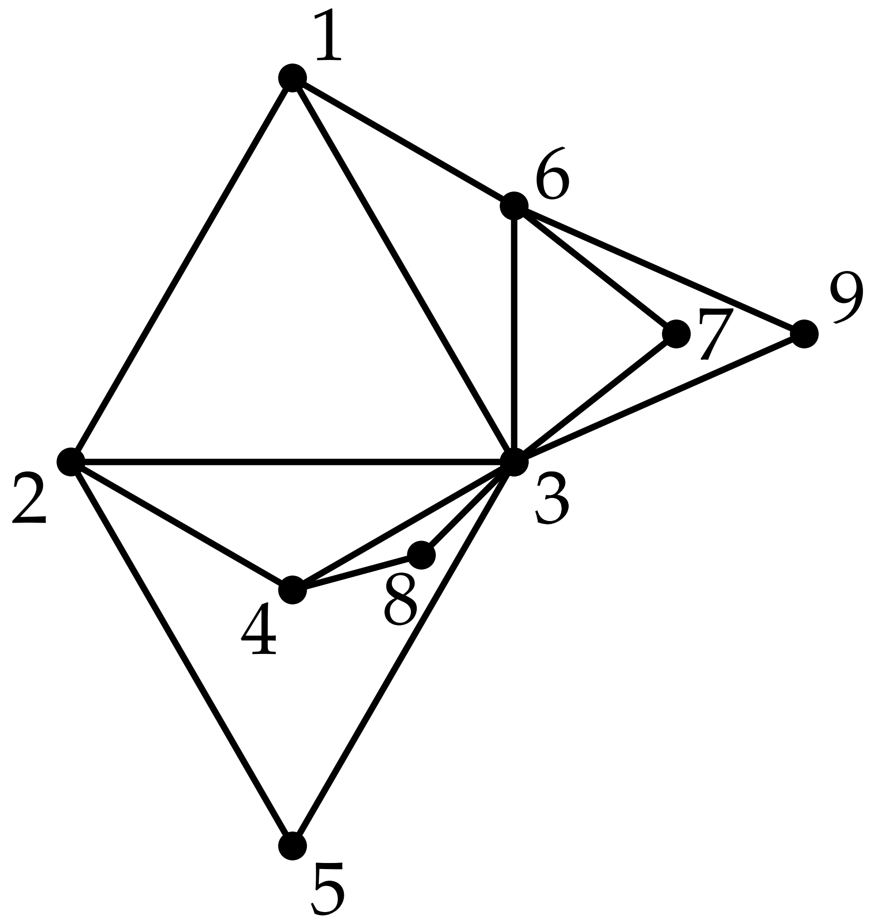


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

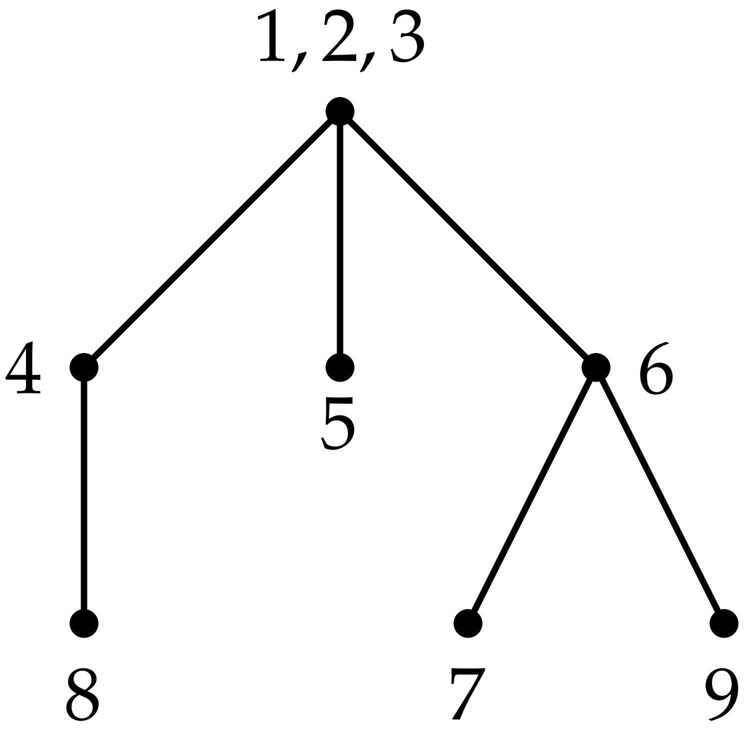


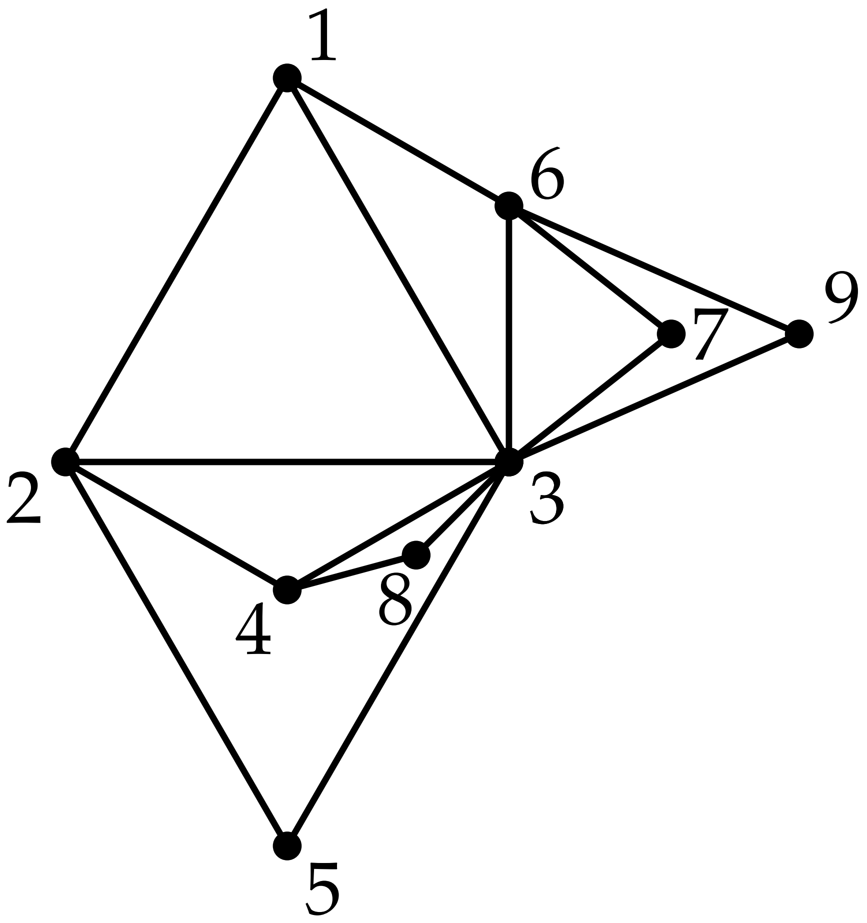


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

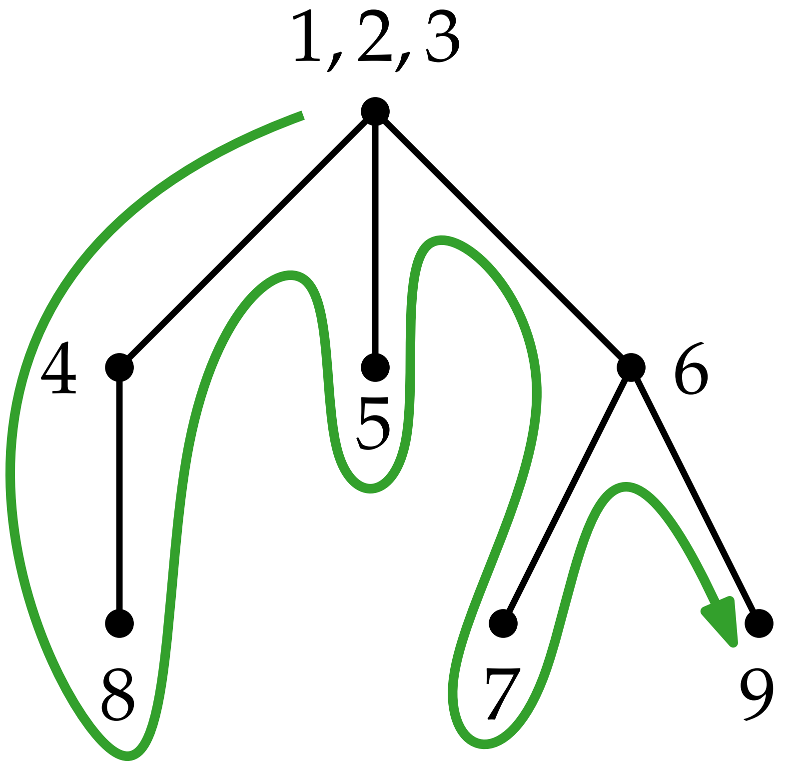


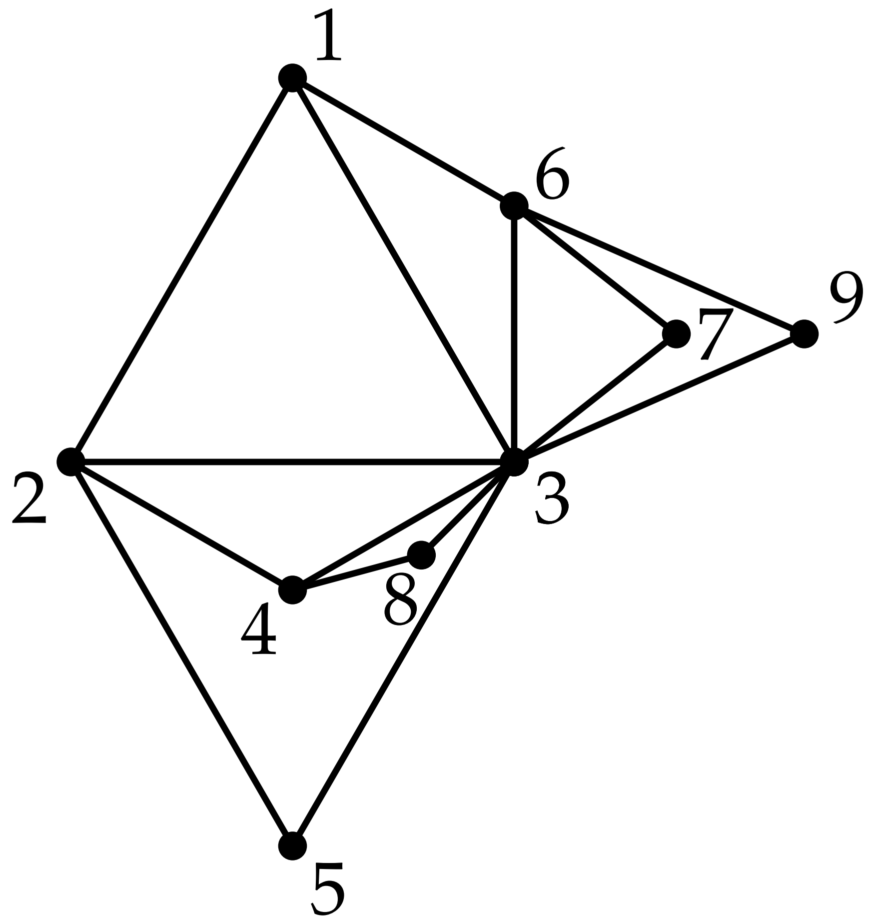


Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.

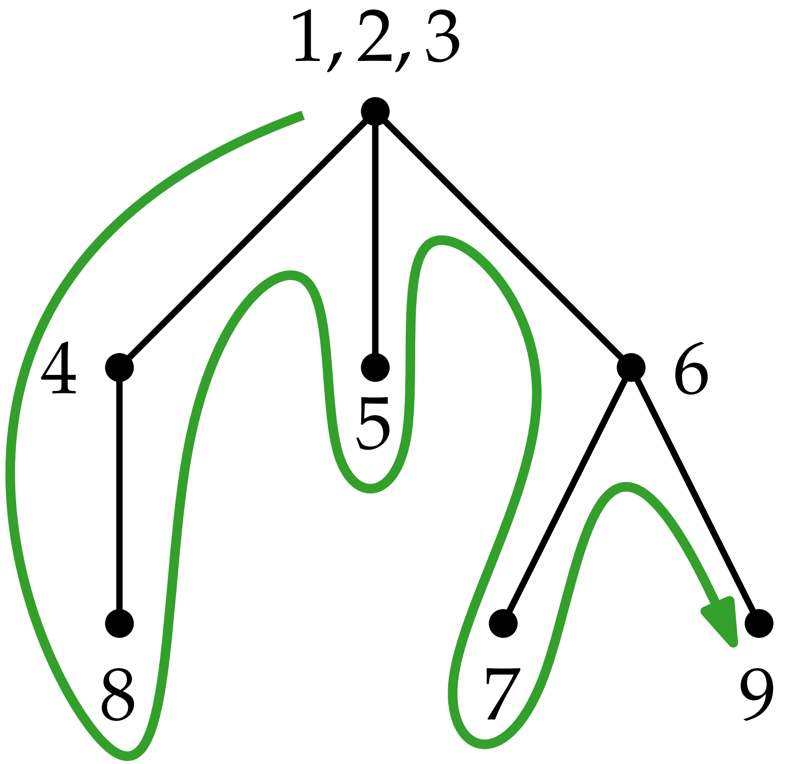




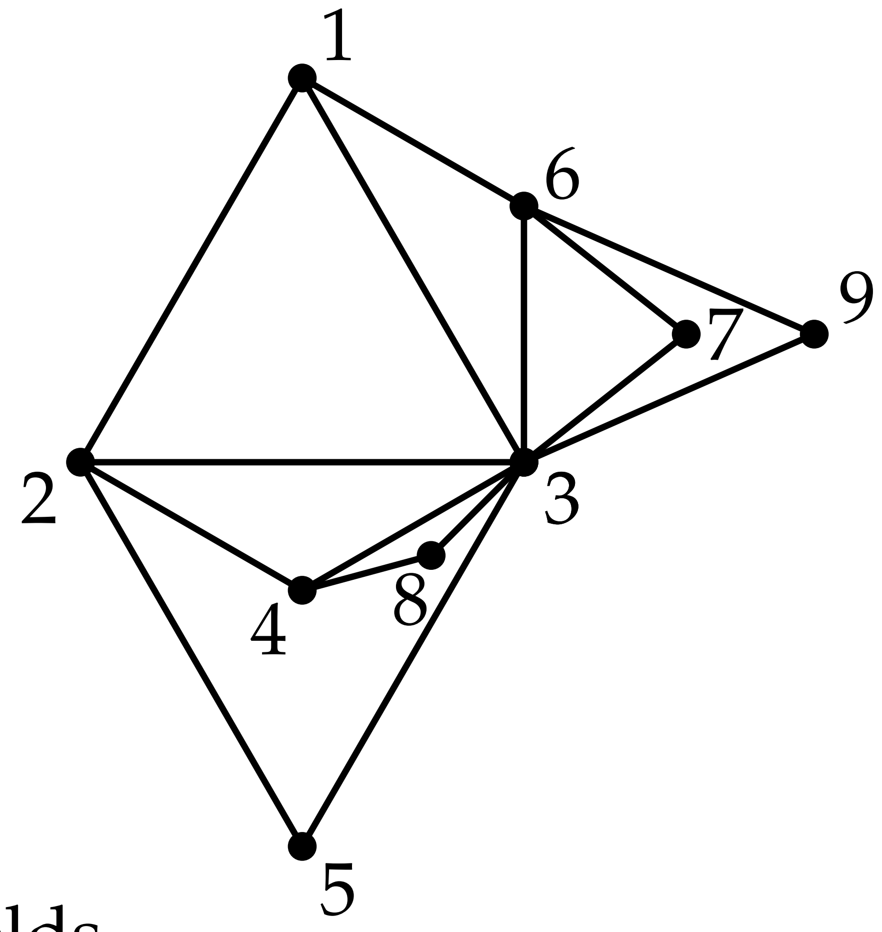
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.



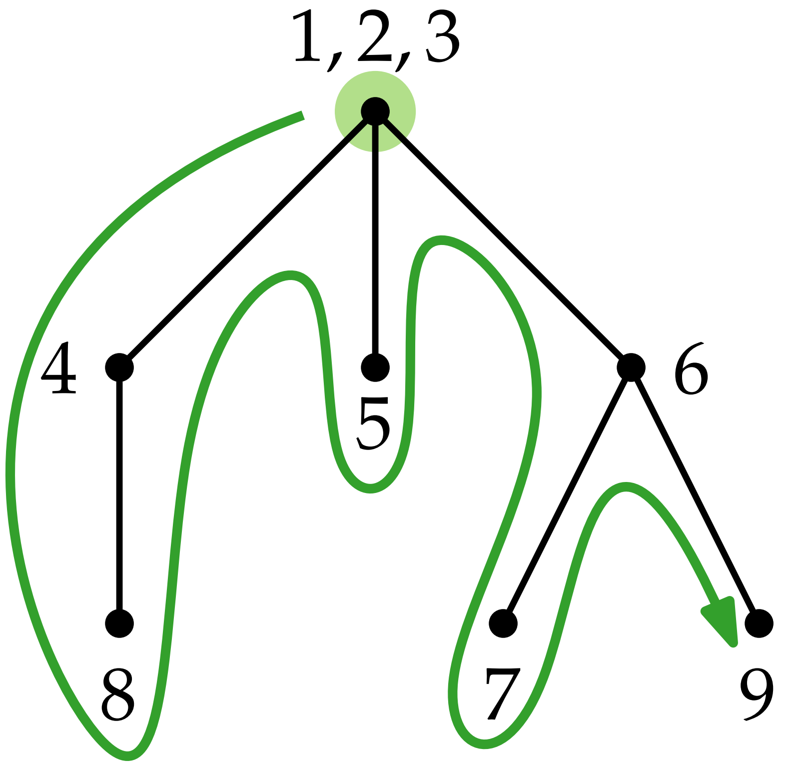
the order that yields
an outerplanar storyplan



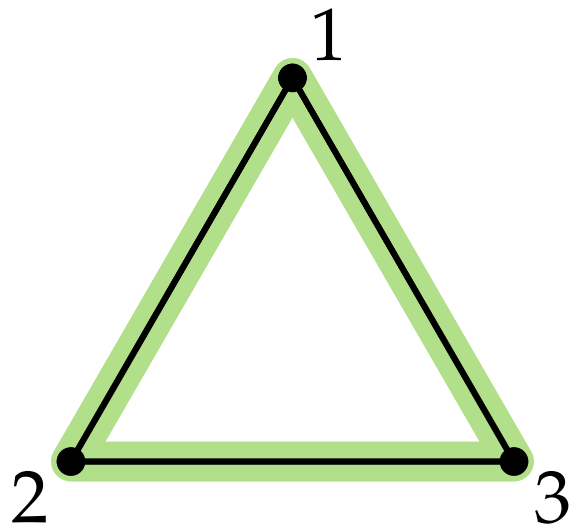
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G
  construct a *tree decomposition* of G.



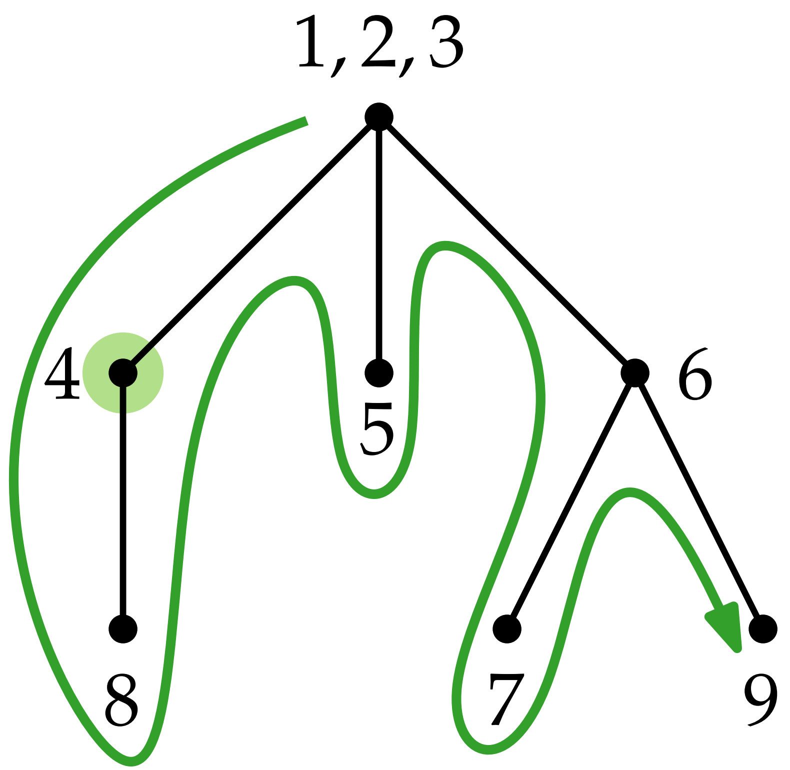
the order that yields
an outerplanar storyplan

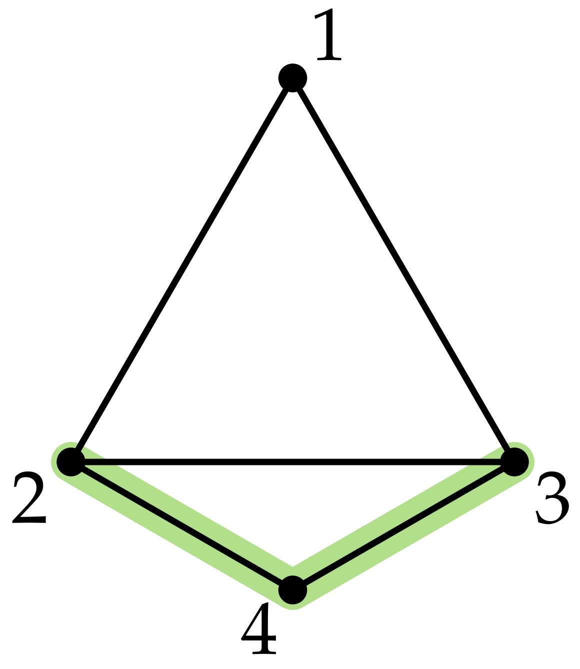


Outerplanar Storyplans of 2-Trees

> **Theorem.**
> Every (partial) **2-tree** admits a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.



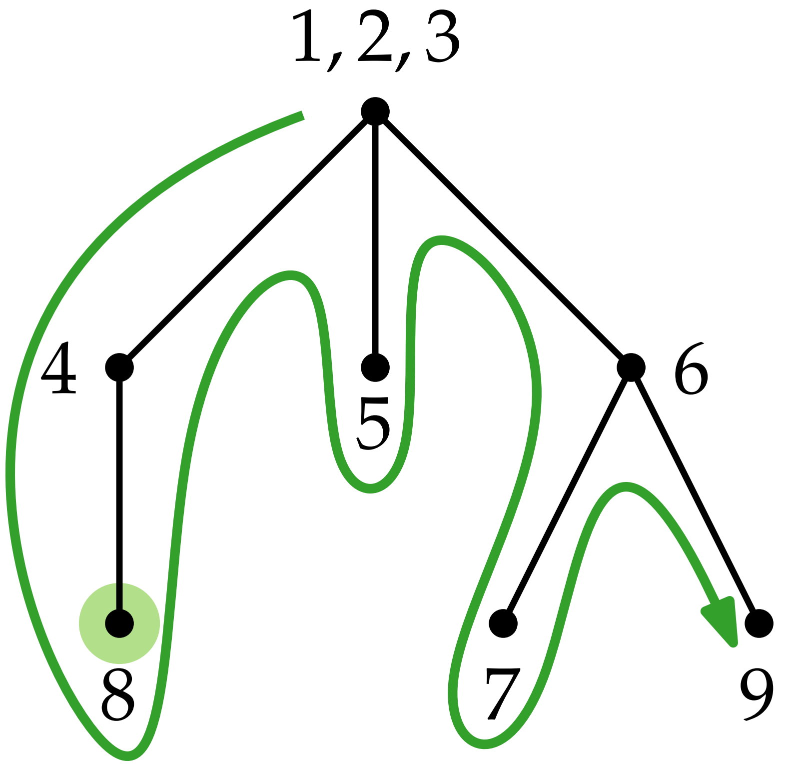


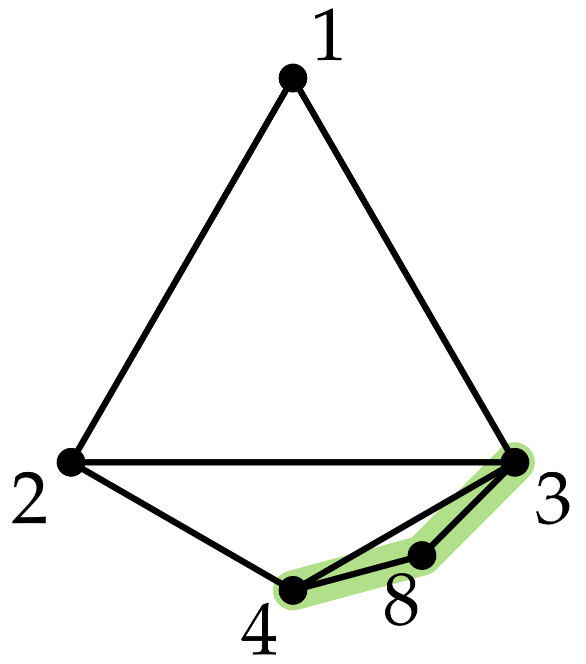
the order that yields an outerplanar storyplan

Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.



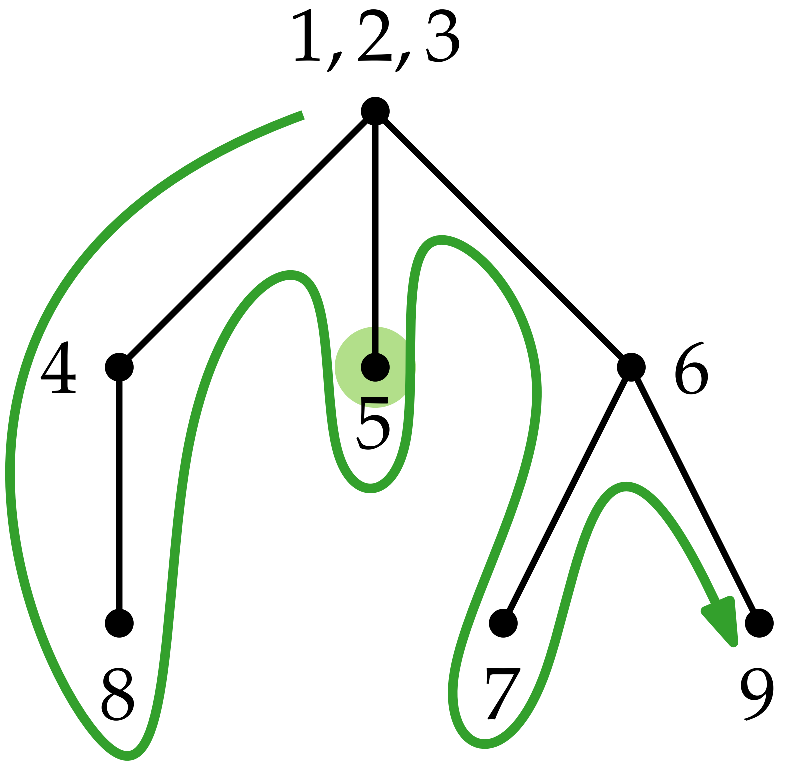


the order that yields an outerplanar storyplan

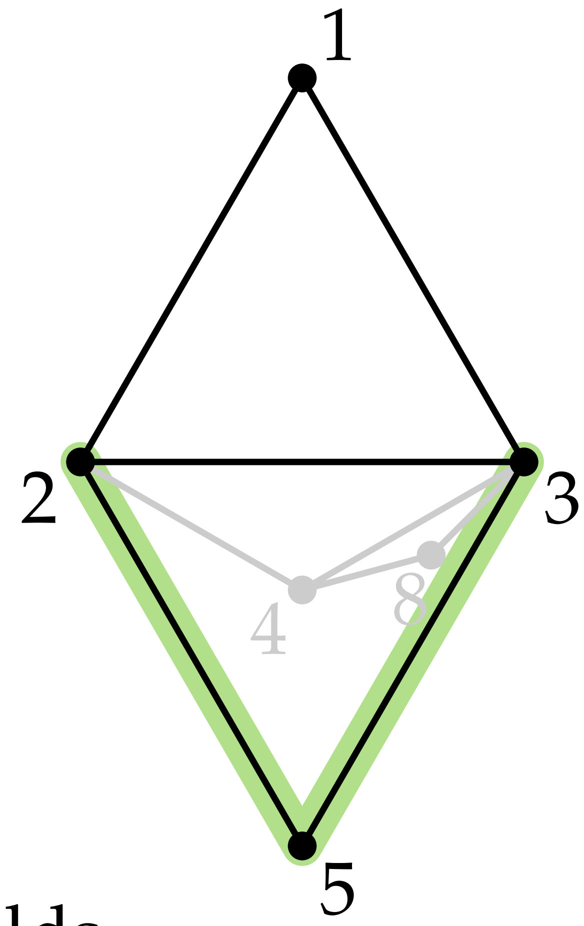
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.



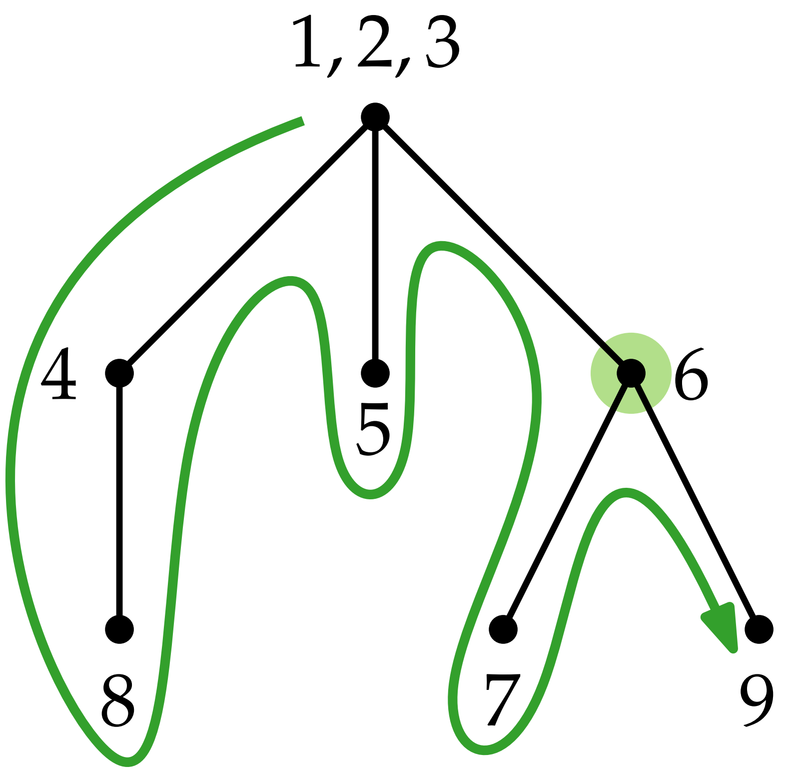
the order that yields
an outerplanar storyplan



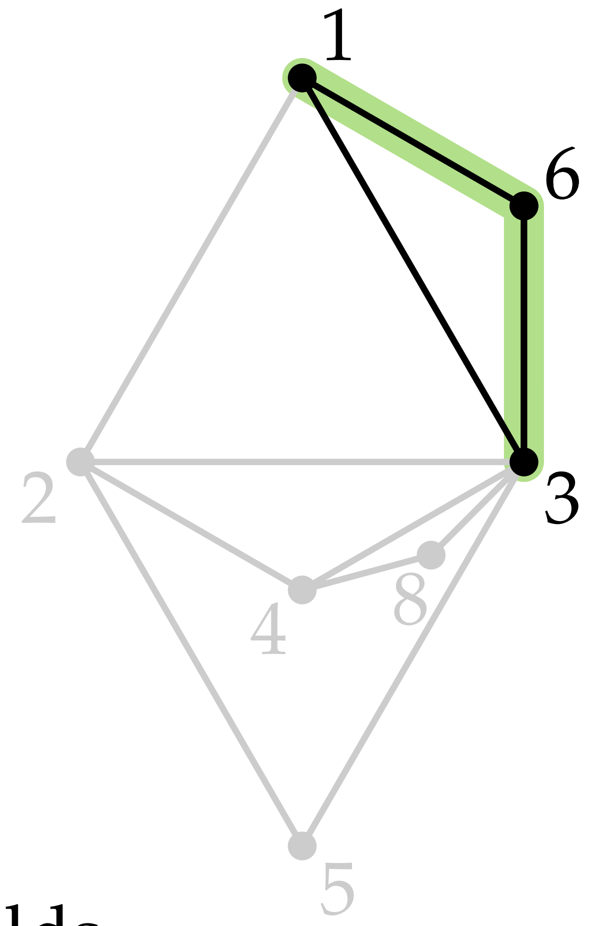
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.



