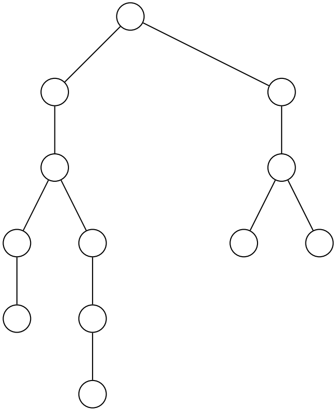
Graph decompositions and their algorithms

Édouard Bonnet

ENS Lyon, LIP

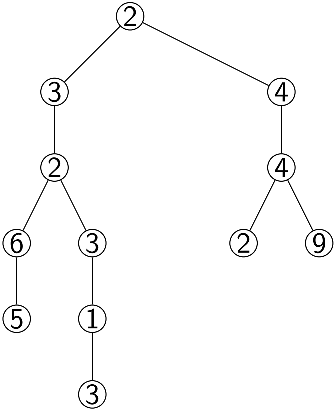
May 5th, 2022, Demi-Journée du Pôle Calcul

Trees



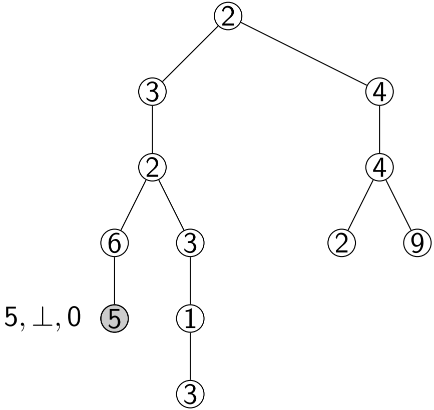
Trees make most NP-hard problems easy

Example of MIN WEIGHTED DOMINATING SET



Trees make most NP-hard problems easy

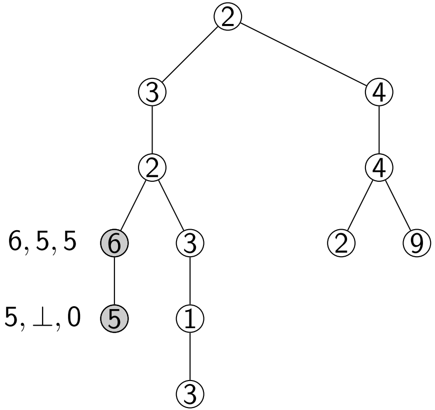
Example of MIN WEIGHTED DOMINATING SET



Idea: keep 3 lightest dominating sets of each subtree (rooted at u) one containing u, one not containing u, and one disregarding u

Trees make most NP-hard problems easy

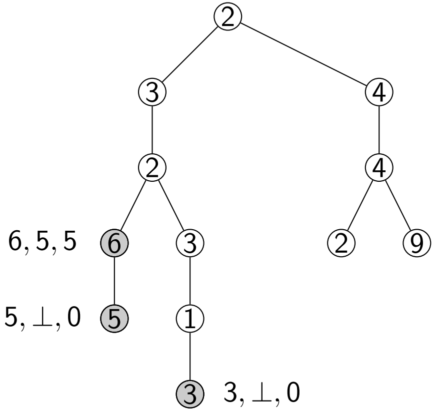
Example of MIN WEIGHTED DOMINATING SET



Idea: keep 3 lightest dominating sets of each subtree (rooted at u)
one containing u, one not containing u, and one disregarding u

Trees make most NP-hard problems easy

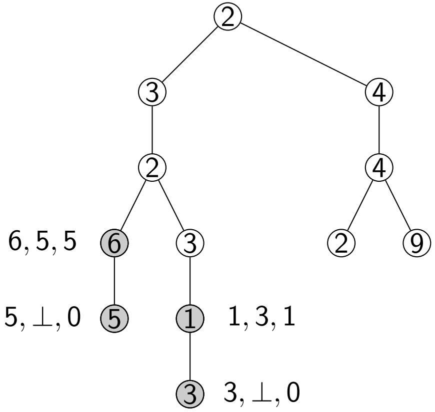
Example of Min Weighted Dominating Set



Idea: keep 3 lightest dominating sets of each subtree (rooted at u)
one containing u, one not containing u, and one disregarding u

Trees make most NP-hard problems easy

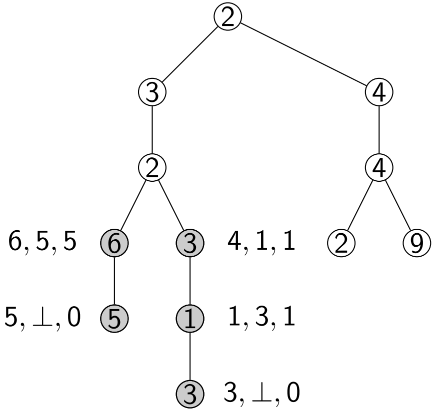
Example of MIN WEIGHTED DOMINATING SET



Idea: keep 3 lightest dominating sets of each subtree (rooted at u)
one containing u, one not containing u, and one disregarding u

Trees make most NP-hard problems easy

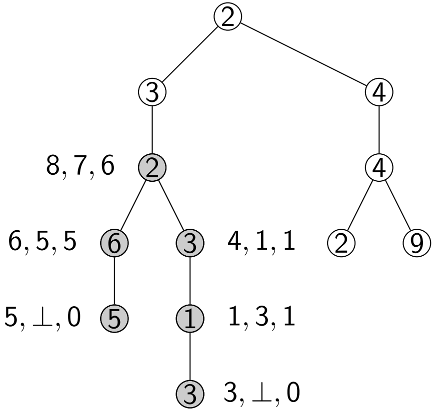
Example of MIN WEIGHTED DOMINATING SET



Idea: keep 3 lightest dominating sets of each subtree (rooted at u)
one containing u, one not containing u, and one disregarding u

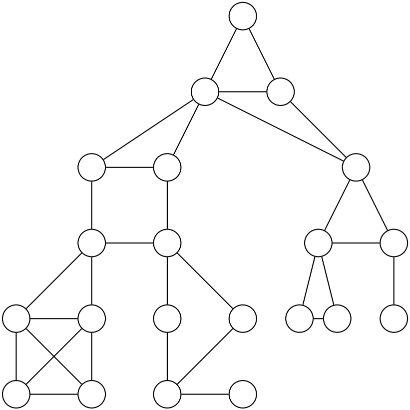
Trees make most NP-hard problems easy

Example of MIN WEIGHTED DOMINATING SET



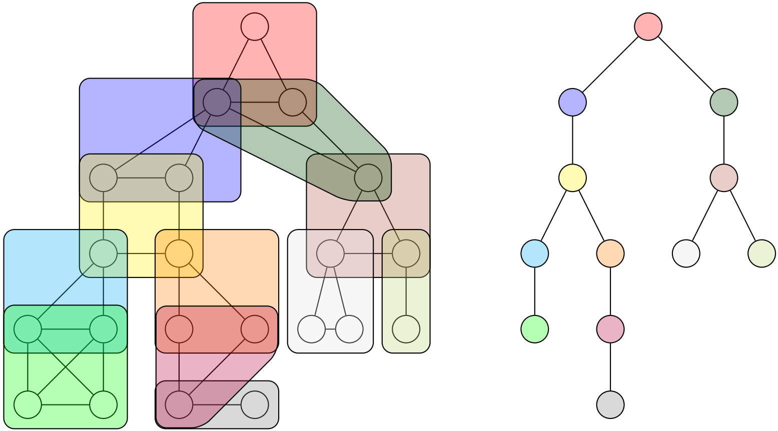
Idea: keep 3 lightest dominating sets of each subtree (rooted at u) one containing u, one not containing u, and one disregarding u

Tree decomposition



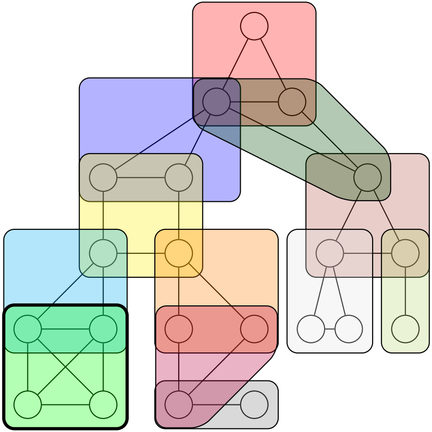
Tree decomposition

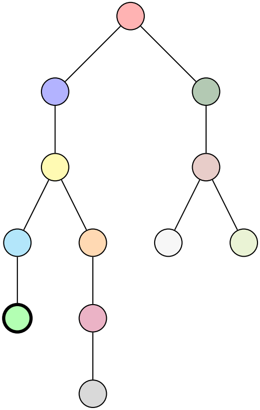
Cover by bags mapping to a tree s.t. each vertex lies in a subtree

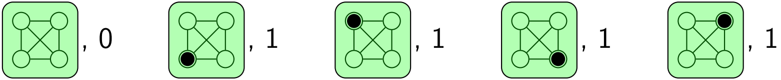


Tree decomposition: solving MAX INDEPENDENT SET

For each trace in each bag, keep a best solution in what is below

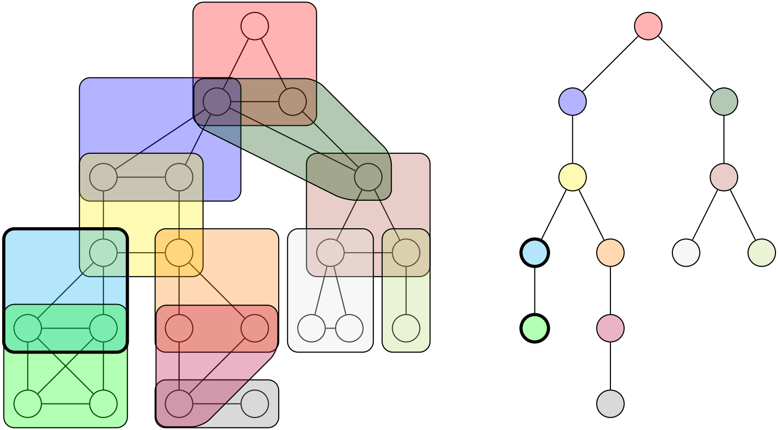


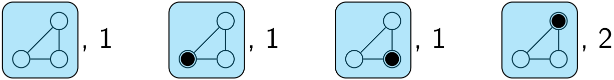




Tree decomposition: solving Max Independent Set

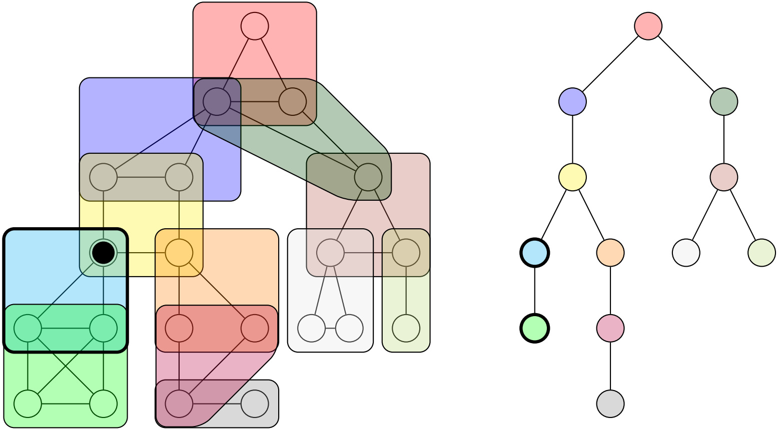
For each trace in each bag, keep a best solution in what is below

