Graph Decompositions via Counting Logics

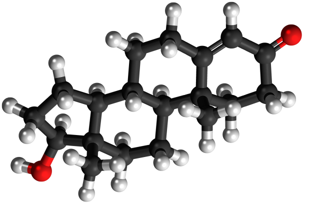
Sandra Kiefer

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Structure Meets Power
Paris, France
July 4, 2022

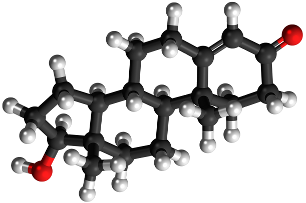
Structure Identification

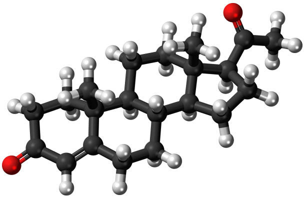
Find a description of **this molecule.**



Structure Identification

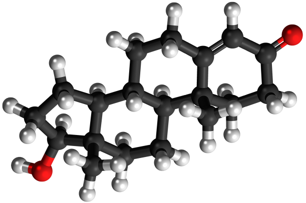
Find a description of **the difference between these molecules.**

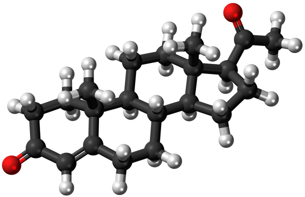




STRUCTURE IDENTIFICATION

Find a description of **the difference between these molecules.**

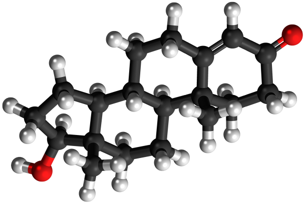




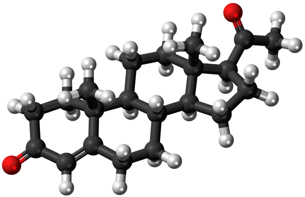
There is 1 red atom with
1 adjacent white atom.

Structure Identification

Find a description of **the difference between these molecules.**



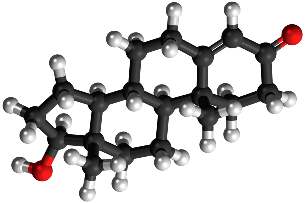
There is 1 red atom with 1 adjacent white atom.



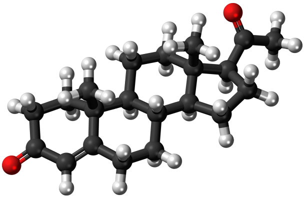
Not here.

Structure Identification

Find a description of **the difference between these molecules.**



$\not\approx$

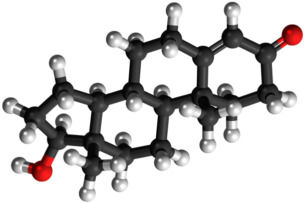


There is 1 red atom with 1 adjacent white atom.

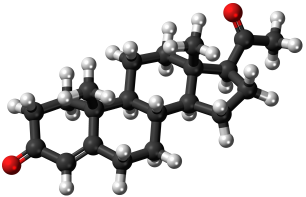
Not here.

Structure Identification

Find a description of **the difference between these molecules.**



$\ncong$



There is 1 red atom with 1 adjacent white atom.

Not here.

$$\exists x \exists y \Big(\mathsf{Red}(x) \wedge \mathsf{White}(y) \wedge E(x,y) \Big)$$

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

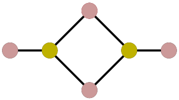
DESCRIPTIVE COMPLEXITY

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

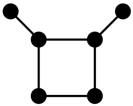
Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

Descriptive Complexity

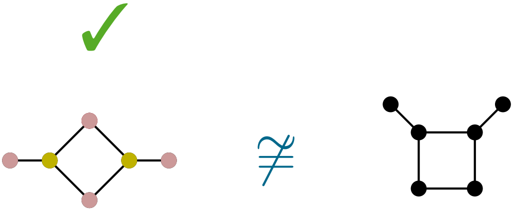


$\ncong$



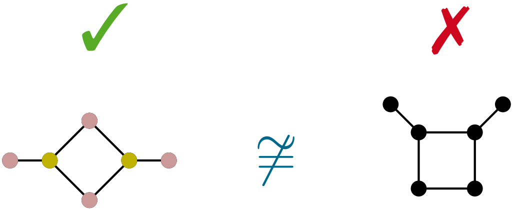
There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

Descriptive Complexity



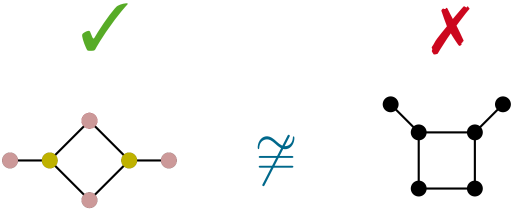
There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

Descriptive Complexity



There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

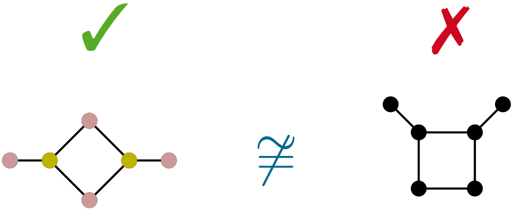
Descriptive Complexity



There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big($$

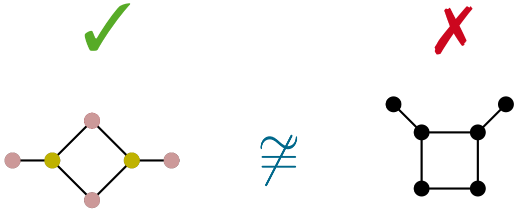
DESCRIPTIVE COMPLEXITY



There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Bigl(\exists^{=2} y \, E(x, y)$$

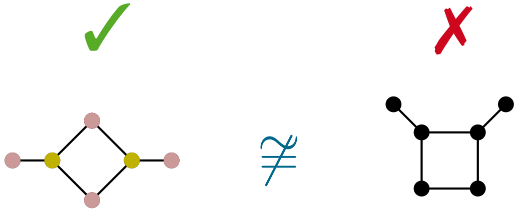
Descriptive Complexity



There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y\, E(x,y) \wedge \forall y\, \big(E(x,y)$$

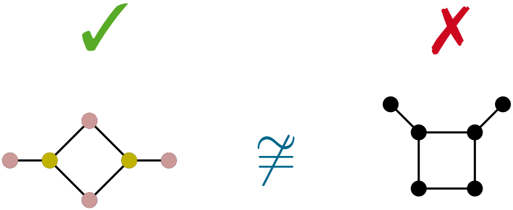
Descriptive Complexity



There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y\, E(x,y) \land \forall y\, \big(E(x,y) \to \exists^{=3} x\, E(y,x) \big) \Big)$$

DESCRIPTIVE COMPLEXITY



There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y\, E(x,y) \land \forall y\, \big(E(x,y) \to \exists^{=3} x\, E(y,x)\big) \Big)$$

2 variables, counting quantifiers, FO $\rightsquigarrow$ C^2-formula

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y\, E(x,y) \wedge \forall y\, \big(E(x,y) \to \exists^{=3} x\, E(y,x)\big) \Big)$$

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \land \forall y \, (E(x,y) \to \exists^{=3} x \, E(y,x)) \Big)$$

How can we measure the complexity of a logical formula?

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \wedge \forall y \, (E(x,y) \to \exists^{=3} x \, E(y,x)) \Big)$$

How can we measure the complexity of a logical formula?

- type/allowed combinations of quantifiers

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \wedge \forall y \, (E(x,y) \to \exists^{=3} x \, E(y,x)) \Big)$$

How can we measure the complexity of a logical formula?

- type/allowed combinations of quantifiers
- number of variables

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \land \forall y \, \big(E(x,y) \to \exists^{=3} x \, E(y,x)\big) \Big)$$

How can we measure the complexity of a logical formula?

- type/allowed combinations of quantifiers
- number of variables
- nesting depth of quantifiers

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \land \forall y \, (E(x,y) \to \exists^{=3} x \, E(y,x)) \Big)$$

How can we measure the complexity of a logical formula?

- type/allowed combinations of quantifiers
- number of variables
- nesting depth of quantifiers
- …

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \wedge \forall y \, (E(x,y) \rightarrow \exists^{=3} x \, E(y,x)) \Big)$$

How can we measure the complexity of a logical formula?

- type/allowed combinations of quantifiers
- number of variables
- nesting depth of quantifiers
- …

The complexity of a defining formula is a measure for the inherent complexity of the graphs.

Descriptive Complexity

There is 1 vertex with exactly 2 neighbours that both have exactly 3 neighbours.

$$\exists x \Big(\exists^{=2} y \, E(x,y) \wedge \forall y \, \big(E(x,y) \to \exists^{=3} x \, E(y,x)\big)\Big)$$

How can we measure the complexity of a logical formula?

- type/allowed combinations of quantifiers
- number of variables
- nesting depth of quantifiers
- …

The complexity of a defining formula is a measure for the inherent complexity of the graphs.

But how do we get from descriptions to actual algorithms?

Algorithmic Logics

Algorithmic Logics

For graphs G, H, the following are equivalent.

1. The logic C^{k+1} distinguishes G and H.
2. The algorithm k-WL distinguishes G and H.

[Cai, Fürer, Immerman ’92]

Colour Refinement

Oldest (?) reference: *The generation of a unique machine description for chemical structures*

[Morgan '65]

Colour Refinement

Oldest (?) reference: *The generation of a unique machine description for chemical structures*

[Morgan '65]

1-WL

- *Initialisation*: All vertices have their initial colours.
- *Refinement*: Recolour vertices depending on colours in their neighbourhoods.
- *Stop* when colouring is stable.

Colour Refinement

Oldest (?) reference: *The generation of a unique machine description for chemical structures*

[Morgan '65]

1-WL

- *Initialisation*: All vertices have their initial colours.
- *Refinement*: Recolour vertices depending on colours in their neighbourhoods.
- *Stop* when colouring is stable.

The induced partition respects orbits, so if two graphs result in different colourings, then they are non-isomorphic.

1-WL has an $O((m + n) \log n)$-implementation.

[Cardon & Crochemore '82]

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Colour Refinement

1-WL

- *Refinement*: v and w obtain different colours $\iff$ there is a colour c such that v and w have different numbers of c-coloured neighbours

Fact

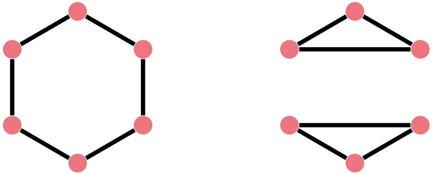
On paths of length n, 1-WL terminates after at most $\frac{n}{2}$ iterations.

Regular graphs

If two graphs result in different colourings, then the graphs are non-isomorphic.

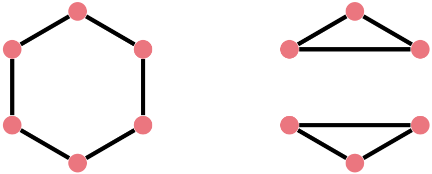
Regular graphs

If two graphs result in different colourings, then the graphs are non-isomorphic.



Regular graphs

If two graphs result in different colourings, then the graphs are non-isomorphic.

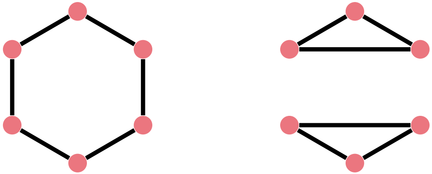


Facts

On every regular graph, 1-WL terminates after one iteration.