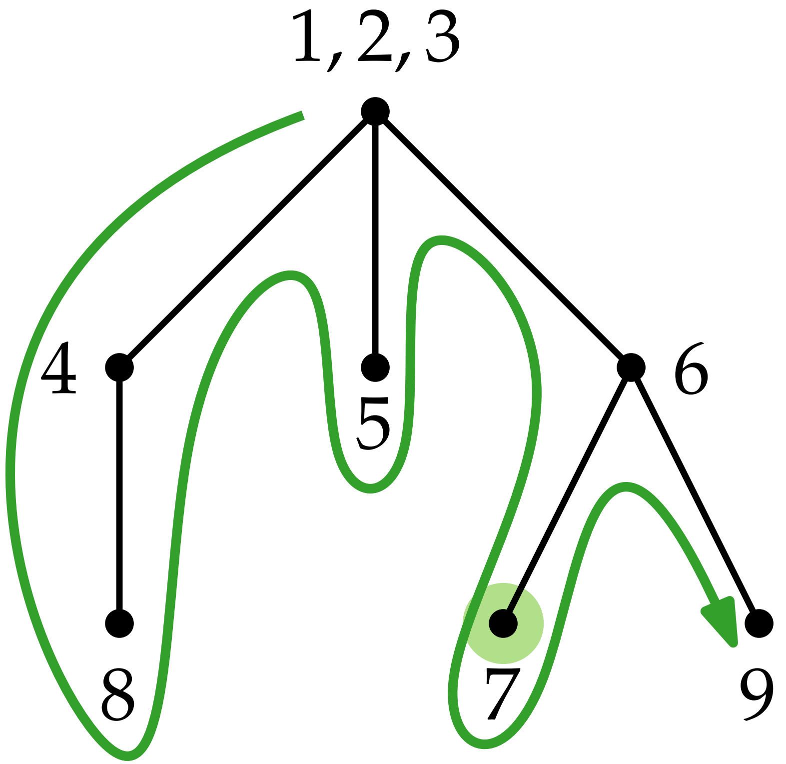
the order that yields
an outerplanar storyplan



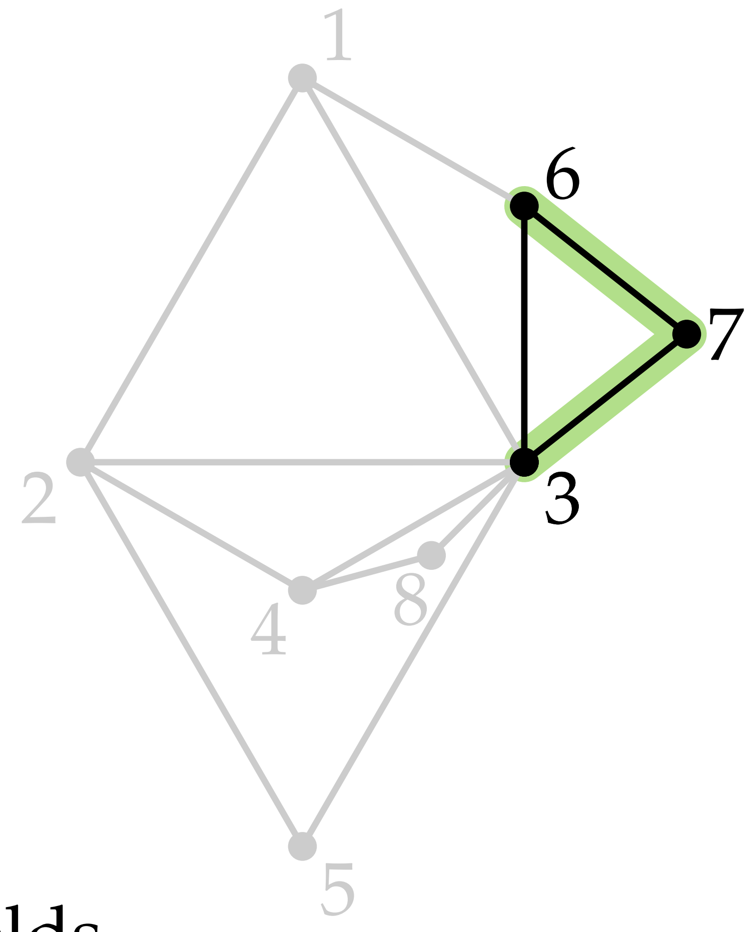
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G construct a *tree decomposition* of G.



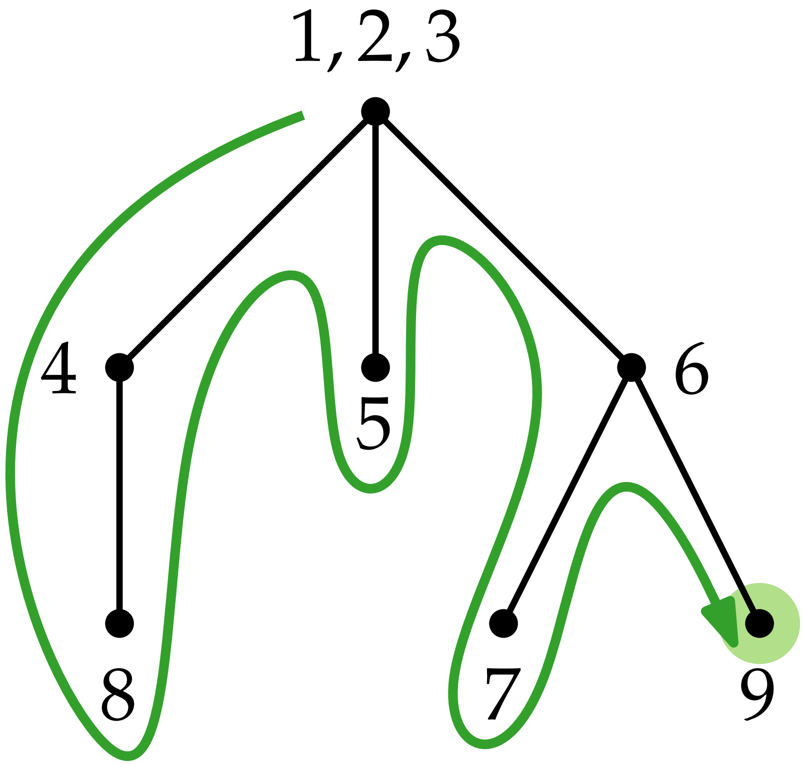
the order that yields
an outerplanar storyplan



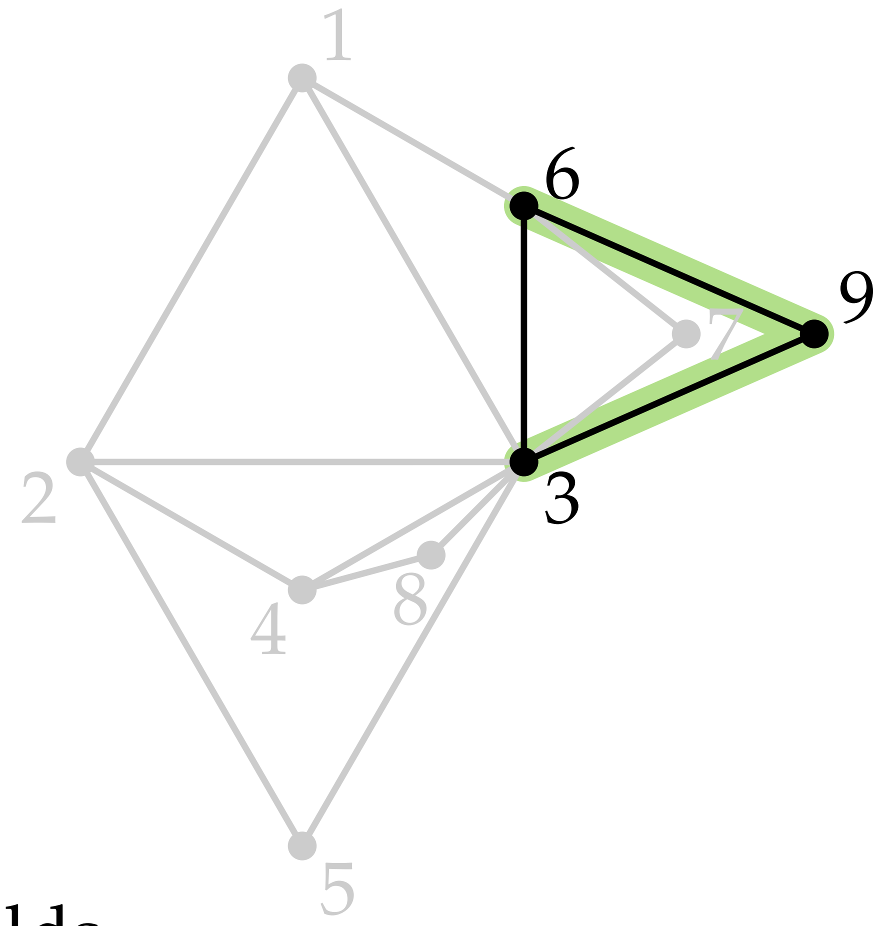
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

- Using the stacking order of G
  construct a *tree decomposition* of G.



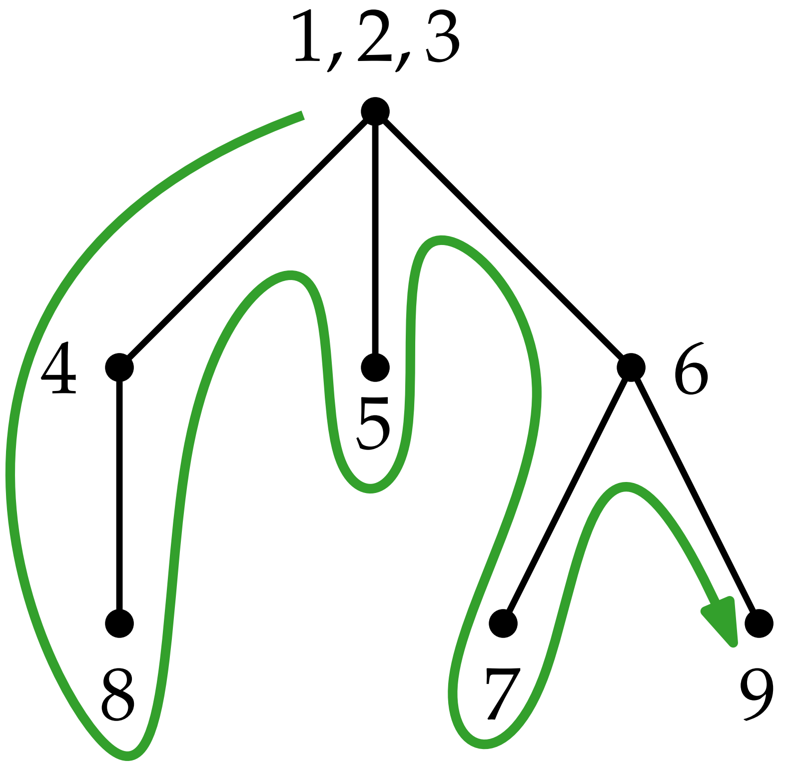
the order that yields
an outerplanar storyplan



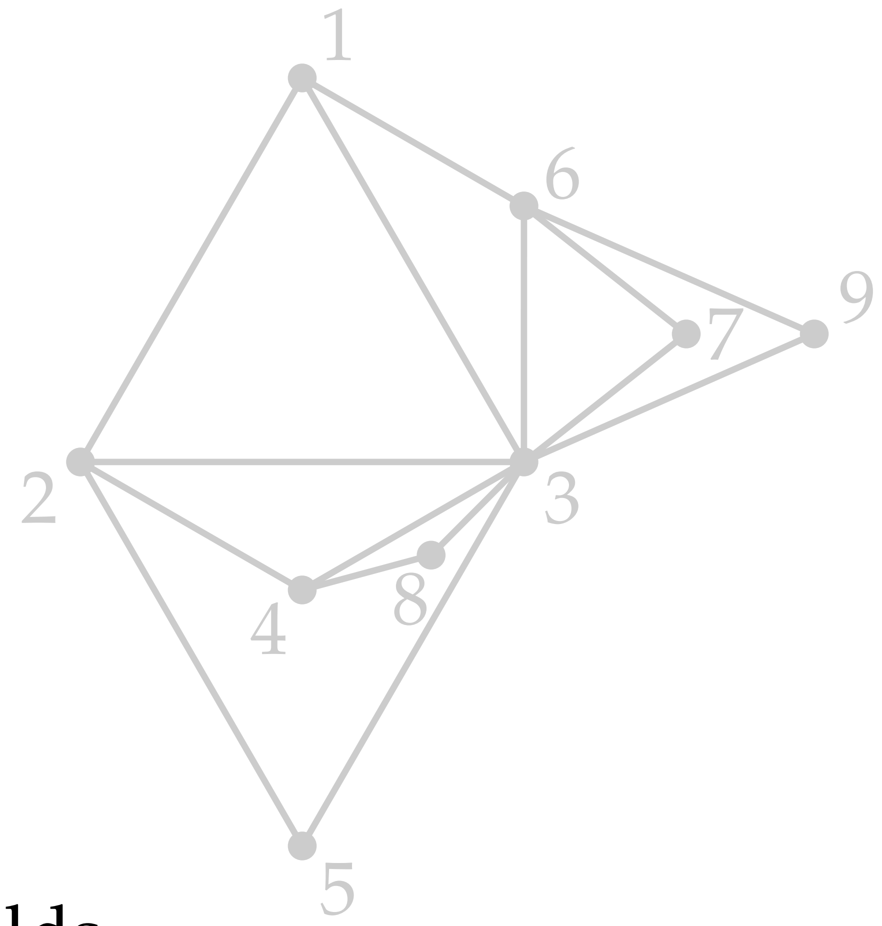
Outerplanar Storyplans of 2-Trees

Theorem.
Every (partial) **2-tree** admits
a straight-line outerplanar storyplan.

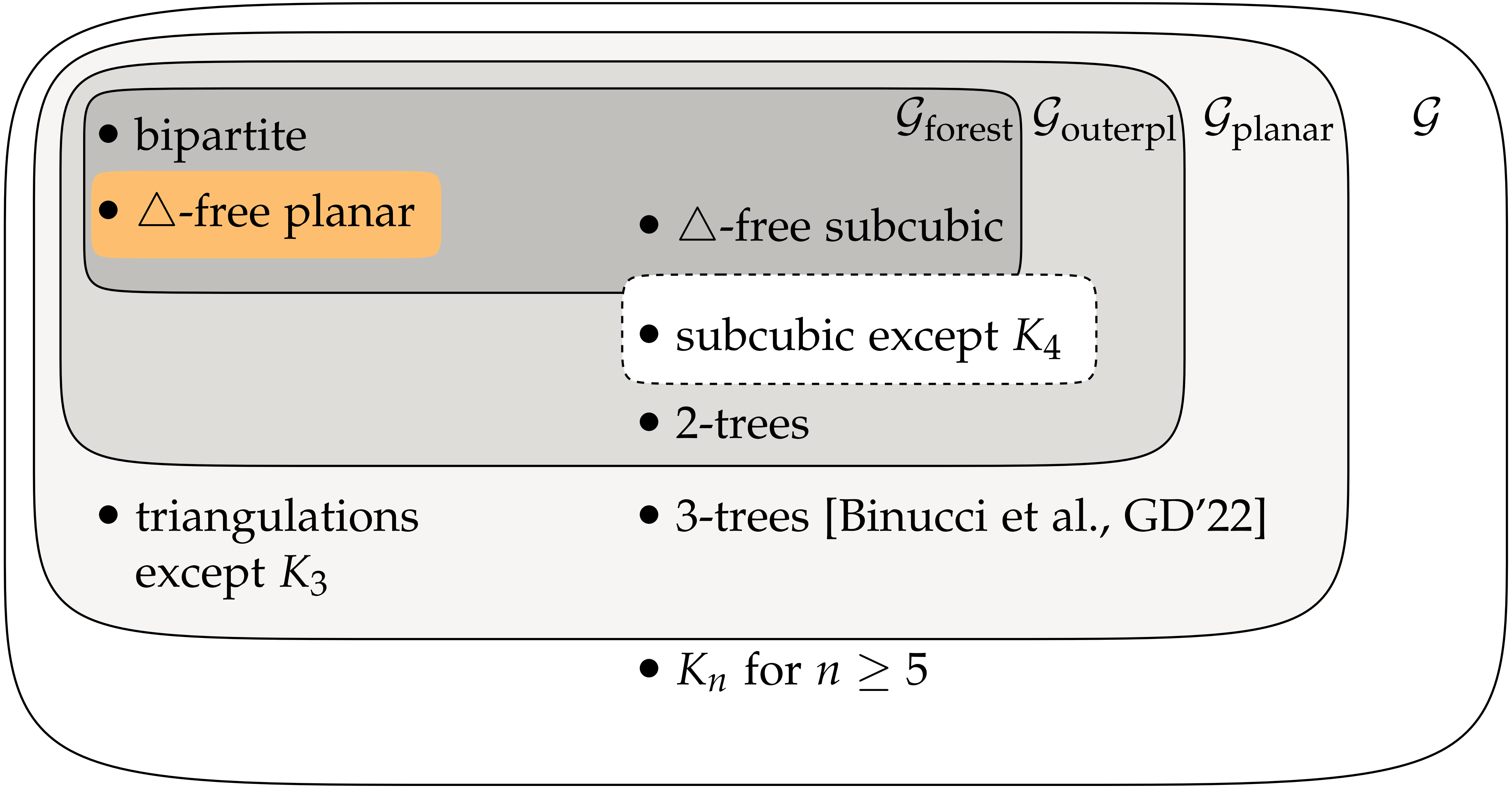
- Using the stacking order of G construct a *tree decomposition* of G.



the order that yields
an outerplanar storyplan



Our Contribution

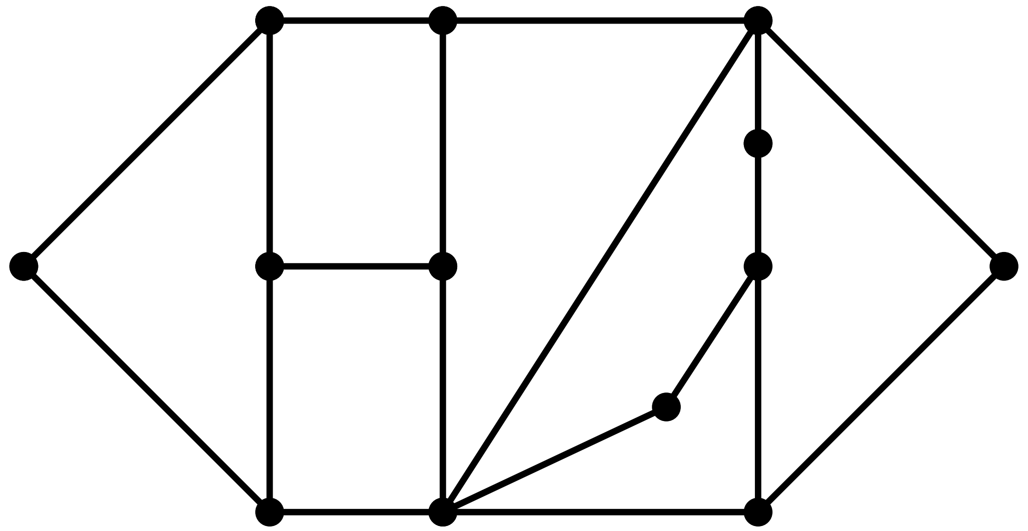


Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

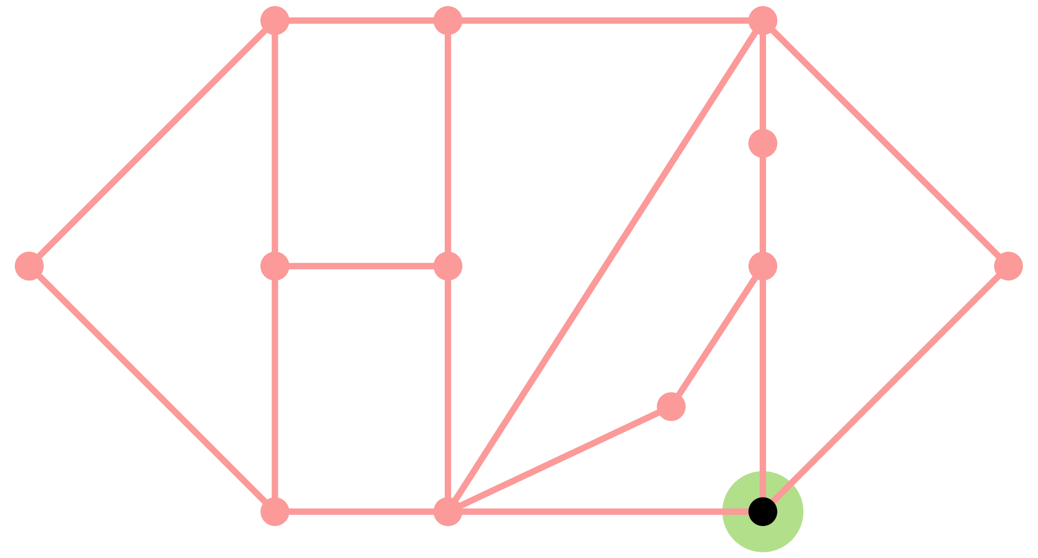


Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.



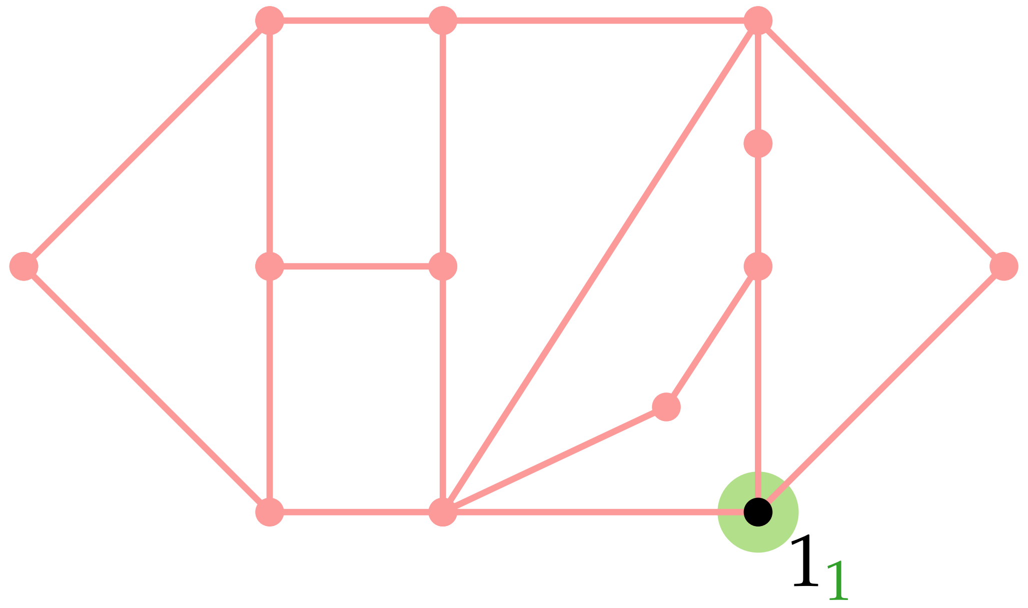
Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.



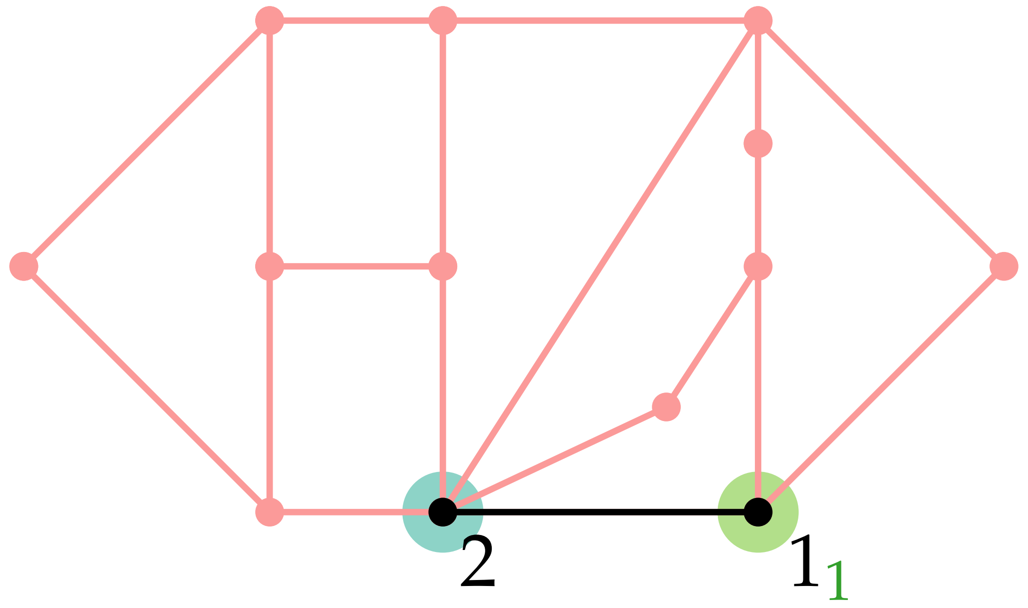
Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.



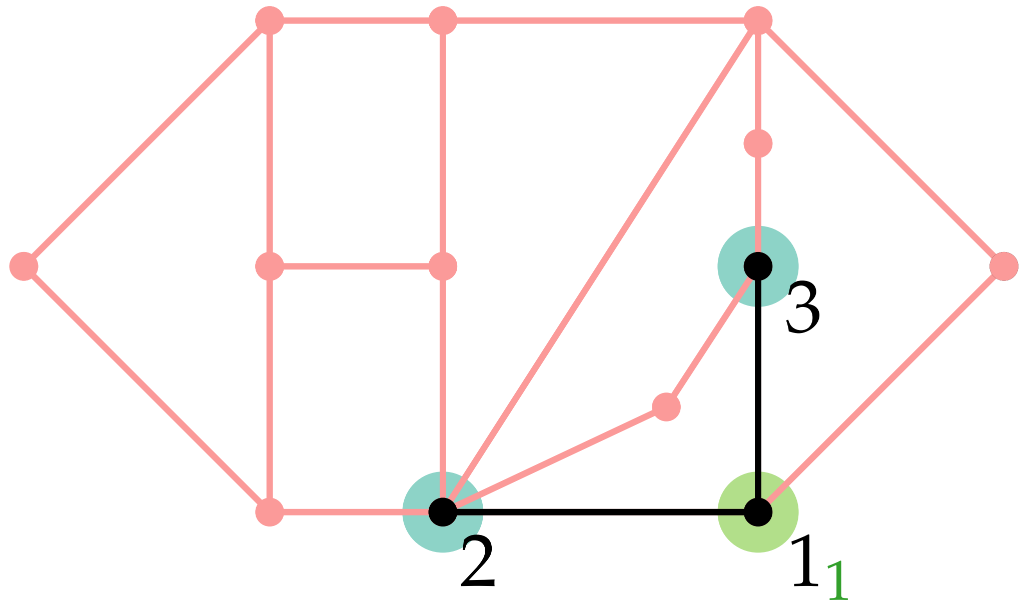
Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.



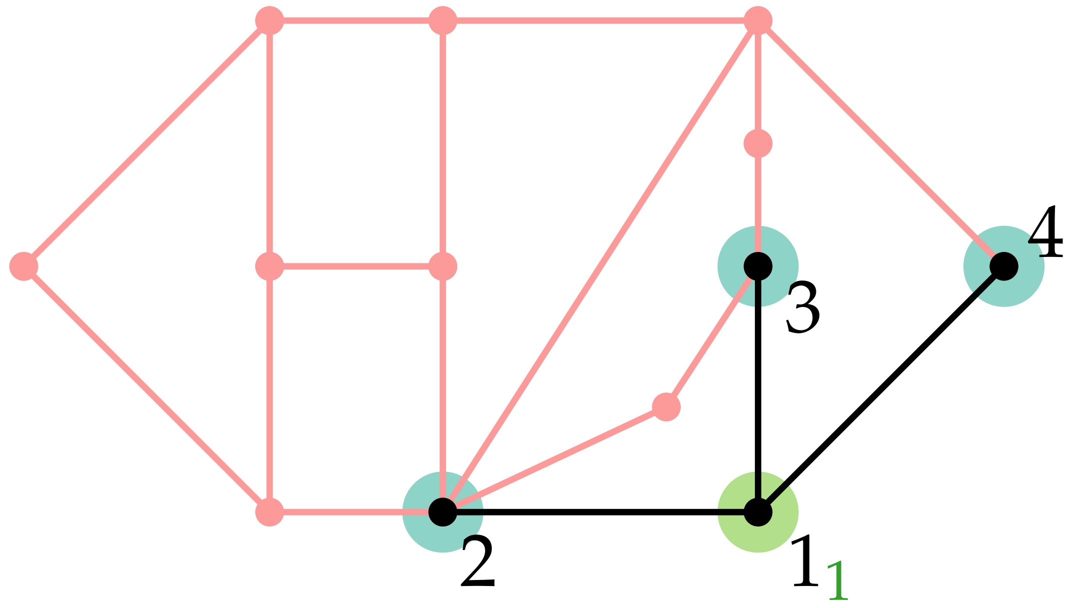
Forest Storyplans of △-Free Planar Graphs

Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.