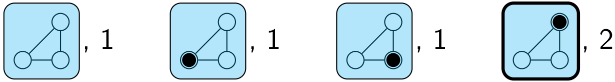




Tree decomposition: solving MAX INDEPENDENT SET

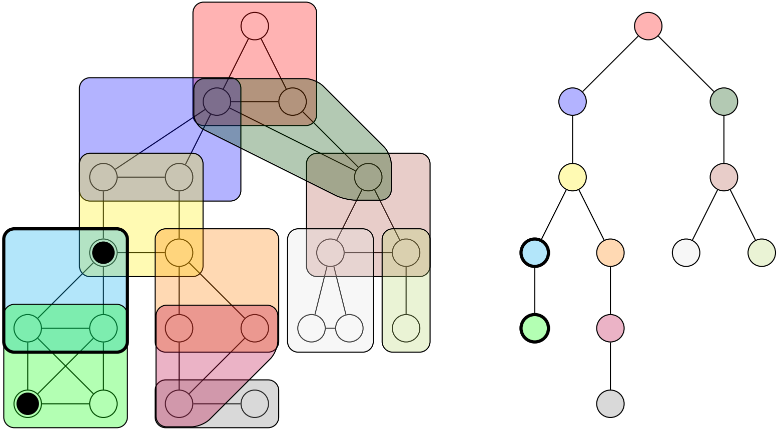
For each trace in each bag, keep a best solution in what is below

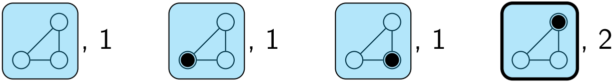




Tree decomposition: solving MAX INDEPENDENT SET

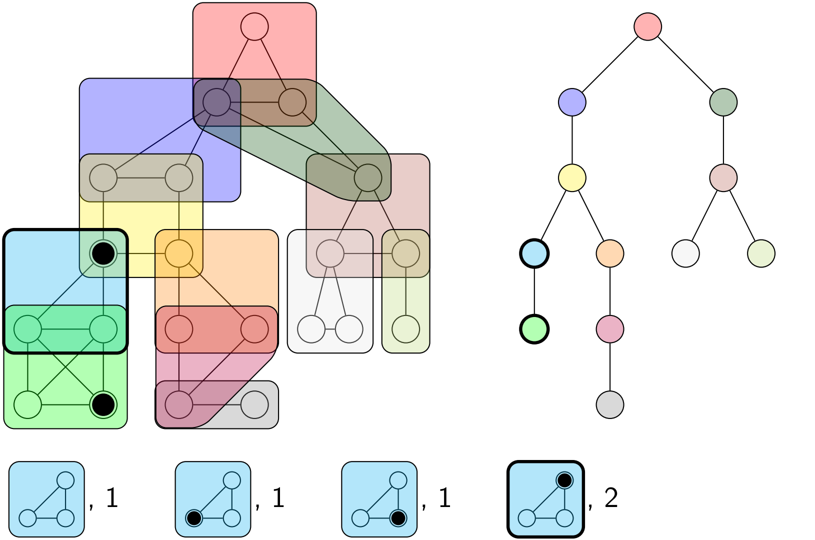
For each trace in each bag, keep a best solution in what is below





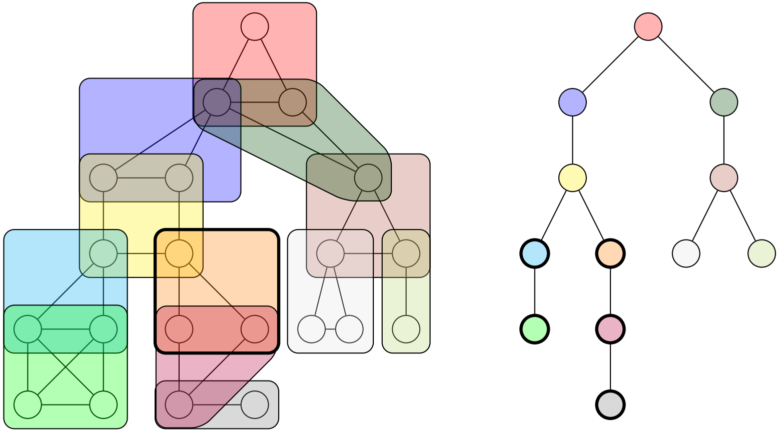
Tree decomposition: solving Max Independent Set

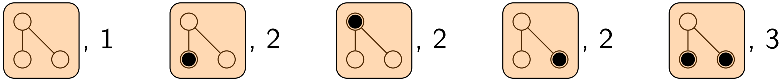
For each trace in each bag, keep a best solution in what is below



Tree decomposition: solving Max Independent Set

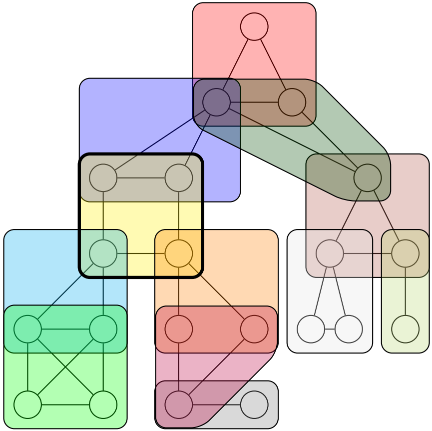
For each trace in each bag, keep a best solution in what is below

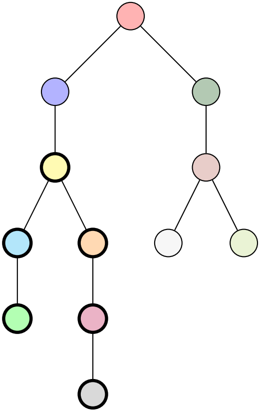


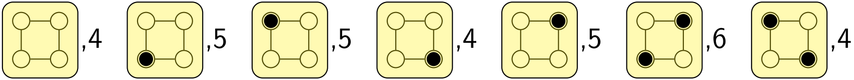


Tree decomposition: solving MAX INDEPENDENT SET

For each trace in each bag, keep a best solution in what is below

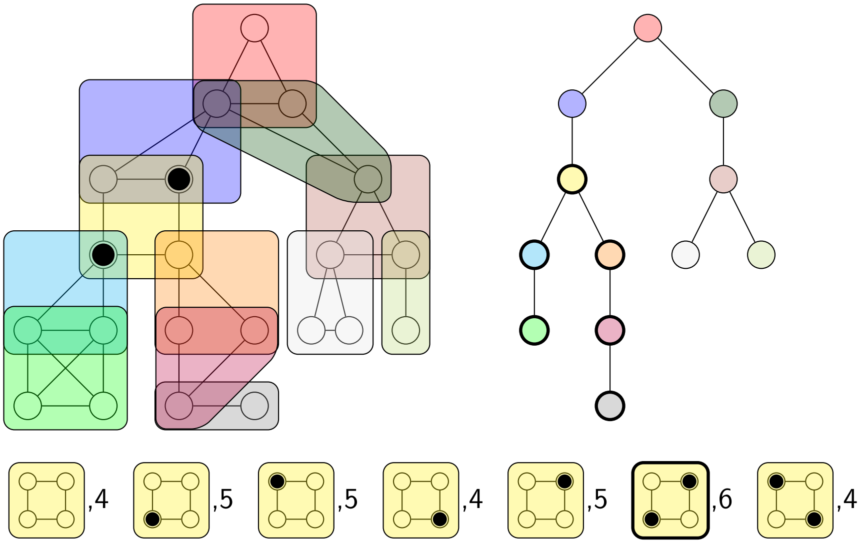






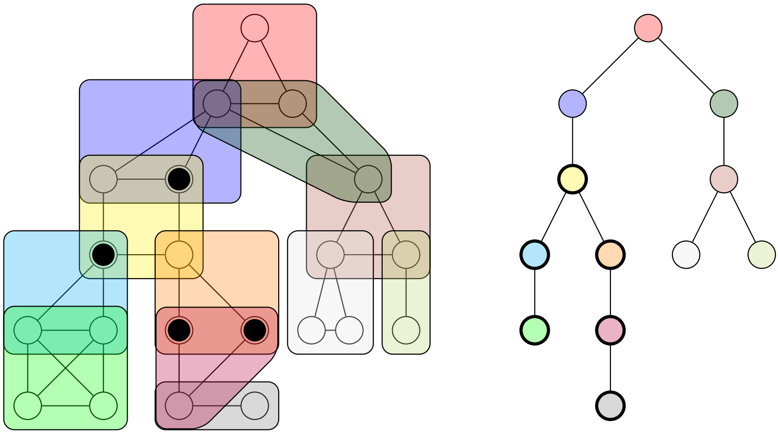
Tree decomposition: solving Max Independent Set

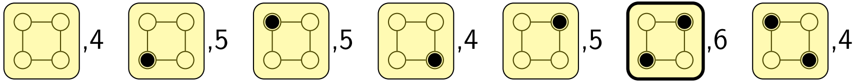
For each trace in each bag, keep a best solution in what is below



Tree decomposition: solving Max Independent Set

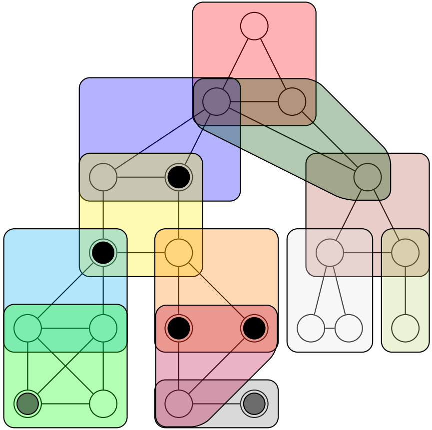
For each trace in each bag, keep a best solution in what is below

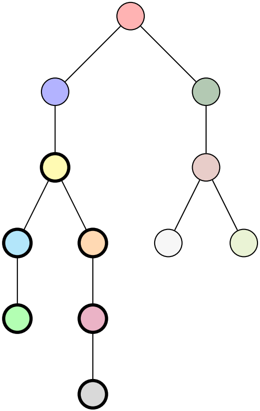


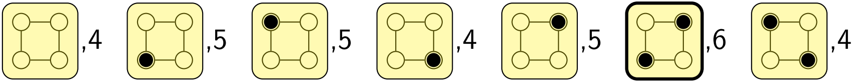


Tree decomposition: solving Max Independent Set

For each trace in each bag, keep a best solution in what is below







Treewidth

Minimum largest bag size over all tree decompositions minus 1

- rediscovered several times in the 70's and 80's...
- made central by *Graph Minors* and algorithmic graph theory
- previous slide: $2^{O(\mathrm{tw})}n$ time with n bags

Treewidth

Minimum largest bag size over all tree decompositions minus 1

- rediscovered several times in the 70's and 80's...
- made central by *Graph Minors* and algorithmic graph theory
- previous slide: $2^{O(\mathrm{tw})}n$ time with n bags

Computing a tree decomposition?