Regular graphs

If two graphs result in different colourings, then the graphs are non-isomorphic.



Facts

On every regular graph, 1-WL terminates after one iteration.

1-WL does not distinguish d-regular graphs of equal order.

The WL-Algorithm

The more powerful k-WL iteratively computes a colouring of V^k.
It can be implemented to run in time $O(n^{k+1} \log n)$.

[Immerman, Lander '90]

The WL-Algorithm

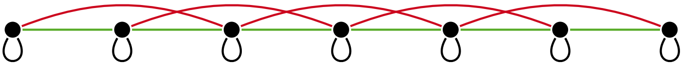
The more powerful k-WL iteratively computes a colouring of V^k.
It can be implemented to run in time $O(n^{k+1} \log n)$.

[Immerman, Lander '90]

The WL-Algorithm

The more powerful k-WL iteratively computes a colouring of V^k.
It can be implemented to run in time $O(n^{k+1} \log n)$.

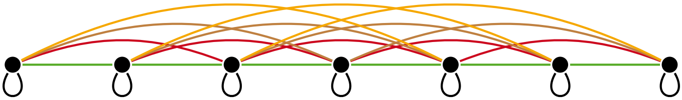
[Immerman, Lander '90]



The WL-Algorithm

The more powerful k-WL iteratively computes a colouring of V^k.
It can be implemented to run in time $O(n^{k+1} \log n)$.

[Immerman, Lander '90]

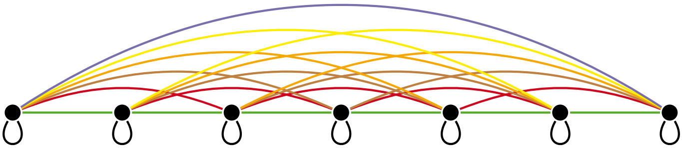


The WL-Algorithm

The more powerful k-WL iteratively computes a colouring of V^k.

It can be implemented to run in time $O(n^{k+1} \log n)$.

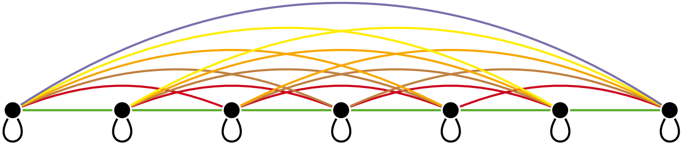
[Immerman, Lander '90]



The WL-Algorithm

The more powerful k-WL iteratively computes a colouring of V^k.
It can be implemented to run in time $O(n^{k+1} \log n)$.

[Immerman, Lander '90]



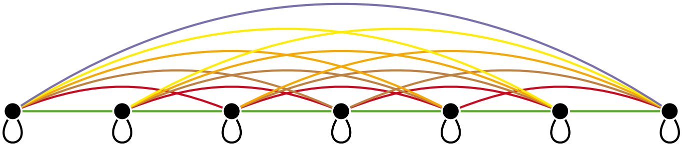
Facts

On strongly regular graph, 2-WL terminates after one iteration.

The WL-Algorithm

The more powerful k-WL iteratively computes a colouring of V^k.
It can be implemented to run in time $O(n^{k+1} \log n)$.

[Immerman, Lander '90]



Facts

On strongly regular graph, 2-WL terminates after one iteration.

2-WL does not distinguish strongly regular graphs with equal parameters.

Applications

Via the WL-algorithm, the logic C has connections to many areas:

Applications

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests

APPLICATIONS

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests
- Linear programming

$$
\widetilde{A} = \left(\begin{array}{ccc|cccc|cc|cccc|c}
3 & -1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 3 & -2 & \frac{1}{2} & \frac{1}{2} & 1 \\
-1 & 1 & 3 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & -2 & 3 & \frac{1}{2} & \frac{1}{2} & 1 \\
1 & 3 & -1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \\
\hline
0 & \frac{1}{3} & \frac{2}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 2 & 0 & 1 & 0 & -1 & 0 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & 0 & 0 & 1 & 0 & -1 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & -1 & 0 & 1 & 0 & 1 \\
\frac{2}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 0 & 2 & 0 & -1 & 0 & 1 & 1 \\
\hline
2 & 2 & 2 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \infty
\end{array}\right)
$$

$$
[\![\widetilde{A}]\!] = [[\widetilde{A}]] = \begin{pmatrix} 4 & 2 & 1 \\ 12 & 4 & \infty \end{pmatrix}
$$

Image source: [Grohe et al. ’14]

Applications

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests
- Linear programming
- Graph kernels

$$\widetilde{A} = \left(\begin{array}{ccc|cccc|cc|cccc|c}
3 & -1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 3 & -2 & \frac{1}{2} & \frac{1}{2} & 1 \\
-1 & 1 & 3 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & -2 & 3 & \frac{1}{2} & \frac{1}{2} & 1 \\
1 & 3 & -1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \\
\hline
0 & \frac{1}{3} & \frac{2}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 2 & 0 & 1 & 0 & -1 & 0 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & 0 & 0 & 1 & 0 & -1 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & -1 & 0 & 1 & 0 & 1 \\
\frac{2}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 0 & 2 & 0 & -1 & 0 & 1 & 1 \\
\hline
2 & 2 & 2 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \infty
\end{array}\right)$$

$$[\![\widetilde{A}]\!] = [[\widetilde{A}]] = \begin{pmatrix} 4 & 2 & 1 \\ 12 & 4 & \infty \end{pmatrix}$$

Image source: [Grohe et al. ’14]

Applications

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests
- Linear programming
- Graph kernels
- Graph neural networks

$$\widetilde{A} = \left(\begin{array}{ccc|cccc|cc|cccc|c}
3 & -1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 3 & -2 & \frac{1}{2} & \frac{1}{2} & 1 \\
-1 & 1 & 3 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & -2 & 3 & \frac{1}{2} & \frac{1}{2} & 1 \\
1 & 3 & -1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \\
\hline
0 & \frac{1}{3} & \frac{2}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 2 & 0 & 1 & 0 & -1 & 0 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & 0 & 0 & 1 & 0 & -1 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & -1 & 0 & 1 & 0 & 1 \\
\frac{2}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 0 & 2 & 0 & -1 & 0 & 1 & 1 \\
\hline
2 & 2 & 2 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \infty
\end{array}\right)$$

$$[\![\widetilde{A}]\!] = [[\widetilde{A}]] = \begin{pmatrix} 4 & 2 & 1 \\ 12 & 4 & \infty \end{pmatrix}$$

Image source: [Grohe et al. '14]

Applications

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests
- Linear programming
- Graph kernels
- Graph neural networks
- Propositional proof complexity

$$\widetilde{A} = \left(\begin{array}{ccc|cccc|cc|cccc|c}
3 & -1 & 1 & \frac14 & \frac14 & \frac14 & \frac14 & 0 & 0 & 3 & -2 & \frac12 & \frac12 & 1 \\
-1 & 1 & 3 & \frac14 & \frac14 & \frac14 & \frac14 & 0 & 0 & -2 & 3 & \frac12 & \frac12 & 1 \\
1 & 3 & -1 & \frac14 & \frac14 & \frac14 & \frac14 & 0 & 0 & \frac12 & \frac12 & \frac12 & \frac12 & 1 \\
\hline
0 & \frac13 & \frac23 & 0 & \frac32 & 0 & \frac32 & 2 & 0 & 1 & 0 & -1 & 0 & 1 \\
\frac13 & \frac13 & \frac13 & \frac32 & 0 & \frac32 & 0 & 2 & 0 & 0 & 1 & 0 & -1 & 1 \\
\frac13 & \frac13 & \frac13 & 0 & \frac32 & 0 & \frac32 & 0 & 2 & -1 & 0 & 1 & 0 & 1 \\
\frac23 & \frac13 & 0 & \frac32 & 0 & \frac32 & 0 & 0 & 2 & 0 & -1 & 0 & 1 & 1 \\
\hline
2 & 2 & 2 & \frac32 & \frac32 & \frac32 & \frac32 & 1 & 1 & \frac12 & \frac12 & \frac12 & \frac12 & \infty
\end{array}\right)$$

$$[\![\widetilde{A}]\!] = [[\widetilde{A}]] = \begin{pmatrix} 4 & 2 & 1 \\ 12 & 4 & \infty \end{pmatrix}$$

Image source: [Grohe et al. ’14]

Applications

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests
- Linear programming
- Graph kernels
- Graph neural networks
- Propositional proof complexity
- Homomorphism counting

$$\widetilde{A} = \left(\begin{array}{ccc|cccc|cc|cccc|c}
3 & -1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 3 & -2 & \frac{1}{2} & \frac{1}{2} & 1 \\
-1 & 1 & 3 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & -2 & 3 & \frac{1}{2} & \frac{1}{2} & 1 \\
1 & 3 & -1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \\
\hline
0 & \frac{1}{3} & \frac{2}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 2 & 0 & 1 & 0 & -1 & 0 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & 0 & 0 & 1 & 0 & -1 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & -1 & 0 & 1 & 0 & 1 \\
\frac{2}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 0 & 2 & 0 & -1 & 0 & 1 & 1 \\
\hline
2 & 2 & 2 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \infty
\end{array}\right)$$

$$[\![\widetilde{A}]\!] = [[\widetilde{A}]] = \begin{pmatrix} 4 & 2 & 1 \\ 12 & 4 & \infty \end{pmatrix}$$

Image source: [Grohe et al. ’14]

APPLICATIONS

Via the WL-algorithm, the logic C has connections to many areas:

- Practical graph-isomorphism tests
- Linear programming
- Graph kernels
- Graph neural networks
- Propositional proof complexity
- Homomorphism counting
- …

$$\widetilde{A} = \left(\begin{array}{ccc|cccc|cc|cccc|c}
3 & -1 & 1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 3 & -2 & \frac{1}{2} & \frac{1}{2} & 1 \\
-1 & 1 & 3 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & -2 & 3 & \frac{1}{2} & \frac{1}{2} & 1 \\
1 & 3 & -1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \\
\hline
0 & \frac{1}{3} & \frac{2}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 2 & 0 & 1 & 0 & -1 & 0 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & 0 & 0 & 1 & 0 & -1 & 1 \\
\frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 2 & -1 & 0 & 1 & 0 & 1 \\
\frac{2}{3} & \frac{1}{3} & 0 & \frac{3}{2} & 0 & \frac{3}{2} & 0 & 0 & 2 & 0 & -1 & 0 & 1 & 1 \\
\hline
2 & 2 & 2 & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & \frac{3}{2} & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \infty
\end{array}\right)$$

$$[\![\widetilde{A}]\!] = [[\widetilde{A}]] = \begin{pmatrix} 4 & 2 & 1 \\ 12 & 4 & \infty \end{pmatrix}$$

Image source: [Grohe et al. ’14]

Algorithmic Logics

For graphs G, H, the following are equivalent.

1. The logic C^{k+1} distinguishes G and H.
2. The algorithm k-WL distinguishes G and H.

[Cai, Fürer, Immerman ’92]

Algorithmic Logics

For graphs G, H, the following are equivalent.

1. The logic C^{k+1} distinguishes G and H.
2. The algorithm k-WL distinguishes G and H.
3. Spoiler wins the $(k+1)$-pebble game on G and H.