Forest Storyplans of △-Free Planar Graphs

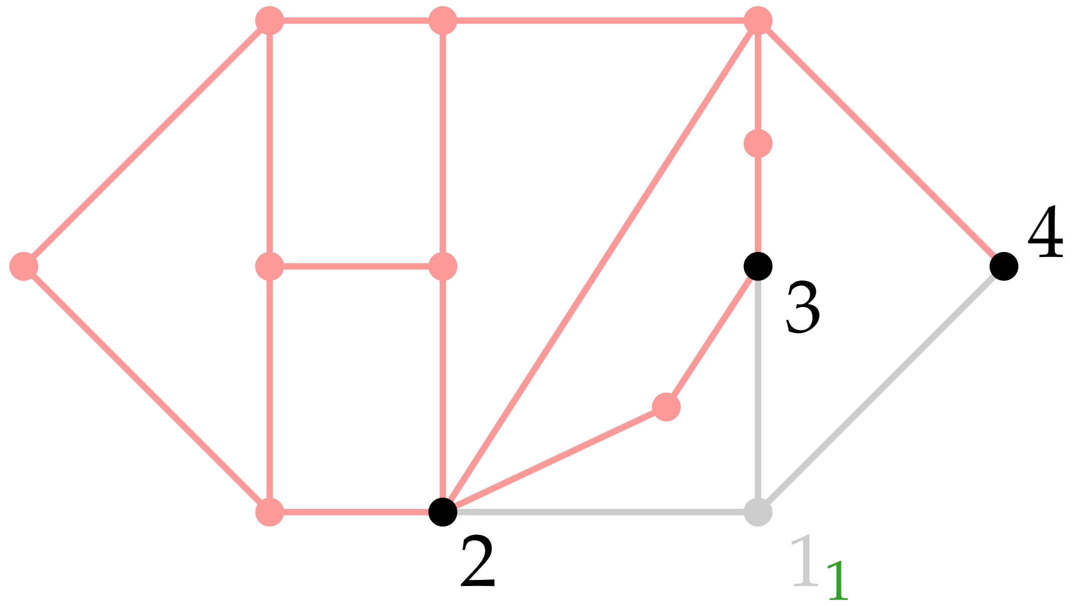
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of $\triangle$-Free Planar Graphs

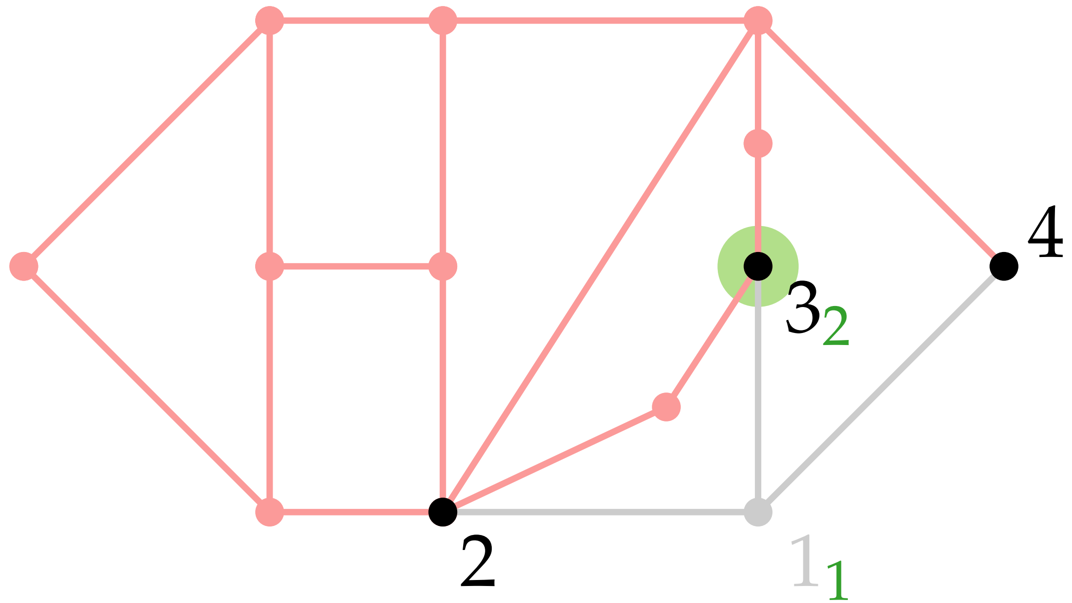
Main theorem.
Every $\triangle$**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

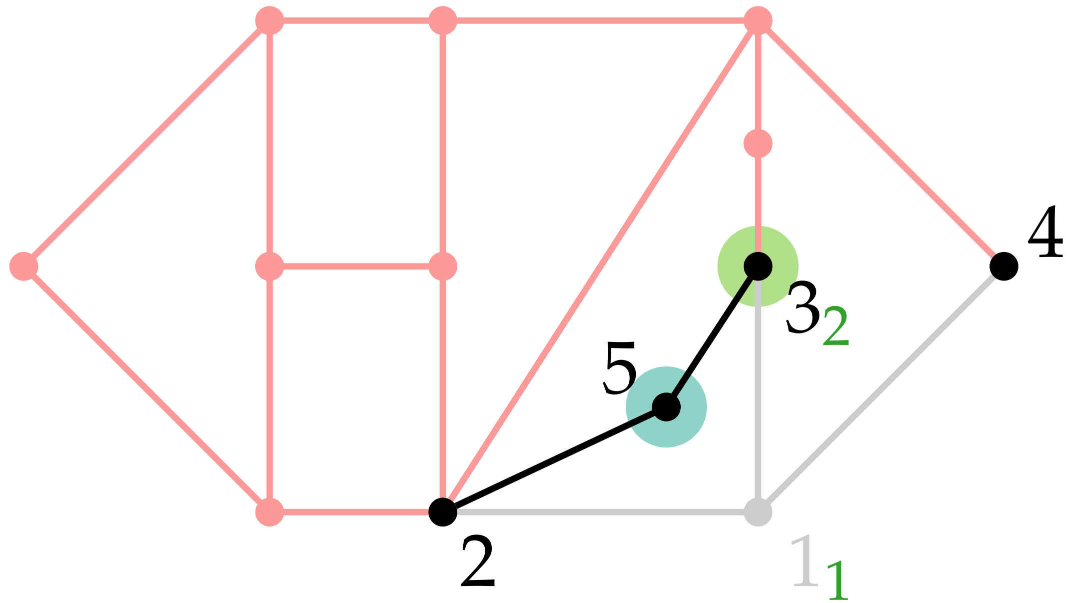
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

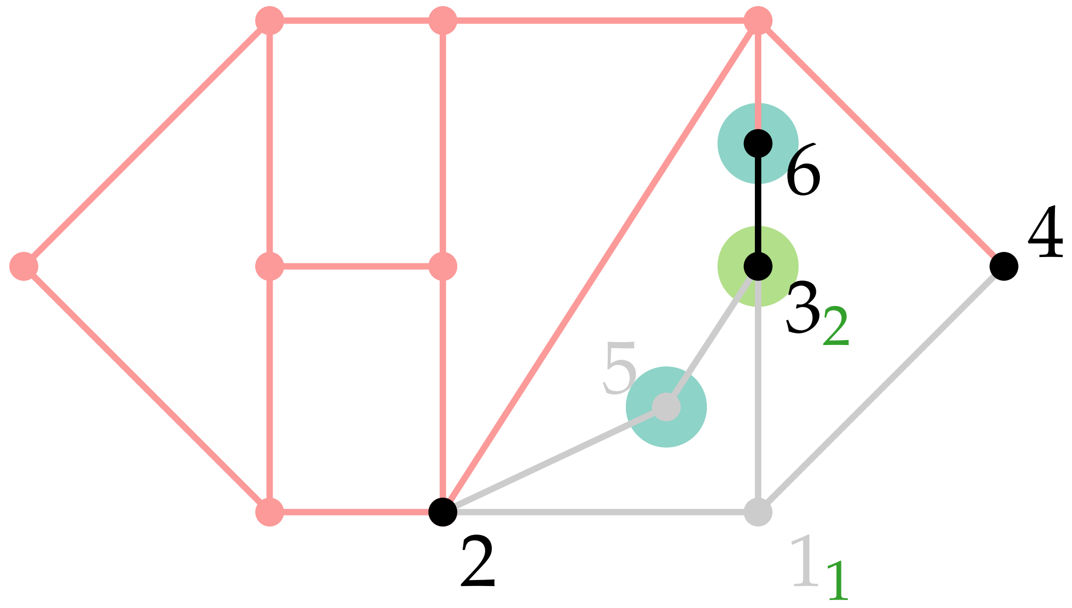
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

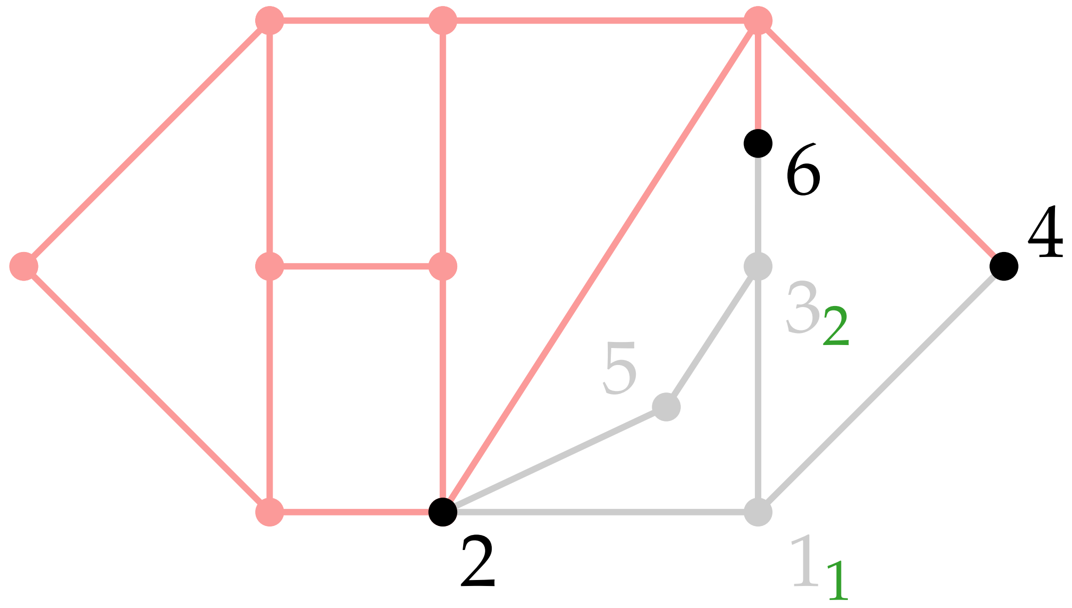
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

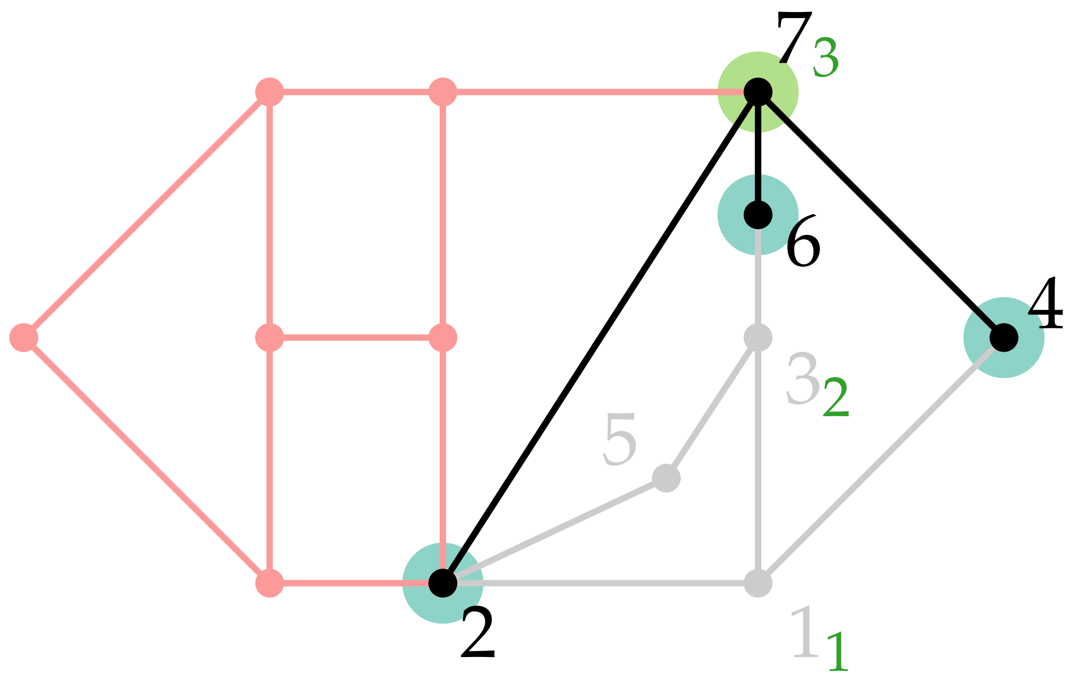
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

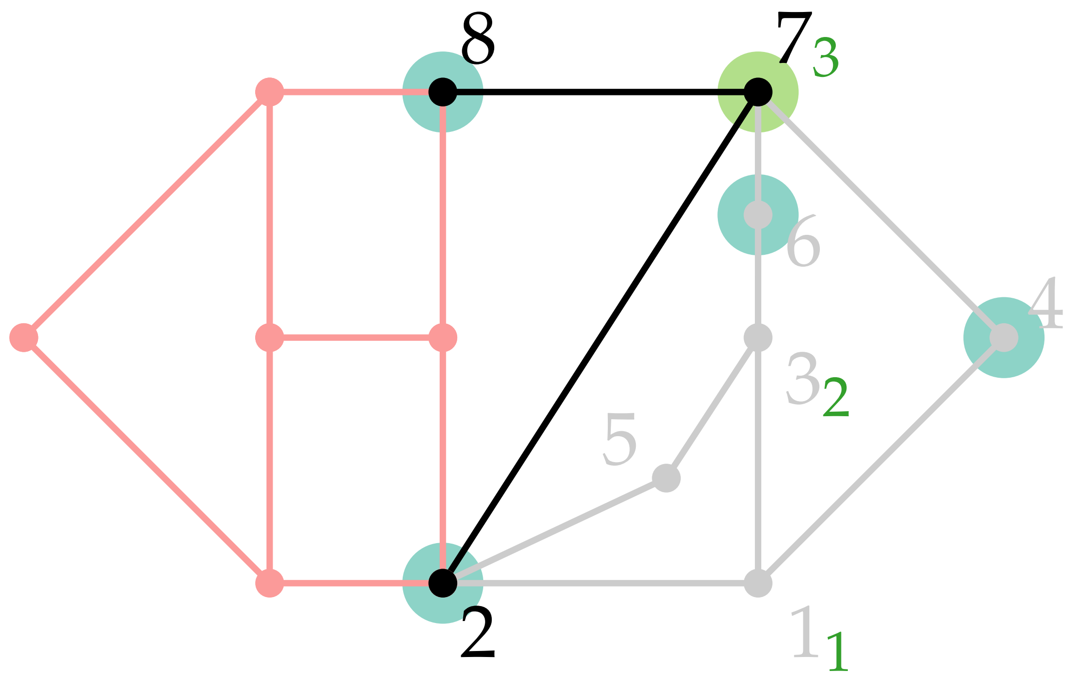
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

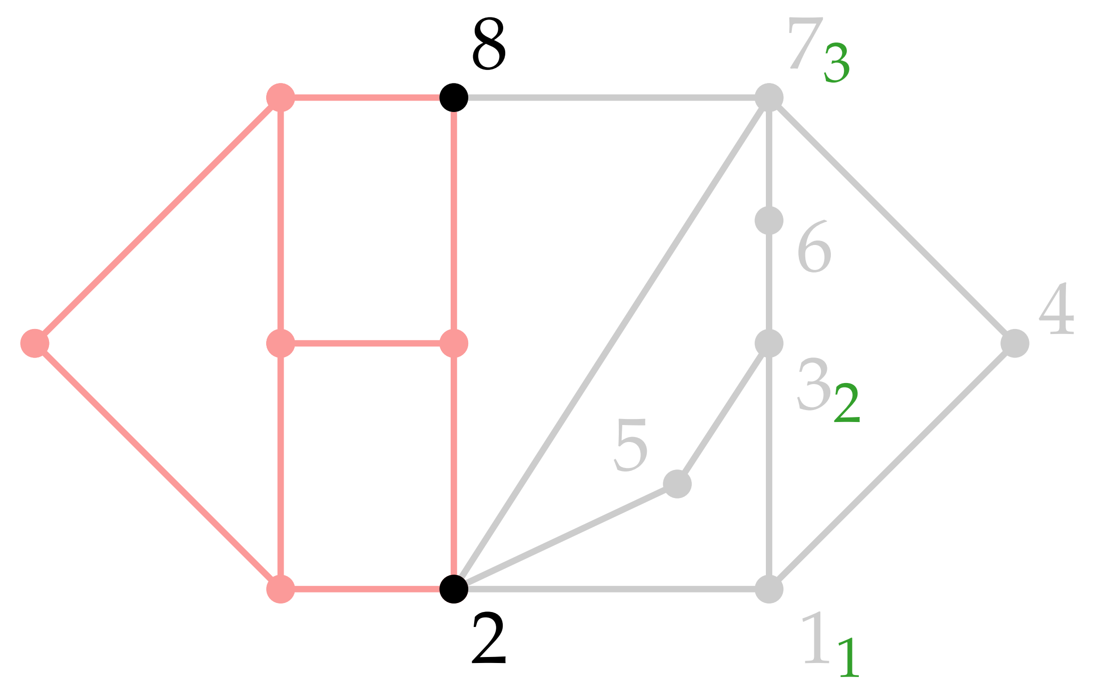
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

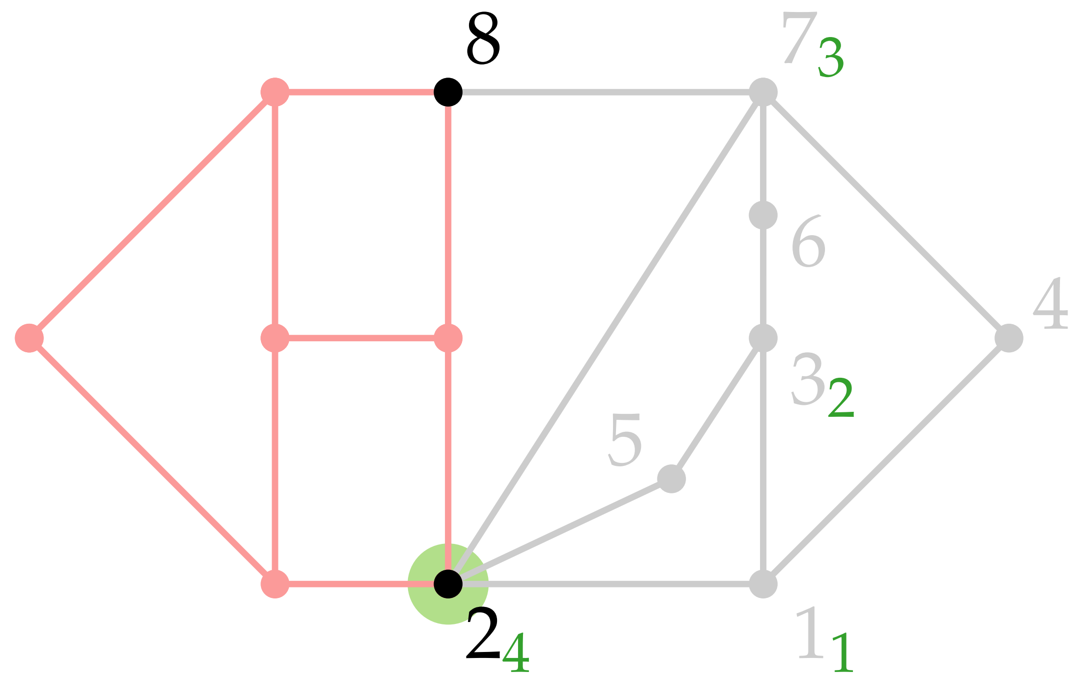
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

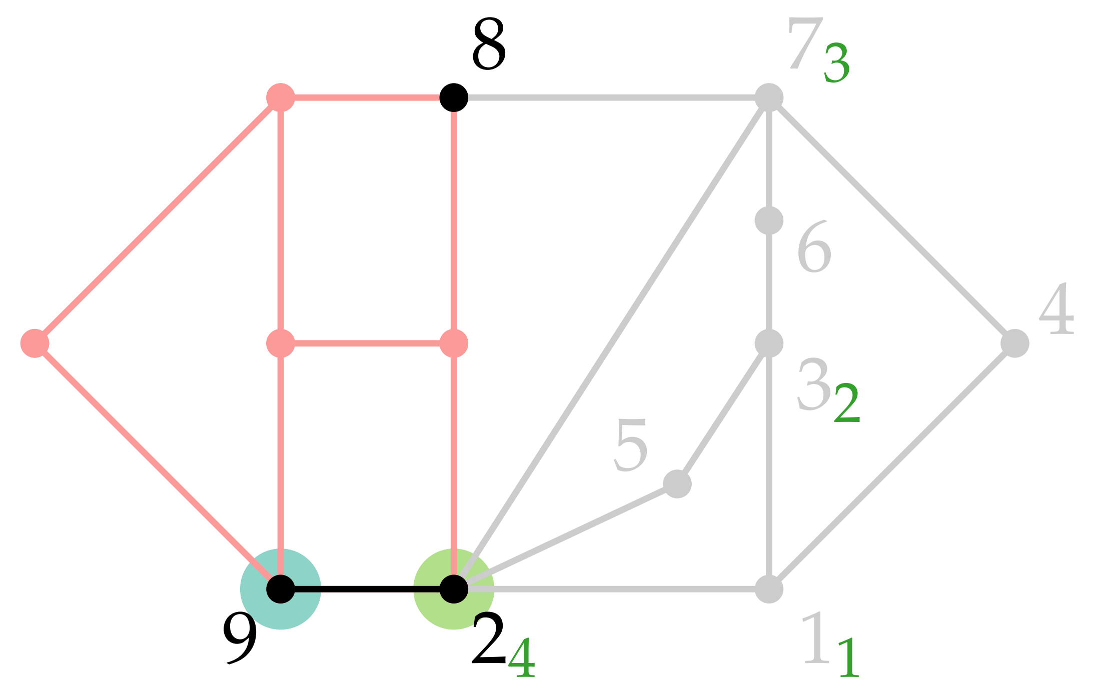
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

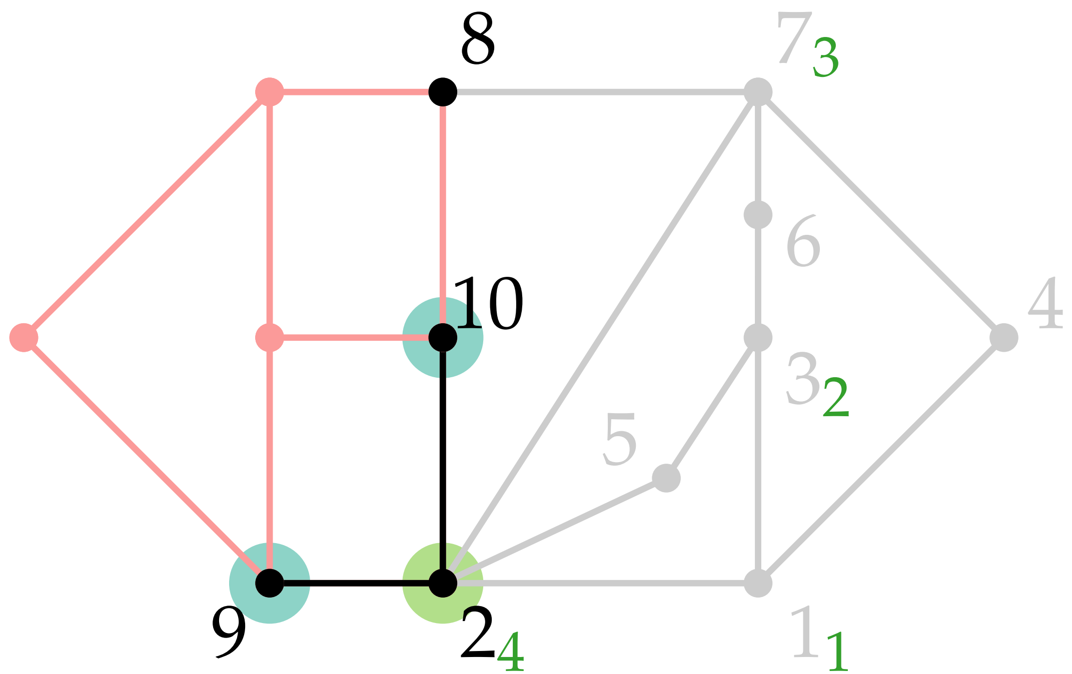
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

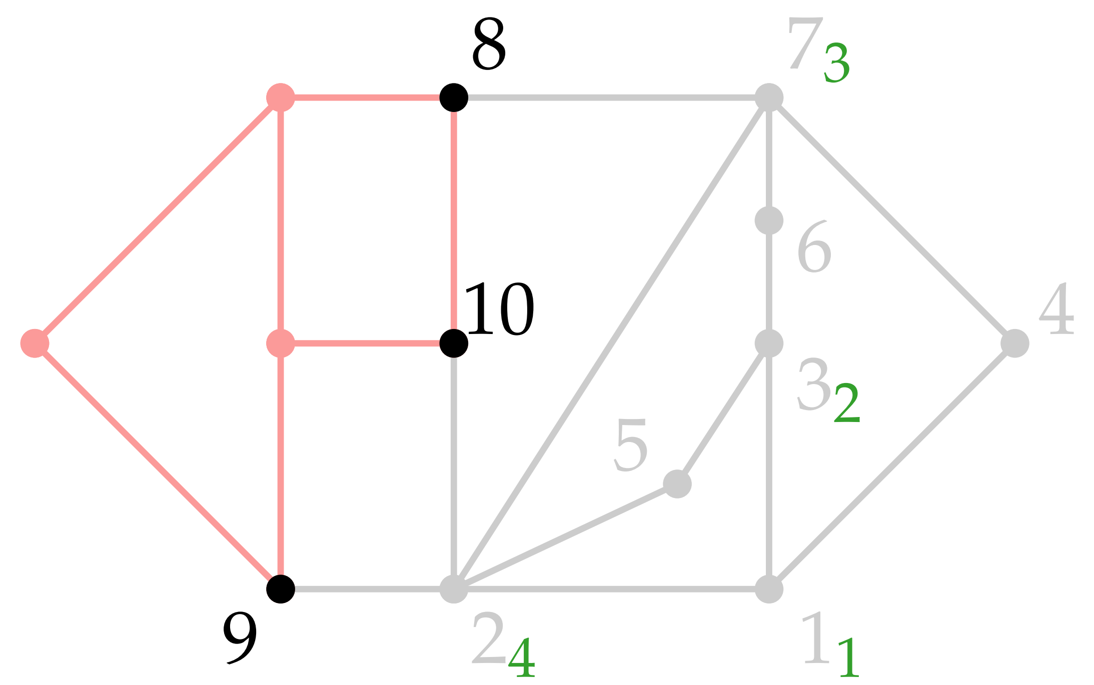
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

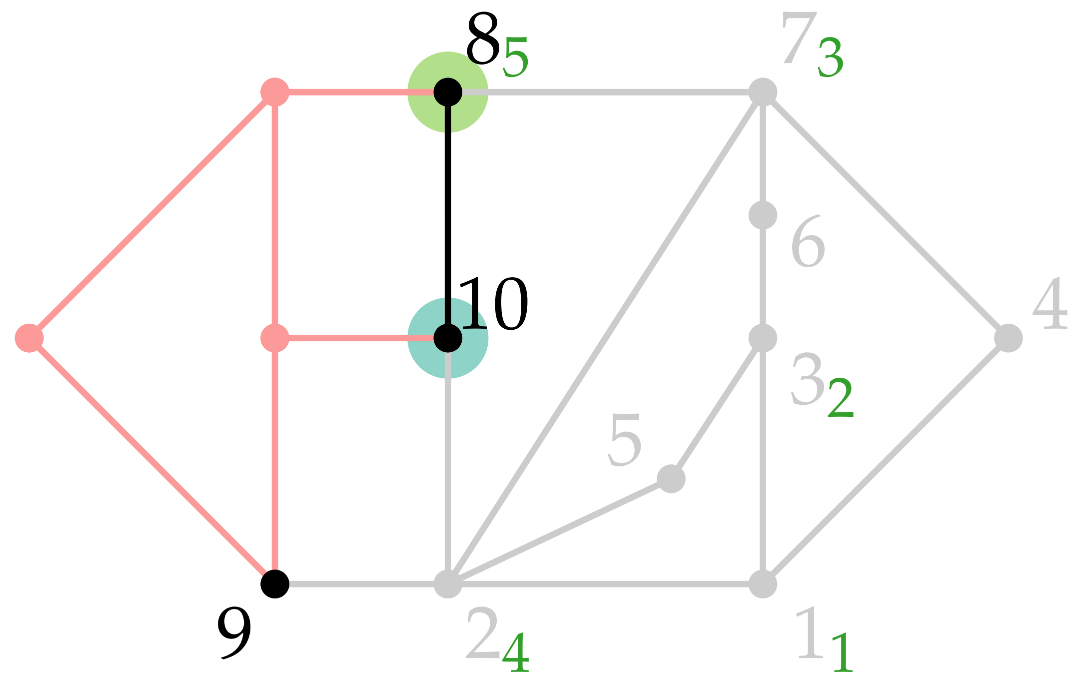
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

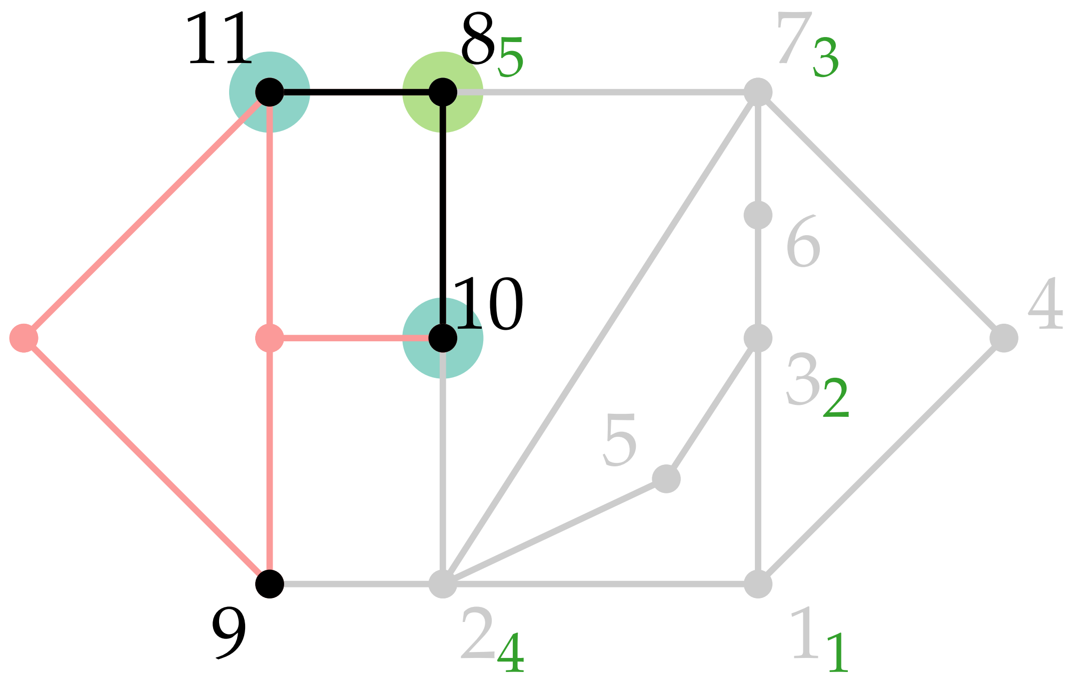
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

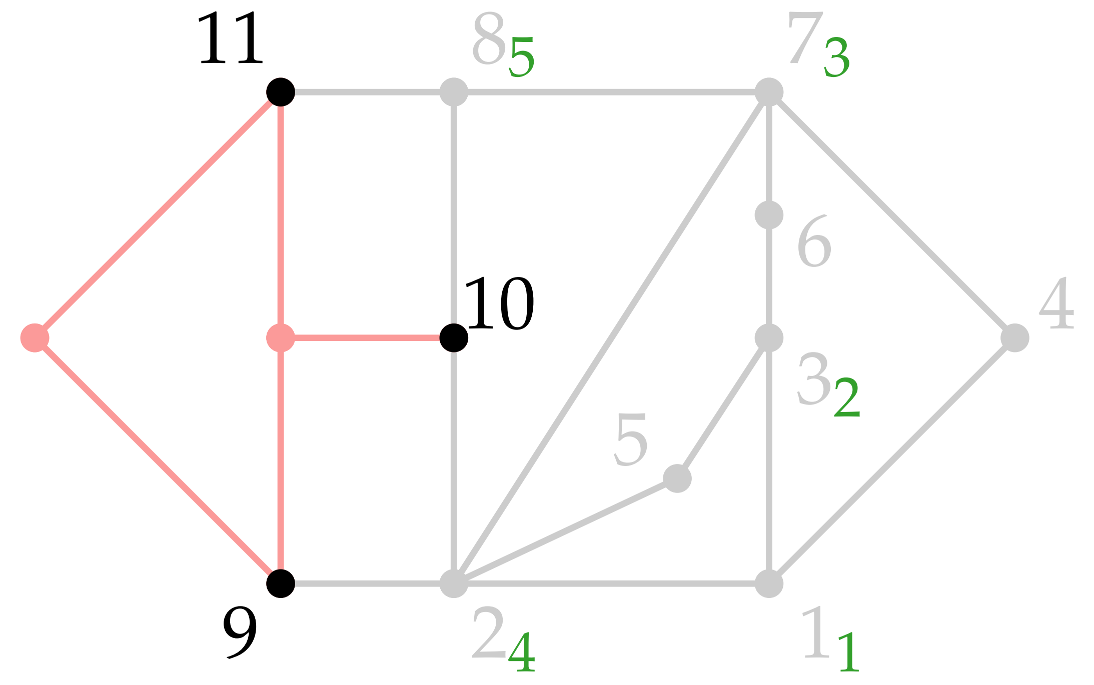
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