Treewidth

Minimum largest bag size over all tree decompositions minus 1

- rediscovered several times in the 70's and 80's...
- made central by *Graph Minors* and algorithmic graph theory
- previous slide: $2^{O(\text{tw})}n$ time with n bags

Computing a tree decomposition? NP-hard but various algorithms

width $2\text{tw}+1$ in $2^{O(\text{tw})}n$

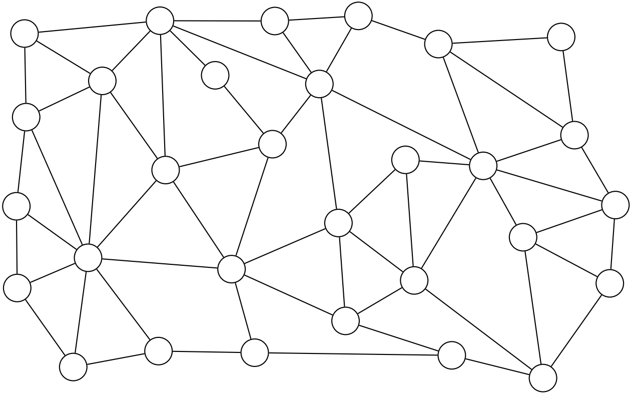
width $5\text{tw}+4$ in $2^{6.76\text{tw}}n\log n$

width tw in $2^{O(\text{tw}^3)}n$

width $\text{tw}\sqrt{\log \text{tw}}$ in $n^{O(1)}$

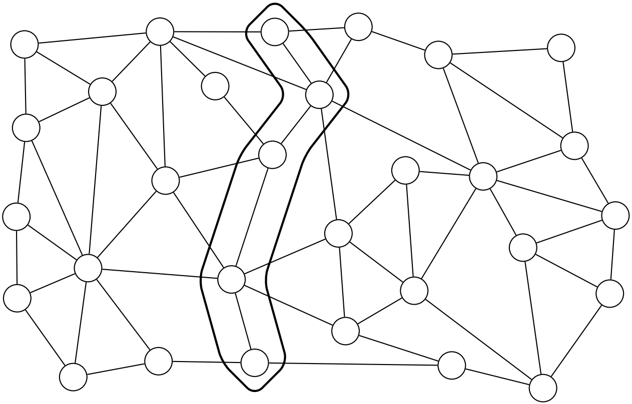
width tw in 1.74^n

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



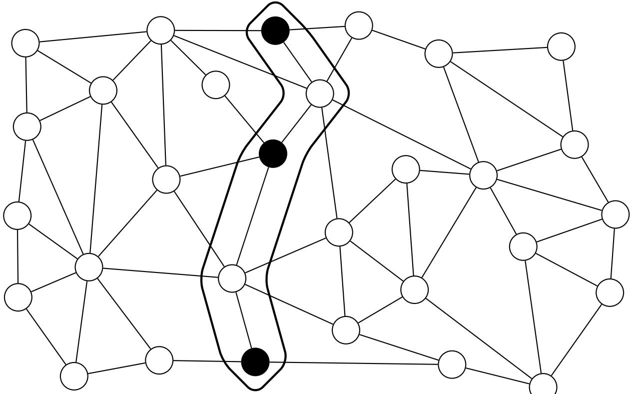
Planar graphs have treewidth $O(\sqrt{n})$

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



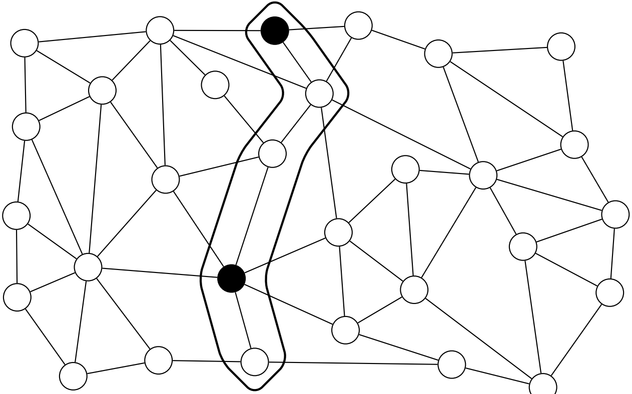
Equivalently $O(\sqrt{n})$ balanced separators, i.e., sides of size $\leqslant 2n/3$

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



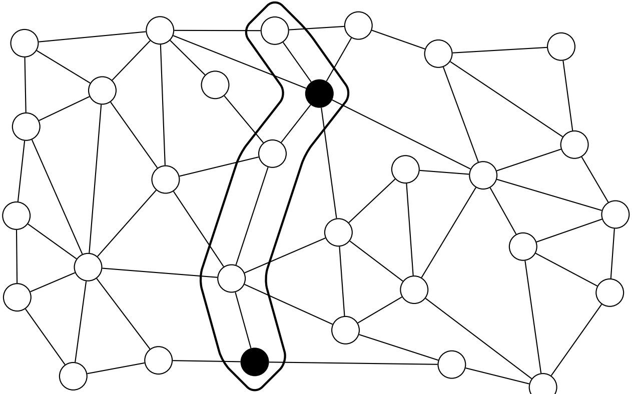
Max Independent Set, 3-Coloring, Hamiltonian Path...

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



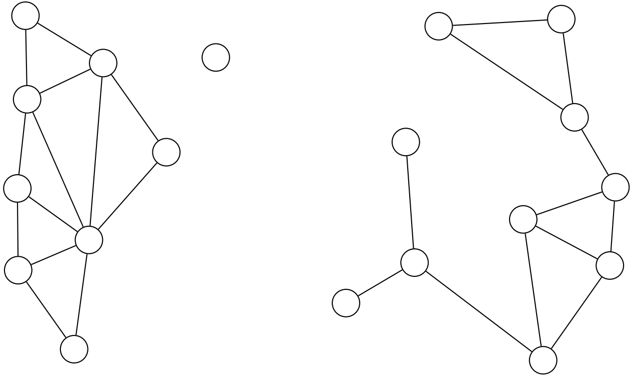
<u>Max Independent Set</u>, 3-Coloring, Hamiltonian Path...

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



<u>Max Independent Set</u>, 3-Coloring, Hamiltonian Path...

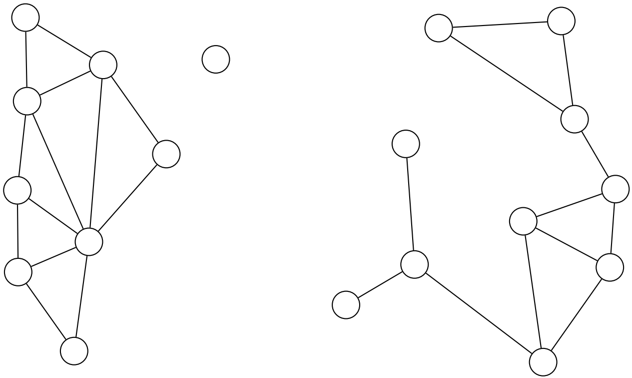
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



<u>MAX INDEPENDENT SET</u>, 3-COLORING, HAMILTONIAN PATH...

$$T(n) \leqslant 2^{O(\sqrt{n})} T(2n/3) \leqslant \ldots \leqslant 2^{O(\sqrt{n}) \sum_i \sqrt{2/3}^i} = 2^{O(\sqrt{n})}$$

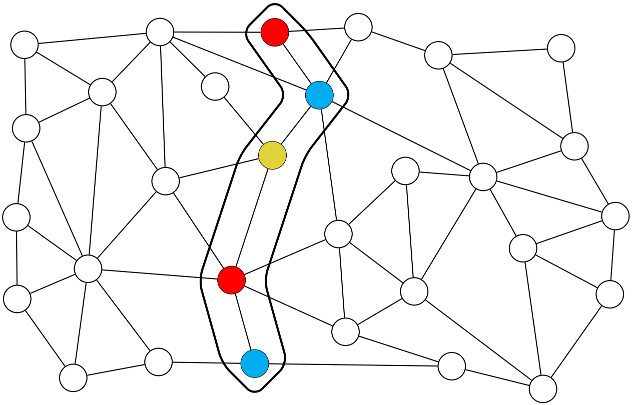
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



<u>Max Independent Set</u>, 3-Coloring, Hamiltonian Path...

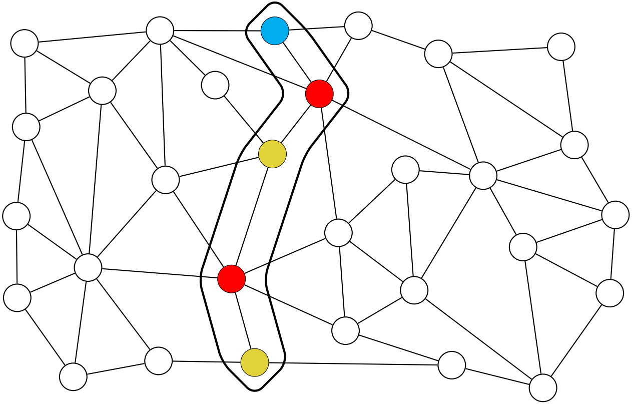
Even polyspace!

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



MAX INDEPENDENT SET, <u>3-COLORING</u>, HAMILTONIAN PATH...

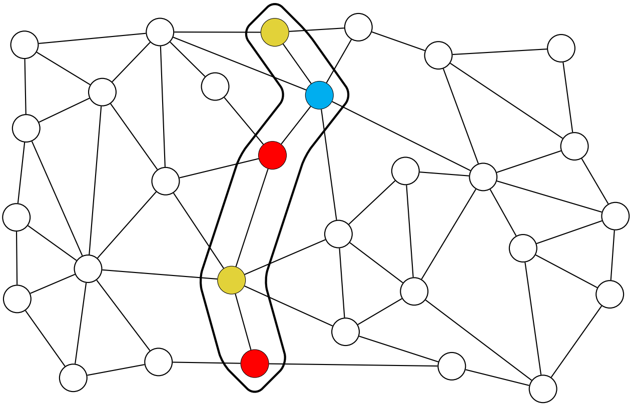
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



Max Independent Set, 3-Coloring, Hamiltonian Path...

solve the extension List 3-Coloring

$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan



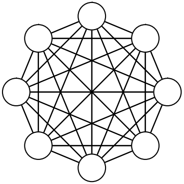
MAX INDEPENDENT SET, <u>3-COLORING</u>, HAMILTONIAN PATH...

solve the extension LIST 3-COLORING

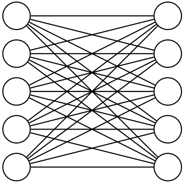
Decomposition of dense graphs?

Graphs with small treewidth have linearly many edges

What about simple dense graphs?



clique

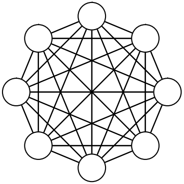


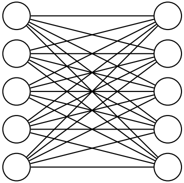
biclique

Decomposition of dense graphs?

Graphs with small treewidth have linearly many edges

What about simple dense graphs?

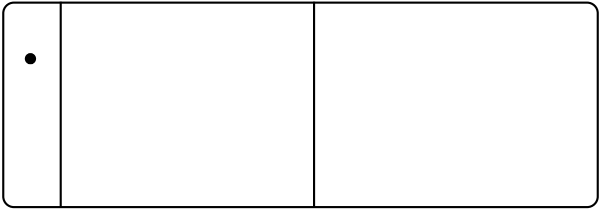

clique


biclique

- cliquewidth defined in the 90's
- allows faster algorithms but hard to compute itself
- rankwidth [Oum, Seymour '05] "equivalent" and approximable

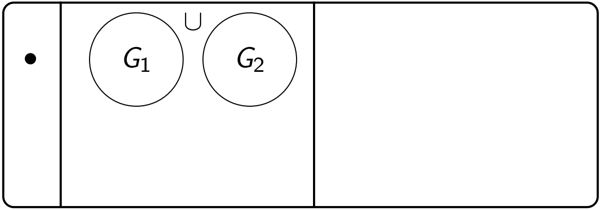
We will see another equivalent definition via *contraction sequences*

Cographs



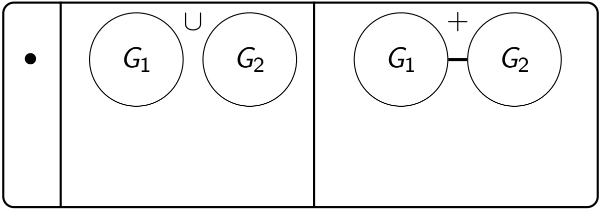
A single vertex is a cograph,

Cographs



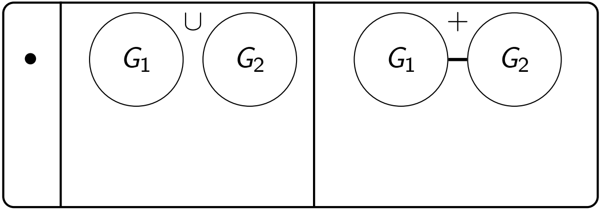
as well as the union of two cographs,

Cographs

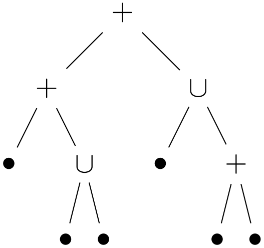


and the complete join of two cographs.