[Cai, Fürer, Immerman '92]

Pebble Game for C^k

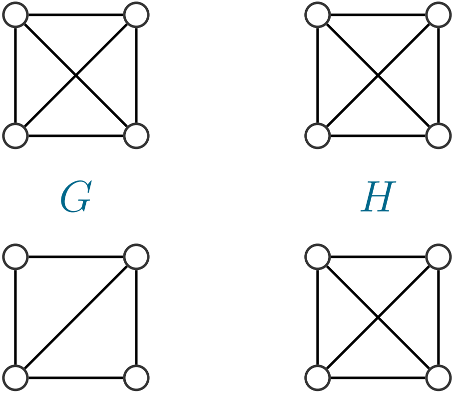
Spoiler and Duplicator dispose of k pairs of pebbles.

1

2

3

4



Pebble Game for C^k

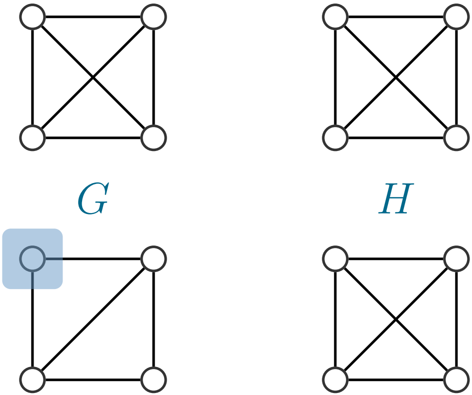
Spoiler and Duplicator dispose of k pairs of pebbles.

1 Spoiler takes a pebble and selects a vertex set S in G or H.

2

3

4



Pebble Game for C^k

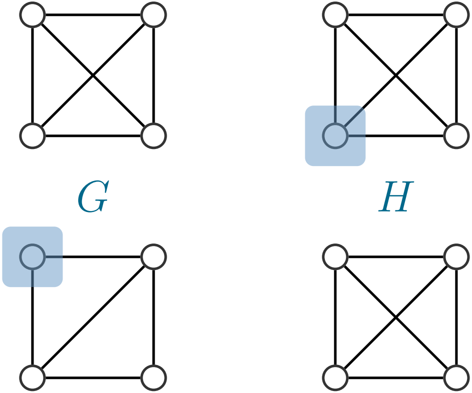
Spoiler and Duplicator dispose of k pairs of pebbles.

1

2 Duplicator takes the other pebble and selects a set S' of equal size in the other graph.

3

4



Pebble Game for C^k

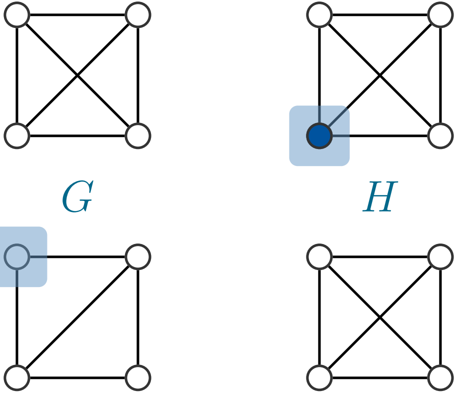
Spoiler and Duplicator dispose of k pairs of pebbles.

1

2

3 Spoiler places his pebble on a vertex in S'.

4



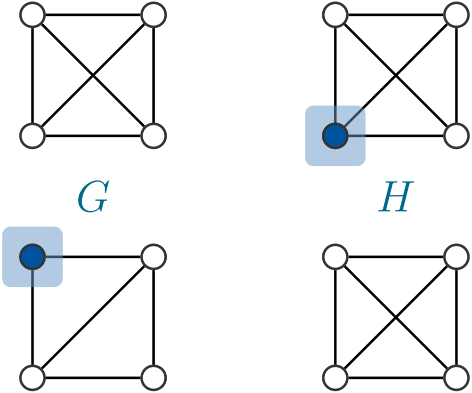
Pebble Game for C^k

Spoiler and Duplicator dispose of k pairs of pebbles.

1

2

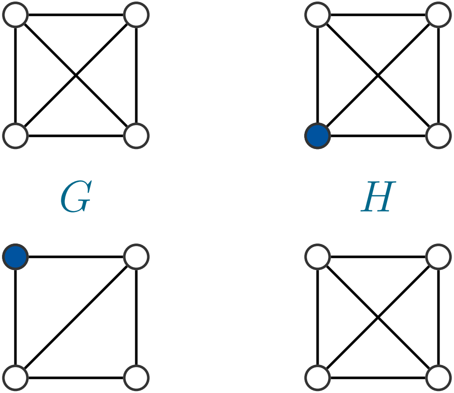
3



4 Duplicator places her pebble on a vertex in S.

Pebble Game for C^k

Spoiler and Duplicator dispose of k pairs of pebbles.



Are the pebbled subgraphs isomorphic? ✓

Pebble Game for C^k

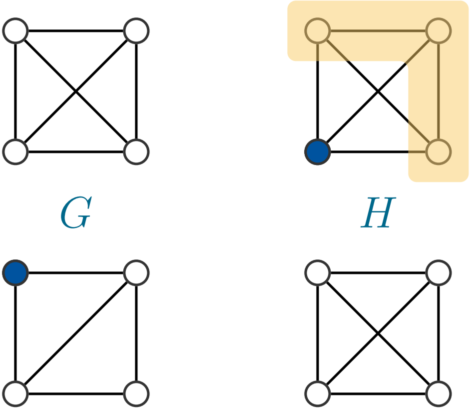
Spoiler and Duplicator dispose of k pairs of pebbles.

1 Spoiler takes a pebble and selects a vertex set S in G or H.

2

3

4

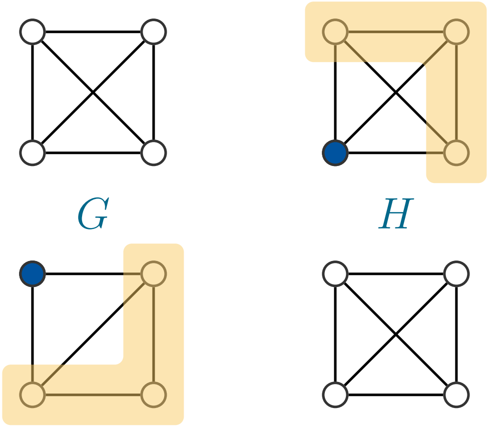


Pebble Game for C^k

Spoiler and Duplicator dispose of k pairs of pebbles.

1

2 Duplicator takes the other pebble and selects a set S' of equal size in the other graph.



3

4

Pebble Game for C^k

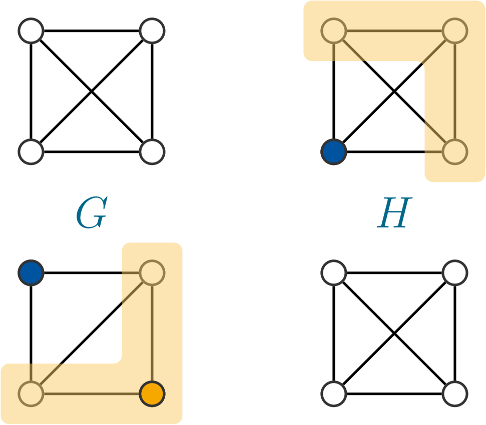
Spoiler and Duplicator dispose of k pairs of pebbles.

1

2

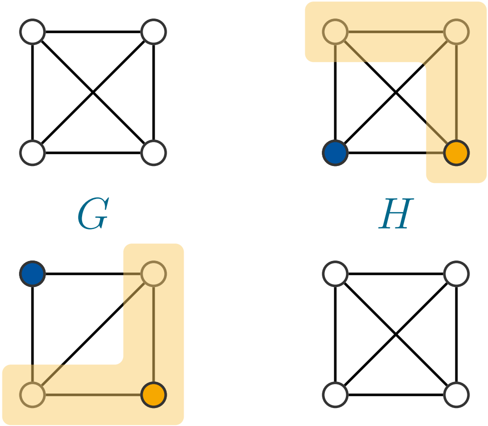
3 Spoiler places his pebble on a vertex in S'.

4



Pebble Game for C^k

Spoiler and Duplicator dispose of k pairs of pebbles.



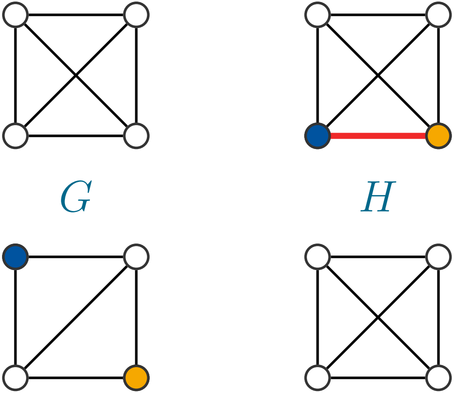
1

2

3

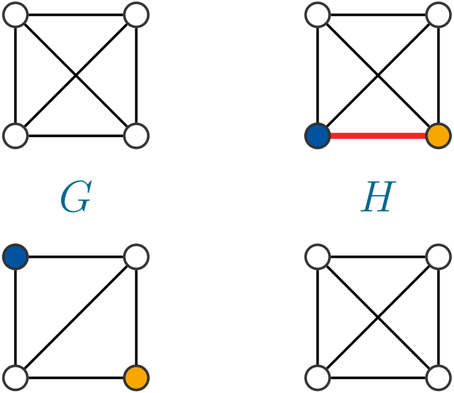
4 Duplicator places her pebble on a vertex in S.

Pebble Game for C^k



Are the pebbled subgraphs isomorphic? ✗

Pebble Game for C^k



Are the pebbled subgraphs isomorphic? ✗
Thus, Spoiler wins.

Algorithmic Logics

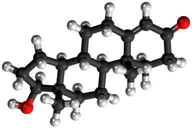
For graphs G, H, the following are equivalent.

1. The logic C^{k+1} distinguishes G and H.
2. The algorithm k-WL distinguishes G and H.
3. Spoiler wins the $(k+1)$-pebble game on G and H.

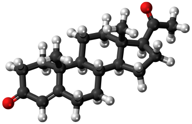
[Cai, Fürer, Immerman ’92]

Nesting depth $\equiv$ Number of iterations $\equiv$ Rounds in game

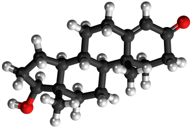
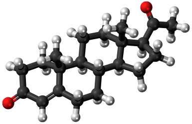
Identification



$\ncong$

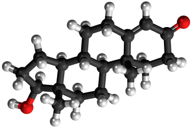
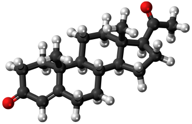


Identification

 $\ncong$ 

$$\exists x \exists y \Big(\mathsf{Red}(x) \land \mathsf{White}(y) \land E(x,y) \Big)$$

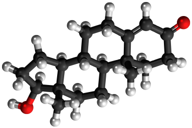
Identification

 $\not\cong$ 

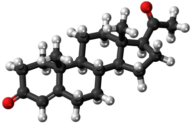
$$\exists x \exists y \Big(\mathsf{Red}(x) \land \mathsf{White}(y) \land E(x,y) \Big)$$

Graphs that are not distinguished by C^k are C^k-*equivalent*.

Identification



$$\ncong$$

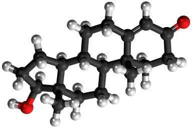
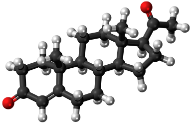


$$\exists x \exists y \Big(\mathsf{Red}(x) \wedge \mathsf{White}(y) \wedge E(x,y) \Big)$$

Graphs that are not distinguished by C^k are C^k*-equivalent*.

G is *identified* by C^k $:\Longleftrightarrow$ Every C^k-equivalent graph is isomorphic to G.