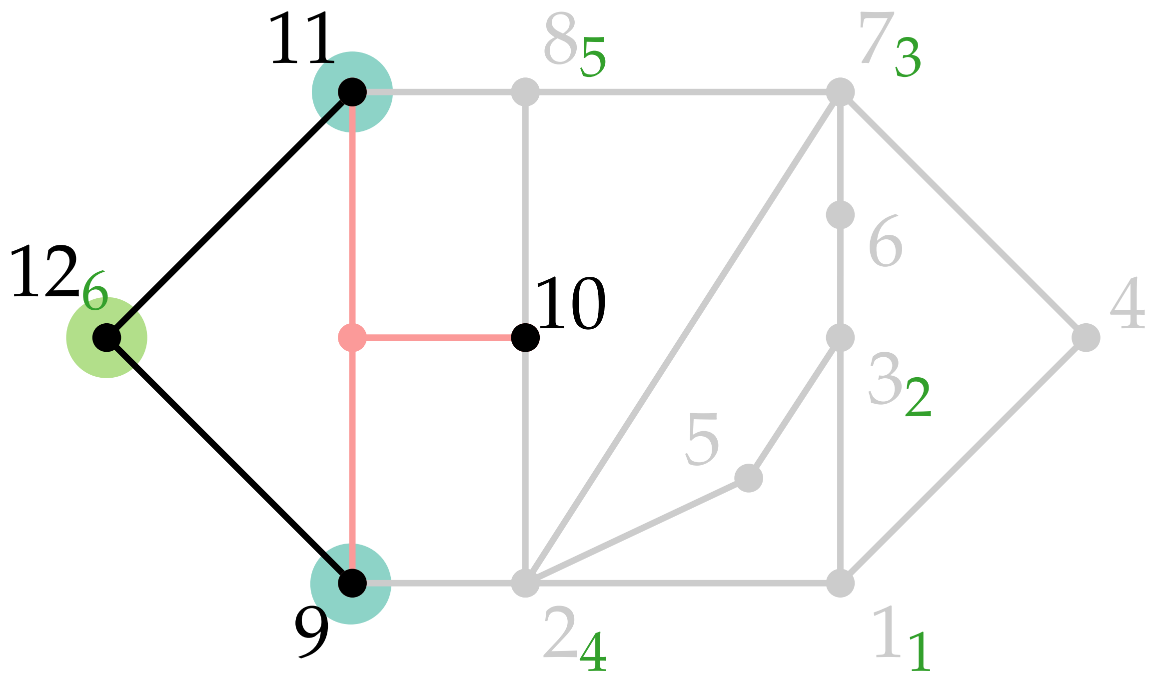
Main theorem.
Every **△-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

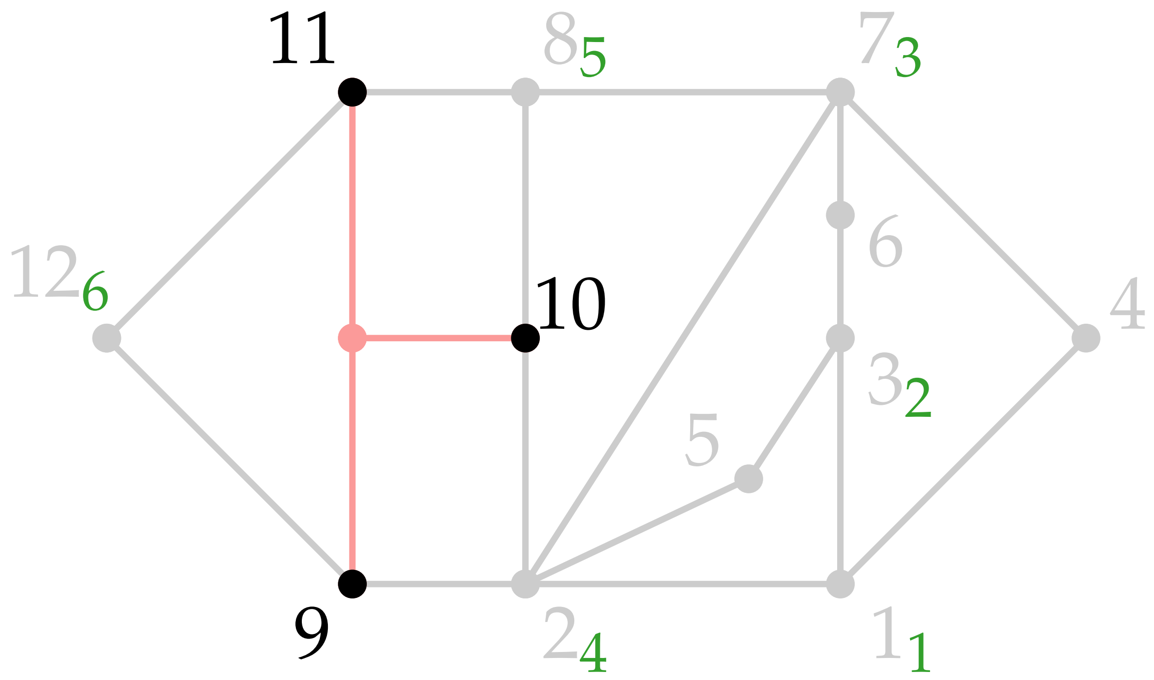
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

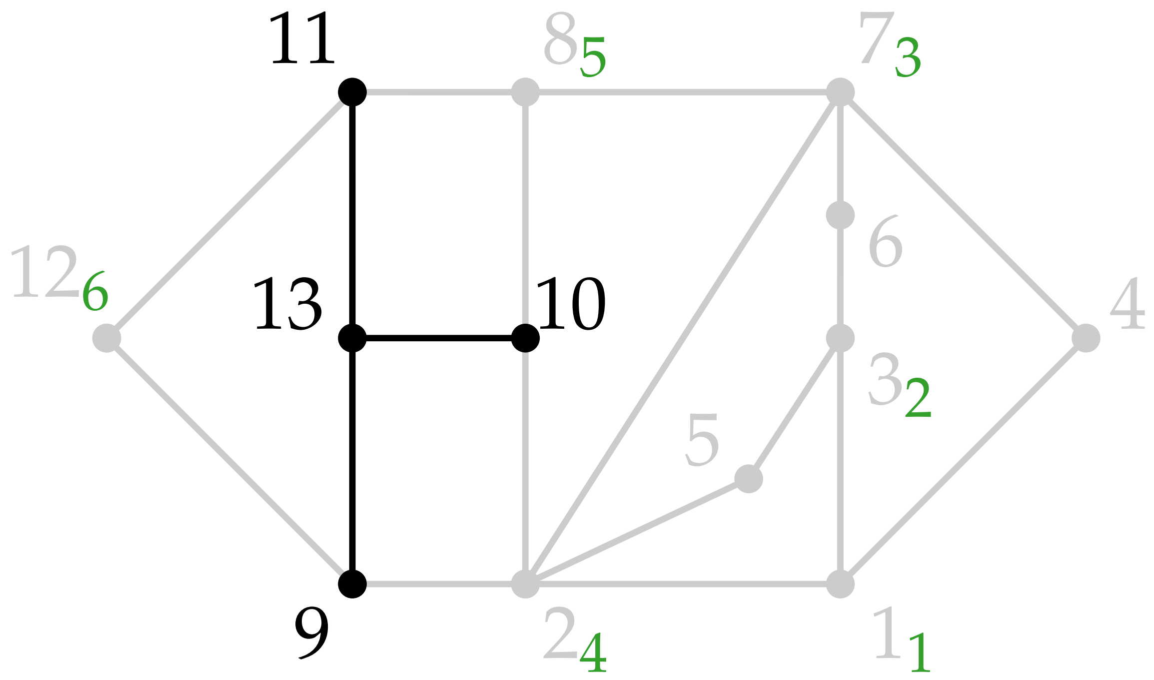
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.



Forest Storyplans of △-Free Planar Graphs

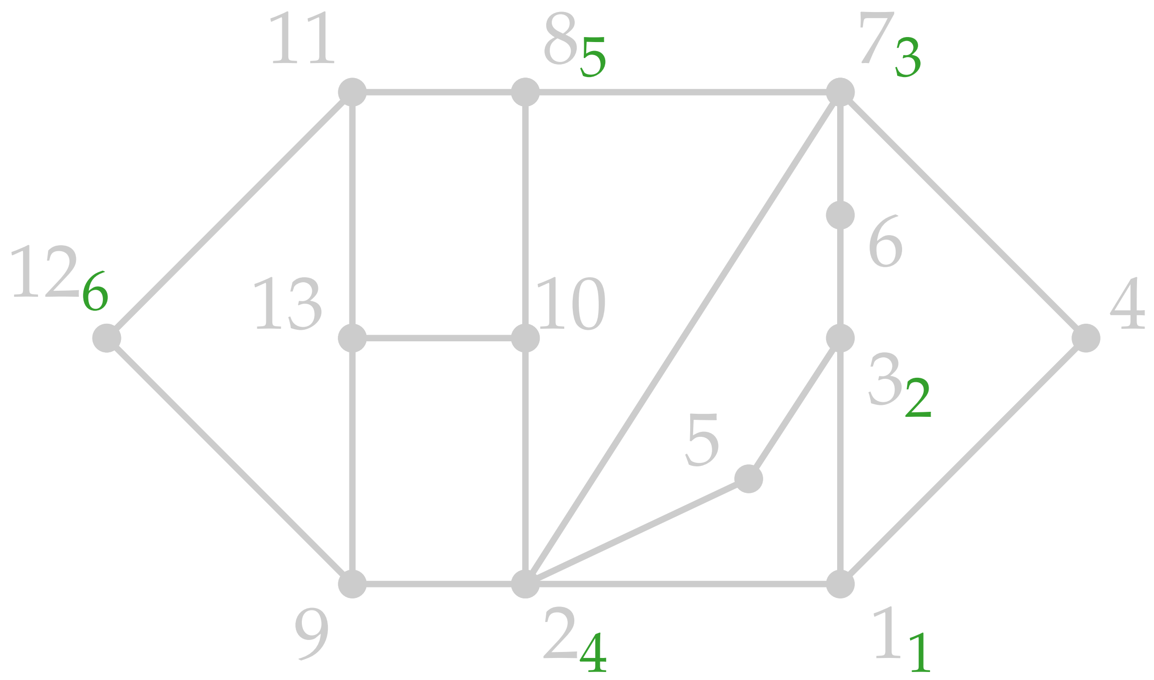
Main theorem.
Every △**-free planar graph** admits a straight-line forest storyplan.

Idea.

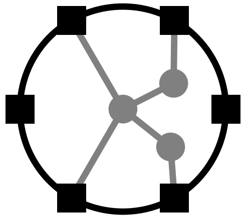
In each iteration we *pick* a vertex on the current outer face that fulfils special properties.

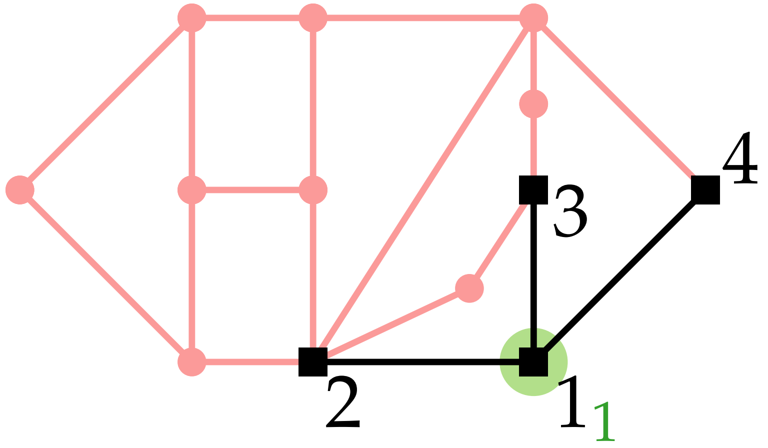
We add the vertex and its neighbors (if they are not visible yet) to the storyplan one by one.

After each iteration at least one vertex dissapears.

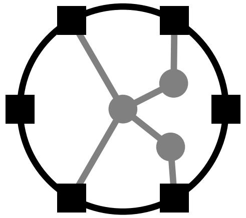


Proof – Invariants

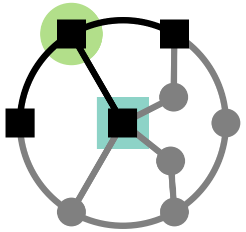
I1  At no point in time, the set of visible edges on the outer face forms a cycle.



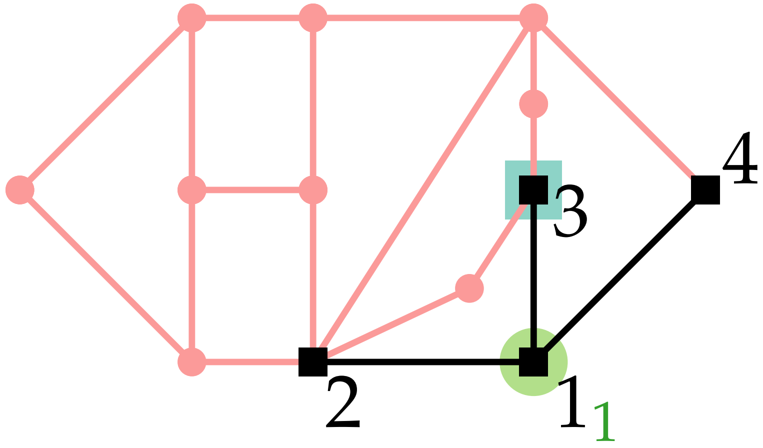
Proof – Invariants

I1 

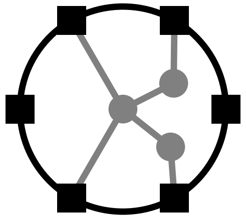
At no point in time, the set of visible edges on the outer face forms a cycle.

I2 

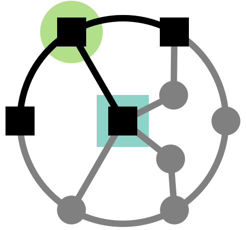
During iteration i, the only inner vertices that may be visible are those that are adjacent to v_i and to no other visible vertex on the outer face.



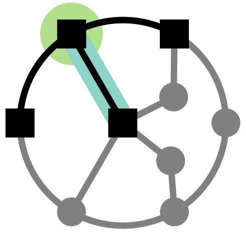
Proof – Invariants

I1 

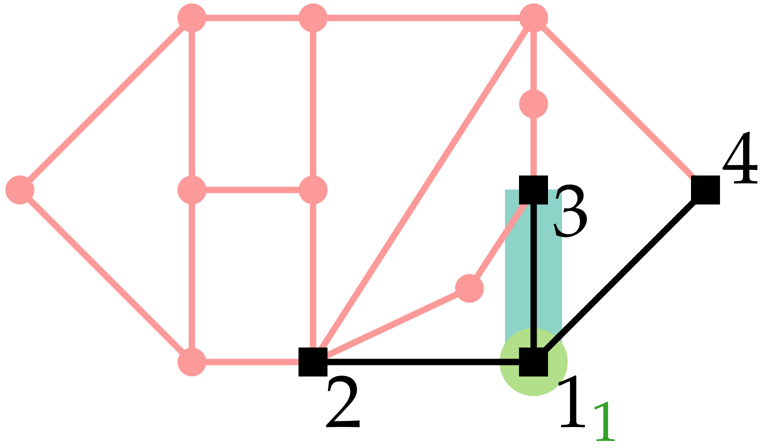
At no point in time, the set of visible edges on the outer face forms a cycle.

I2 

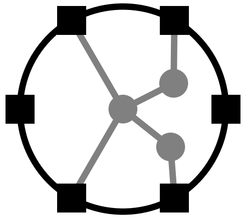
During iteration i, the only inner vertices that may be visible are those that are adjacent to v_i and to no other visible vertex on the outer face.

I3 

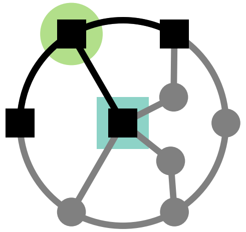
During iteration i, the only inner edges that may be visible are those that are incident to v_i and to no other vertex on the outer face.



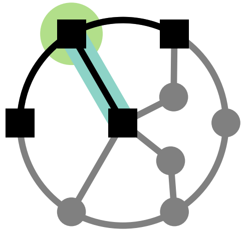
Proof – Invariants

I1 

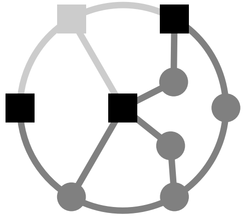
At no point in time, the set of visible edges on the outer face forms a cycle.

I2 

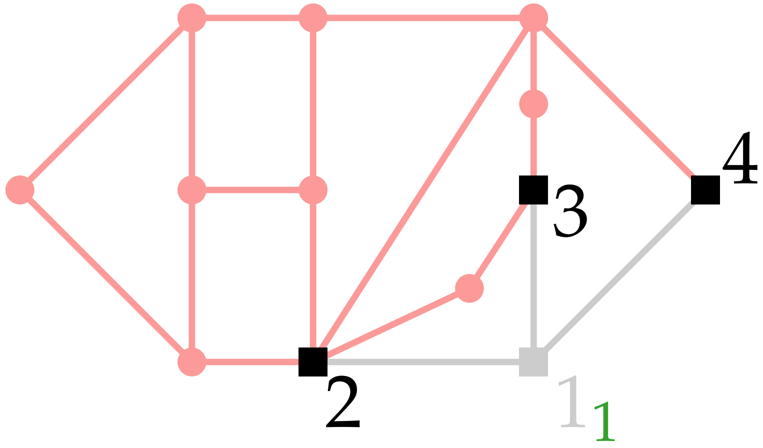
During iteration i, the only inner vertices that may be visible are those that are adjacent to v_i and to no other visible vertex on the outer face.

I3 

During iteration i, the only inner edges that may be visible are those that are incident to v_i and to no other vertex on the outer face.

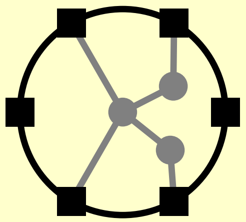
I4 

At the end of each iteration, only vertices and edges incident with the outer face are visible.

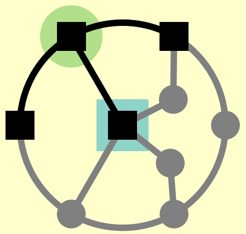


Proof – Rules

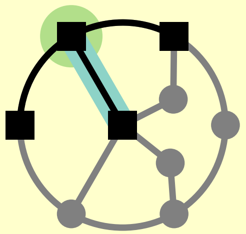
I1



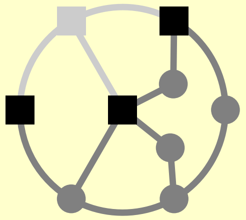
I2



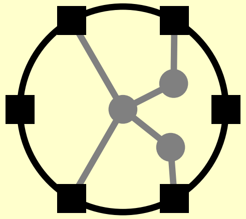
I3

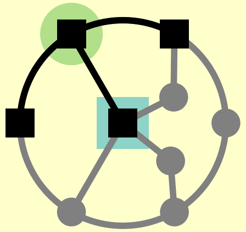


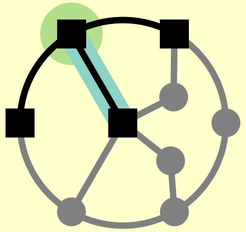
I4

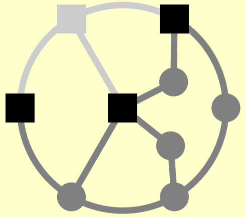


Proof – Rules

I1


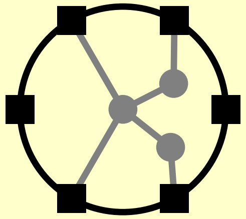
I2


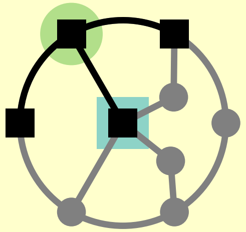
I3


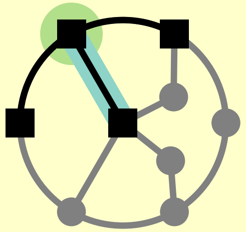
I4


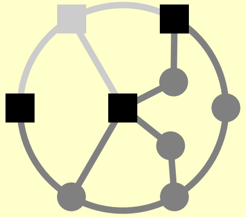
We always pick a vertex from the “current” outer face.

Proof – Rules

I1


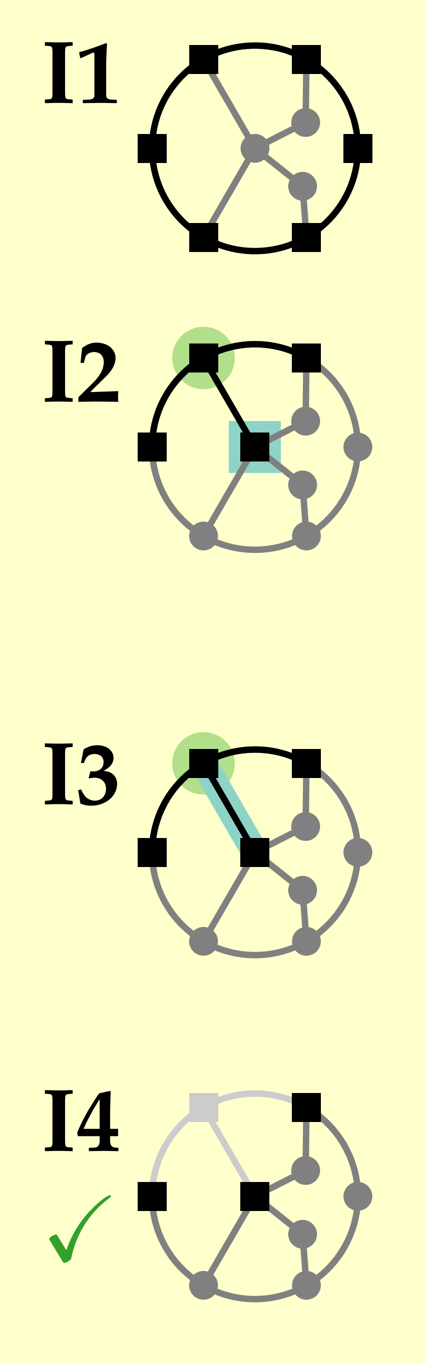
I2


I3


I4 ✓


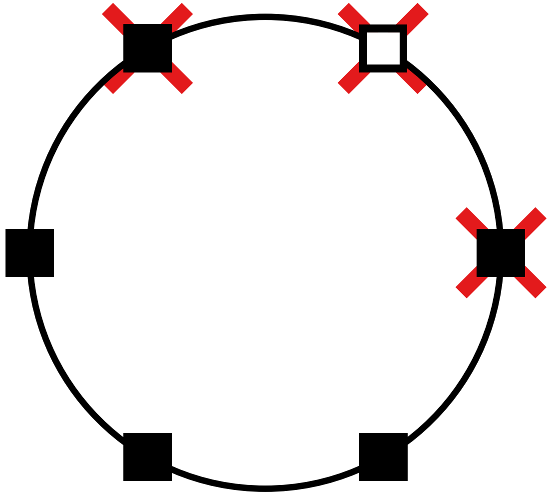
We always pick a vertex from the “current” outer face.

Proof – Rules



R1

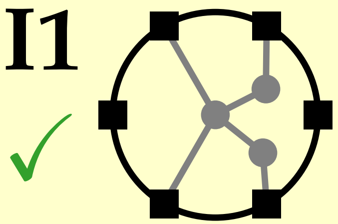
Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.



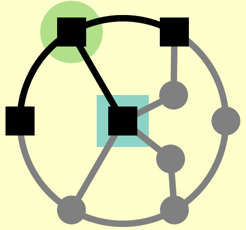
■ visible
□ invisible

Proof – Rules

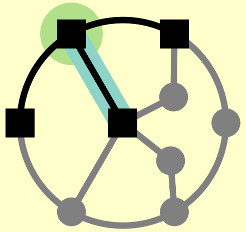
I1



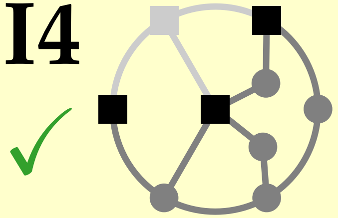
I2



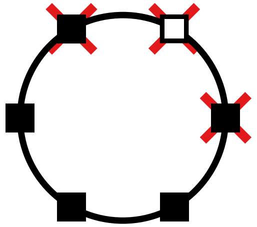
I3



I4

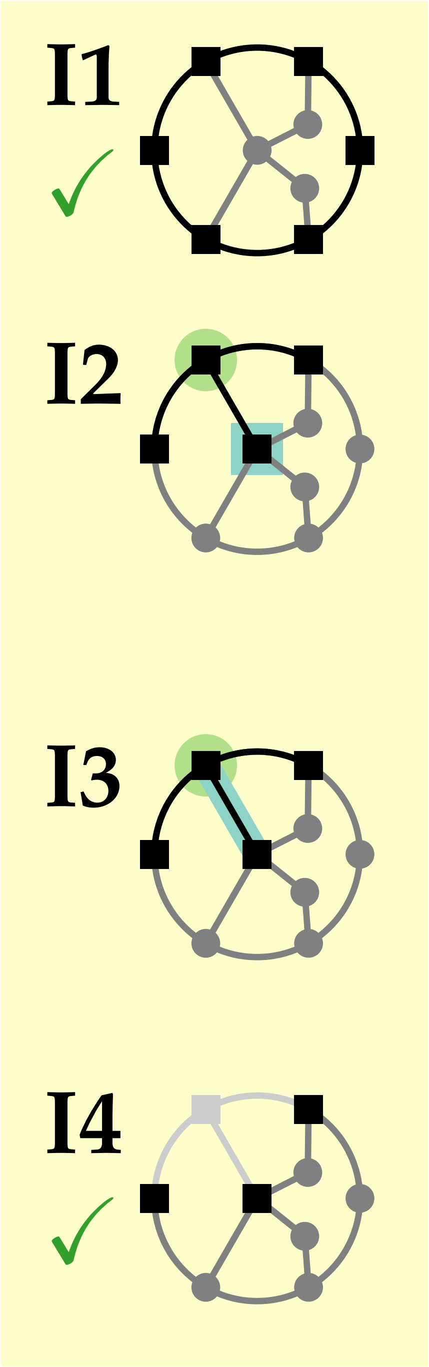


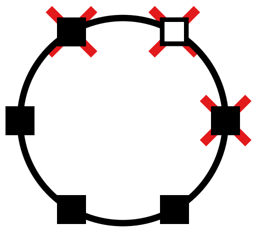
R1



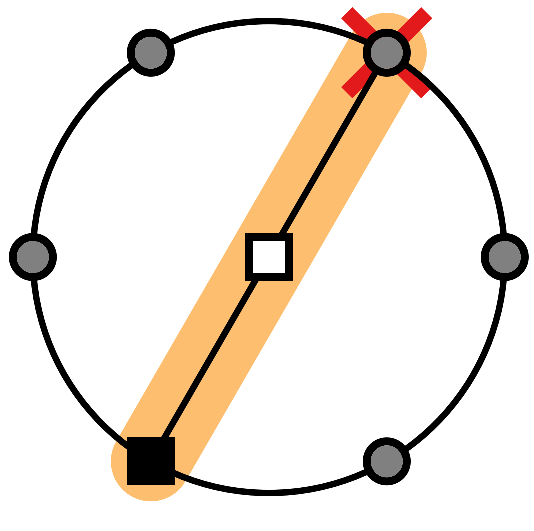
Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

Proof – Rules



R1  Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

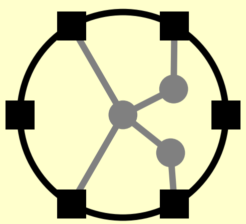
R2 Do not pick an endpoint of a *half-chord* if the other endpoint is visible.



Proof – Rules

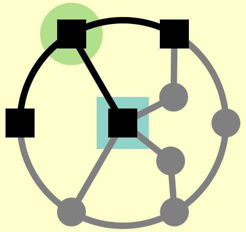
■ visible
□ invisible
● any

I1



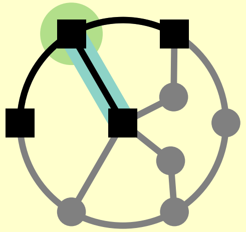
✓

I2

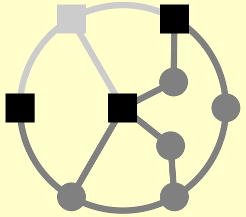


✓

I3

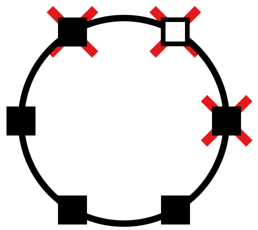


I4



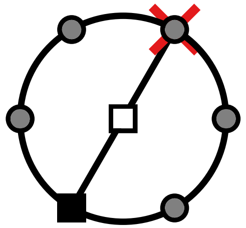
✓

R1



Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

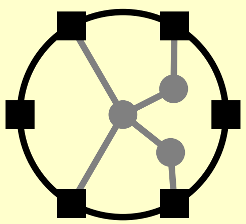
R2

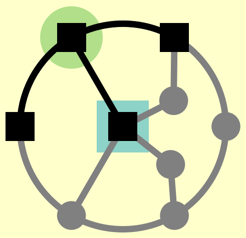


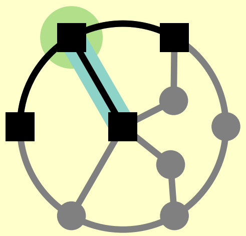
Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

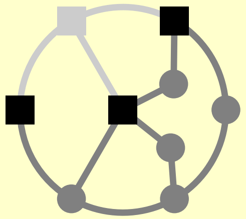
Proof – Rules

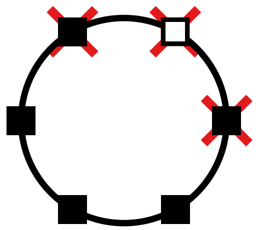
■ visible
□ invisible
● any

I1  ✓

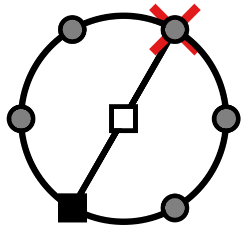
I2  ✓

I3 

I4  ✓

R1 

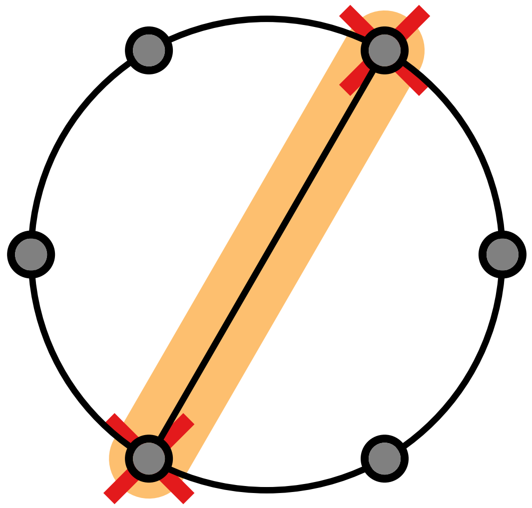
Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

R2 

Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

R3

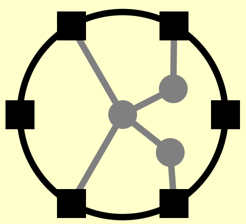
Do not pick an endpoint of a *chord*.