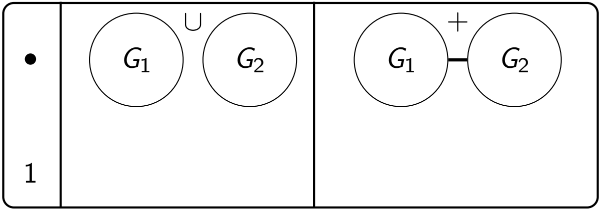
Cographs



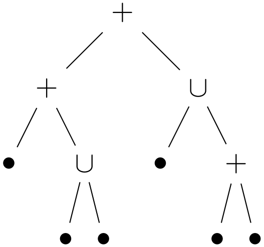
Many NP-hard problems are polytime solvable on cographs



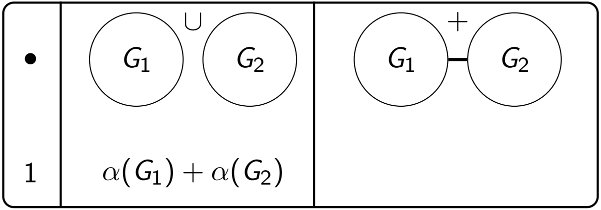
Cographs



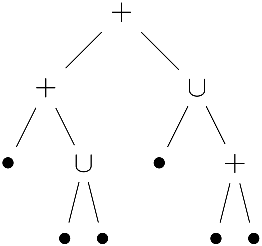
For instance the independence number $\alpha(G)$ is polytime



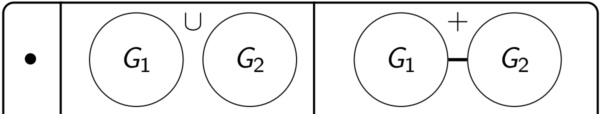
Cographs



In case of a disjoint union: combine the solutions

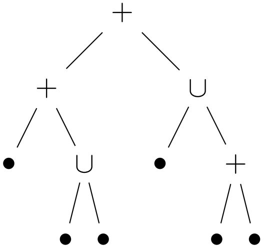


Cographs

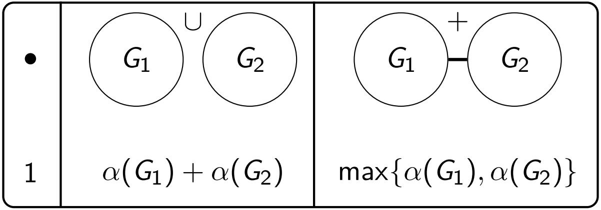


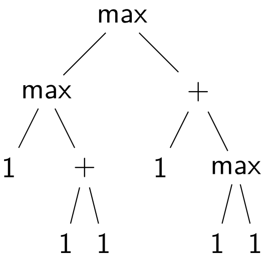
1	$\alpha(G_1) + \alpha(G_2)$	$\max\{\alpha(G_1), \alpha(G_2)\}$

In case of a complete join: pick the larger one



Cographs


∪
G_1 G_2
+
G_1 G_2
1
$\alpha(G_1) + \alpha(G_2)$
$\max\{\alpha(G_1), \alpha(G_2)\}$

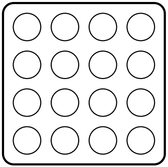

max
max
+
1
+
1
max
1 1
1 1

Another cograph definition

Every induced subgraph has two twins

Another cograph definition

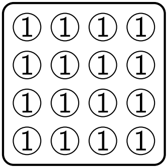
Every induced subgraph has two twins



Is there another algorithmic scheme based on this definition?

Another cograph definition

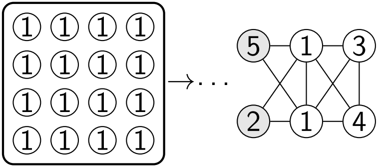
Every induced subgraph has two twins



We store in each vertex its inner max independent set

Another cograph definition

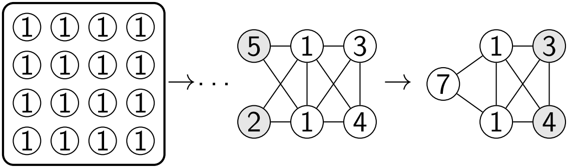
Every induced subgraph has two twins



We can find a pair of false/true twins

Another cograph definition

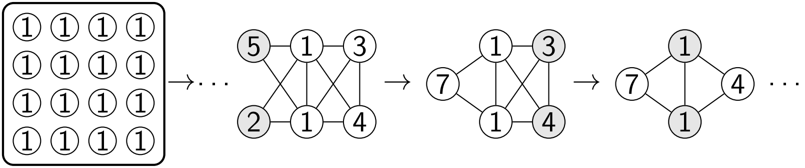
Every induced subgraph has two twins



Sum them if they are false twins

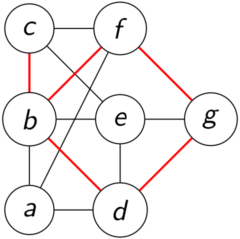
Another cograph definition

Every induced subgraph has two twins



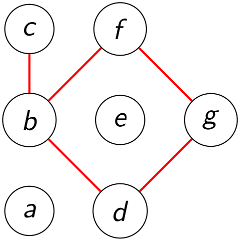
Max them if they are true twins

Trigraphs



Three outcomes between a pair of vertices:
edge, or non-edge, or red edge

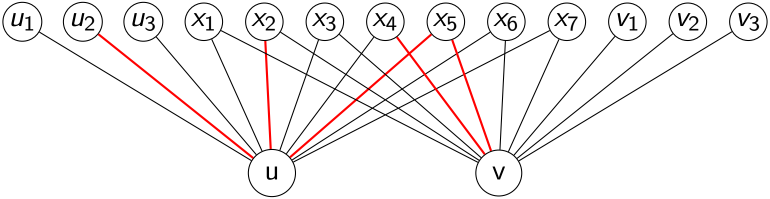
Trigraphs



Three outcomes between a pair of vertices:
edge, or non-edge, or red edge

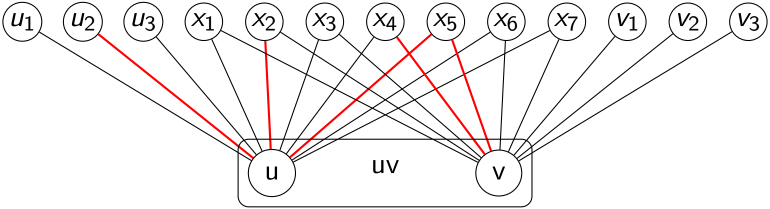
Red graph: trigraph minus its black edges

Contractions in trigraphs



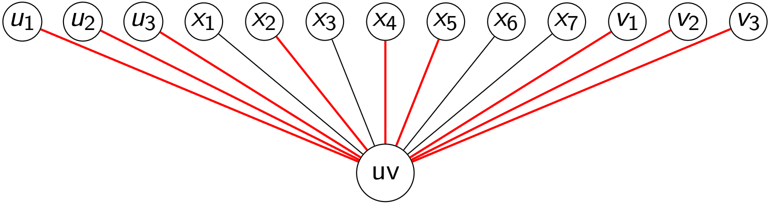
Identification of two non-necessarily adjacent vertices

Contractions in trigraphs



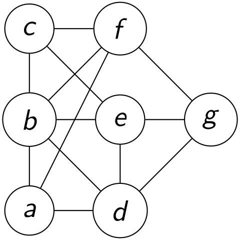
Identification of two non-necessarily adjacent vertices

Contractions in trigraphs



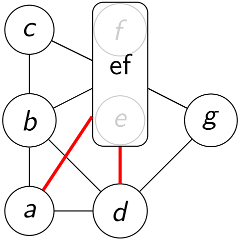
edges to $N(u) \triangle N(v)$ turn red, for $N(u) \cap N(v)$ red is absorbing

Contraction sequence



A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that
G_i is obtained by performing one contraction in G_{i+1}.

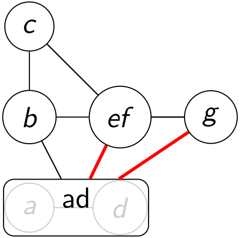
Contraction sequence



A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \dots, G_2, G_1$ such that
G_i is obtained by performing one contraction in G_{i+1}.

partition viewpoint: $G_i \leftrightsquigarrow (G, \mathcal{P}_i)$, vertex $\leftrightsquigarrow$ part
$G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$

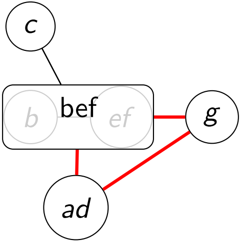
Contraction sequence



A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that
G_i is obtained by performing one contraction in G_{i+1}.

partition viewpoint: $G_i \leftrightsquigarrow (G, \mathcal{P}_i)$, vertex $\leftrightsquigarrow$ part
$G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$

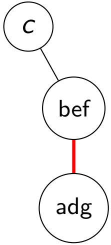
Contraction sequence



A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \dots, G_2, G_1$ such that
G_i is obtained by performing one contraction in G_{i+1}.

partition viewpoint: $G_i \leftrightsquigarrow (G, \mathcal{P}_i)$, vertex $\leftrightsquigarrow$ part
$G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$

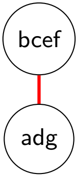
Contraction sequence



A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that G_i is obtained by performing one contraction in G_{i+1}.

partition viewpoint: $G_i \leftrightsquigarrow (G, \mathcal{P}_i)$, vertex $\leftrightsquigarrow$ part
$G\langle S \rangle = G[\bigcup$ vertices of G contracted into a vertex of $S]$

Contraction sequence



A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that
G_i is obtained by performing one contraction in G_{i+1}.

partition viewpoint: $G_i \leftrightsquigarrow (G, \mathcal{P}_i)$, vertex $\leftrightsquigarrow$ part
$G\langle S \rangle = G[\bigcup$ vertices of G contracted into a vertex of $S]$

Contraction sequence

A contraction sequence of G:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that
G_i is obtained by performing one contraction in G_{i+1}.

partition viewpoint: $G_i \leftrightsquigarrow (G, \mathcal{P}_i)$, vertex $\leftrightsquigarrow$ part
$G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$

Reduced parameters

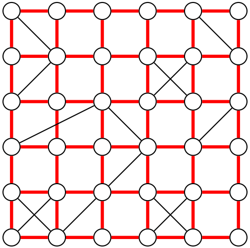
A graph class has bounded reduced X if all its members admit a contraction sequence whose red graphs have bounded X

Reduced parameters

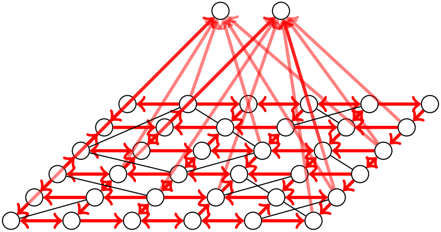
A graph class has bounded reduced X if all its members admit a contraction sequence whose red graphs have bounded X

red graphs have bounded ...	characterize bounded ...
degree	**twin-width**
component size	**cliquewidth** (sparse: treewidth)
number of edges*	linear cliquewidth (sparse: pathwidth)
outdegree	(oriented) twin-width
degree + treewidth	?
cutwidth	?
bandwidth	?