Identification

 $\ncong$ 

$$\exists x \exists y \Big(\mathsf{Red}(x) \wedge \mathsf{White}(y) \wedge E(x,y) \Big)$$

Graphs that are not distinguished by C^k are *C^k-equivalent*.

G is *identified* by C^k $:\Longleftrightarrow$ Every C^k-equivalent graph is isomorphic to G.

C^2 identifies almost all graphs. [Babai, Erdös, Selkow '80]

But it fails on very simple graphs! ($\mathsf{C}^2 \equiv 1$-WL)

Identification

Theorem (K., Schweitzer, Selman 2015)

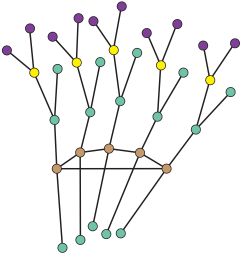
1-WL identifies G. $\iff$ The flip of G is a bouquet forest.

Identification

Theorem (K., Schweitzer, Selman 2015)

1-WL identifies G. $\iff$ The flip of G is a bouquet forest.

Bouquet: copies $(T_1, v_1), \ldots, (T_5, v_5)$ of a vertex-coloured tree (T, v), connected via a 5-cycle on $v_1, \ldots, v_5$



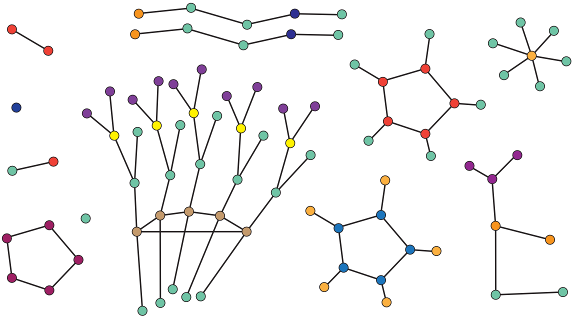
IDENTIFICATION

Theorem (K., Schweitzer, Selman 2015)

1-*WL identifies* G. $\iff$ *The flip of* G *is a bouquet forest.*

Bouquet: copies $(T_1, v_1), \ldots, (T_5, v_5)$ of a vertex-coloured tree (T, v), connected via a 5-cycle on $v_1, \ldots, v_5$

Bouquet forest: disjoint union of vertex-coloured trees and **non-isomorphic** bouquets



WL-Dimension

1-WL has an $O((m+n)\log n)$-implementation.

[Cardon & Crochemore '82]

k-WL can be implemented to run in time $O(n^{k+1}\log n)$.

[Immerman, Lander '90]

WL-Dimension

1-WL has an $O((m+n)\log n)$-implementation.

[Cardon & Crochemore '82]

k-WL can be implemented to run in time $O(n^{k+1}\log n)$.

[Immerman, Lander '90]

How powerful is $\boldsymbol{k}$-WL for fixed $\boldsymbol{k \geq 2}$?

WL-Dimension

1-WL has an $O((m+n)\log n)$-implementation.

[Cardon & Crochemore '82]

k-WL can be implemented to run in time $O(n^{k+1}\log n)$.

[Immerman, Lander '90]

How powerful is $\boldsymbol{k}$-WL for fixed $\boldsymbol{k \geq 2}$?

There is no k such that k-WL distinguishes every pair of non-isomorphic graphs.

[Cai, Fürer, Immerman '92]

WL-Dimension

1-WL has an $O((m+n)\log n)$-implementation.

[Cardon & Crochemore '82]

k-WL can be implemented to run in time $O(n^{k+1}\log n)$.

[Immerman, Lander '90]

How powerful is k-WL for fixed $k \geq 2$?

There is no k such that k-WL distinguishes every pair of non-isomorphic graphs.

[Cai, Fürer, Immerman '92]

$\leadsto$ **What if we restrict ourselves to certain graph classes?**

WL-Dimension

1-WL has an $O((m+n)\log n)$-implementation.

[Cardon & Crochemore '82]

k-WL can be implemented to run in time $O(n^{k+1}\log n)$.

[Immerman, Lander '90]

How powerful is k-WL for fixed $k \geq 2$?

There is no k such that k-WL distinguishes every pair of non-isomorphic graphs.

[Cai, Fürer, Immerman '92]

$\leadsto$ **What if we restrict ourselves to certain graph classes?**

Definition

A graph G has *WL-dimension at most* k if k-WL identifies G.

WL-dimension

Definition

A graph G has *WL-dimension at most k* if k-WL identifies G.

WL-DIMENSION

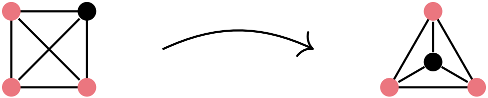
Definition

A graph G has *WL-dimension at most k* if k-WL identifies G.

Graph class	WL-dimension		
	lower bound	upper bound	
Trees	1	1	
Interval graphs	2	2	[Evdokimov, Ponomarenko, Tinhofer '00]
Excluded minor H	$\Omega(\lvert V(H)\rvert)$	$f(H)$	[Grohe '10]
Planar graphs	2	$\cancel{14}$ **3**	**[K., Ponomarenko, Schweitzer '17]**
Treewidth k	$\cancel{\Omega(k)}$ $\frac{k}{2}-2$	$\cancel{k+2}$ k	**[K., Neuen '19]**
Genus g	$\Omega(g)$	$4g+3$	[Grohe, K. '19]
Clique width k	$\Omega(k)$	$3k+4$	[Grohe, Neuen '19]
Rank width k	$\Omega(k)$	$3k+4$	[Grohe, Neuen '19]

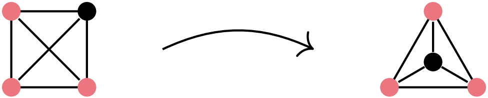
Planar Graphs

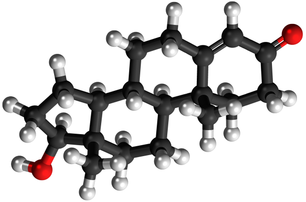
G is *planar* $:\Longleftrightarrow$ G can be embedded in the plane without any edge crossings.



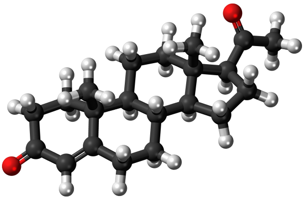
Planar Graphs

G is *planar* $:\Longleftrightarrow$ G can be embedded in the plane without any edge crossings.

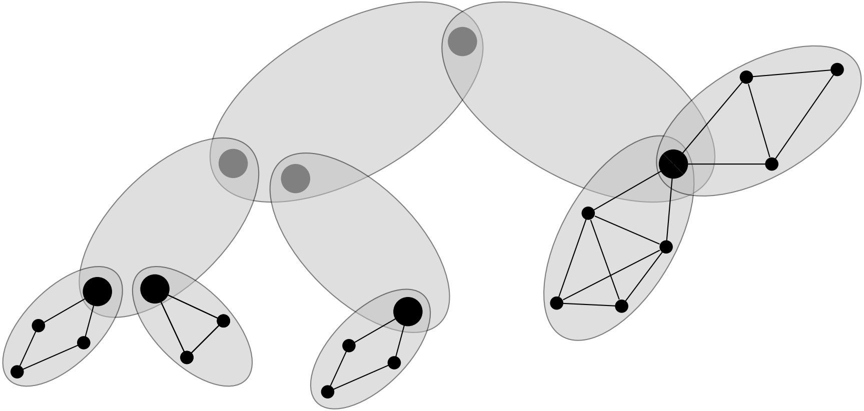




$\ncong$

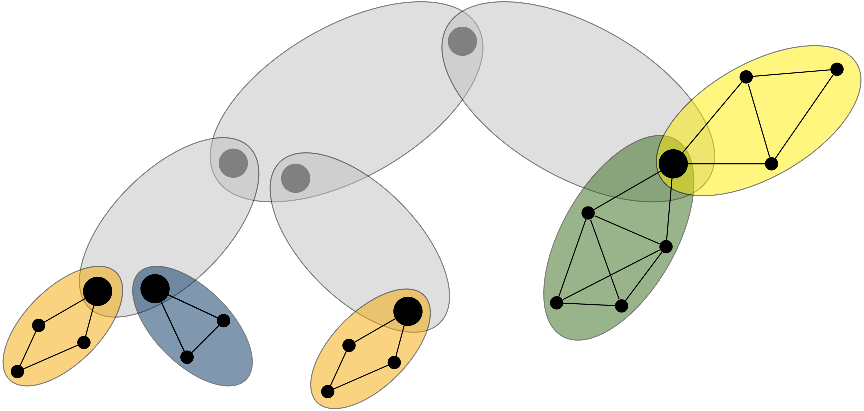


DECOMPOSITIONS



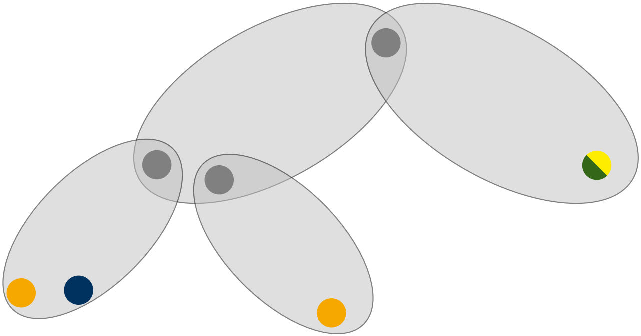
A decomposition of a connected graph into
2-connected components and cut vertices

Decompositions

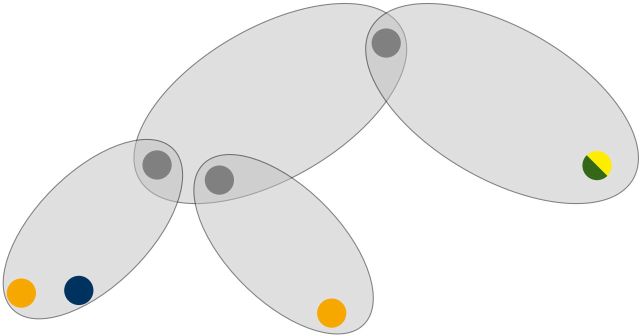


A decomposition of a connected graph into
2-connected components and cut vertices

DECOMPOSITIONS



Decompositions



Reduction scheme:

1. planar $\leq$ vertex-coloured 2-connected planar
2. vertex-col. 2-conn. planar $\leq$ arc-col. 3-conn. planar
3. arc-coloured 3-connected planar case

Planar graphs

Theorem (K., Ponomarenko, Schweitzer 2017)

Planar graphs have WL-dimension at most 3.

Planar graphs

Theorem (K., Ponomarenko, Schweitzer 2017)

Planar graphs have WL-dimension at most 3.

Reduction scheme:

1. planar $\leq$ vertex-coloured 2-connected planar **2-WL**
2. **vertex-col. 2-conn. planar $\leq$ arc-col. 3-conn. planar** **3-WL**
3. arc-coloured 3-connected planar case **3-WL**

Planar graphs

Theorem (K., Ponomarenko, Schweitzer 2017)

Planar graphs have WL-dimension at most 3.

Reduction scheme:

1. planar $\leq$ vertex-coloured 2-connected planar **2-WL**
2. **vertex-col. 2-conn. planar $\leq$ arc-col. 3-conn. planar** **2-WL**
3. arc-coloured 3-connected planar case **3-WL**

Planar graphs

Theorem (K., Ponomarenko, Schweitzer 2017)

Planar graphs have WL-dimension at most 3.