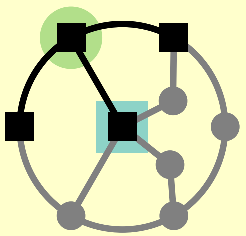
Proof – Rules

■ visible
□ invisible
● any

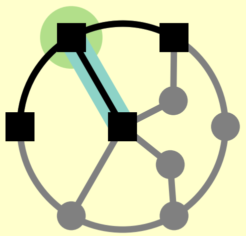
I1 ✓



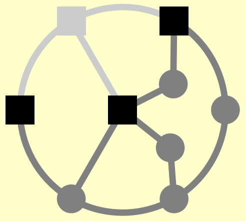
I2 ✓



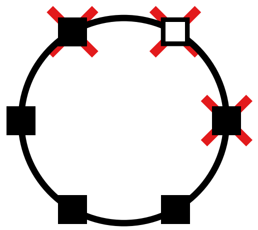
I3



I4 ✓

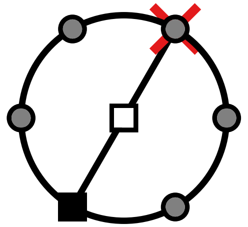


R1



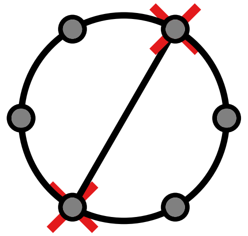
Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

R2



Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

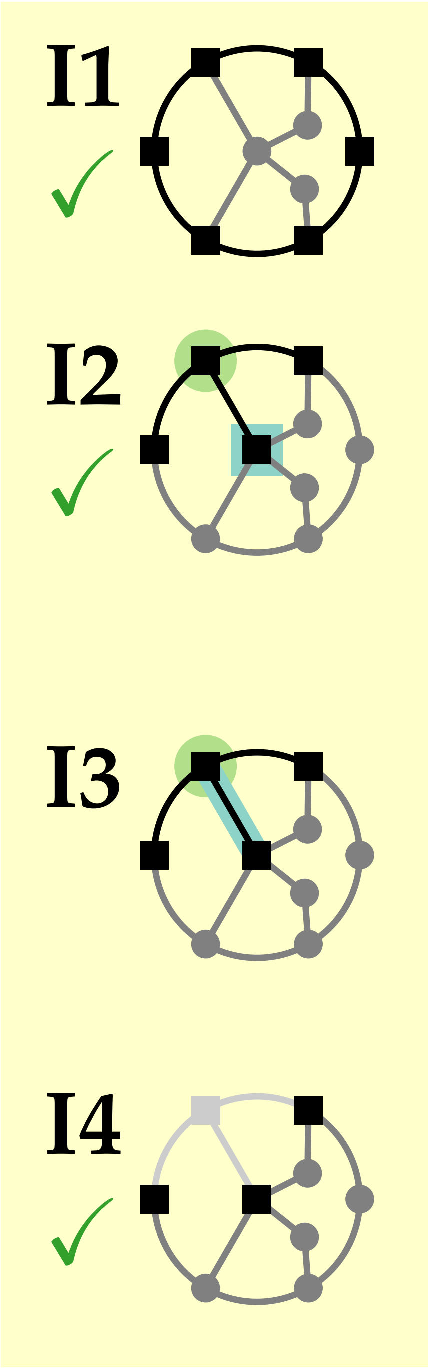
R3

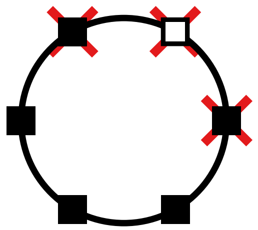


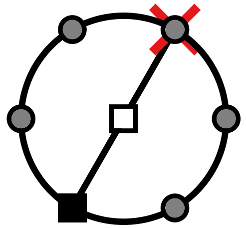
Do not pick an endpoint of a *chord*.

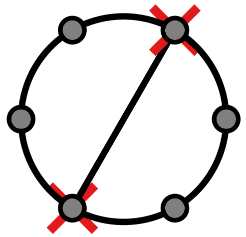
Proof – Rules

- visible
- invisible
- any

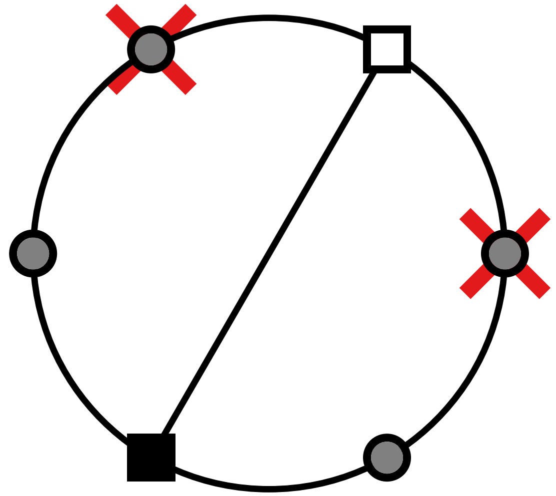


R1  Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

R2  Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

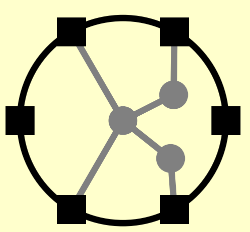
R3  Do not pick an endpoint of a *chord*.

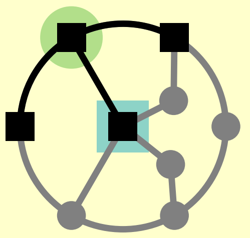
R4 Do not pick a neighbor of an endpoint of a chord if the other endpoint of that chord is visible.

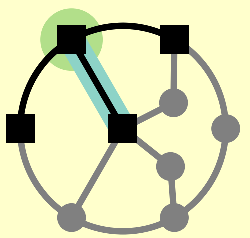


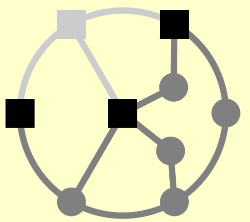
Proof – Rules

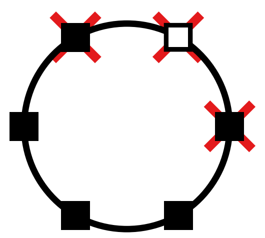
■ visible
□ invisible
● any

I1 
✓

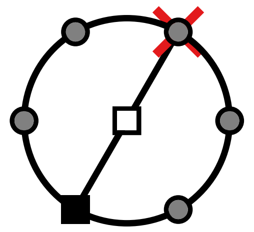
I2 
✓

I3 

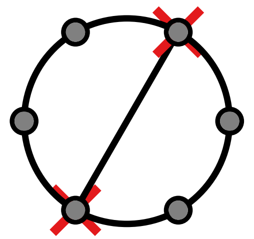
I4 
✓

R1 

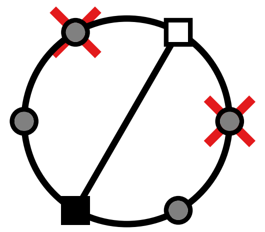
Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

R2 

Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

R3 

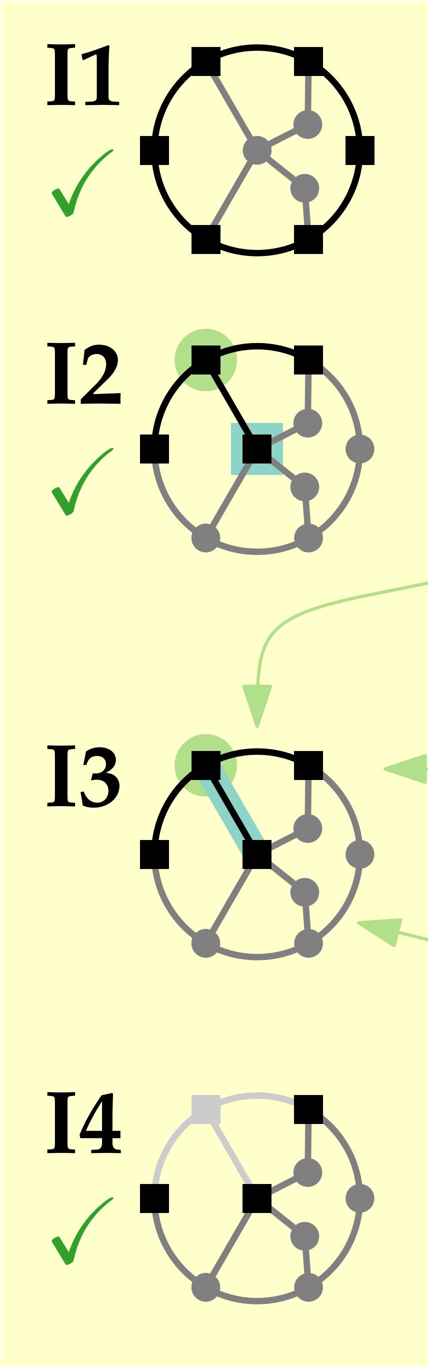
Do not pick an endpoint of a *chord*.

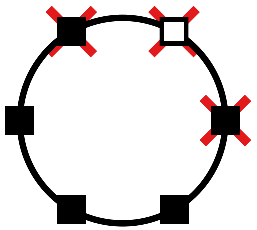
R4 

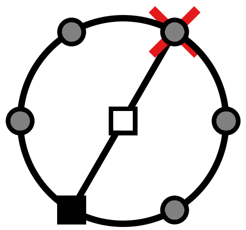
Do not pick a neighbor of an endpoint of a chord if the other endpoint of that chord is visible.

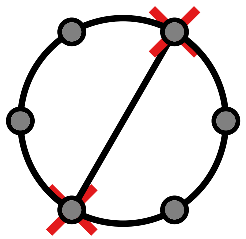
Proof – Rules

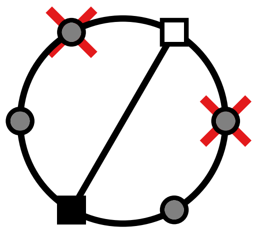
■ visible
□ invisible
● any



R1  Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

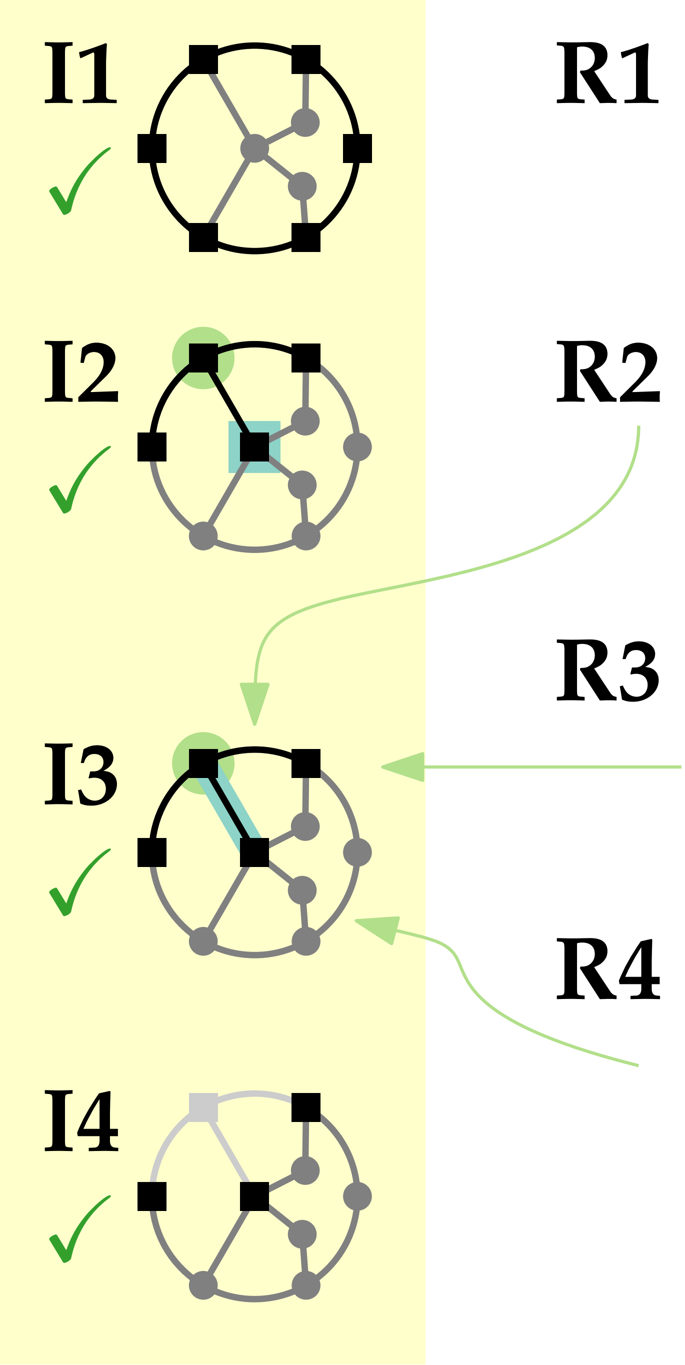
R2  Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

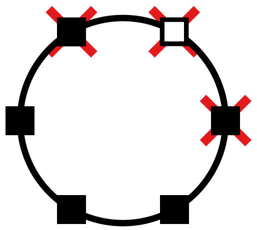
R3  Do not pick an endpoint of a *chord*.

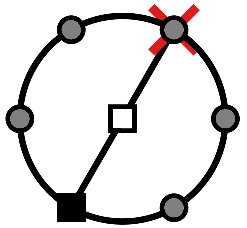
R4  Do not pick a neighbor of an endpoint of a chord if the other endpoint of that chord is visible.

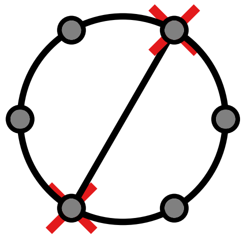
Proof – Rules

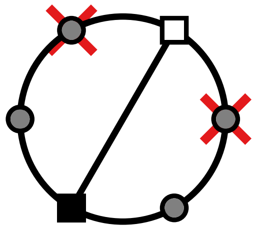
■ visible
□ invisible
● any



R1  Do not pick a vertex whose closed neighborhood contains all invisible vertices of the outer face.

R2  Do not pick an endpoint of a *half-chord* if the other endpoint is visible.

R3  Do not pick an endpoint of a *chord*.

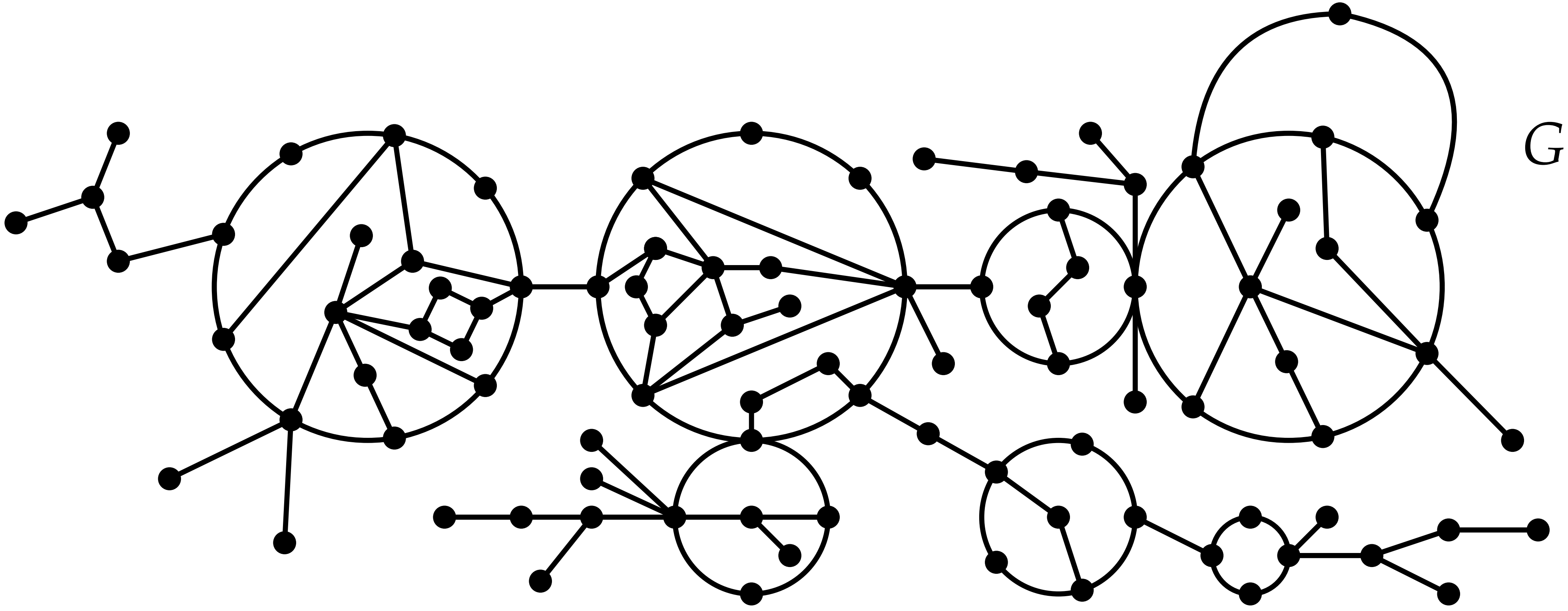
R4  Do not pick a neighbor of an endpoint of a chord if the other endpoint of that chord is visible.

Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Proof – Reducing the Graph

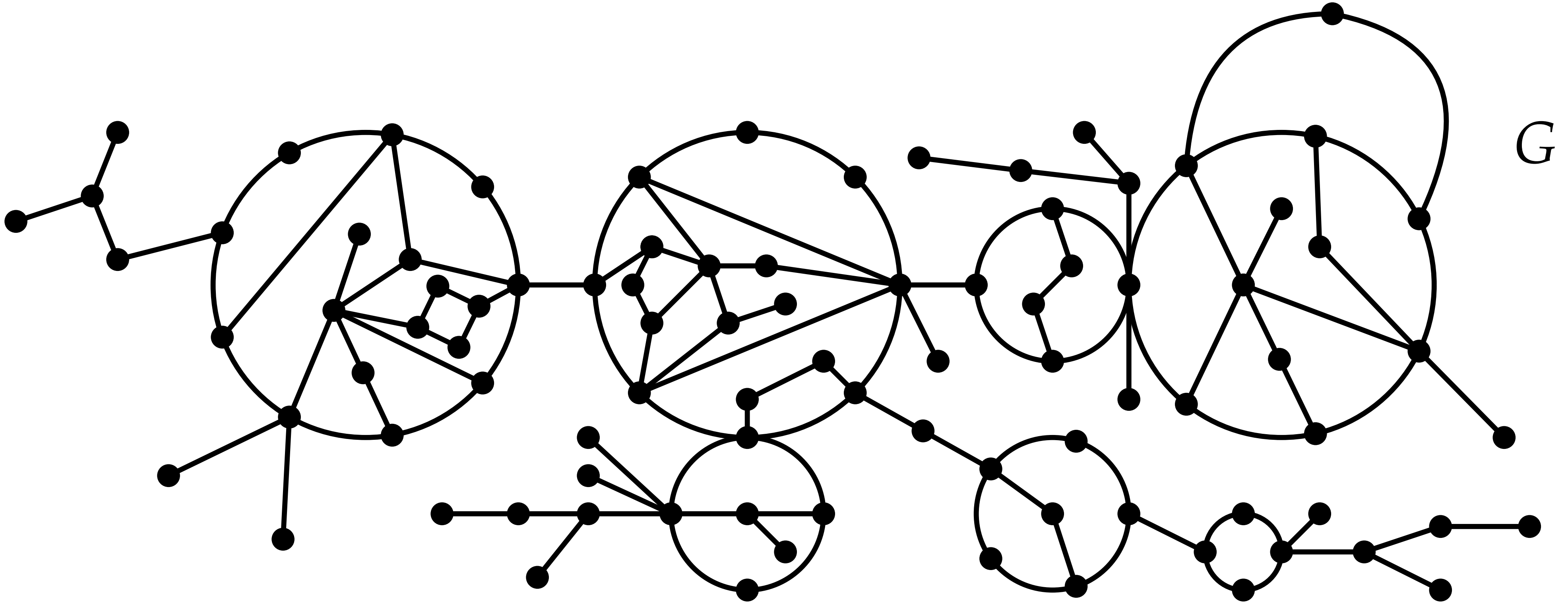
To show: There always is a vertex that can be picked.



Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

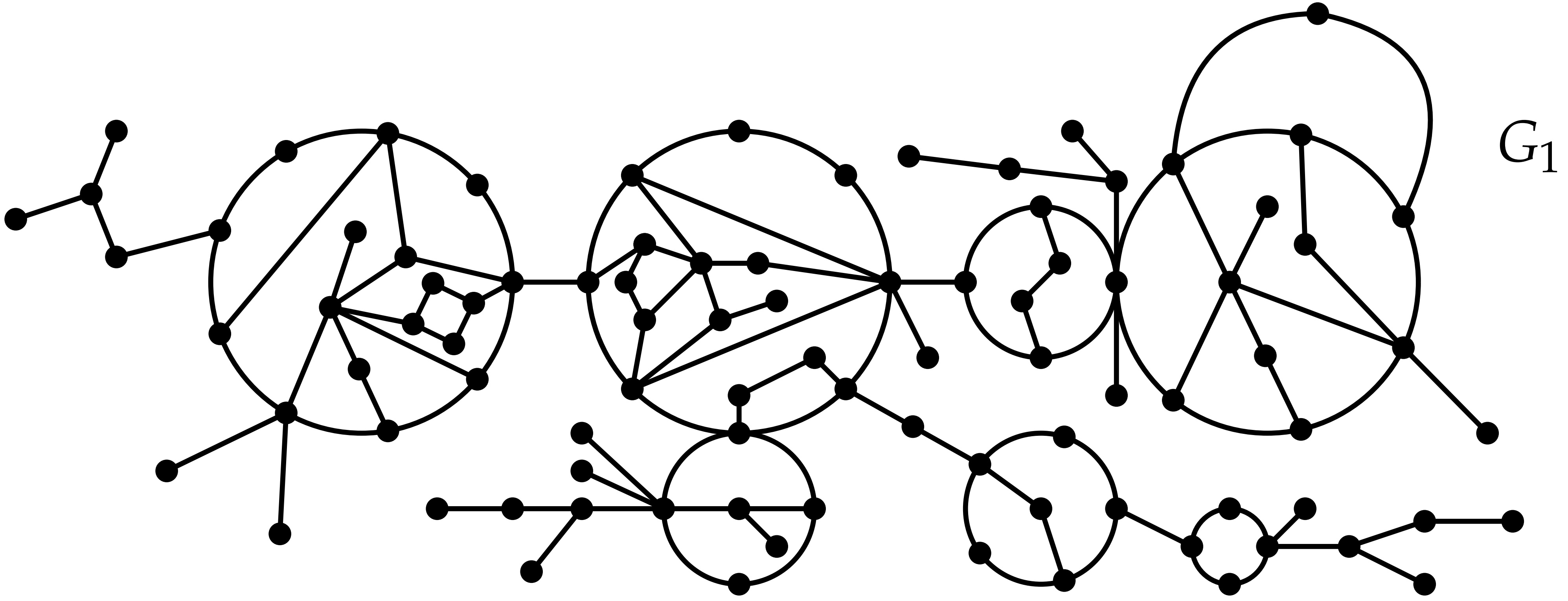
Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.



Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

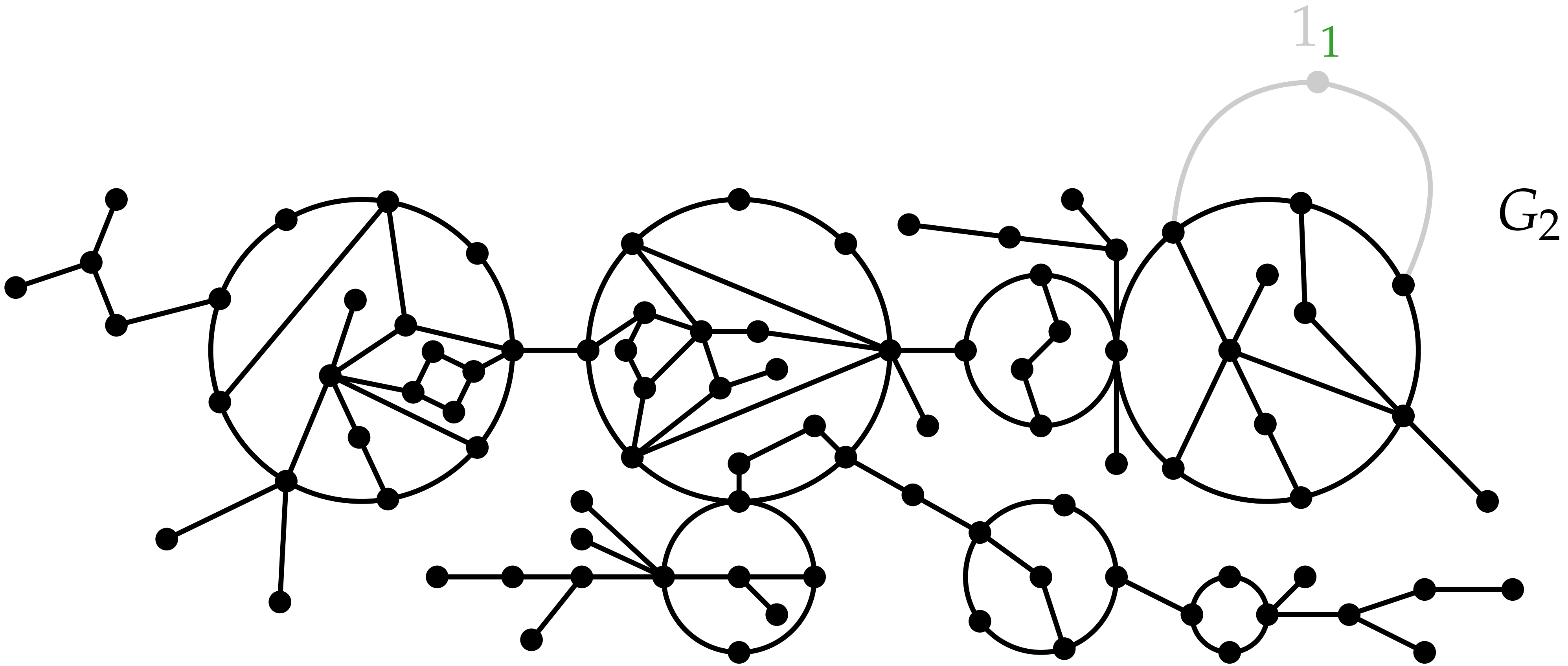
Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.



Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

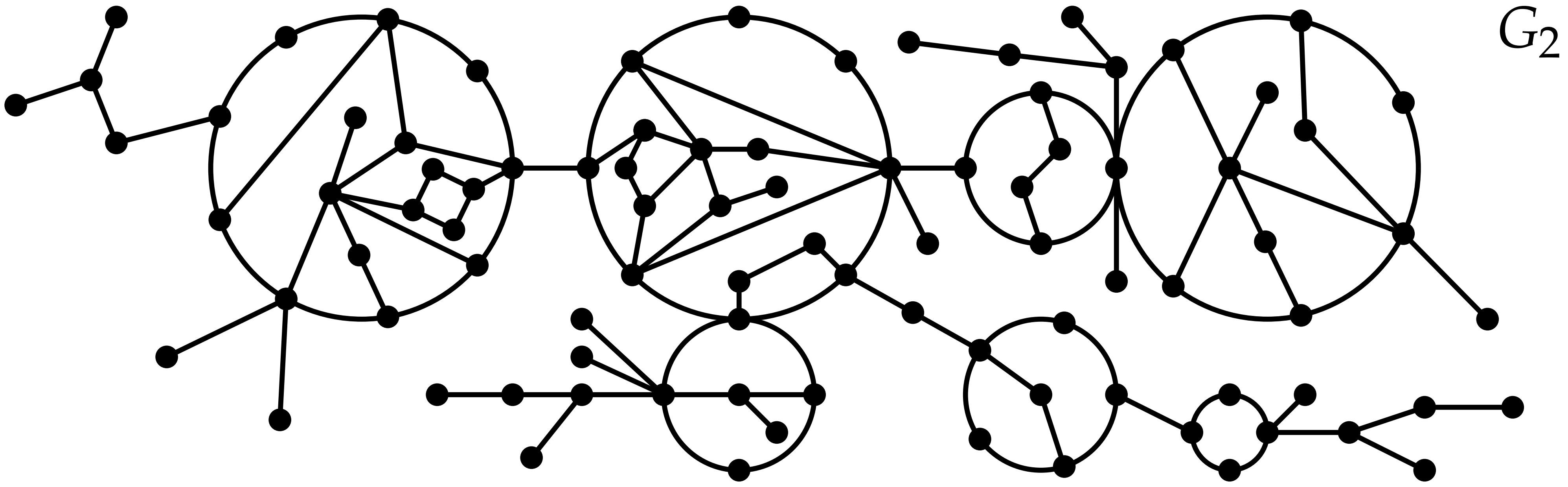
Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.



Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

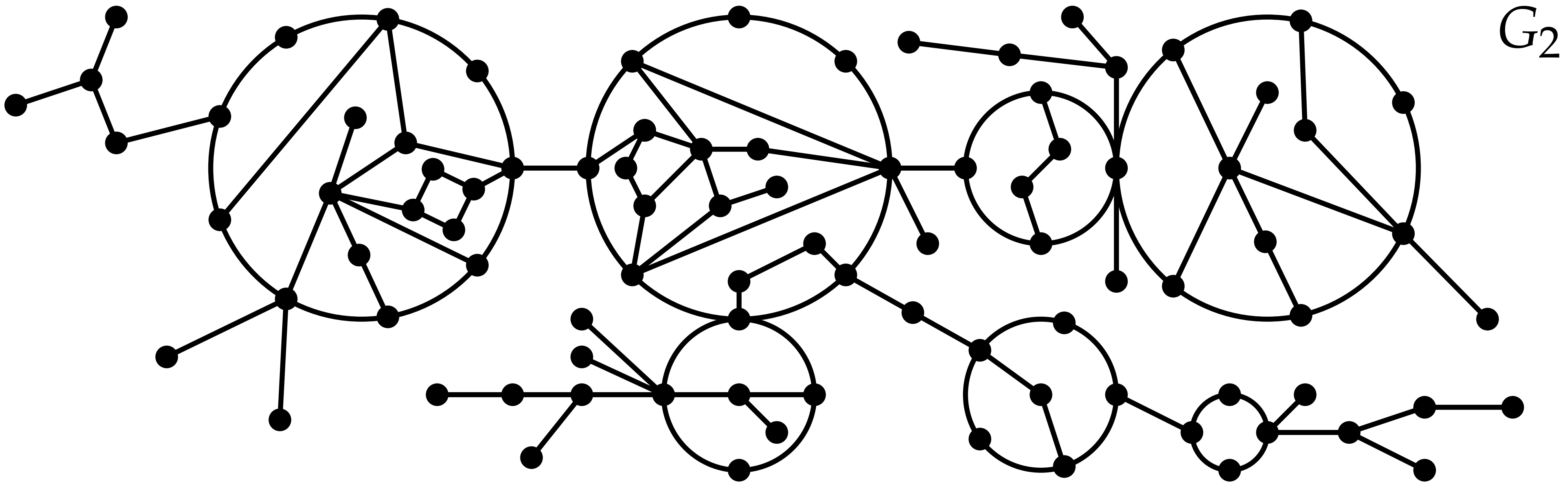


Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G'_i be the subgraph of G_i that consists of:



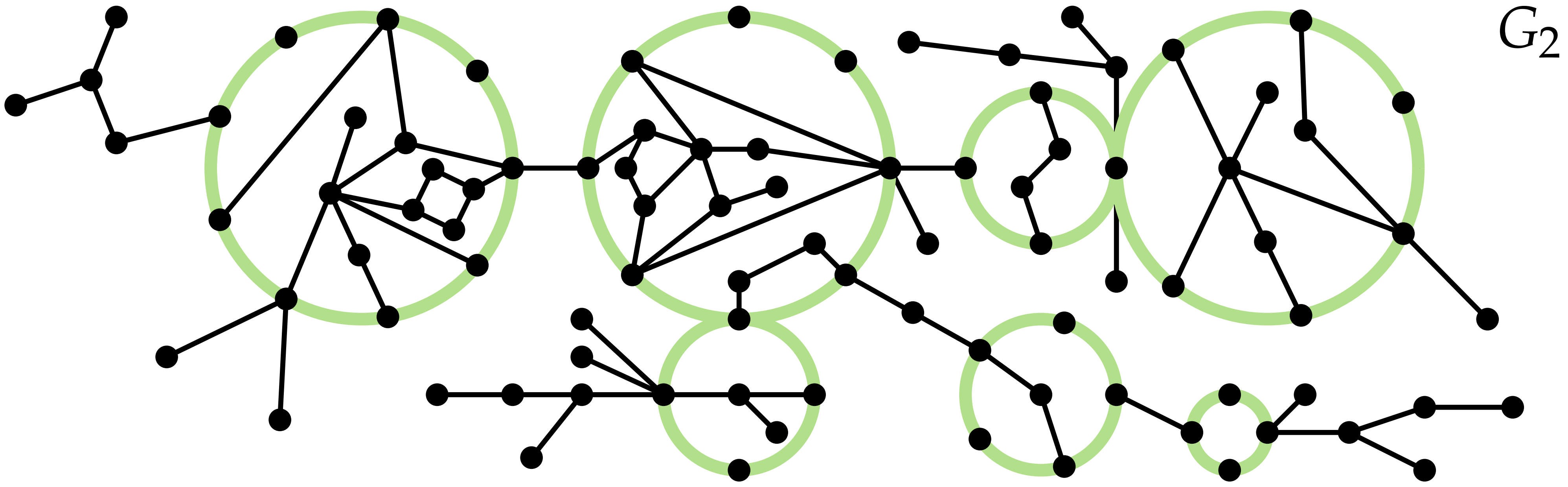
Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G'_i be the subgraph of G_i that consists of:

- vertices and edges that lie on a simple cycle that bounds the outer face of G_i,



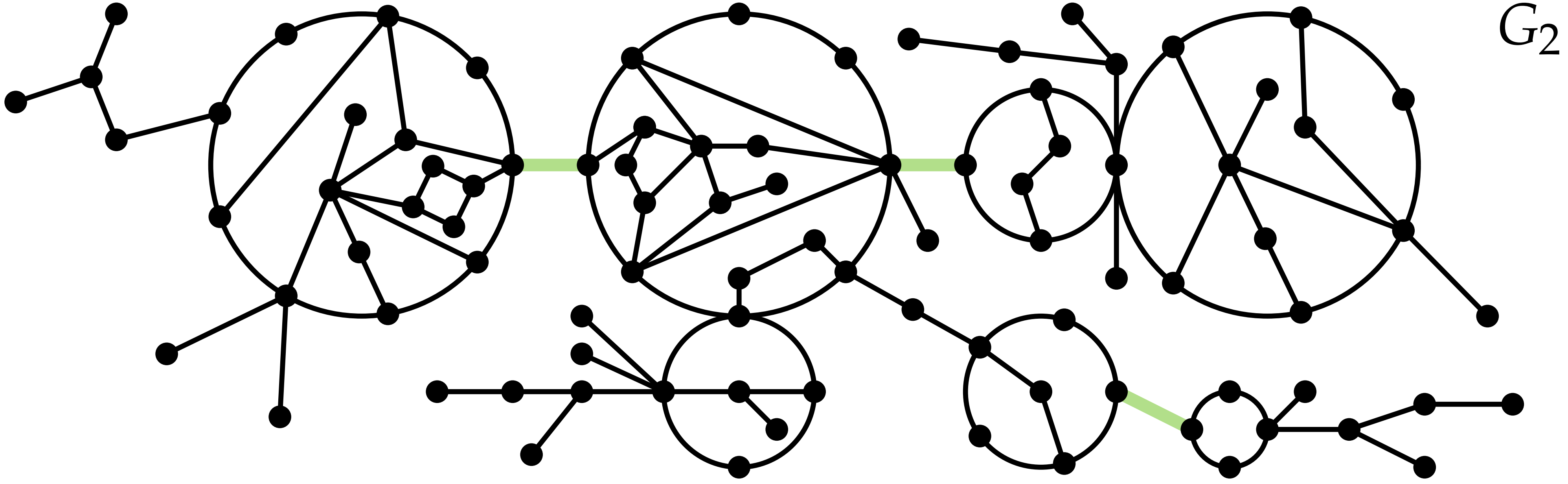
Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G_i' be the subgraph of G_i that consists of:

- vertices and edges that lie on a simple cycle that bounds the outer face of G_i,
- every edge that connects two cycles,



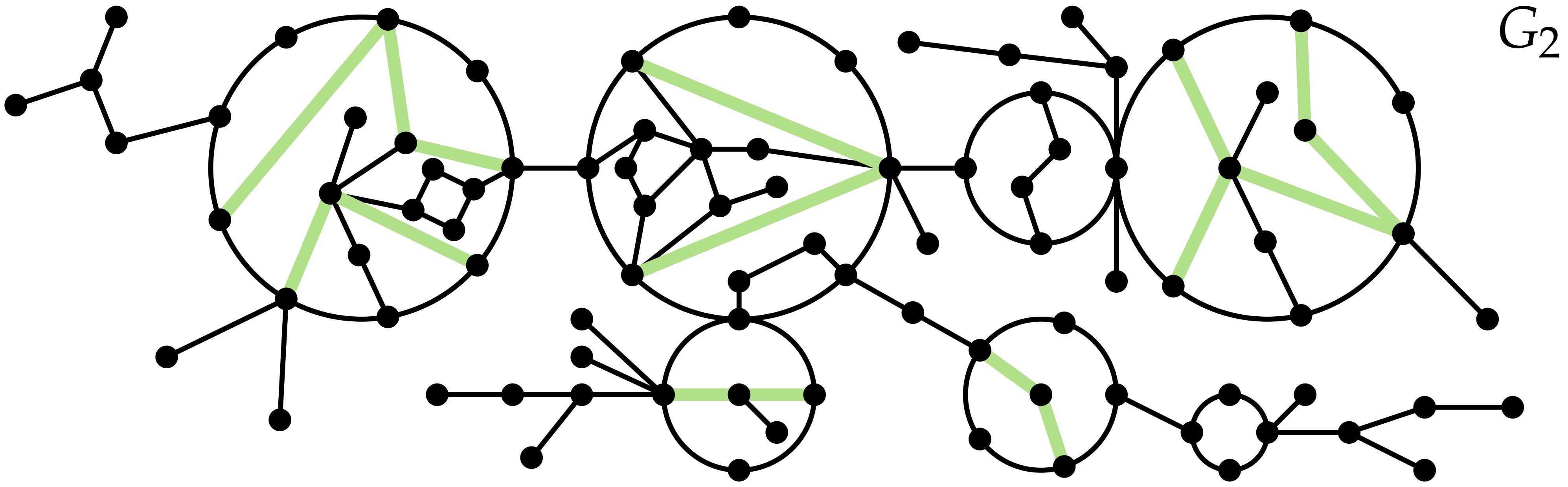
Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G_i' be the subgraph of G_i that consists of:

- vertices and edges that lie on a simple cycle that bounds the outer face of G_i,
- every edge that connects two cycles,
- all chords and half-chords of G_i.



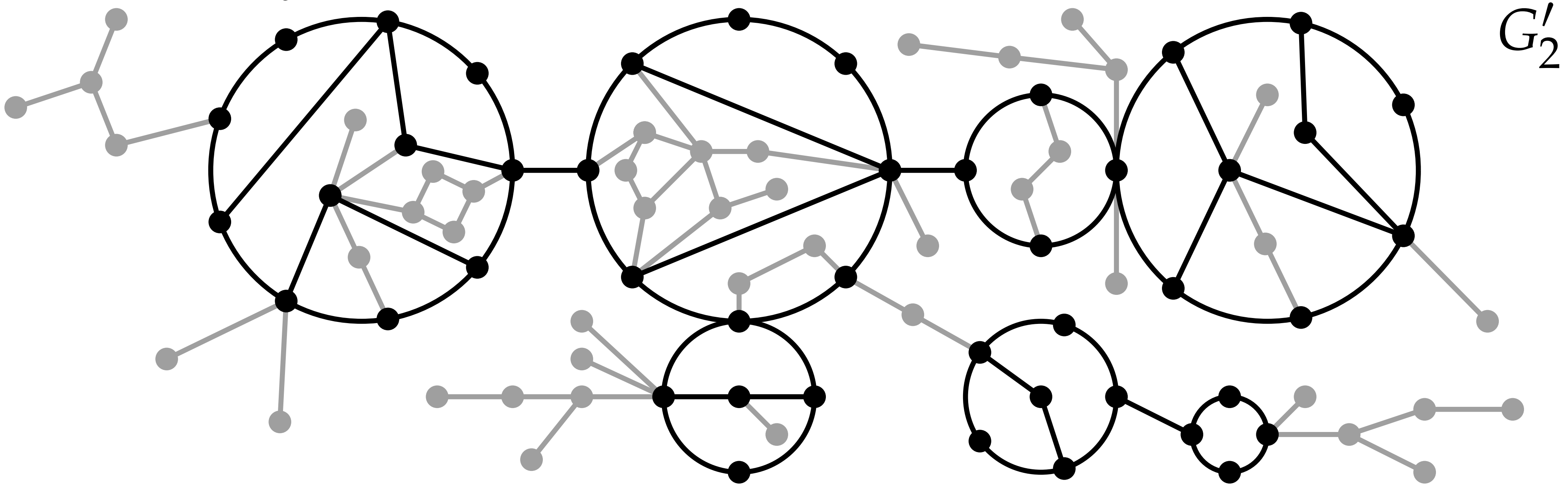
Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G_i' be the subgraph of G_i that consists of:

- vertices and edges that lie on a simple cycle that bounds the outer face of G_i,
- every edge that connects two cycles,
- all chords and half-chords of G_i.



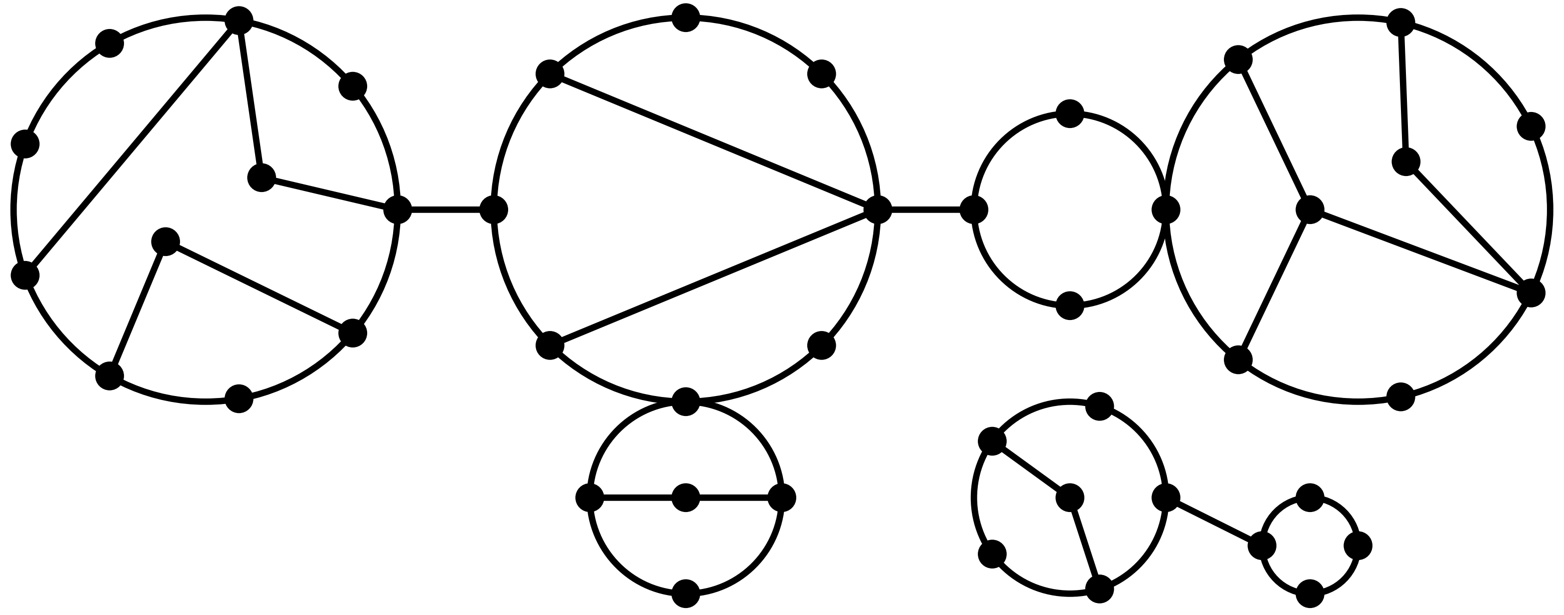
Proof – Reducing the Graph

To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G_i' be the subgraph of G_i that consists of:

- vertices and edges that lie on a simple cycle that bounds the outer face of G_i,
- every edge that connects two cycles,
- all chords and half-chords of G_i.



Proof – Reducing the Graph

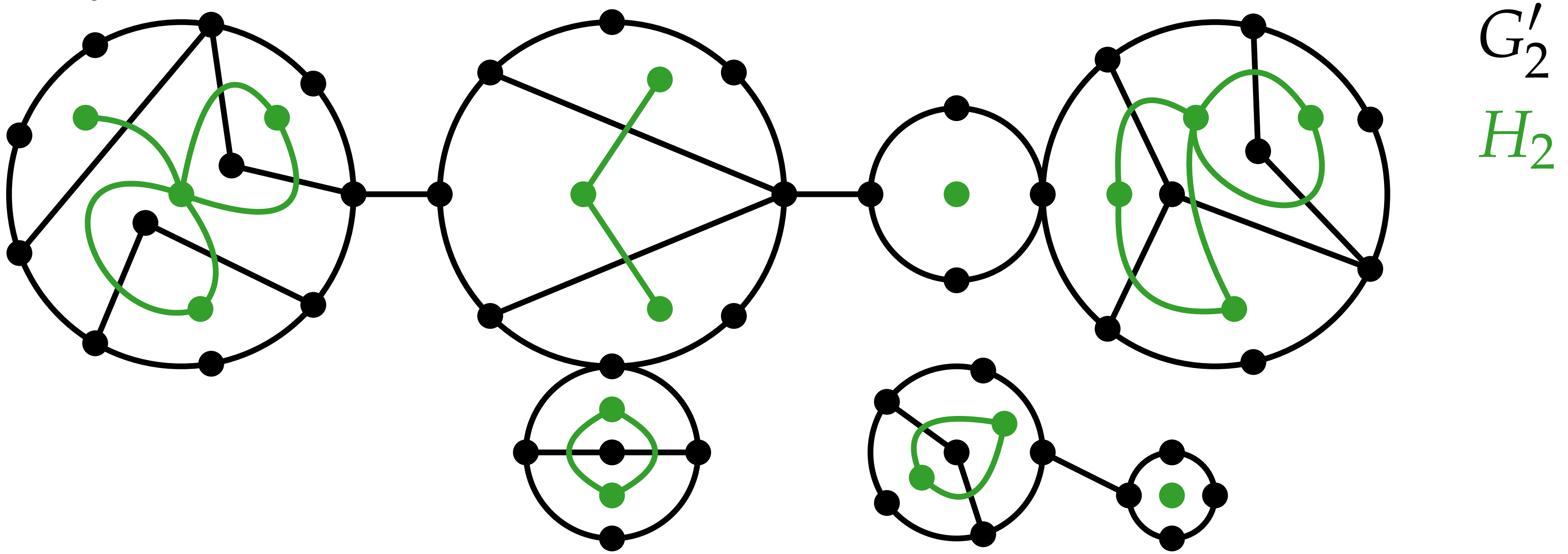
To show: There always is a vertex that can be picked.

Let $G_1 = G$ and, for $i \in \{1, 2, \dots\}$, let G_{i+1} be the subgraph of G_i that we obtain after removing the vertices (and the edges incident to them) that disappear in iteration i.

Let G'_i be the subgraph of G_i that consists of:

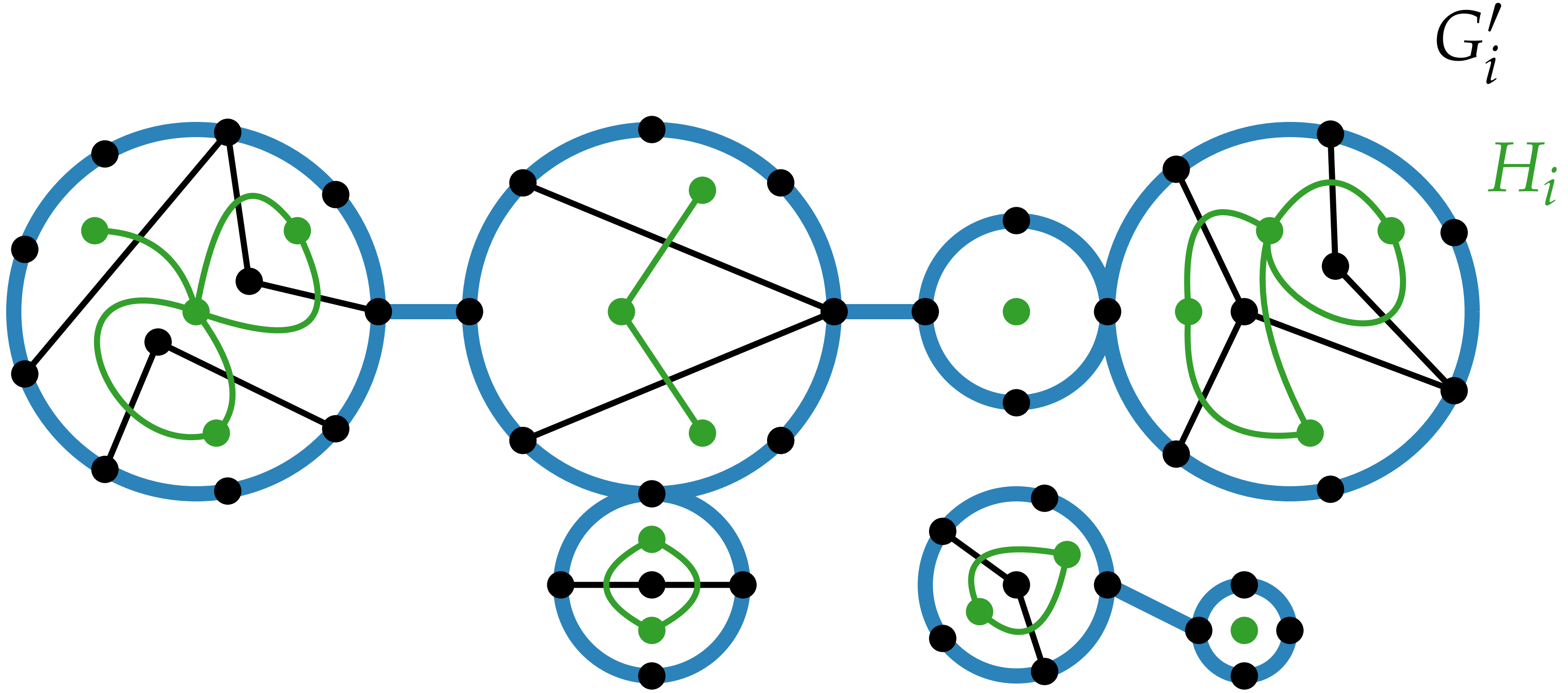
- vertices and edges that lie on a simple cycle that bounds the outer face of G_i,
- every edge that connects two cycles,
- all chords and half-chords of G_i.

Let H_i be the weak dual of G'_i.



Proof – General Case for the Outer Face

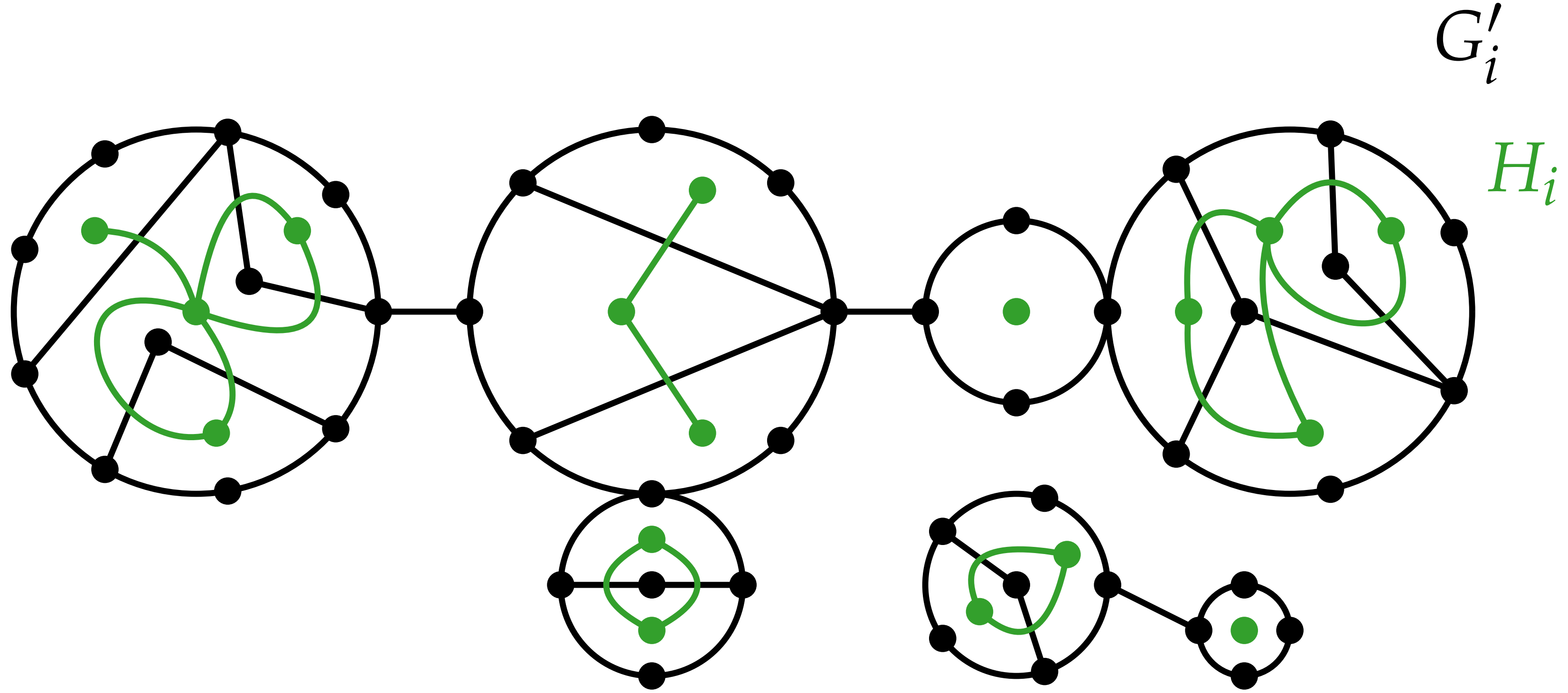
Note that in general the outer face of G_i' is a *cactus* forest.



Proof – General Case for the Outer Face

Note that in general the outer face of G'_i is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G'_i is a simple cycle.

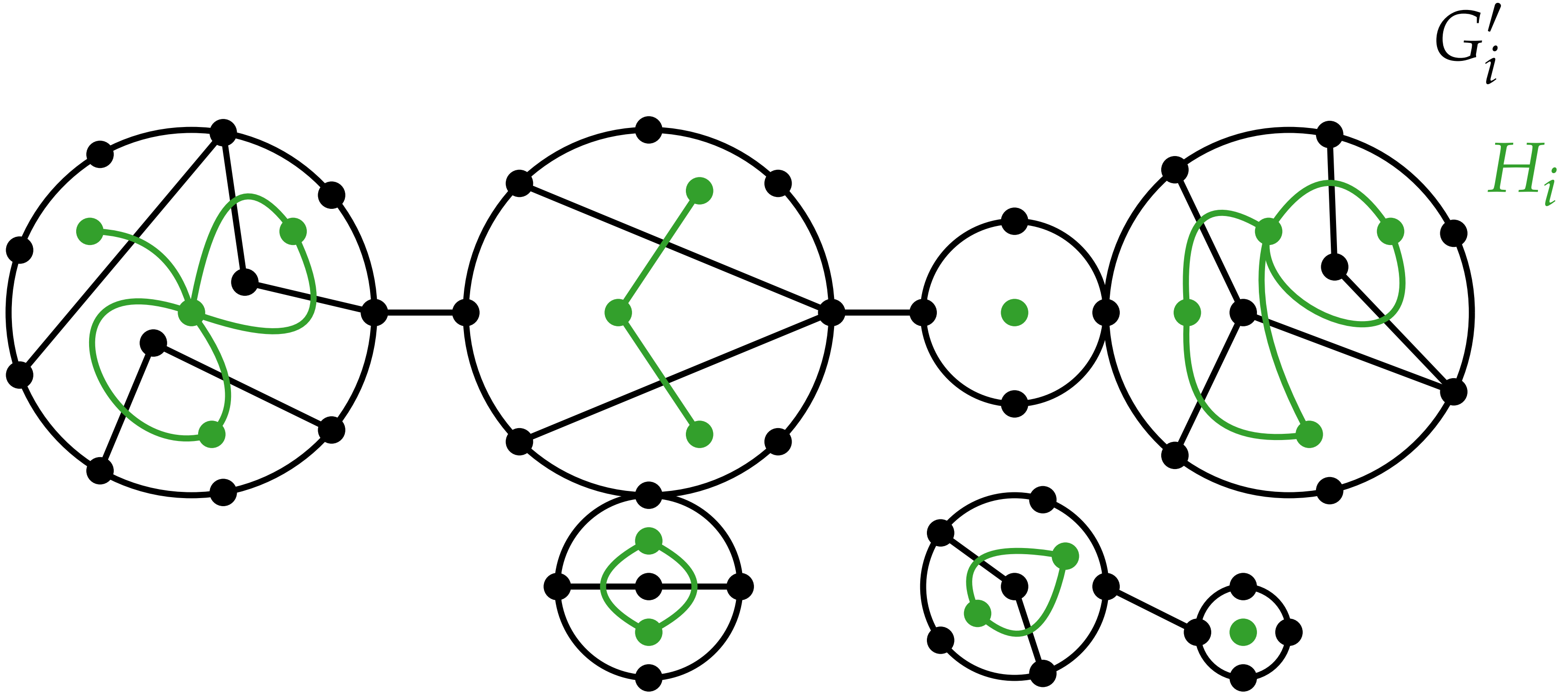


Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

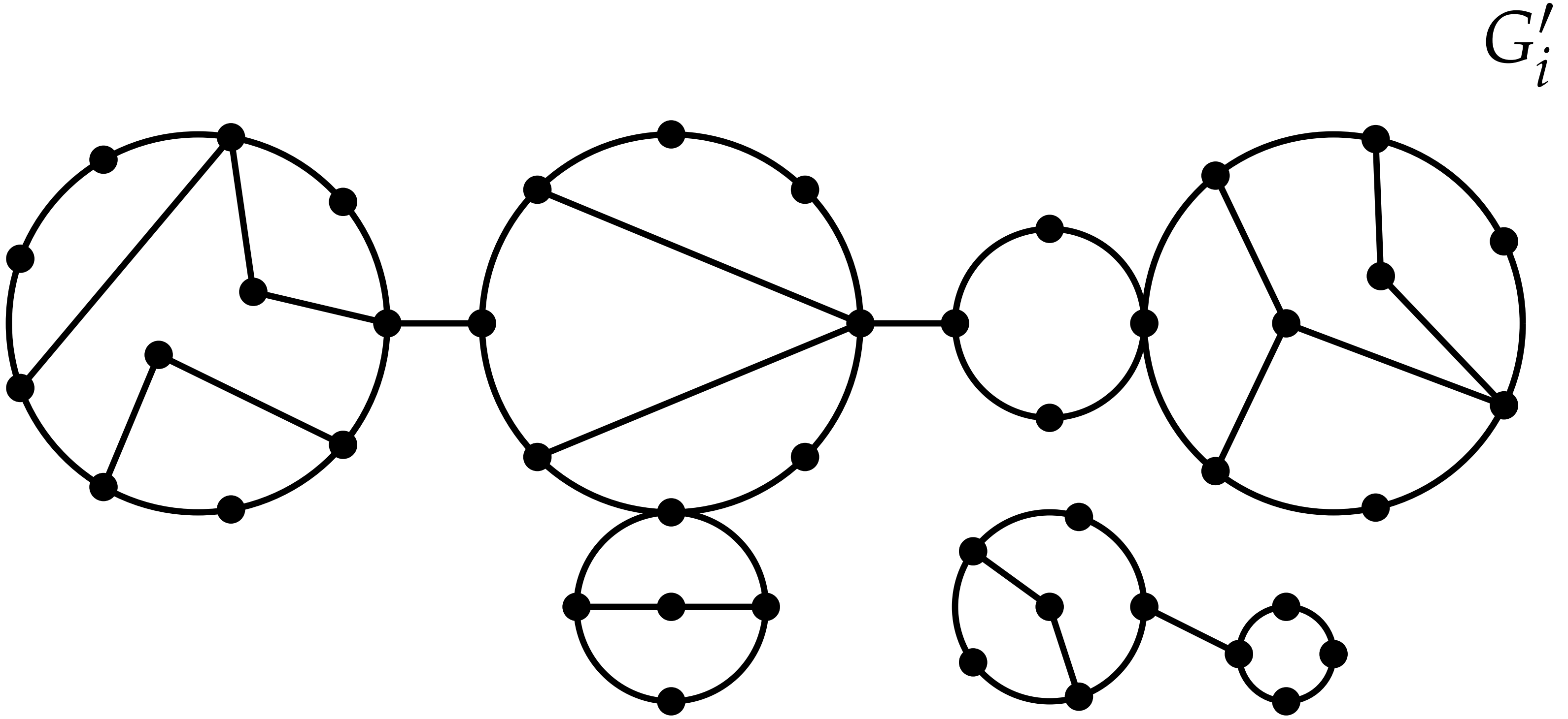


Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

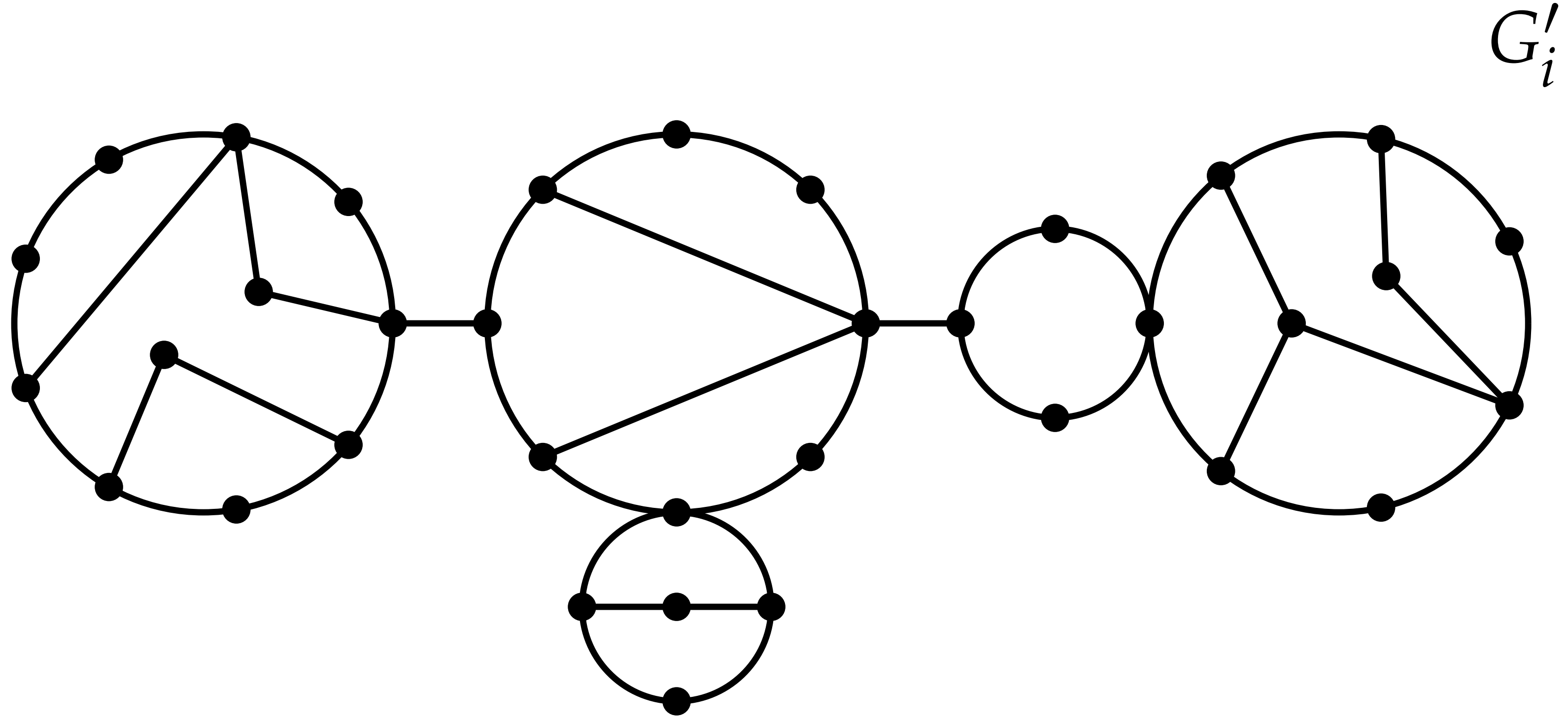


Proof – General Case for the Outer Face

Note that in general the outer face of G'_i is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G'_i is a simple cycle.