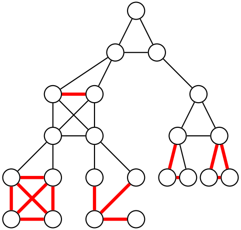
Different conditions imposed in the sequence of red graphs



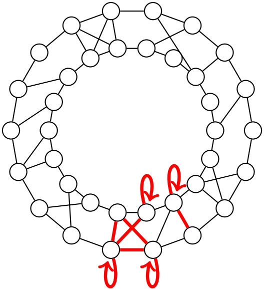
bd degree: defines bd twin-width



bd outdegree: defines bd oriented twin-width

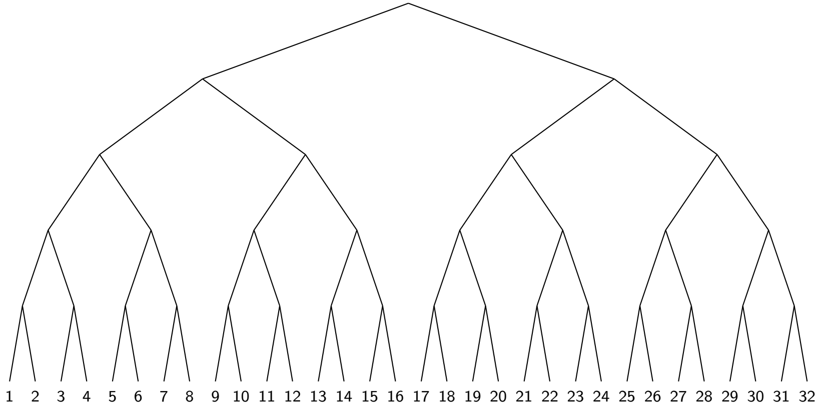


bd component: redefines bd cliquewidth



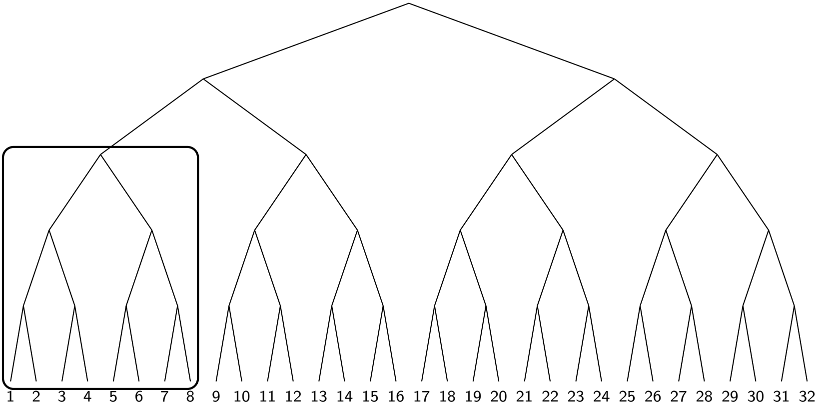
bd #edges: redefines bd linear cliquewidth

Bd boolean-width ⇒ bd component twin-width



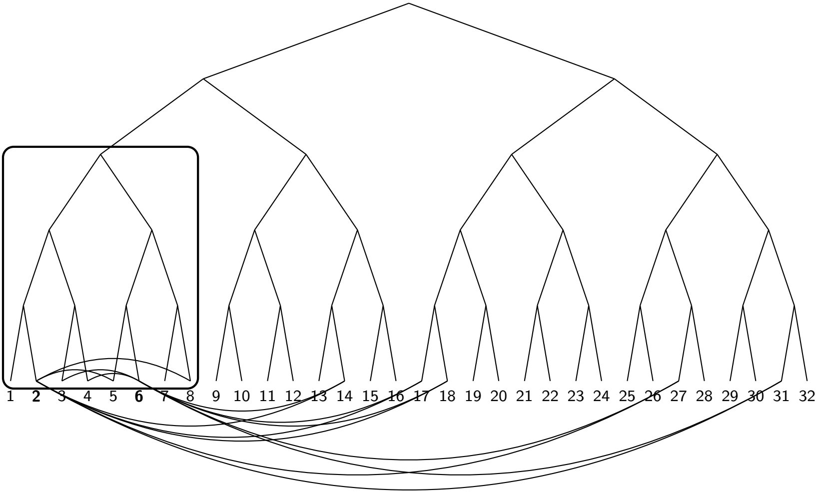
Bd boolean-width: binary tree layout s.t. every edge cut in the tree induces a bipartition with bd # distinct neighborhoods

Bd boolean-width ⇒ bd component twin-width



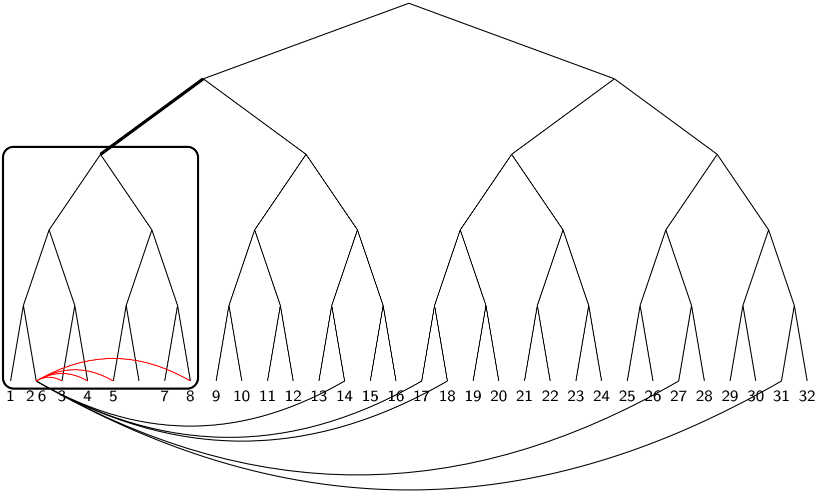
There is a subtree on $\ell \in [d+1, 2d]$ leaves, where d bounds the number of single-vertex neighborhoods in a bipartition

Bd boolean-width ⇒ bd component twin-width



Two vertices have the same neighborhood outside of this subtree

Bd boolean-width ⇒ bd component twin-width



Contracting them preserves the upper bound at $2d$
on the size of red connected components

Component twin-width and boolean-width are tied

Theorem (B., Kim, Reinald, Thomassé '22)

A class has bounded component twin-width iff it has bounded boolean-width/cliquewidth/rank-width.

Proof.

We just saw one direction.

Component twin-width and boolean-width are tied

Theorem (B., Kim, Reinald, Thomassé '22)

A class has bounded component twin-width iff it has bounded boolean-width/cliquewidth/rank-width.

Proof.

We just saw one direction.
Conversely, build the binary tree layout based on the contractions.
When red components merge, their subtree gets a same parent. □

Component twin-width and boolean-width are tied

Theorem (B., Kim, Reinald, Thomassé '22)

A class has bounded component twin-width iff it has bounded boolean-width/cliquewidth/rank-width.

Proof.

We just saw one direction.
Conversely, build the binary tree layout based on the contractions.
When red components merge, their subtree gets a same parent. □

Theorem (B., Kim, Reinald, Thomassé '22)

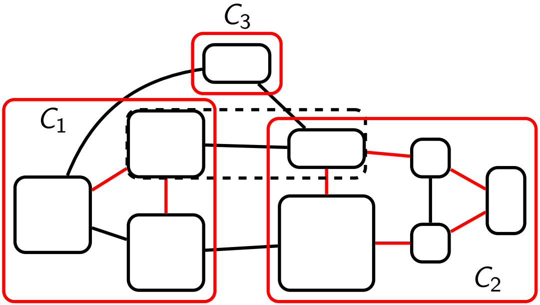
A class has bounded total twin-width iff it has bounded linear boolean-width/cliquewidth/rank-width.

Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph G with a contraction sequence s.t. all red graphs have components of size at most d

Is it easier to design algorithms via this characterization?

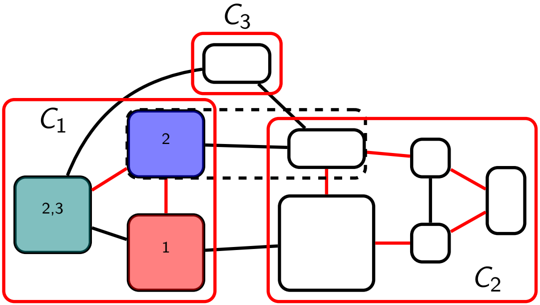
Solve 3-Coloring on a graph G with a contraction sequence s.t. all red graphs have components of size at most d



For every red component C keep every profile $V(C) \to 2^{\{1,2,3\}} \setminus \{\emptyset\}$ realizable by a proper 3-coloring of $G\langle C \rangle$

Is it easier to design algorithms via this characterization?

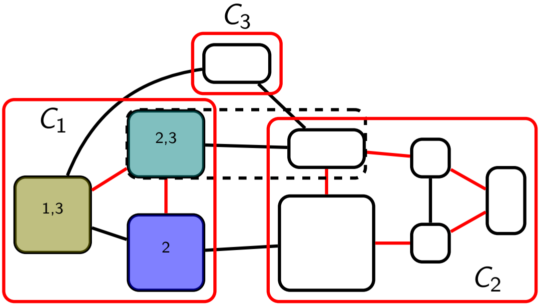
Solve 3-Coloring on a graph G with a contraction sequence s.t. all red graphs have components of size at most d



For every red component C keep every profile $V(C) \to 2^{\{1,2,3\}} \setminus \{\emptyset\}$ realizable by a proper 3-coloring of $G\langle C \rangle$

Is it easier to design algorithms via this characterization?

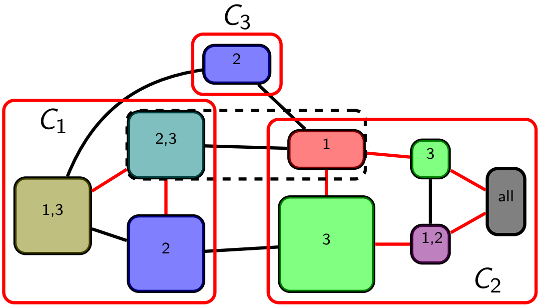
Solve 3-COLORING on a graph G with a contraction sequence s.t. all red graphs have components of size at most d



For every red component C keep every profile $V(C) \to 2^{\{1,2,3\}} \setminus \{\emptyset\}$ realizable by a proper 3-coloring of $G\langle C \rangle$

Is it easier to design algorithms via this characterization?

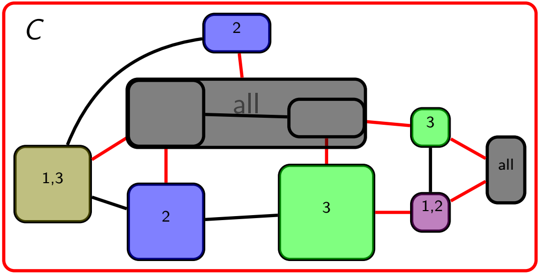
Solve 3-Coloring on a graph G with a contraction sequence s.t. all red graphs have components of size at most d



Some tuples of the at most $d+1$ profiles corresponding to merging red components are compatible

Is it easier to design algorithms via this characterization?