Reduction scheme:

1. planar $\leq$ vertex-coloured 2-connected planar **2-WL**
2. **vertex-col. 2-conn. planar $\leq$ arc-col. 3-conn. planar** **2-WL**
3. arc-coloured 3-connected planar case **3-WL**

Major ingredient for 2:

Theorem (K., Neuen 2019)

2-WL detects 2-separators in graphs.

Planar graphs

Theorem (K., Ponomarenko, Schweitzer 2017)

Planar graphs have WL-dimension at most 3.

Reduction scheme:

1. planar $\leq$ vertex-coloured 2-connected planar — **2-WL**
2. **vertex-col. 2-conn. planar $\leq$ arc-col. 3-conn. planar** — **2-WL**
3. arc-coloured 3-connected planar case — **3-WL**

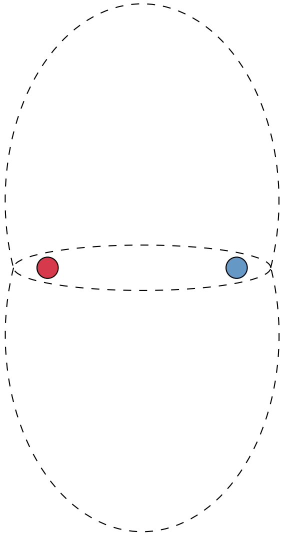
Major ingredient for 2:

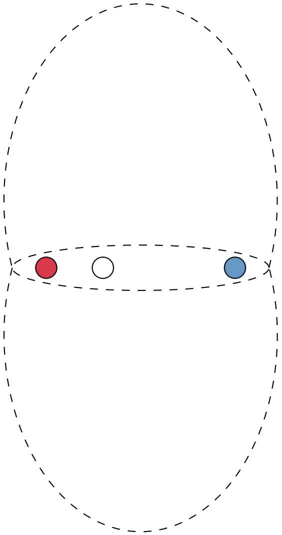
Theorem (K., Neuen 2019)

2-WL detects 2-separators in graphs.

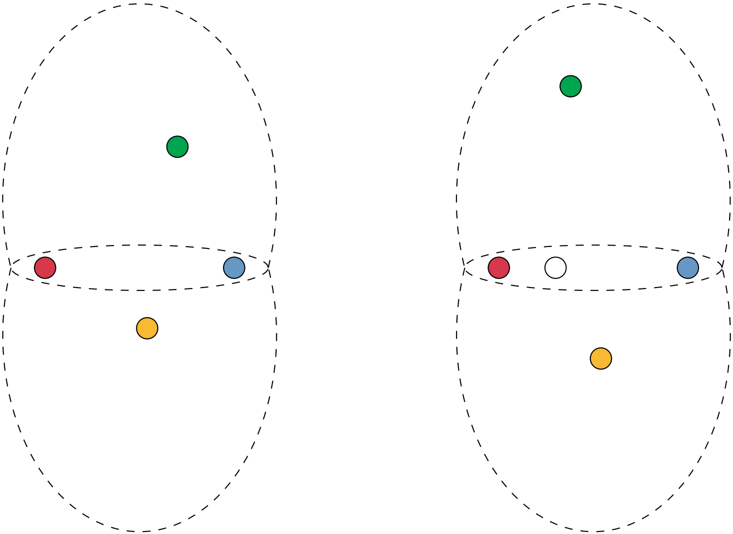
We show: 2-separators can be detected with 4-WL. (5 pebbles)

Detecting 2-Separators With 5 Pebbles

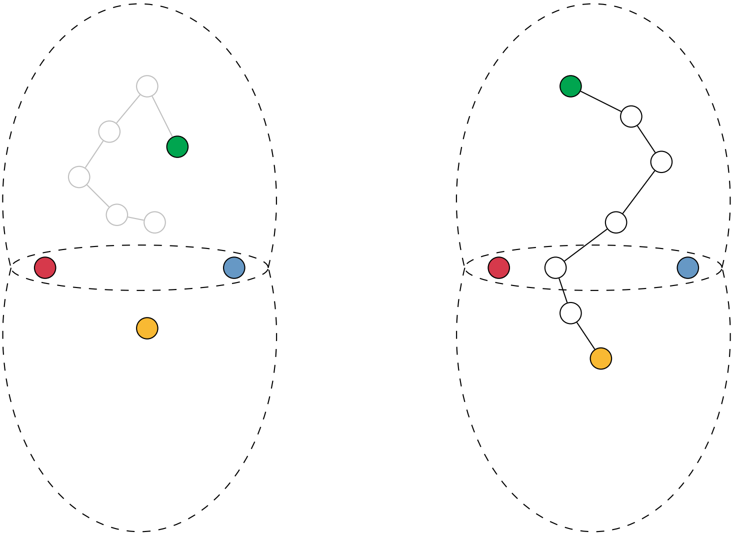




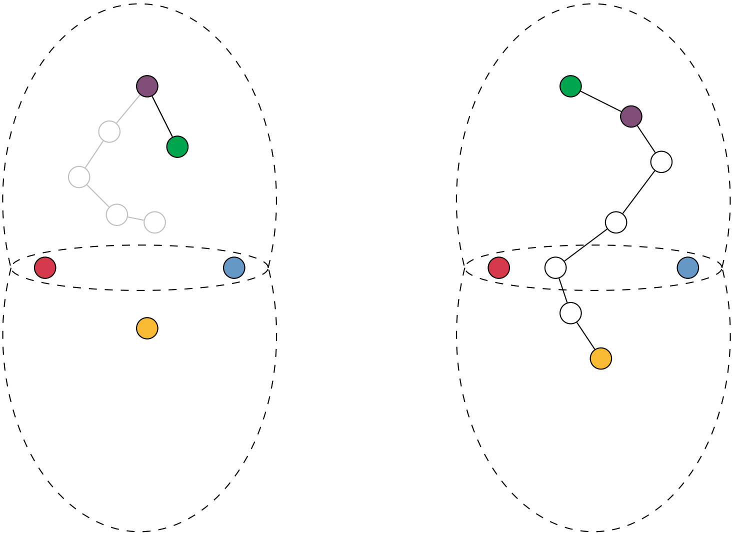
Detecting 2-Separators With 5 Pebbles



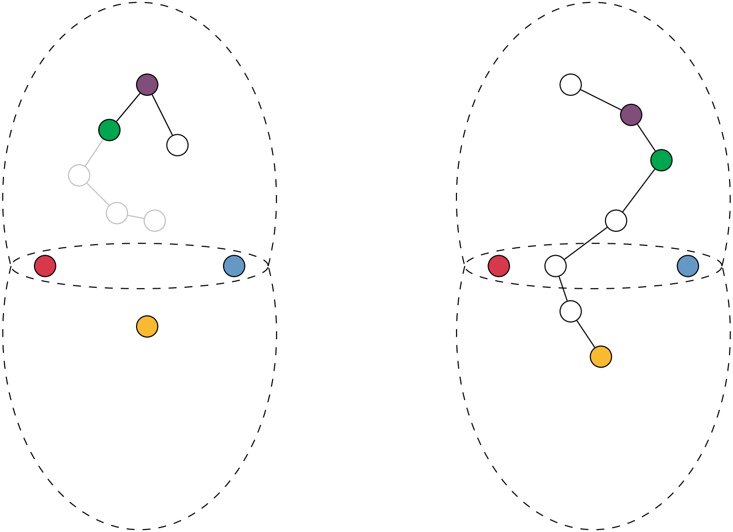
Detecting 2-Separators With 5 Pebbles



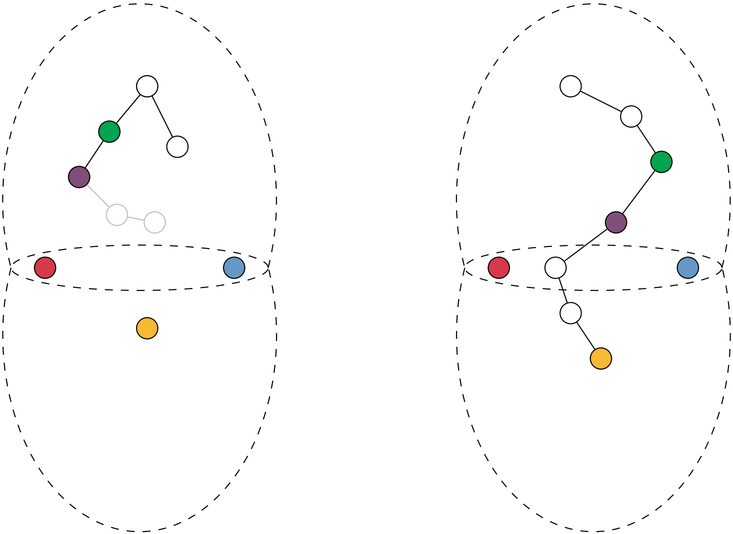
Detecting 2-Separators With 5 Pebbles



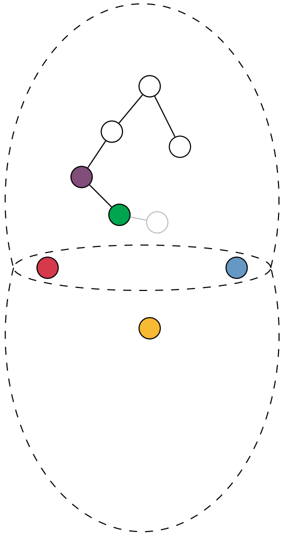
Detecting 2-Separators With 5 Pebbles

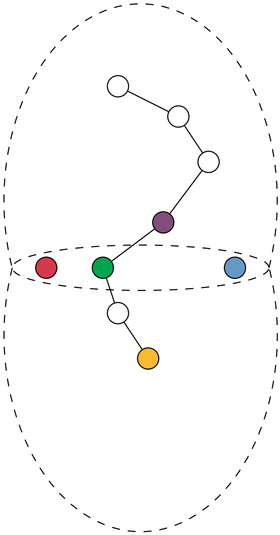


Detecting 2-Separators With 5 Pebbles

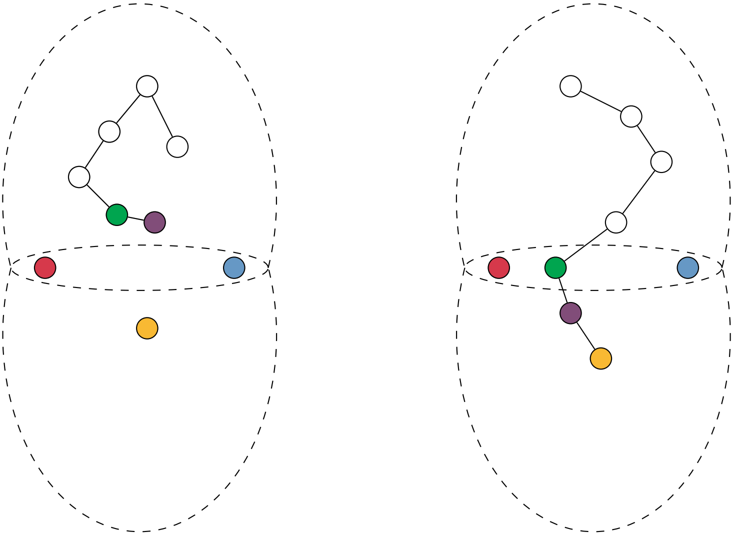


Detecting 2-Separators With 5 Pebbles





Detecting 2-Separators With 5 Pebbles



Spoiler wins.