We skip this part.

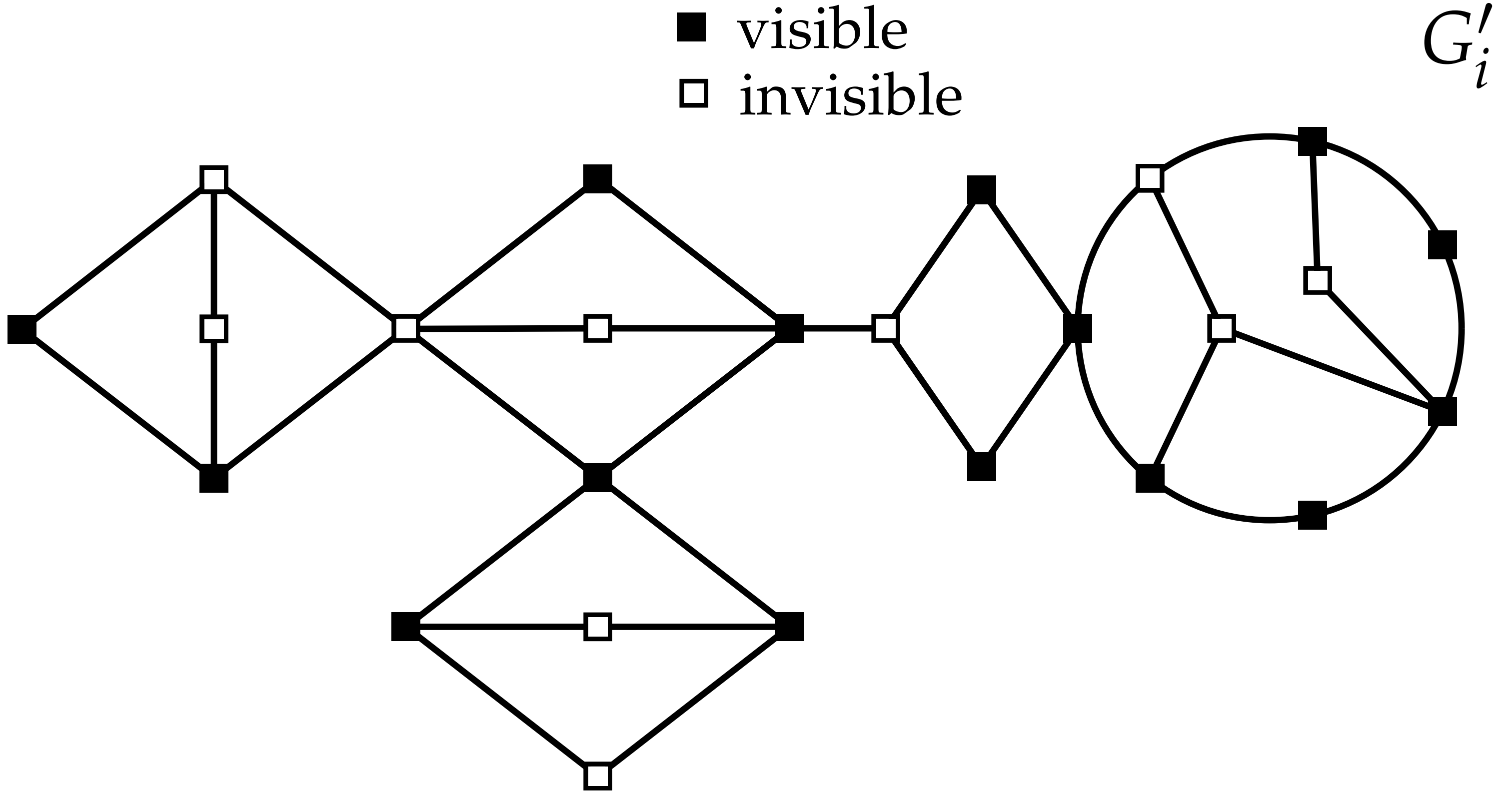


Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.



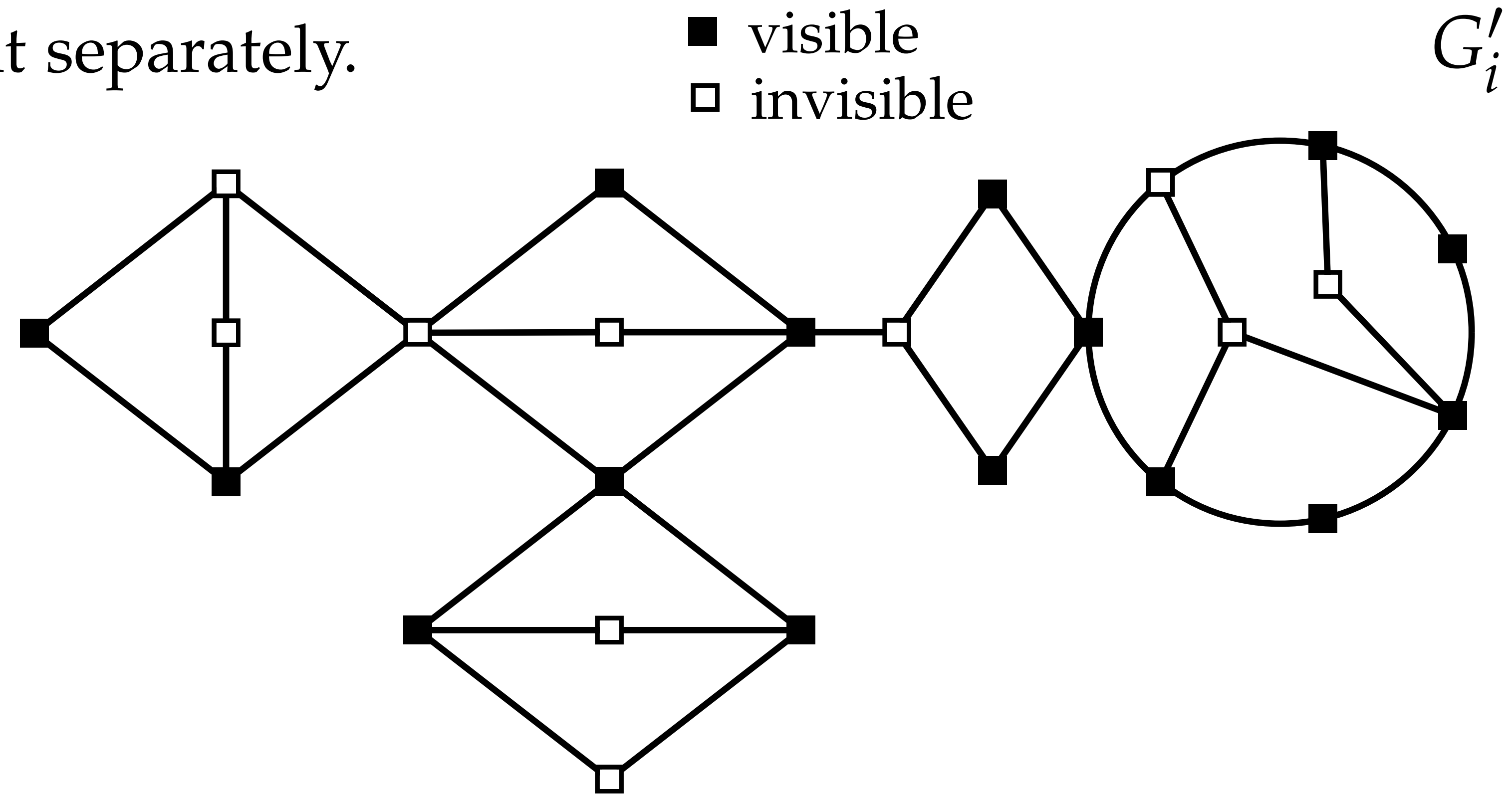
Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.



Proof – General Case for the Outer Face

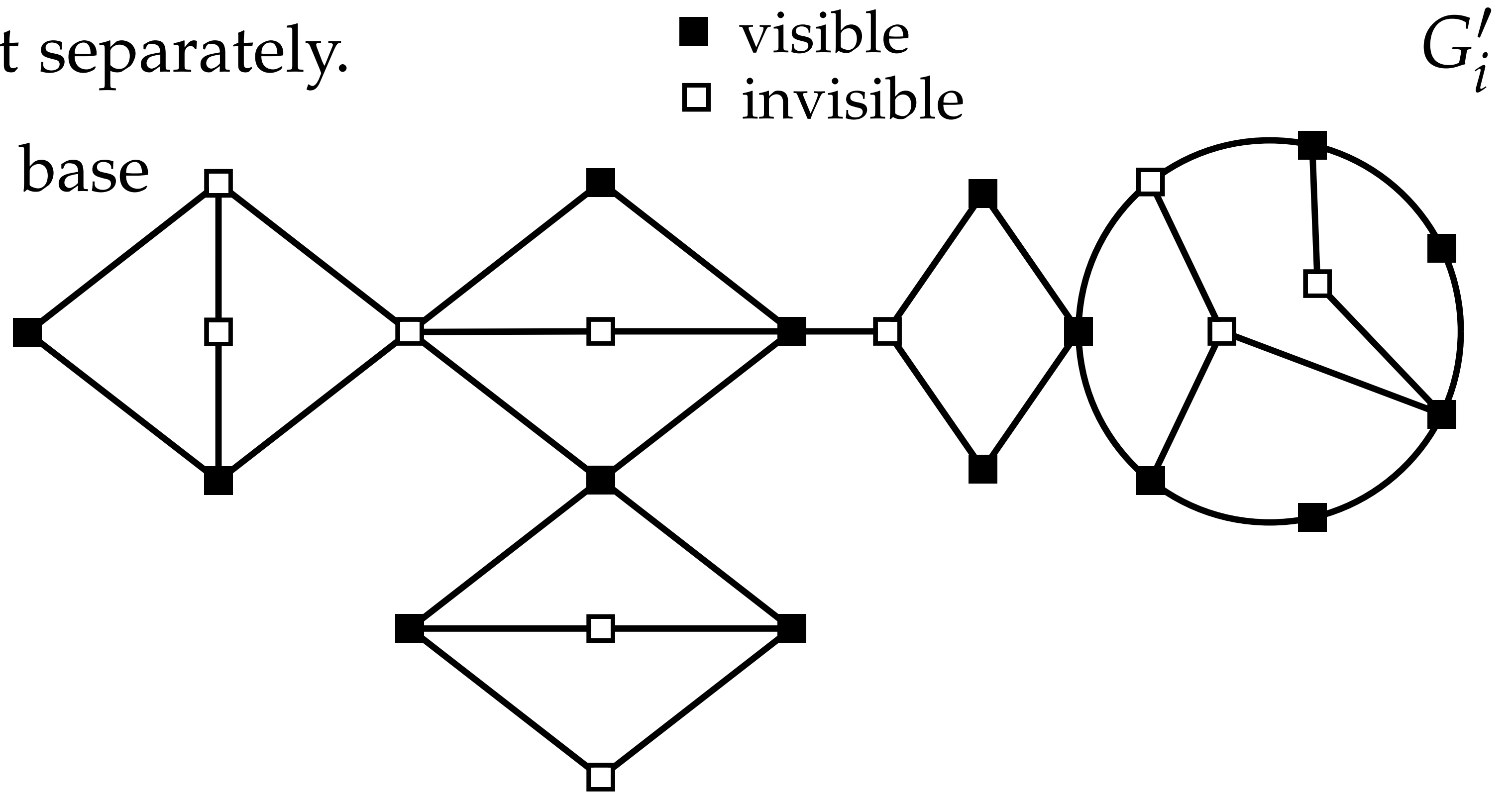
Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.



Proof – General Case for the Outer Face

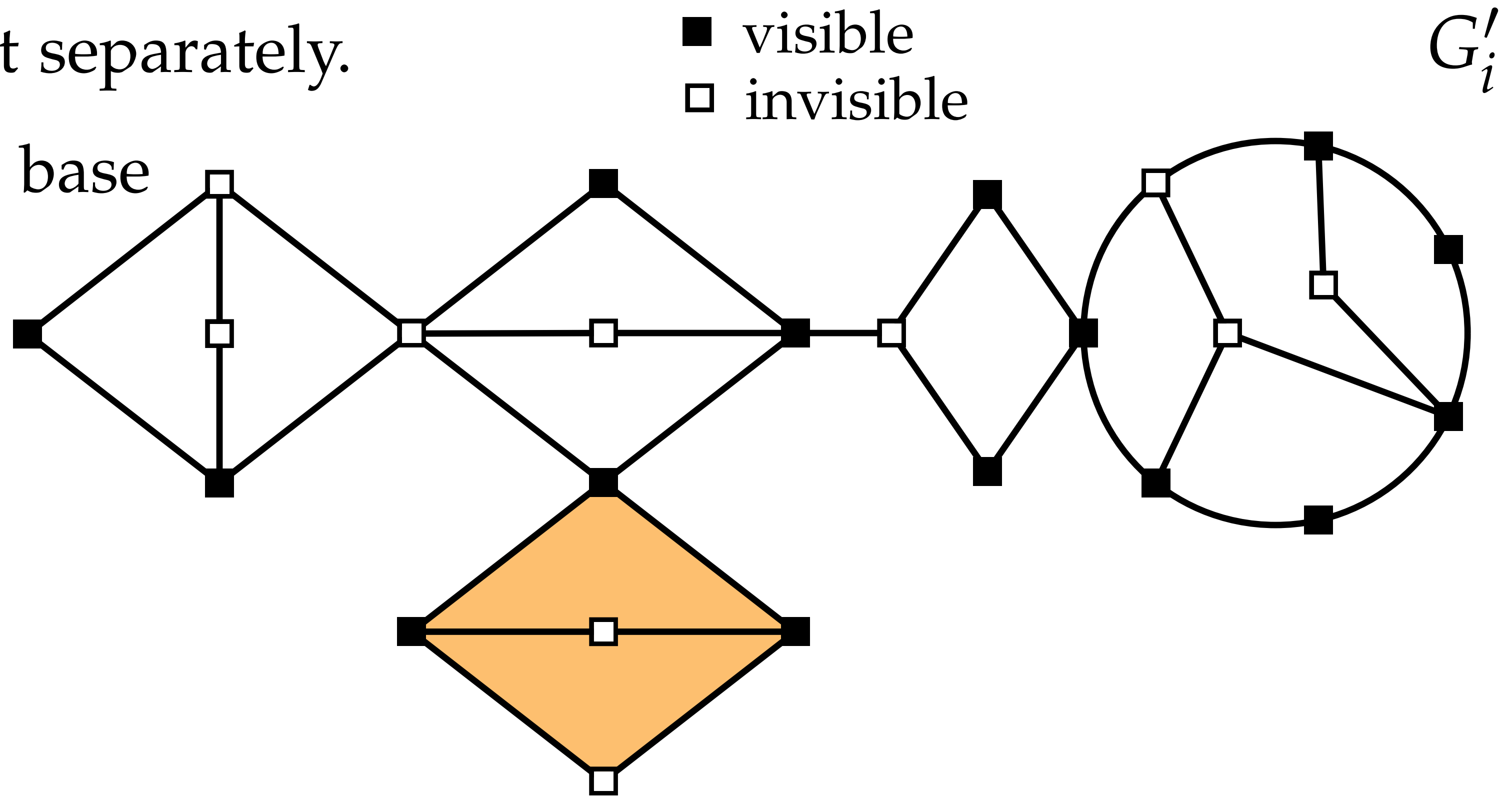
Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.



Proof – General Case for the Outer Face

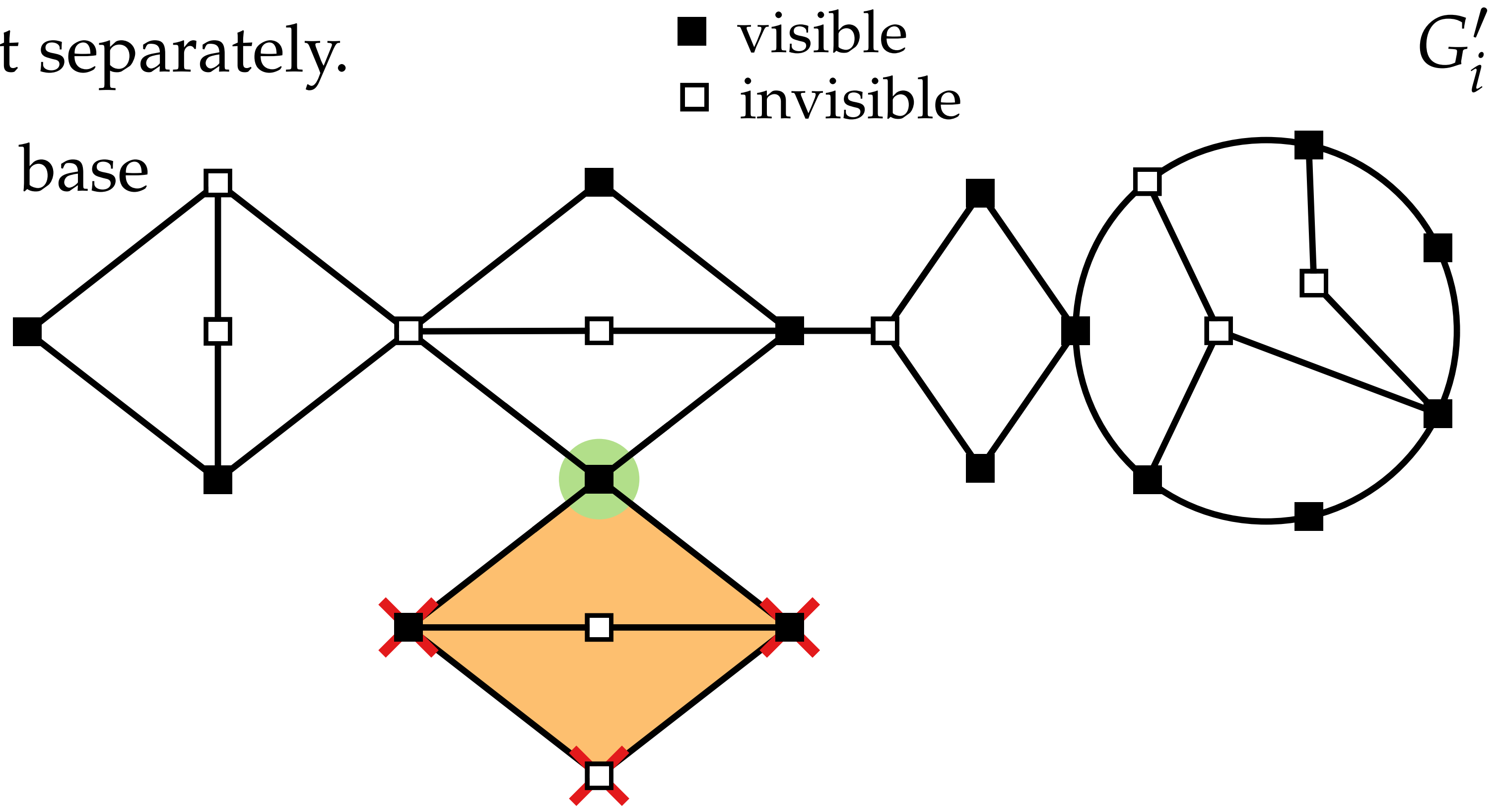
Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

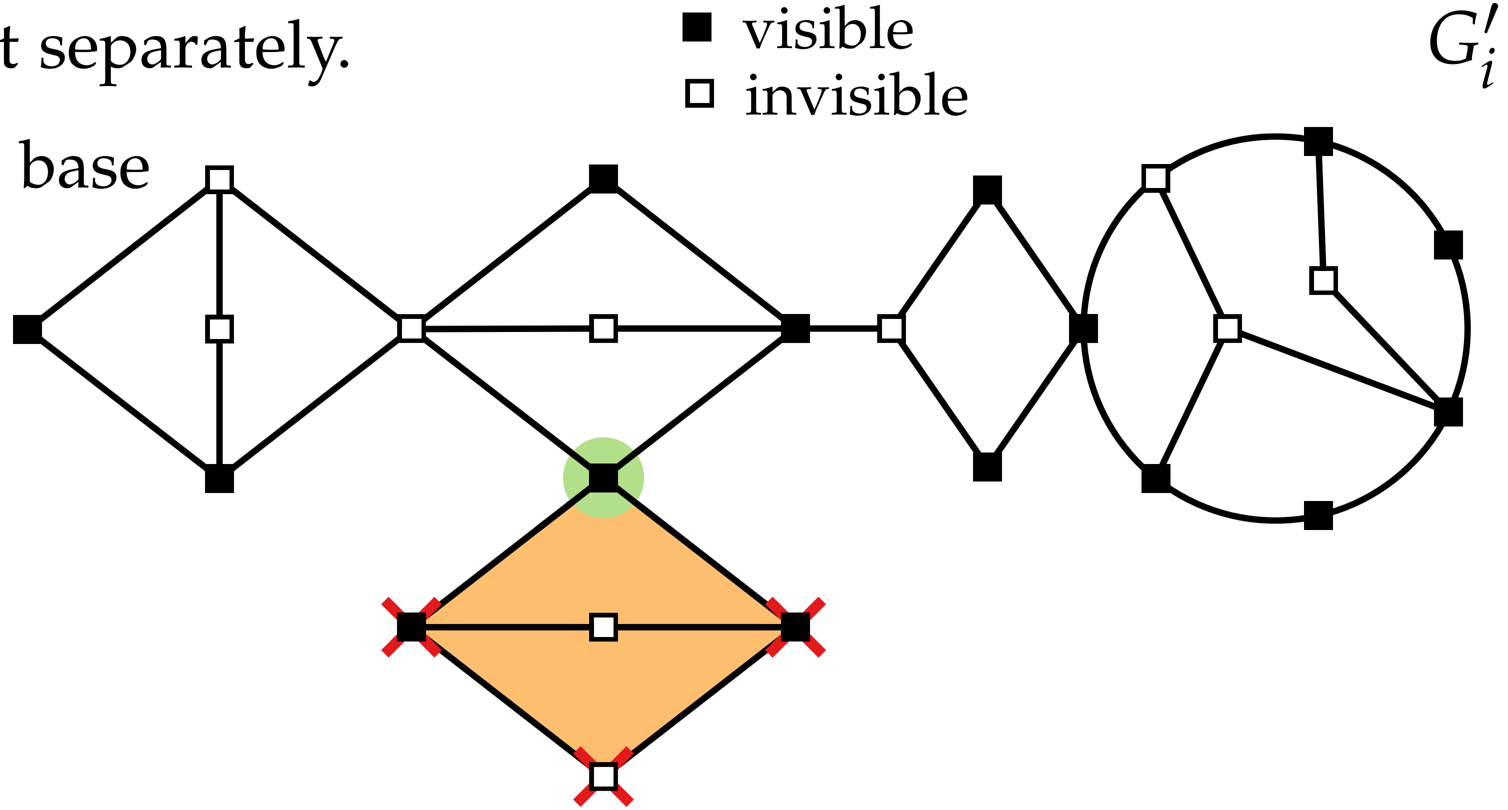
With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

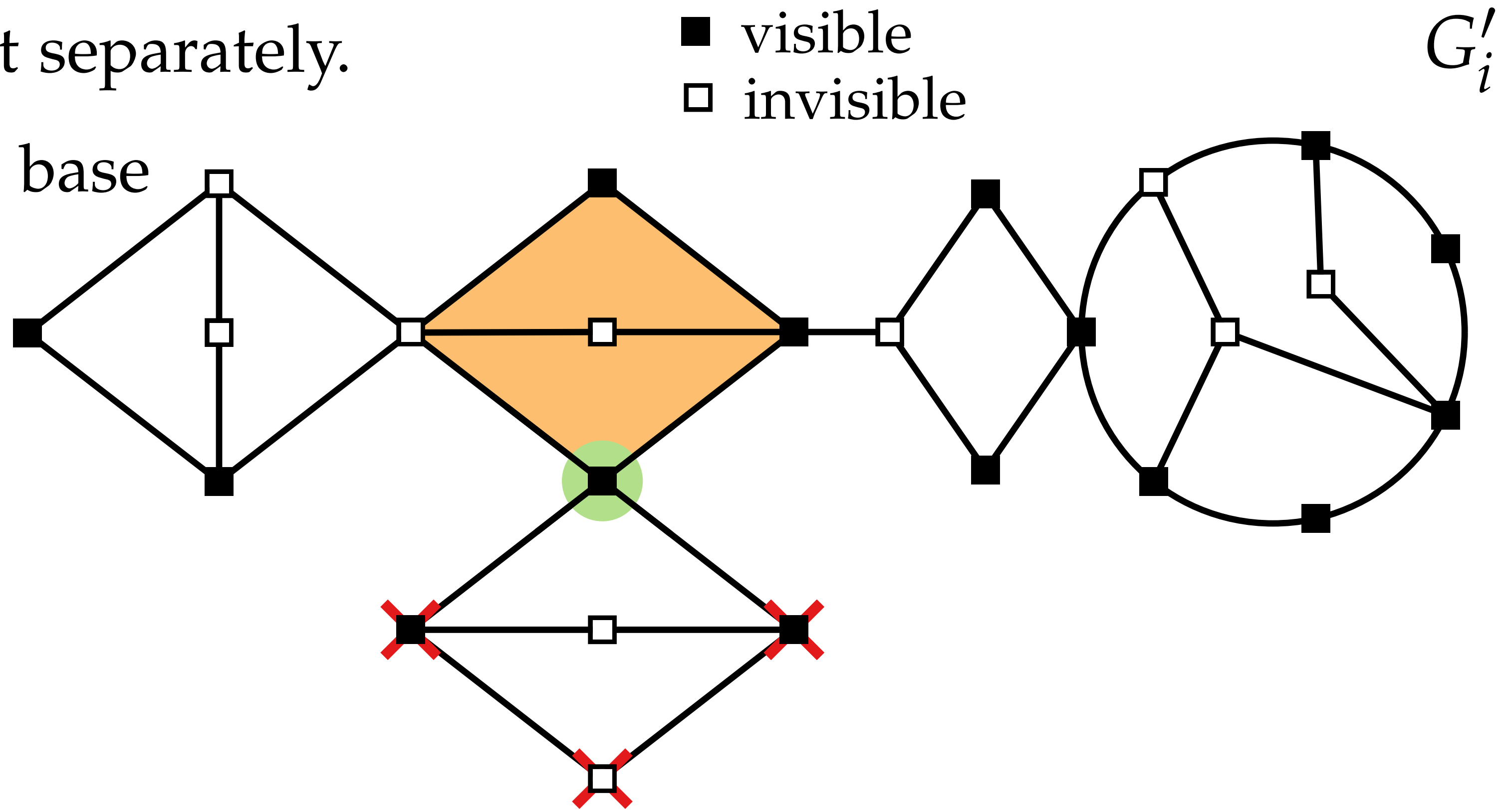
With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

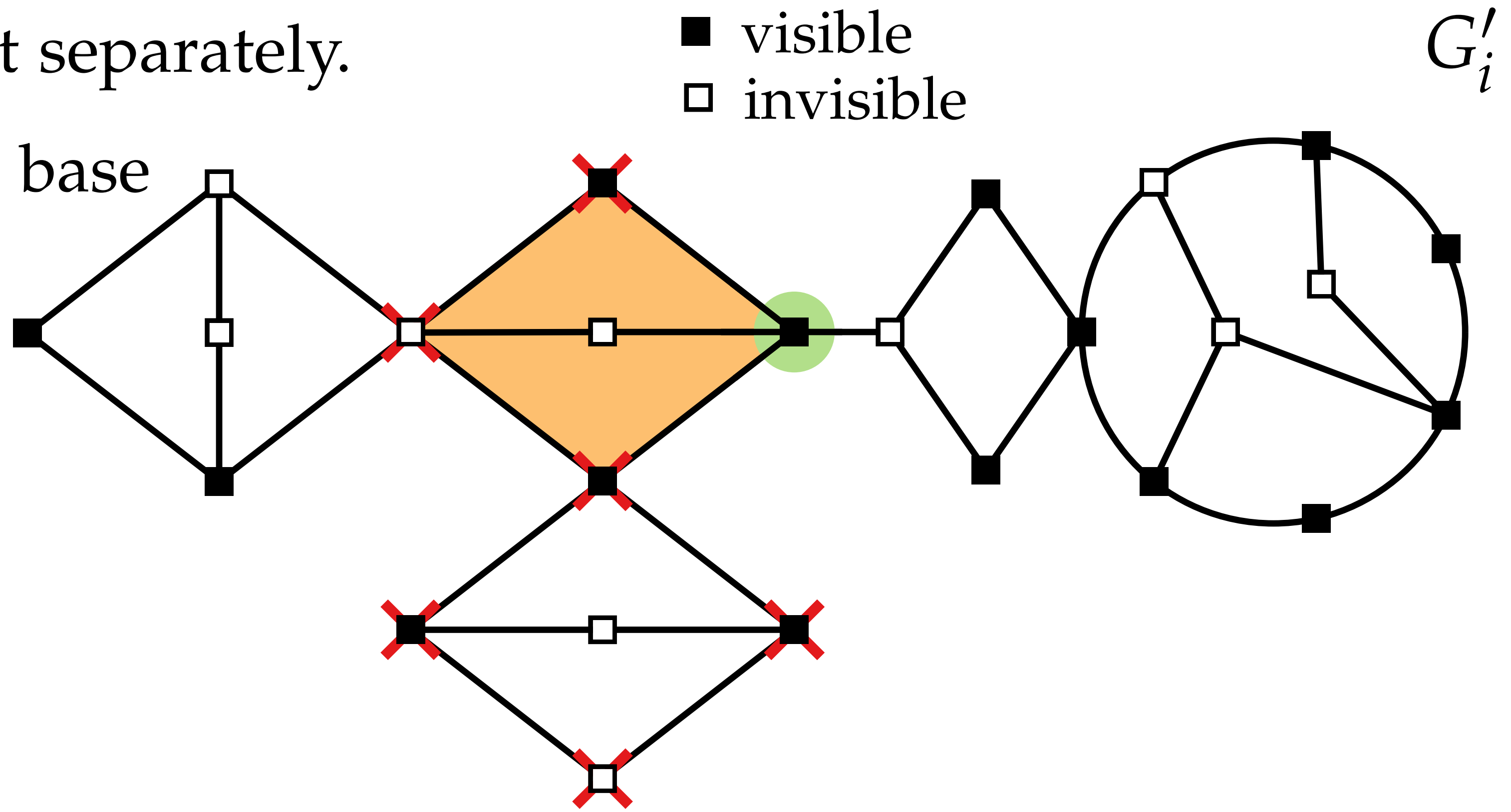
With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