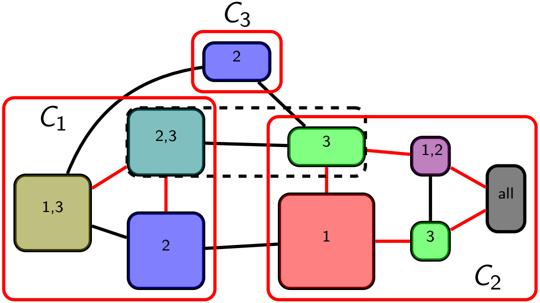
Solve 3-Coloring on a graph G with a contraction sequence s.t.
all red graphs have components of size at most d



Some tuples of the at most $d + 1$ profiles
corresponding to merging red components are compatible

Is it easier to design algorithms via this characterization?

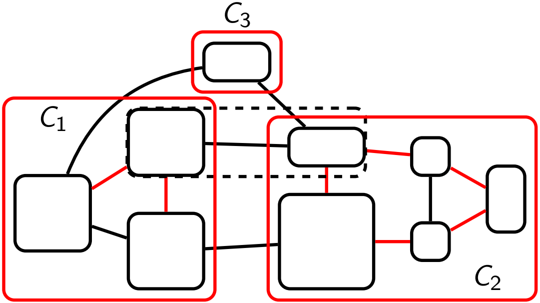
Solve 3-Coloring on a graph G with a contraction sequence s.t. all red graphs have components of size at most d



Some tuples of the at most $d + 1$ profiles corresponding to merging red components are incompatible

Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph G with a contraction sequence s.t. all red graphs have components of size at most d



Initialization: time $3n$
Update: time $7^d d^2$
Total: time $7^d d^2 n$
End: still a profile on the single vertex *containing* the whole graph?

Formulas, sentences, and model checking

GRAPH FO/MSO MODEL CHECKING	**Parameter:** $|\varphi|$

Input: A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$
Question: $G \models \varphi$?

Formulas, sentences, and model checking

> Graph FO/MSO Model Checking **Parameter:** $|\varphi|$
> **Input:** A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$
> **Question:** $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} x = x_i \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i) \vee E(x_i, x)$$

$G \models \varphi$? $\Leftrightarrow$

Formulas, sentences, and model checking

GRAPH FO/MSO MODEL CHECKING **Parameter:** $|\varphi|$
Input: A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$
Question: $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \forall x \bigvee_{1 \leqslant i \leqslant k} x = x_i \vee \bigvee_{1 \leqslant i \leqslant k} E(x, x_i) \vee E(x_i, x)$$

$G \models \varphi$? $\Leftrightarrow$ k-DOMINATING SET

Formulas, sentences, and model checking

Graph FO/MSO Model Checking **Parameter:** $|\varphi|$
Input: A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$
Question: $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \bigwedge_{1 \leqslant i < j \leqslant k} \neg(x_i = x_j) \wedge \neg E(x_i, x_j) \wedge \neg E(x_j, x_i)$$

$G \models \varphi$? $\Leftrightarrow$

Formulas, sentences, and model checking

> GRAPH FO/MSO MODEL CHECKING **Parameter:** $|\varphi|$
> **Input:** A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$
> **Question:** $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \bigwedge_{1 \leqslant i < j \leqslant k} \neg(x_i = x_j) \wedge \neg E(x_i, x_j) \wedge \neg E(x_j, x_i)$$

$G \models \varphi$? $\Leftrightarrow$ k-INDEPENDENT SET

Formulas, sentences, and model checking

> GRAPH FO/MSO MODEL CHECKING **Parameter:** $|\varphi|$
> **Input:** A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$
> **Question:** $G \models \varphi$?

Example:

$$\varphi = \exists X_1 \exists X_2 \exists X_3 (\forall x \bigvee_{1 \leqslant i \leqslant 3} X_i(x)) \wedge \forall x \forall y \bigwedge_{1 \leqslant i \leqslant 3} (X_i(x) \wedge X_i(y) \rightarrow \neg E(x, y))$$

$G \models \varphi? \Leftrightarrow$

Formulas, sentences, and model checking

Graph FO/MSO Model Checking	**Parameter:** $\lvert\varphi\rvert$
Input: A graph G and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$	
Question: $G \models \varphi$?	

Example:

$$\varphi = \exists X_1 \exists X_2 \exists X_3 (\forall x \bigvee_{1 \leqslant i \leqslant 3} X_i(x)) \wedge \forall x \forall y \bigwedge_{1 \leqslant i \leqslant 3} (X_i(x) \wedge X_i(y) \rightarrow \neg E(x, y))$$

$G \models \varphi$? $\Leftrightarrow$ 3-Coloring

Courcelle's theorems

We will reprove with contraction sequences:

Theorem (Courcelle, Makowsky, Rotics '00)

MSO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ *given a witness that the clique-width/component twin-width of the input* G *is at most* d.

generalizes

Theorem (Courcelle '90)

MSO model checking can be solved in time $f(|\varphi|, t) \cdot |V(G)|$ *on graphs* G *of treewidth at most* t.

Courcelle's theorems

We will reprove with contraction sequences:

Theorem (Courcelle, Makowsky, Rotics '00)

MSO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ given a witness that the clique-width/component twin-width of the input G is at most d.

generalizes

Theorem (Courcelle '90)

MSO model checking can be solved in time $f(|\varphi|, t) \cdot |V(G)|$ on graphs G of treewidth at most t.

Instead of maintaining all the possible profiles of 3-colorings, mantain all the sentences of quantifier depth $\leqslant q$ satisfied by a red component!

Rank-k m-types

Sets of non-equivalent formulas/sentences of quantifier rank at most k satisfied by a fixed structure:

$$\mathrm{tp}_k^{\mathcal{L}}(\mathscr{A}, \vec{a} \in A^m) = \{\varphi(\vec{x}) \in \mathcal{L}[k] : \mathscr{A} \models \varphi(\vec{a})\},$$

$$\mathrm{tp}_k^{\mathcal{L}}(\mathscr{A}) = \{\varphi \in \mathcal{L}[k] : \mathscr{A} \models \varphi\}.$$

Rank-k m-types

Sets of non-equivalent formulas/sentences of quantifier rank at most k satisfied by a fixed structure:

$$\mathsf{tp}_k^{\mathcal{L}}(\mathscr{A}, \vec{a} \in A^m) = \{\varphi(\vec{x}) \in \mathcal{L}[k] : \mathscr{A} \models \varphi(\vec{a})\},$$

$$\mathsf{tp}_k^{\mathcal{L}}(\mathscr{A}) = \{\varphi \in \mathcal{L}[k] : \mathscr{A} \models \varphi\}.$$

Theorem (folklore)

For $\mathcal{L} \in \{FO, MSO\}$, the number of rank-$k$ m-types is bounded by a function of k and m only.

Proof.

"$\mathcal{L}[k+1]$ are Boolean combinations of $\exists x \mathcal{L}[k]$." □

Rank-k m-types

Sets of non-equivalent formulas/sentences of quantifier rank at most k satisfied by a fixed structure:

$$\mathrm{tp}_k^{\mathcal{L}}(\mathscr{A}, \vec{a} \in A^m) = \{\varphi(\vec{x}) \in \mathcal{L}[k] : \mathscr{A} \models \varphi(\vec{a})\},$$

$$\mathrm{tp}_k^{\mathcal{L}}(\mathscr{A}) = \{\varphi \in \mathcal{L}[k] : \mathscr{A} \models \varphi\}.$$

Theorem (folklore)

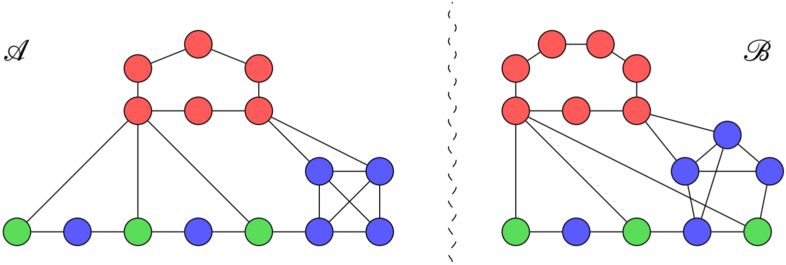
For $\mathcal{L} \in \{FO, MSO\}$, the number of rank-$k$ m-types is bounded by a function of k and m only.

Proof.

"$\mathcal{L}[k+1]$ are Boolean combinations of $\exists x \mathcal{L}[k]$." □

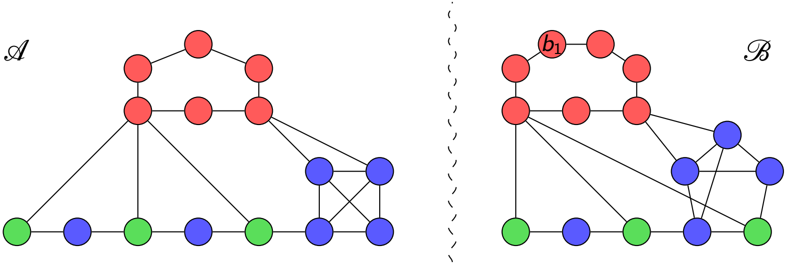
Rank-k types partition the graphs into $g(k)$ classes.
Efficient Model Checking = quickly finding the class of the input.

FO Ehrenfeucht-Fraissé game



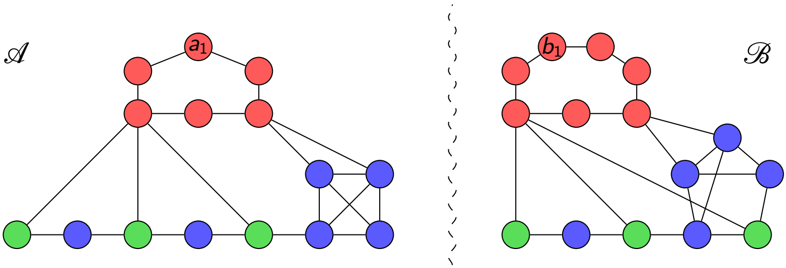
2-player game on two σ-structures $\mathscr{A}, \mathscr{B}$ (for us, colored graphs)

FO Ehrenfeucht-Fraissé game



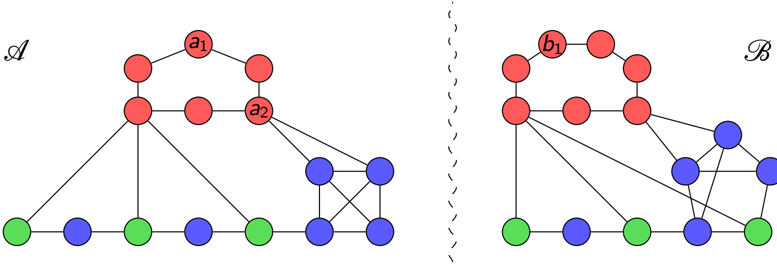
At each round, Spoiler picks a structure ($\mathscr{B}$) and a vertex therein