Detecting Separators

Thus, 2-separators can be detected with **5 pebbles**. (4-WL)

Lemma (K., Ponomarenko, Schweitzer 2017)

*2-Separators can be detected with **4 pebbles**.* *(3-WL)*

Lemma (K., Neuen 2019)

*2-Separators can be detected with **3 pebbles**.* *(2-WL)*

Detecting Separators

Thus, 2-separators can be detected with **5 pebbles**. (4-WL)

Lemma (K., Ponomarenko, Schweitzer 2017)

*2-Separators can be detected with **4 pebbles**.* *(3-WL)*

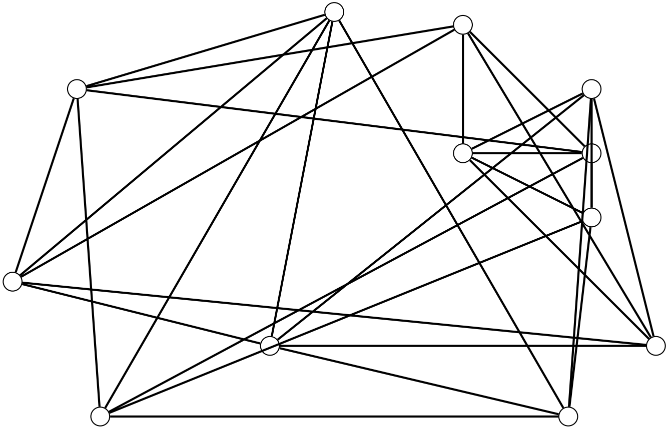
Lemma (K., Neuen 2019)

*2-Separators can be detected with **3 pebbles**.* *(2-WL)*

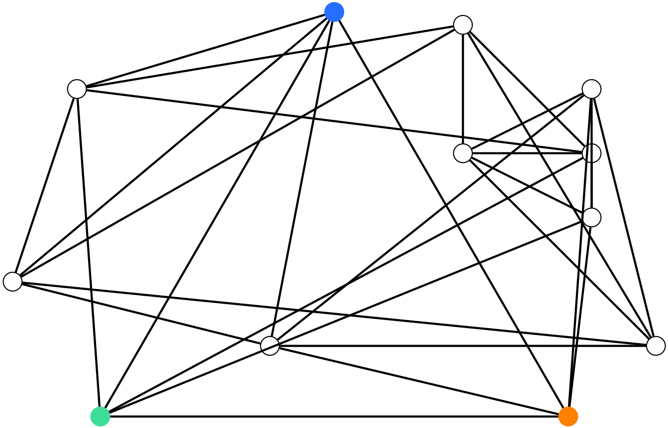
Reduction scheme:

1. planar $\leq$ vertex-coloured 2-connected planar **2-WL**
2. **vertex-col. 2-conn. planar $\leq$ arc-col. 3-conn. planar** **2-WL**
3. arc-coloured 3-connected planar case **3-WL**

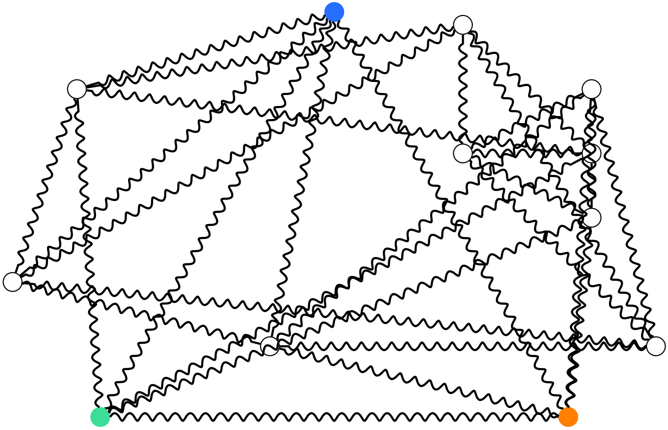
3-CONNECTED PLANAR GRAPHS



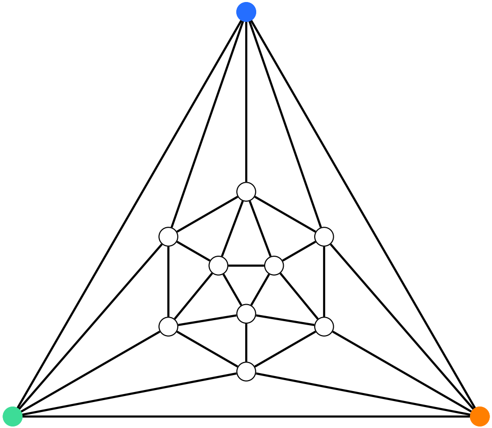
3-CONNECTED PLANAR GRAPHS



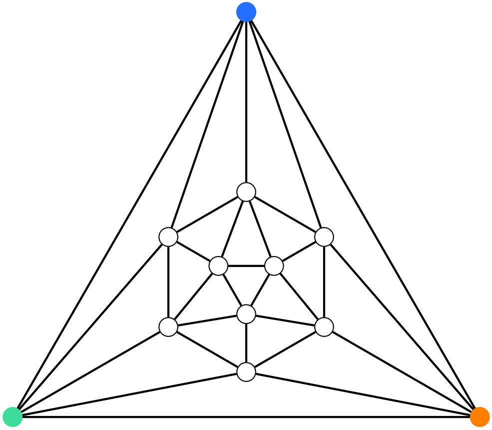
3-CONNECTED PLANAR GRAPHS



3-CONNECTED PLANAR GRAPHS



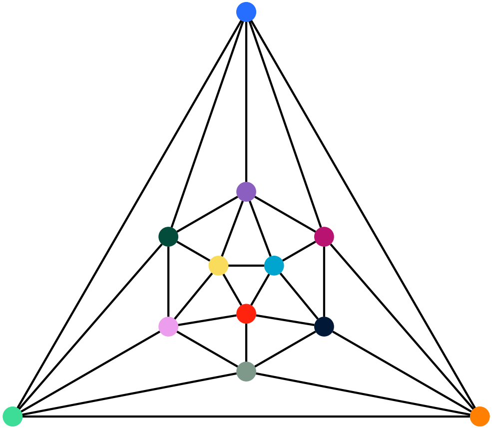
3-CONNECTED PLANAR GRAPHS



In **Tutte's Spring Embedding**, no two vertices are mapped to the same location.

We show: this implies that they get different colours w.r.t. 1-WL.

3-CONNECTED PLANAR GRAPHS



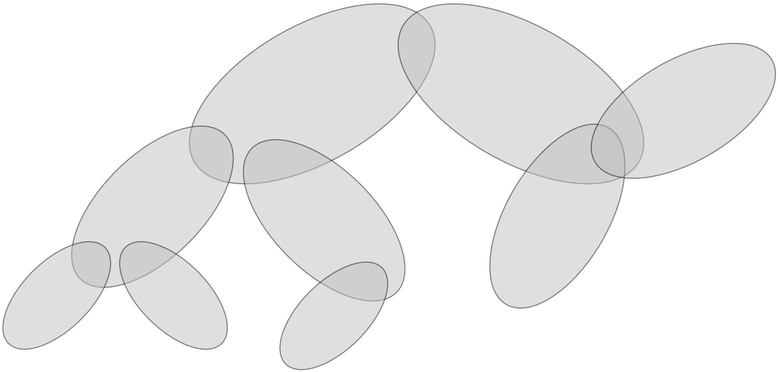
In **Tutte’s Spring Embedding**, no two vertices are mapped to the same location.

We show: this implies that they get different colours w.r.t. 1-WL.

Bounded-Treewidth Graphs

Theorem (K., Neuen 2019)

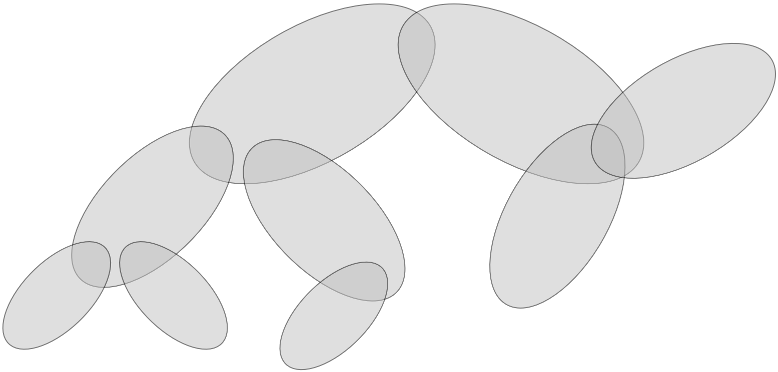
Let G be a graph of treewidth at most $k \geq 2$. Then k-WL identifies G.



Bounded-Treewidth Graphs

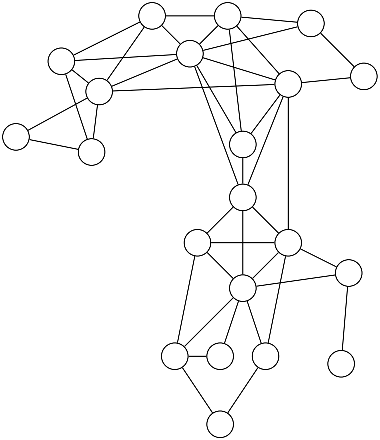
Theorem (K., Neuen 2019)

Let G be a graph of treewidth at most $k \geq 2$. Then k-WL identifies G.



Spoiler enforces a descent of a tree decomposition of width at most k of G.

BOUNDED-TREEWIDTH GRAPHS



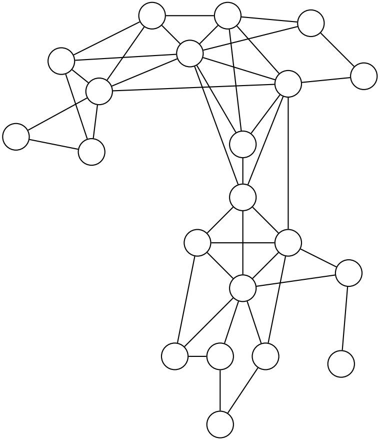
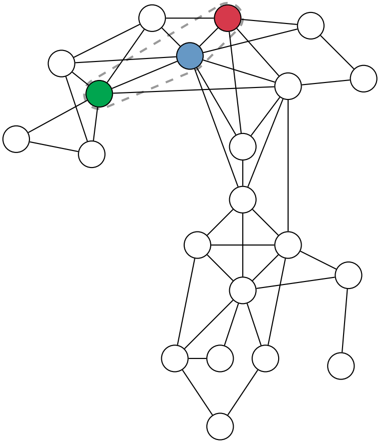


Image source: [Neuen]

Bounded-Treewidth Graphs



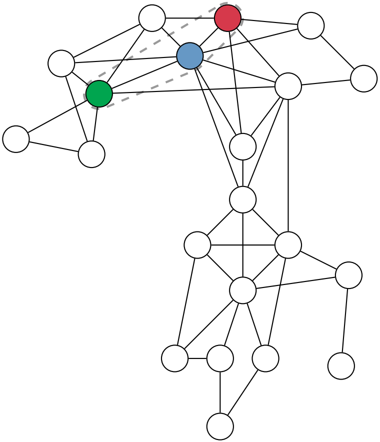


Image source: [Neuen]