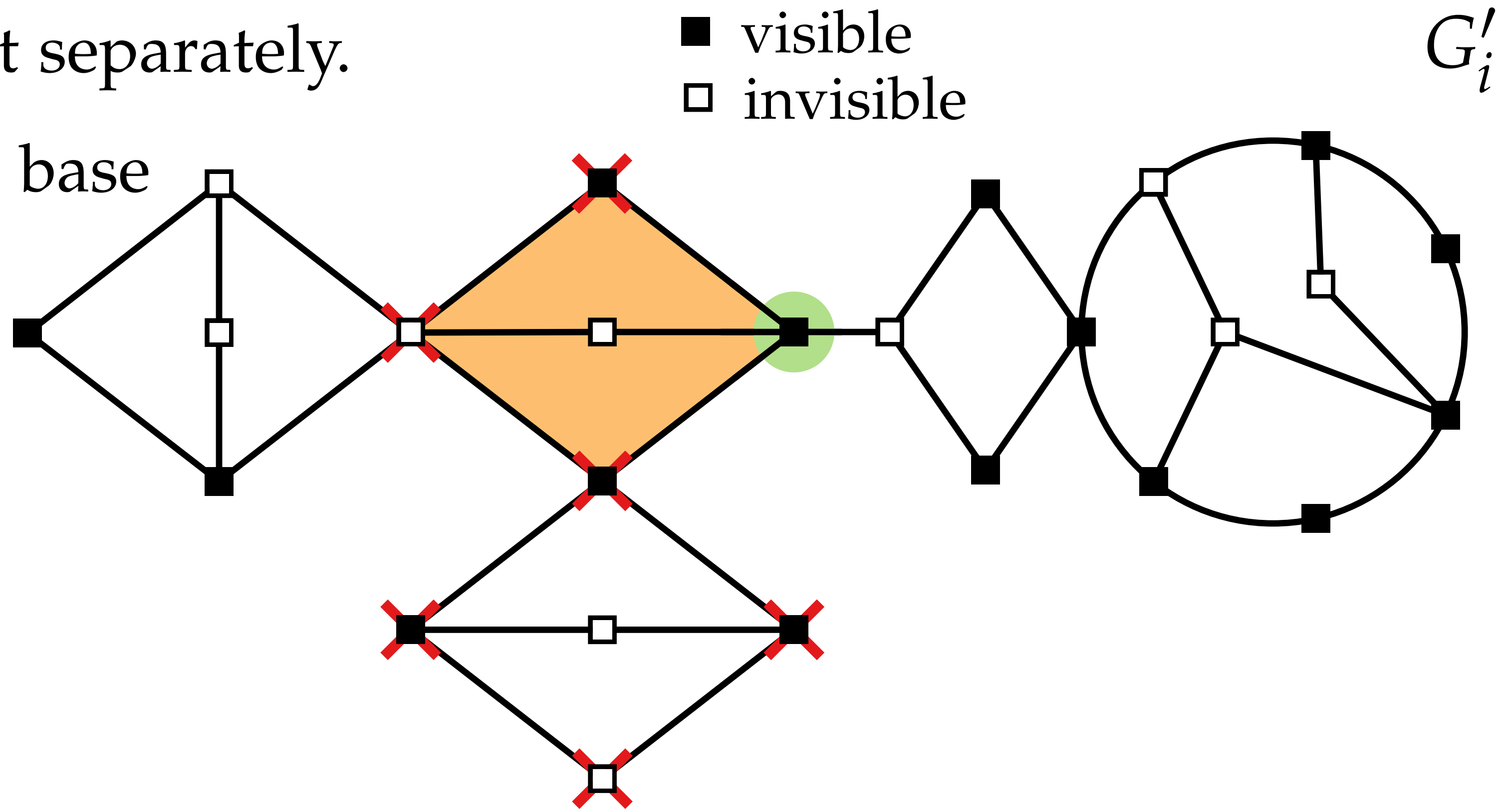
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

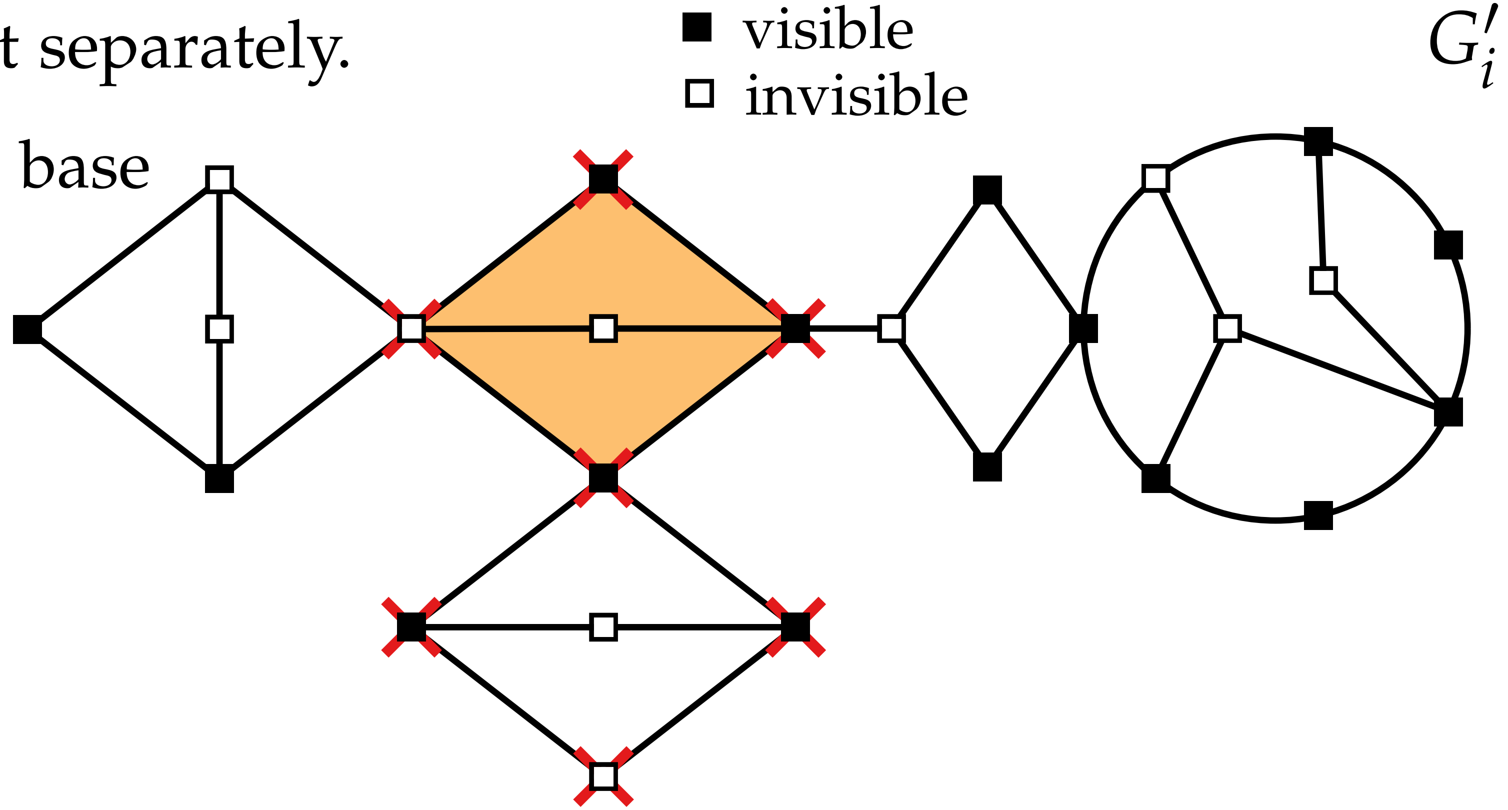
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

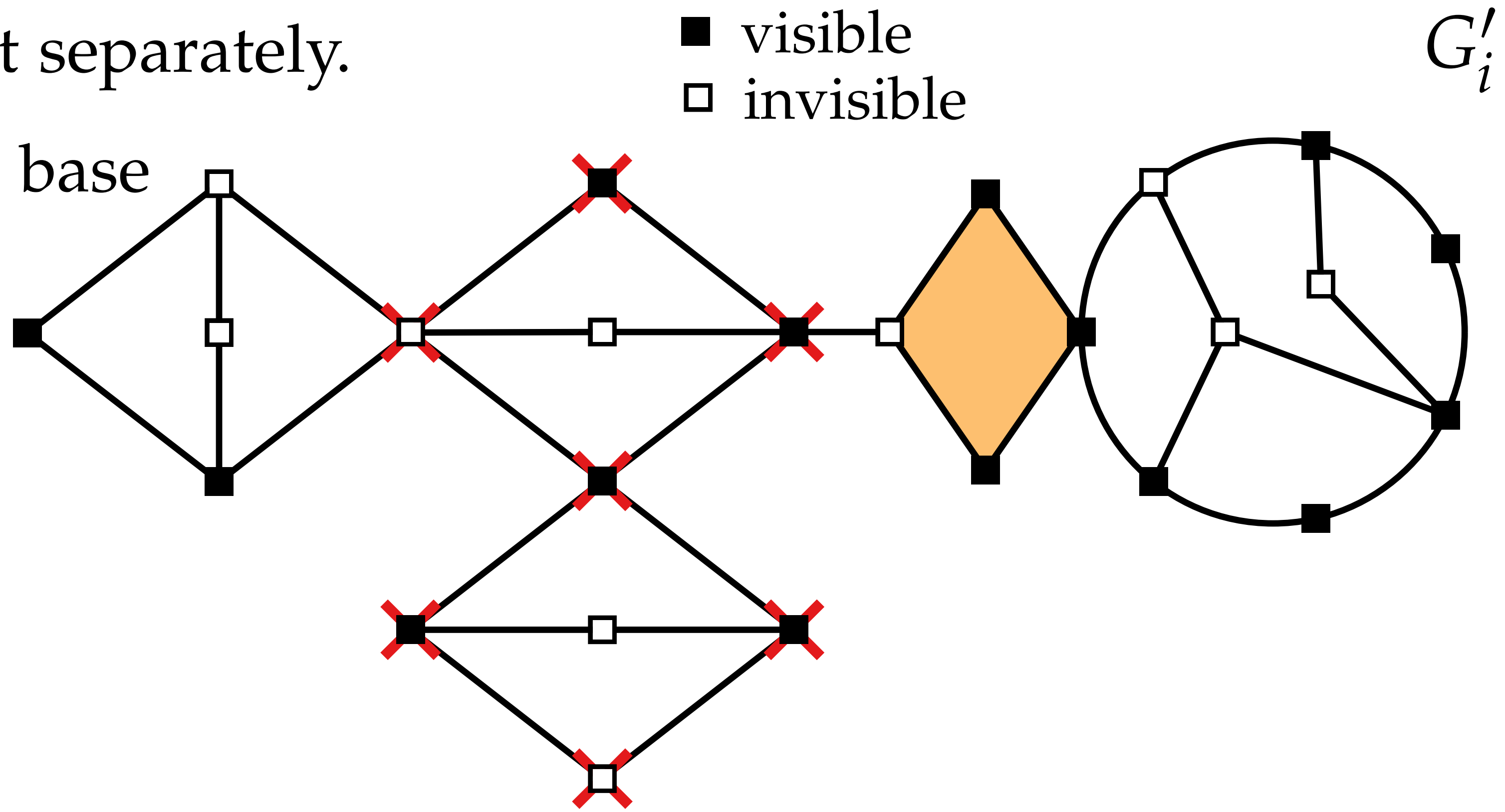
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

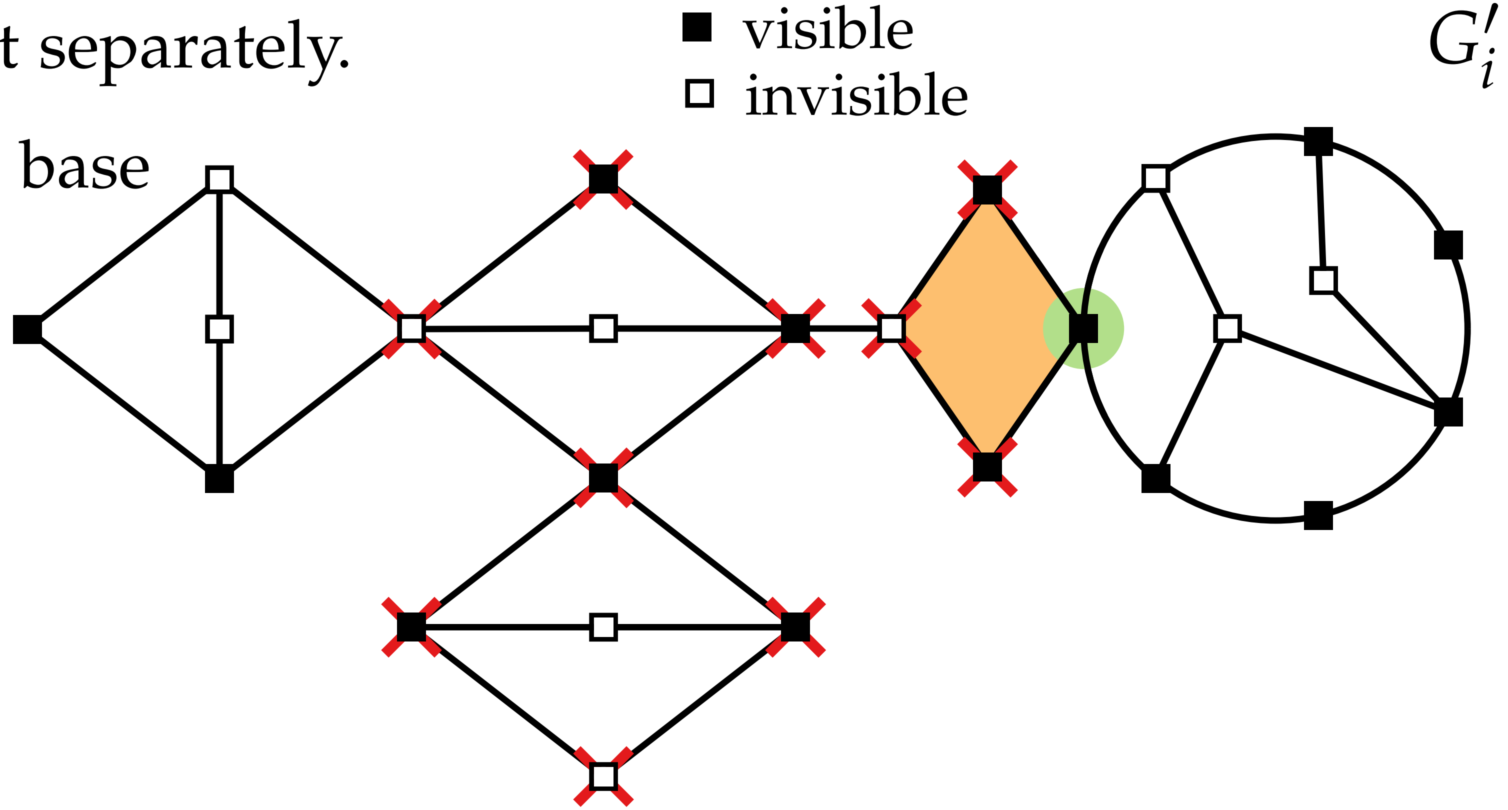
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

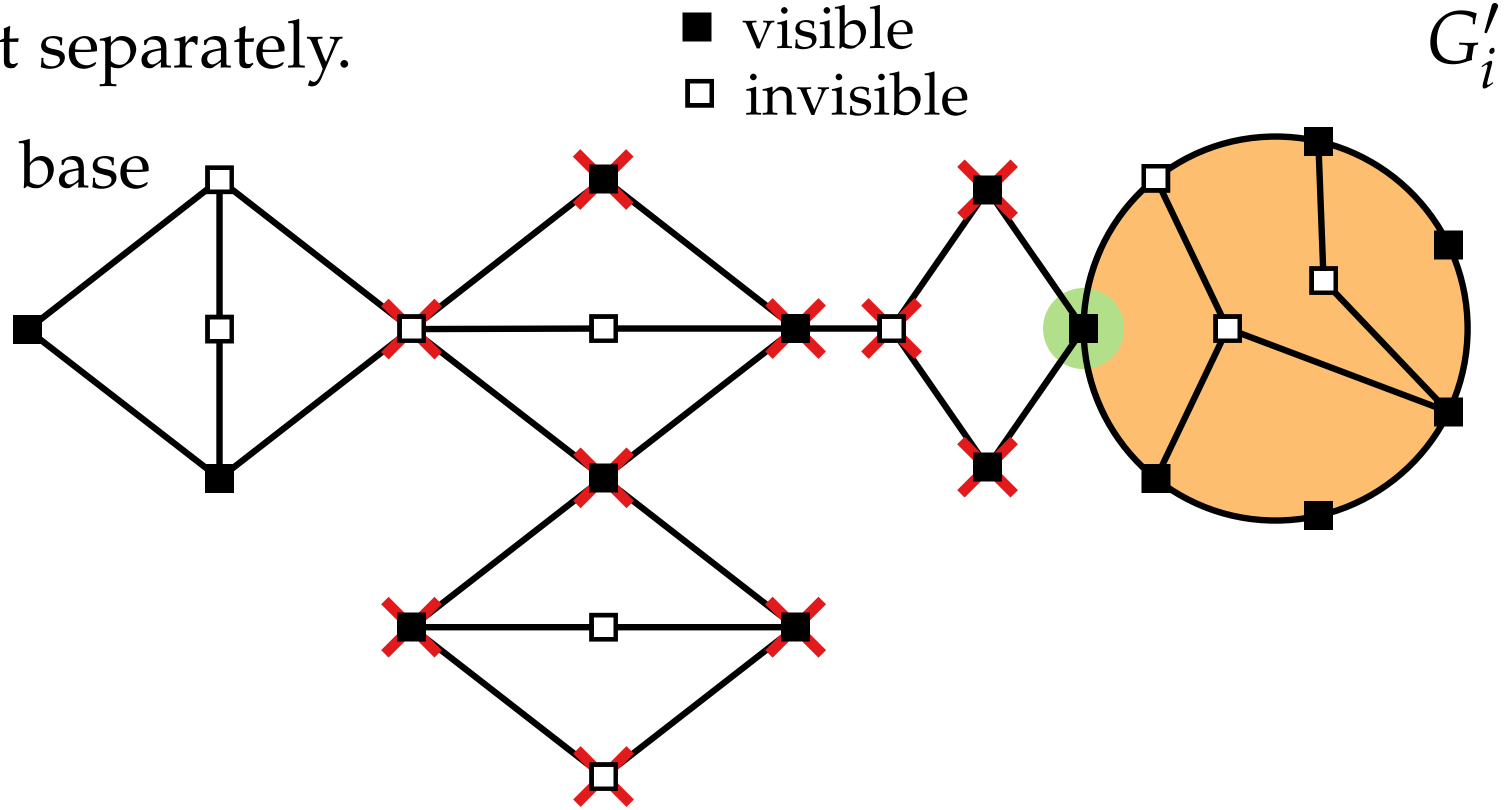
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

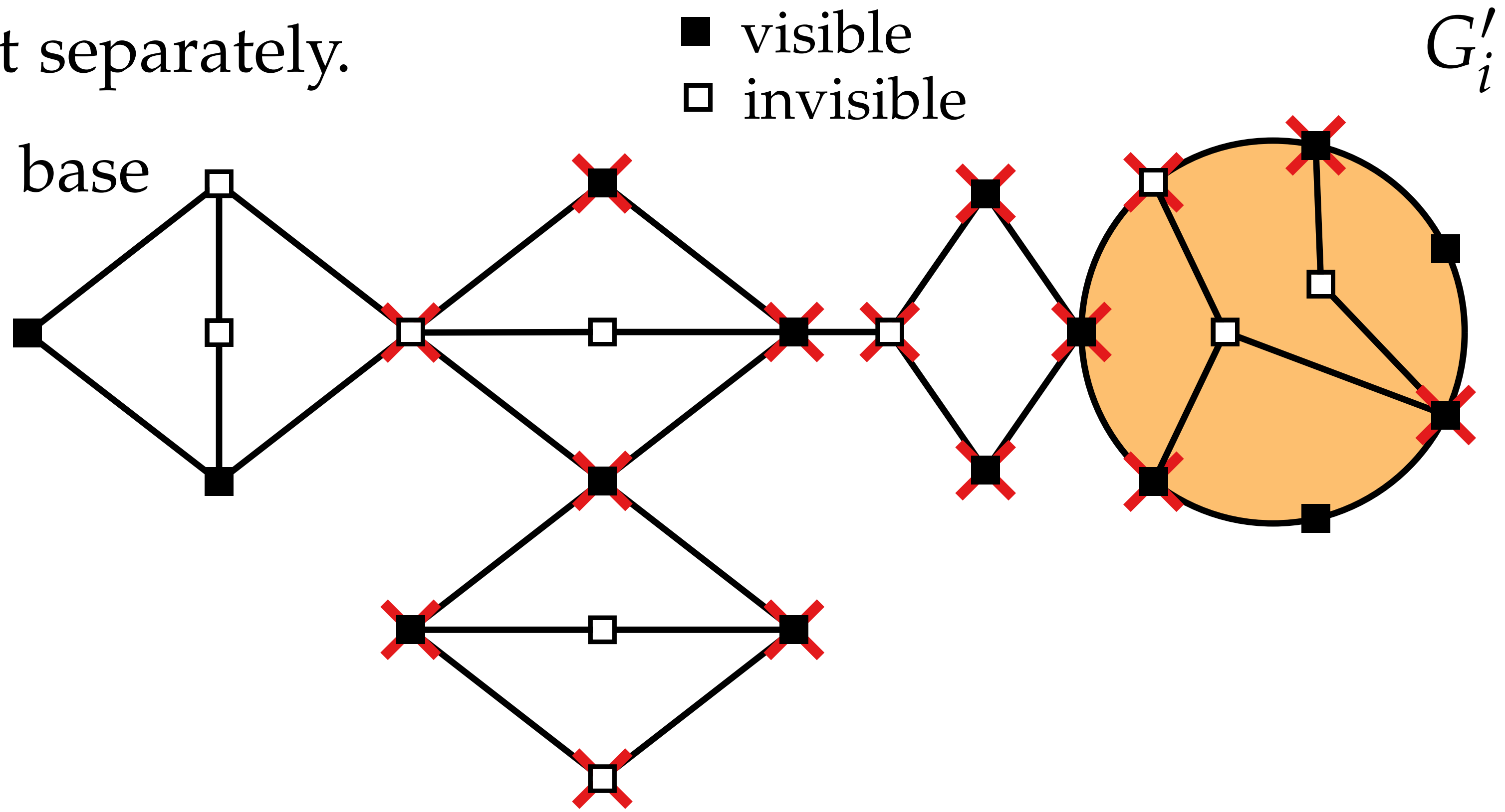
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

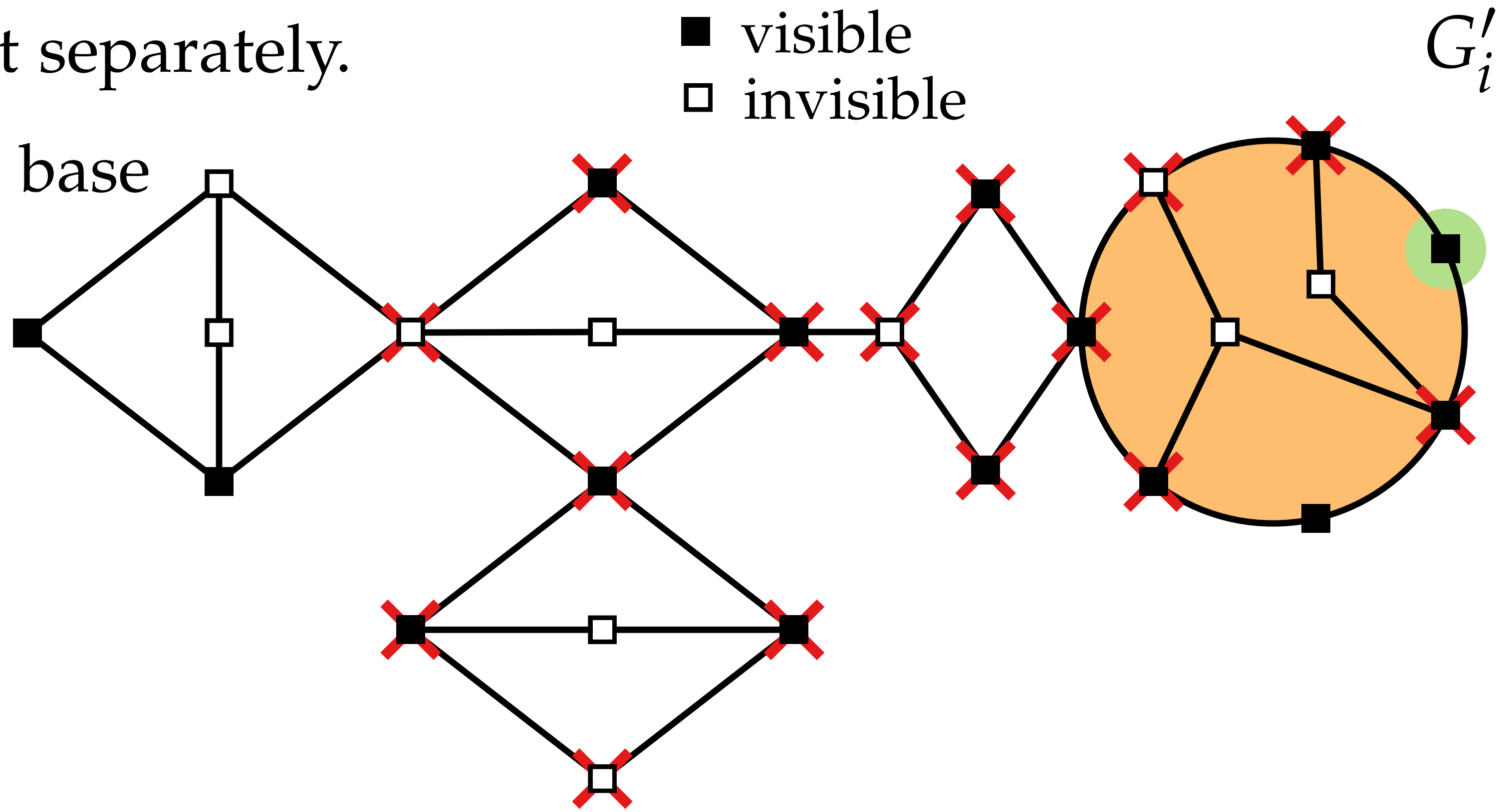
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Proof – General Case for the Outer Face

Note that in general the outer face of G_i' is a *cactus* forest.

With the help of H_i we can find a vertex that observes the rules in the *base case*, that is, the outer face of G_i' is a simple cycle.

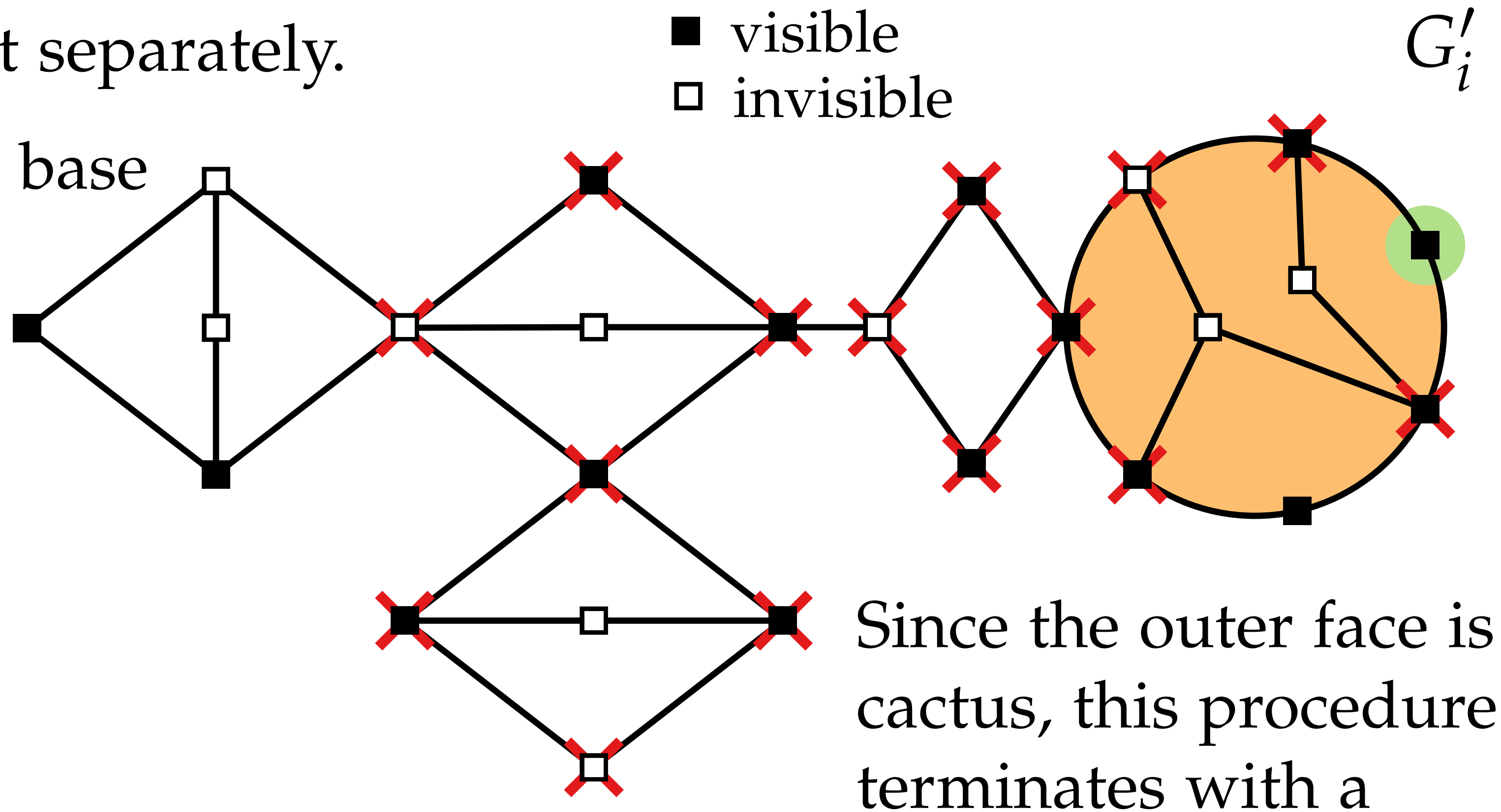
We skip this part.

Consider each biconnected component separately.

Repeatedly apply the argument of the base case to different components.

Check if the found vertex v can be picked in all other components that contain v.

Check for each neighbor w of v from diferent component whether making w visible violates one of the invariants.



Since the outer face is a cactus, this procedure terminates with a globally “good” vertex.

Open Problems

Recently we showed that
– deciding whether a given graph admits an outerplanar storyplan, and
– deciding whether a given graph admits a forest storyplan
are both NP-hard problems.

Open Problems

Recently we showed that
– deciding whether a given graph admits an outerplanar storyplan, and
– deciding whether a given graph admits a forest storyplan
are both NP-hard problems.

- Are there FPT algorithms for outerplanar or forest storyplans, e.g., w.r.t. treewidth/pathwidth?

Open Problems

Recently we showed that
– deciding whether a given graph admits an outerplanar storyplan, and
– deciding whether a given graph admits a forest storyplan
are both NP-hard problems.

- Are there FPT algorithms for outerplanar or forest storyplans, e.g., w.r.t. treewidth/pathwidth?

- Does every graph that admits a planar/outerplanar/forest storyplan, admit also a *straight-line* planar/outerplanar/forest storyplan?

Open Problems

Recently we showed that
– deciding whether a given graph admits an outerplanar storyplan, and
– deciding whether a given graph admits a forest storyplan
are both NP-hard problems.

- Are there FPT algorithms for outerplanar or forest storyplans, e.g., w.r.t. treewidth/pathwidth?

- Does every graph that admits a planar/outerplanar/forest storyplan, admit also a *straight-line* planar/outerplanar/forest storyplan?

- What about other variants of storyplans? For example, 1-planar storyplans.

Open Problems

Recently we showed that
– deciding whether a given graph admits an outerplanar storyplan, and
– deciding whether a given graph admits a forest storyplan
are both NP-hard problems.

- Are there FPT algorithms for outerplanar or forest storyplans, e.g., w.r.t. treewidth/pathwidth?

- Does every graph that admits a planar/outerplanar/forest storyplan, admit also a *straight-line* planar/outerplanar/forest storyplan?

- What about other variants of storyplans? For example, 1-planar storyplans.

The problem is NP-hard.
[Jan Sulejmani, Bachelor Thesis, 2024]

Open Problems

Thank you!

Recently we showed that
– deciding whether a given graph admits an outerplanar storyplan, and
– deciding whether a given graph admits a forest storyplan
are both NP-hard problems.

- Are there FPT algorithms for outerplanar or forest storyplans, e.g., w.r.t. treewidth/pathwidth?

- Does every graph that admits a planar/outerplanar/forest storyplan, admit also a *straight-line* planar/outerplanar/forest storyplan?

- What about other variants of storyplans? For example, 1-planar storyplans.

The problem is NP-hard.
[Jan Sulejmani, Bachelor Thesis, 2024]