FO Ehrenfeucht-Fraissé game



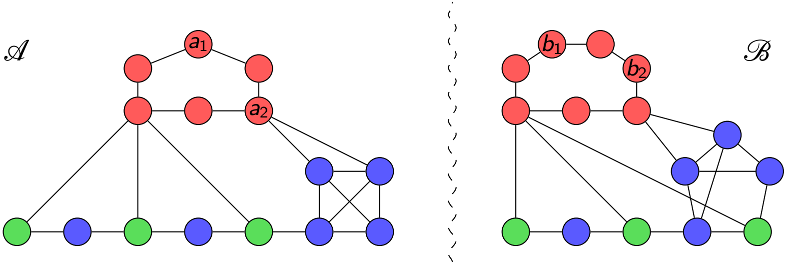
Duplicator answers with a vertex in the other structure

FO Ehrenfeucht-Fraissé game



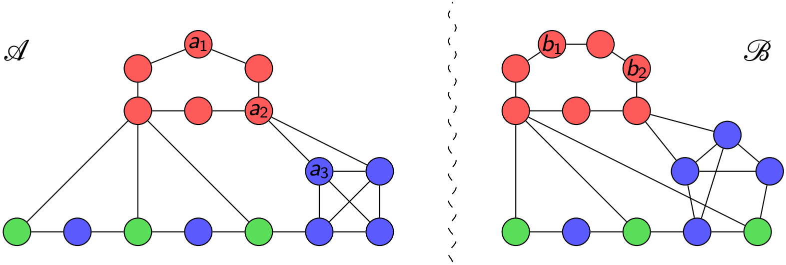
After q rounds, Duplicator wishes that $a_i \mapsto b_i$ is an isomorphism between $\mathscr{A}[a_1, \ldots, a_k]$ and $\mathscr{B}[b_1, \ldots, b_k]$

FO Ehrenfeucht-Fraissé game



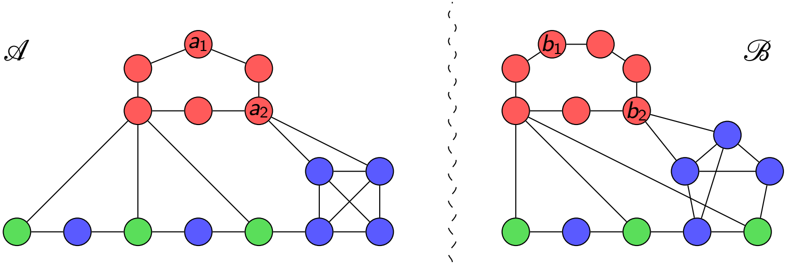
After q rounds, Duplicator wishes that $a_i \mapsto b_i$ is an isomorphism between $\mathscr{A}[a_1, \ldots, a_k]$ and $\mathscr{B}[b_1, \ldots, b_k]$

FO Ehrenfeucht-Fraissé game



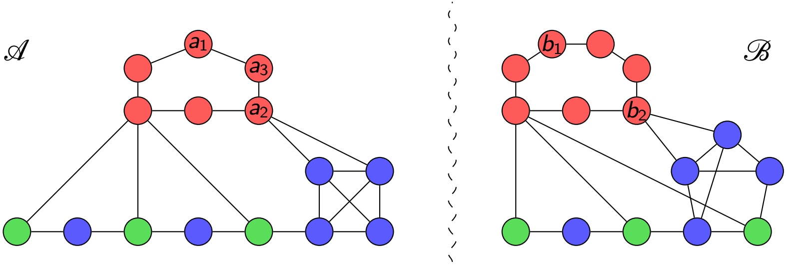
When no longer possible, Spoiler wins

FO Ehrenfeucht-Fraissé game



When no longer possible, Spoiler wins

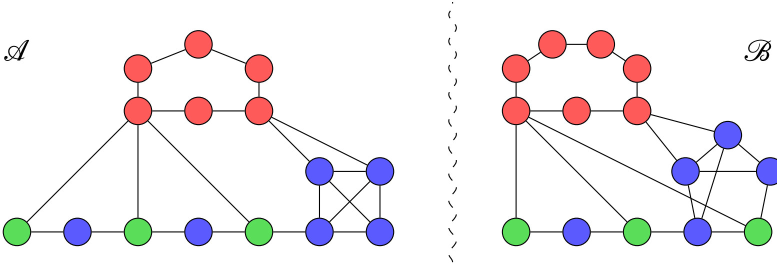
FO Ehrenfeucht-Fraissé game



If Duplicator can survive k rounds, we write $\mathscr{A} \equiv_k^{\mathsf{FO}} \mathscr{B}$

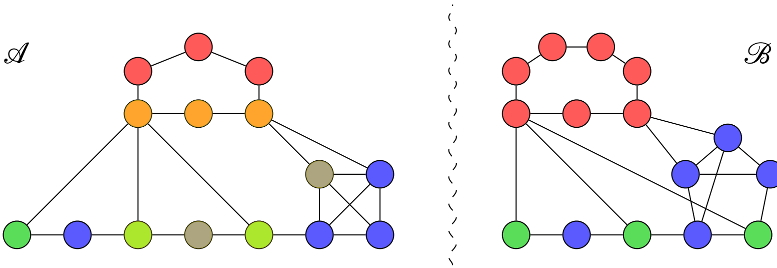
Here $\mathscr{A} \equiv_2^{\mathsf{FO}} \mathscr{B}$ and $\mathscr{A} \not\equiv_3^{\mathsf{FO}} \mathscr{B}$

MSO Ehrenfeucht-Fraissé game



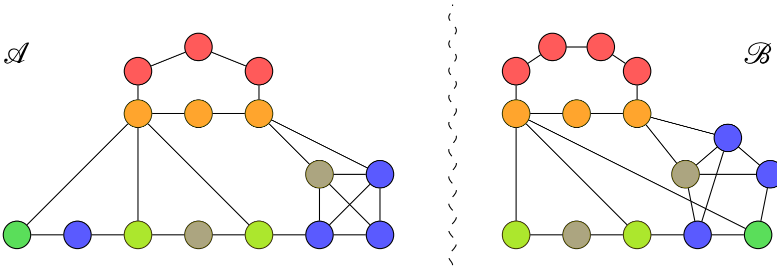
Same game but Spoiler can now play set moves

MSO Ehrenfeucht-Fraissé game



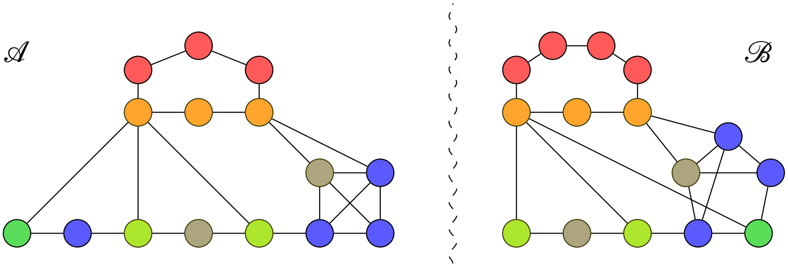
Same game but Spoiler can now play set moves

MSO Ehrenfeucht-Fraissé game



To which Duplicator answers a set in the other structure

MSO Ehrenfeucht-Fraissé game



Again we write $\mathscr{A} \equiv_k^{\mathsf{MSO}} \mathscr{B}$ if Duplicator can survive k rounds

k-round EF games capture rank-k types

Theorem (Ehrenfeucht-Fraissé)

For every σ-structures $\mathscr{A}, \mathscr{B}$ and logic $\mathcal{L} \in \{FO, MSO\}$,

$$\mathscr{A} \equiv_k^{\mathcal{L}} \mathscr{B} \text{ if and only if } tp_k^{\mathcal{L}}(\mathscr{A}) = tp_k^{\mathcal{L}}(\mathscr{B}).$$

k-round EF games capture rank-k types

Theorem (Ehrenfeucht-Fraissé)

For every σ-structures $\mathscr{A}, \mathscr{B}$ and logic $\mathcal{L} \in \{FO, MSO\}$,

$$\mathscr{A} \equiv_k^{\mathcal{L}} \mathscr{B} \text{ if and only if } tp_k^{\mathcal{L}}(\mathscr{A}) = tp_k^{\mathcal{L}}(\mathscr{B}).$$

Proof.

Induction on k.

($\Rightarrow$) $\mathcal{L}[k+1]$ formulas are Boolean combinations of $\exists x \varphi$ or $\exists X \varphi$ where $\varphi \in \mathcal{L}[k]$. Use the answer of Duplicator to $x = a$ or $X = A$.

k-round EF games capture rank-k types

Theorem (Ehrenfeucht-Fraissé)

For every σ-structures $\mathscr{A}, \mathscr{B}$ and logic $\mathcal{L} \in \{FO, MSO\}$,

$$\mathscr{A} \equiv_k^{\mathcal{L}} \mathscr{B} \text{ if and only if } tp_k^{\mathcal{L}}(\mathscr{A}) = tp_k^{\mathcal{L}}(\mathscr{B}).$$

Proof.

Induction on k.

($\Rightarrow$) $\mathcal{L}[k+1]$ formulas are Boolean combinations of $\exists x \varphi$ or $\exists X \varphi$ where $\varphi \in \mathcal{L}[k]$. Use the answer of Duplicator to $x = a$ or $X = A$.

($\Leftarrow$) If $\text{tp}_{k+1}^{\mathcal{L}}(\mathcal{A}) = \text{tp}_{k+1}^{\mathcal{L}}(\mathcal{B})$, then the type $\text{tp}_k^{\mathcal{L}}(\mathcal{A}, a)$ is equal to some $\text{tp}_k^{\mathcal{L}}(\mathcal{B}, b)$. Move a can be answered by playing b. □

MSO model checking for component twin-width d

Partitioned sentences: sentences on $(E, U_1, \ldots, U_d)$-structures, interpreted as a graph vertex partitioned in d parts

Maintain for every red component C of every trigraph G_i

$$\text{tp}_k^{\text{MSO}}(G, \mathcal{P}_i, C) = \{\varphi \in \text{MSO}_{E,U_1,\ldots,U_d}(k) : (G\langle C\rangle, \mathcal{P}_i\langle C\rangle) \models \varphi\}.$$

MSO model checking for component twin-width d

Partitioned sentences: sentences on $(E, U_1, \ldots, U_d)$-structures, interpreted as a graph vertex partitioned in d parts

Maintain for every red component C of every trigraph G_i

$$\text{tp}_k^{\text{MSO}}(G, \mathcal{P}_i, C) = \{\varphi \in \text{MSO}_{E,U_1,\ldots,U_d}(k) : (G\langle C\rangle, \mathcal{P}_i\langle C\rangle) \models \varphi\}.$$

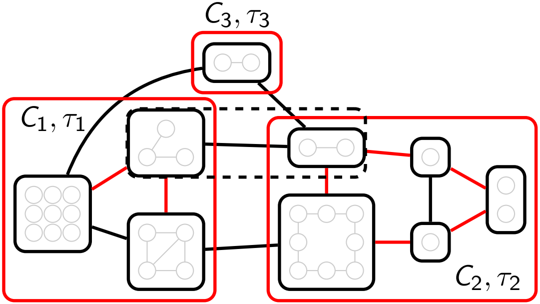
For each $v \in V(G)$, $\text{tp}_k(G, \mathcal{P}_n, \{v\}) =$ type of K_1
$\text{tp}_k(G, \mathcal{P}_1, \{V(G)\}) =$ type of G

MSO model checking for component twin-width d

Partitioned sentences: sentences on $(E, U_1, \ldots, U_d)$-structures, interpreted as a graph vertex partitioned in d parts

Maintain for every red component C of every trigraph G_i

$$\text{tp}_k^{\text{MSO}}(G, \mathcal{P}_i, C) = \{\varphi \in \text{MSO}_{E,U_1,\ldots,U_d}(k) : (G\langle C\rangle, \mathcal{P}_i\langle C\rangle) \models \varphi\}.$$



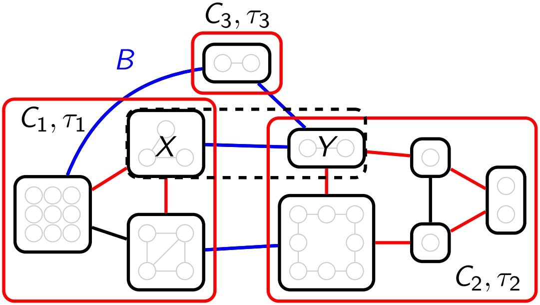
$\tau = \text{tp}_k^{\text{MSO}}(G, \mathcal{P}_i, C)$ based on the $\tau_j = \text{tp}_k^{\text{MSO}}(G, \mathcal{P}_{i+1}, C_j)$?