Bounded-Treewidth Graphs

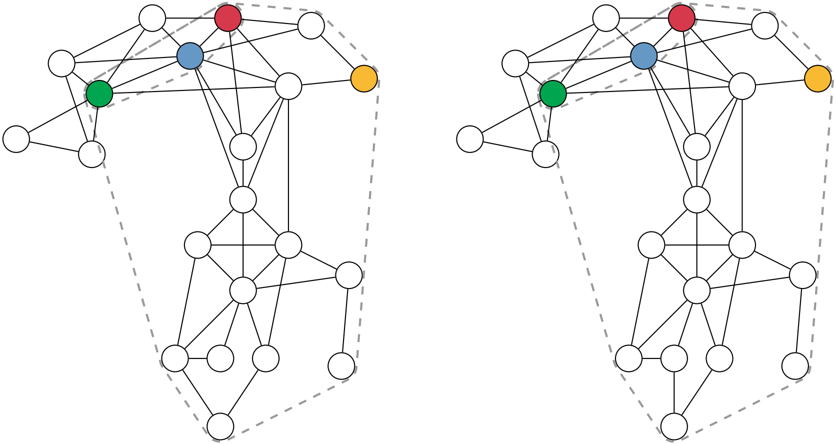


Image source: [Neuen]

Bounded-Treewidth Graphs

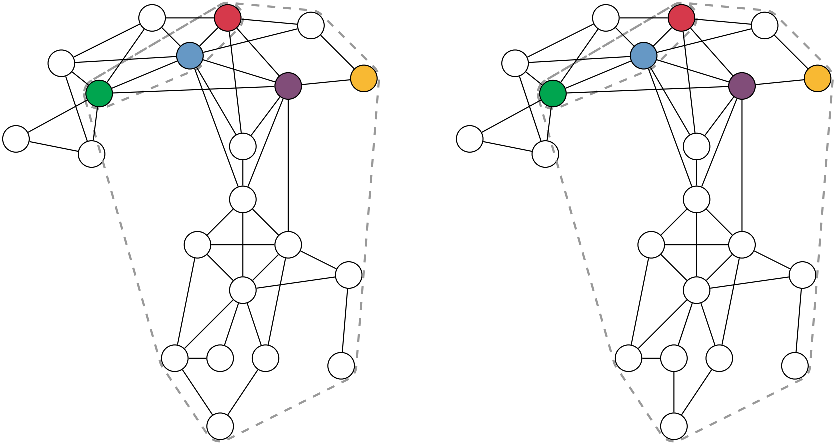


Image source: [Neuen]

Bounded-Treewidth Graphs

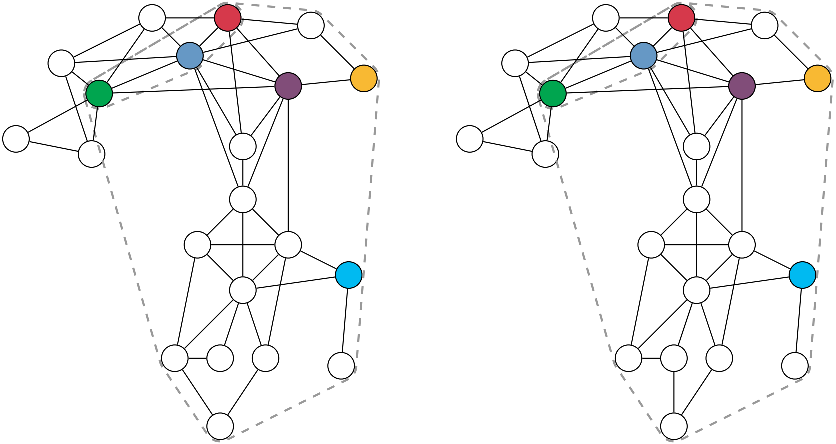


Image source: [Neuen]

Bounded-Treewidth Graphs

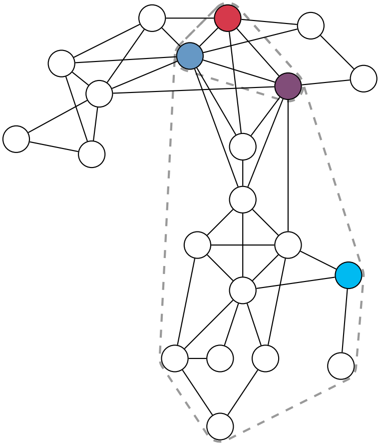
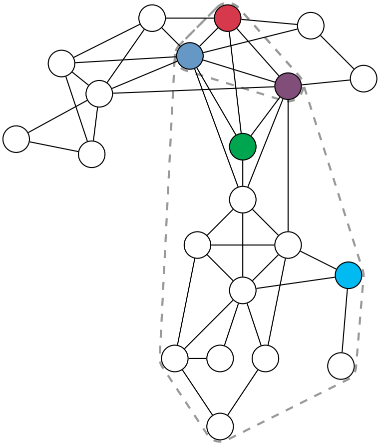


Image source: [Neuen]

Bounded-Treewidth Graphs



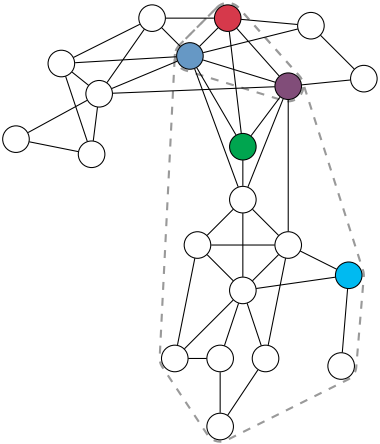


Image source: [Neuen]

Bounded-Treewidth Graphs

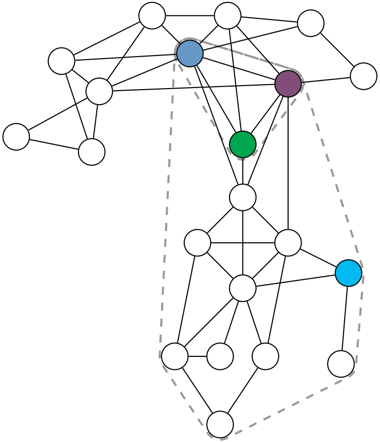


Image source: [Neuen]

Bounded-Treewidth Graphs

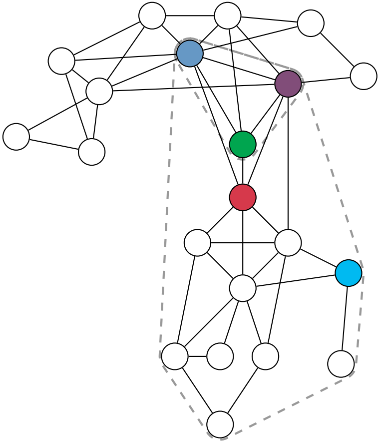
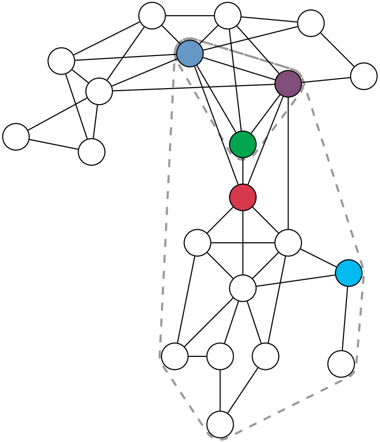



Image source: [Neuen]

Bounded-Treewidth Graphs

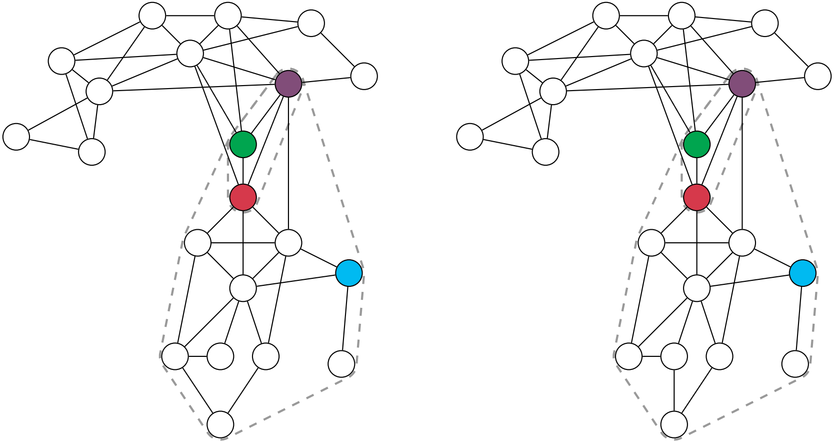


Image source: [Neuen]

Bounded-Treewidth Graphs

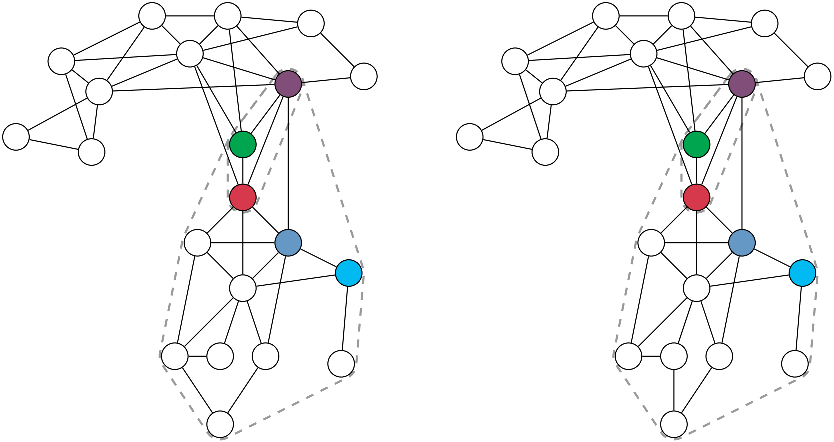


Image source: [Neuen]

Bounded-Treewidth Graphs

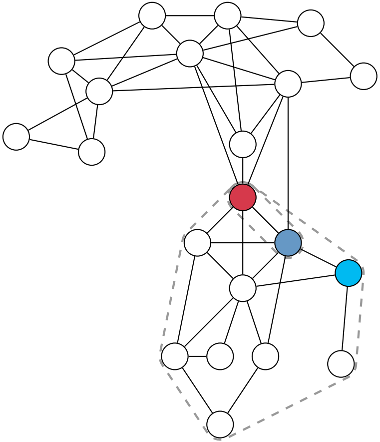


Image source: [Neuen]

Bounded-Treewidth Graphs

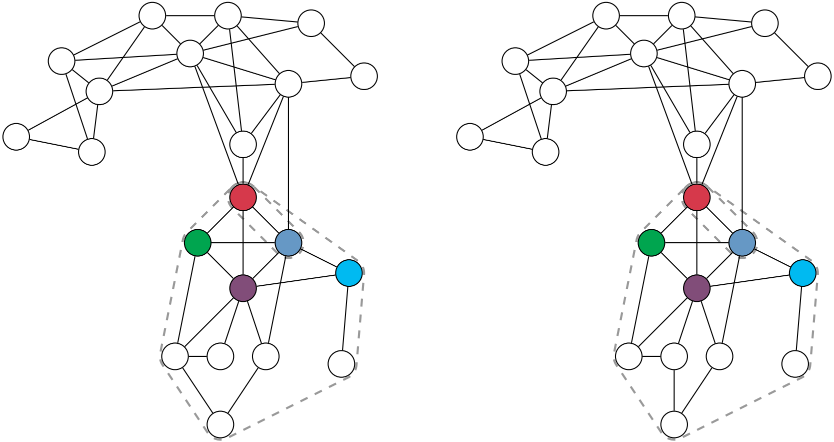
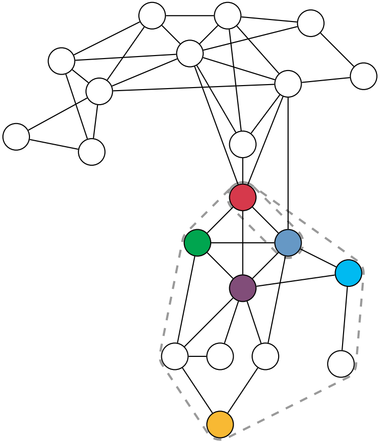


Image source: [Neuen]

Bounded-Treewidth Graphs



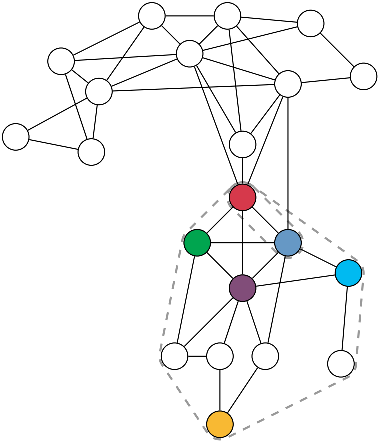


Image source: [Neuen]

Bounded-Treewidth Graphs

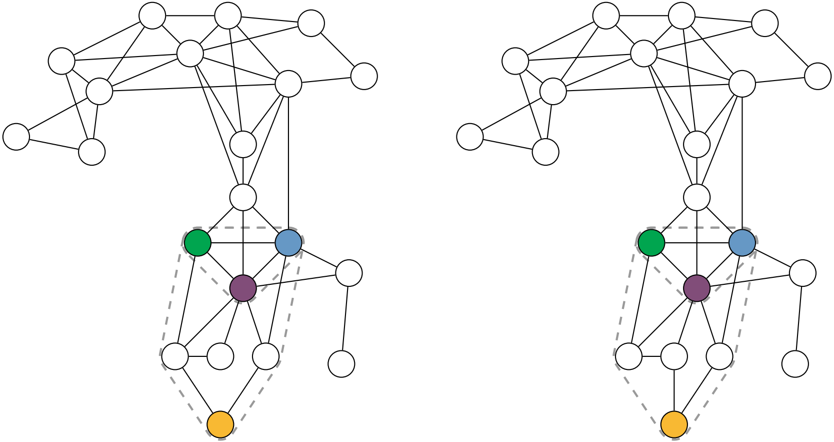
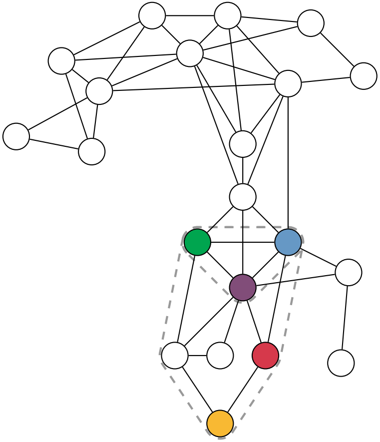


Image source: [Neuen]

Bounded-Treewidth Graphs



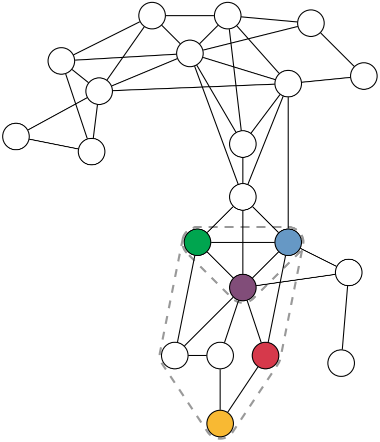


Image source: [Neuen]

Bounded-Treewidth Graphs

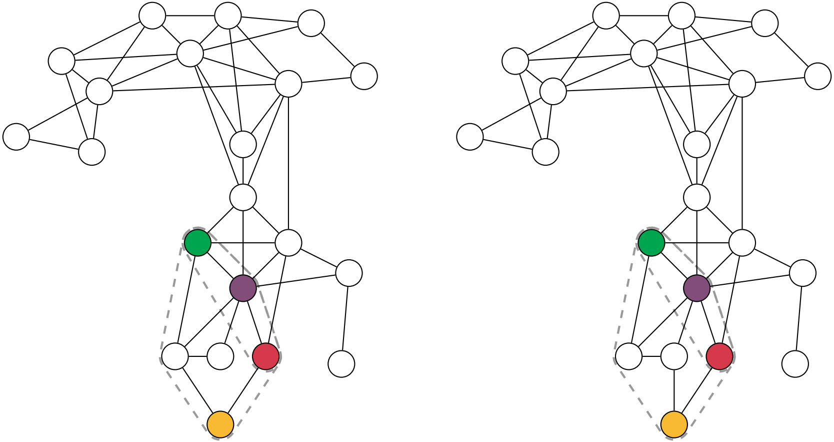
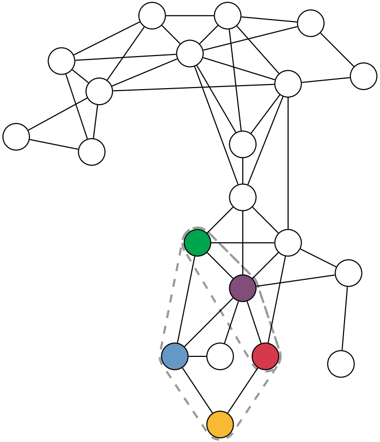


Image source: [Neuen]

Bounded-Treewidth Graphs



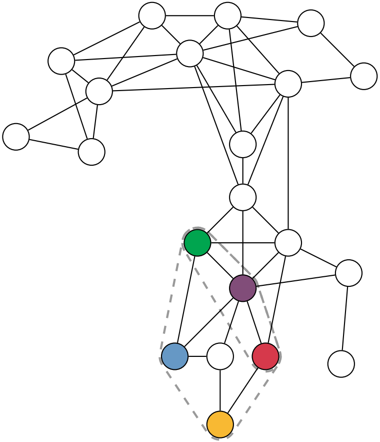
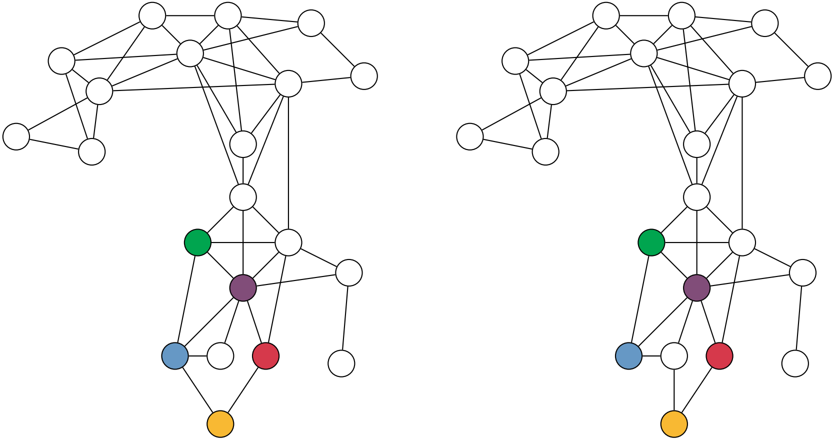


Image source: [Neuen]

Bounded-Treewidth Graphs



Spoiler wins.

Image source: [Neuen]

WL-DIMENSION

Definition

A graph G has *WL-dimension at most k* if k-WL identifies G.

Graph class	WL-dimension		
	lower bound	upper bound	
Trees	1	1	
Interval graphs	2	2	[Evdokimov, Ponomarenko, Tinhofer '00]
Excluded minor H	$\Omega(\lvert V(H)\rvert)$	$f(H)$	[Grohe '10]
Planar graphs	2	$\cancel{14}$ $\mathbf{3}$	**[K., Ponomarenko, Schweitzer '17]**
Treewidth k	$\cancel{\Omega(k)}$ $\frac{k}{2} - 2$	$\cancel{k+2}$ k	**[K., Neuen '19]**
Genus g	$\Omega(g)$	$4g+3$	[Grohe, K. '19]
Clique width k	$\Omega(k)$	$3k+4$	[Grohe, Neuen '19]
Rank width k	$\Omega(k)$	$3k+4$	[Grohe, Neuen '19]

Conclusion

Decompositions can be very helpful to bound the logical and algorithmic complexity of deciding isomorphism between graphs.

Conclusion

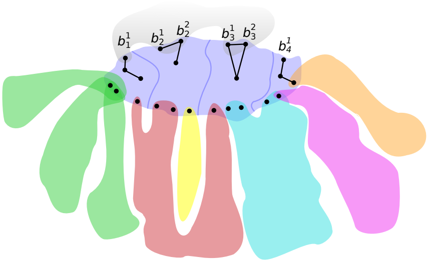
Decompositions can be very helpful to bound the logical and algorithmic complexity of deciding isomorphism between graphs.

They can also be useful for bounding the number of WL-iterations.

Conclusion

Decompositions can be very helpful to bound the logical and algorithmic complexity of deciding isomorphism between graphs.

They can also be useful for bounding the number of WL-iterations.



Theorem (Grohe, K. 2021)

There is a $k \in \mathbb{N}$ such that k-WL identifies all planar n-vertex graphs in $O(\log n)$ iterations.

Conclusion

The WL-dimension to distinguish two graphs is at most the dimension that distinguishes their decompositions into 3-connected components.

Conclusion

The WL-dimension to distinguish two graphs is at most the dimension that distinguishes their decompositions into 3-connected components.

WL-Dimension of Planar Graphs

1. Reduction to 2-connected graphs **2-WL**
2. Reduction to 3-connected graphs **3-WL**
3. 3-connected planar graphs **3-WL**

Conclusion

The WL-dimension to distinguish two graphs is at most the dimension that distinguishes their decompositions into 3-connected components.

WL-Dimension of Planar Graphs

1. Reduction to 2-connected graphs **2-WL**
2. Reduction to 3-connected graphs **2-WL**
3. 3-connected planar graphs **3-WL**

Conclusion

The WL-dimension to distinguish two graphs is at most the dimension that distinguishes their decompositions into 3-connected components.

WL-Dimension of Planar Graphs

1. Reduction to 2-connected graphs **2-WL**
2. Reduction to 3-connected graphs **2-WL**
3. 3-connected planar graphs **3-WL**

- **What is the exact WL-dimension of planar graphs?**
- What about other graph classes?
- **What other useful decompositions does C detect**?

Conclusion

The WL-dimension to distinguish two graphs is at most the dimension that distinguishes their decompositions into 3-connected components.

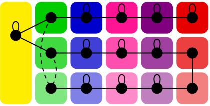
WL-Dimension of Planar Graphs

1. Reduction to 2-connected graphs **2-WL**
2. Reduction to 3-connected graphs **2-WL**
3. 3-connected planar graphs **3-WL**

- **What is the exact WL-dimension of planar graphs?**
- What about other graph classes?
- **What other useful decompositions does C detect**?

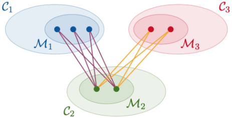
$\rightsquigarrow$ **ICALP-talk on Thursday** ☺

WL-Complexity



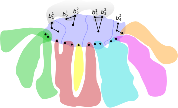
1-WL

Graphs with $n{-}1$ iterations



2-WL

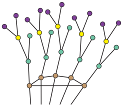
First nontrivial upper bound



Planar graphs

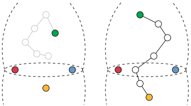
Logarithmic upper bound

WL-Power



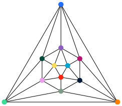
1-WL

Complete characterisation



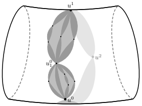
2-WL

Detects 2-separators



Planar graphs

WL-dim ≤ 3



Euler genus

WL-dim ≤ $4g{+}3$