MSO model checking for component twin-width d

Partitioned sentences: sentences on $(E, U_1, \ldots, U_d)$-structures, interpreted as a graph vertex partitioned in d parts

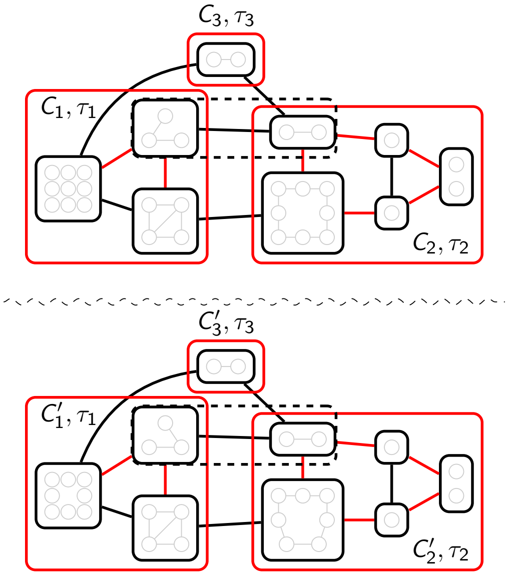
Maintain for every red component C of every trigraph G_i

$$\mathrm{tp}_k^{\mathrm{MSO}}(G, \mathcal{P}_i, C) = \{\varphi \in \mathrm{MSO}_{E,U_1,\ldots,U_d}(k) : (G\langle C\rangle, \mathcal{P}_i\langle C\rangle) \models \varphi\}.$$



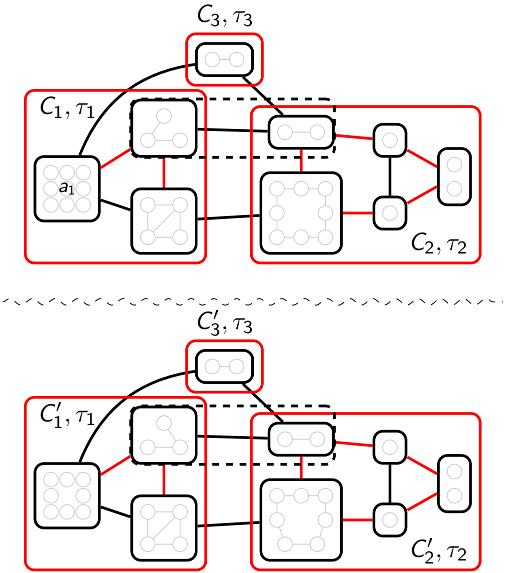
C arises from $C_1, \ldots, C_{d'}$: $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



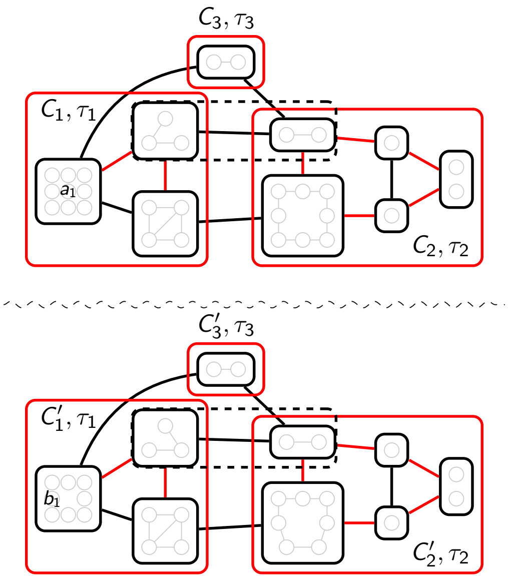
Duplicator combines her strategies in the red components

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



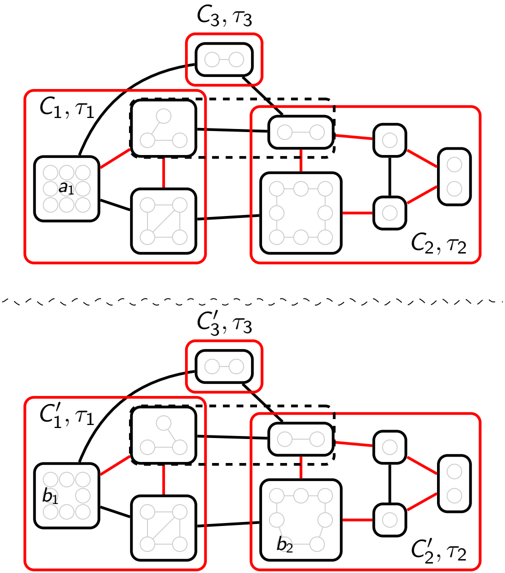
If Spoiler plays a vertex in the component of type τ_1,

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



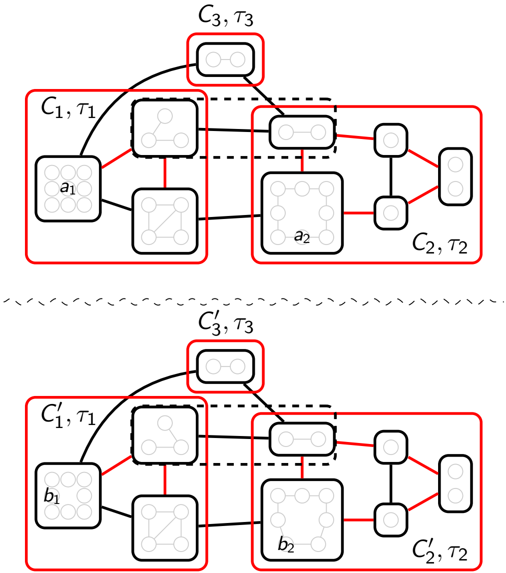
Duplicator answers the corresponding winning move

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



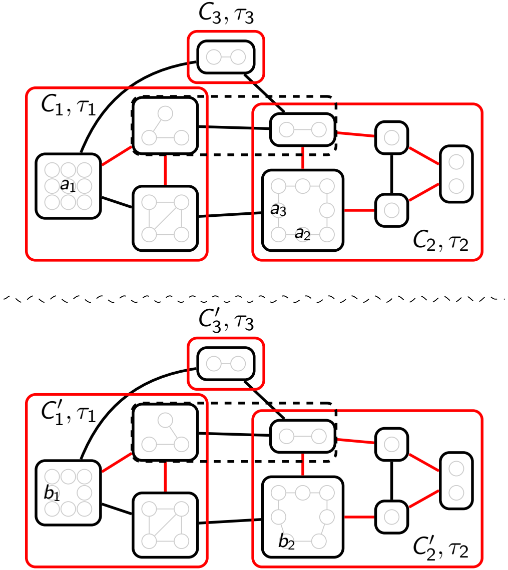
Same in the component of type τ_2

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



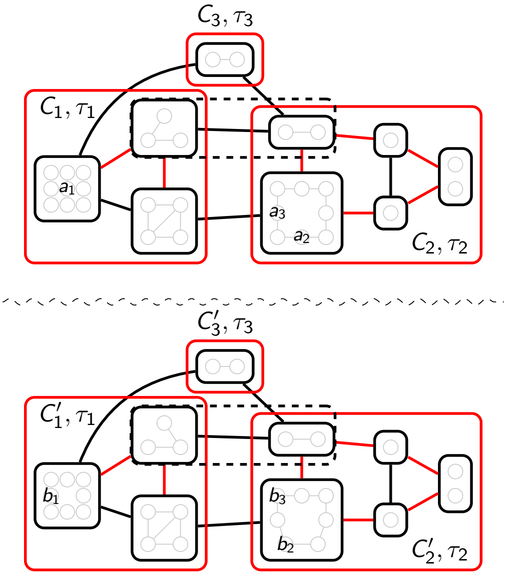
Same in the component of type τ_2

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



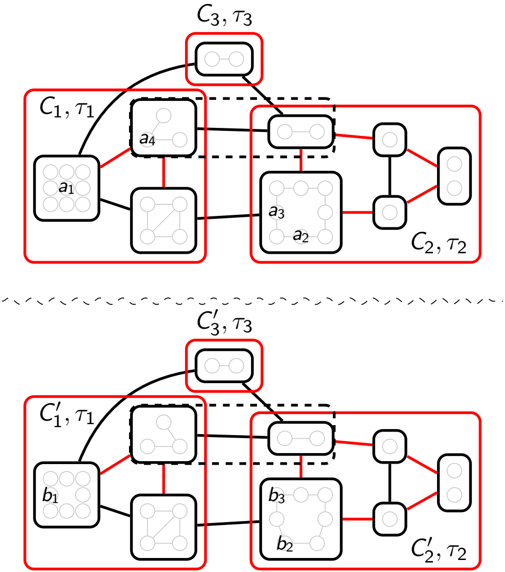
Same in the component of type τ_2

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



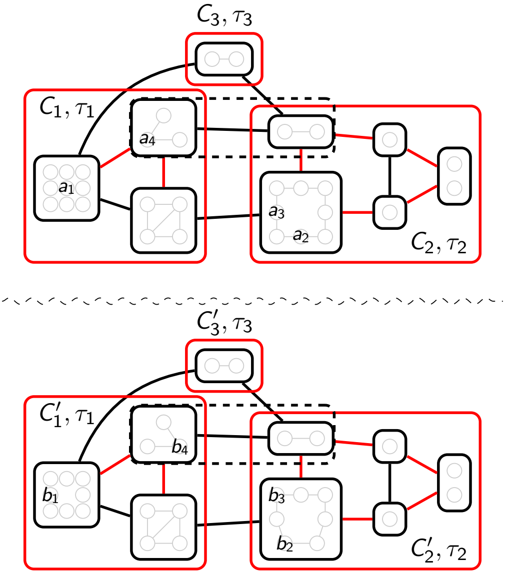
Same in the component of type τ_2

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



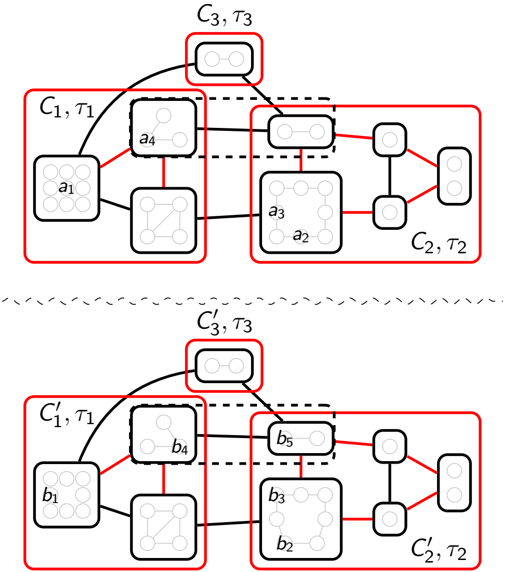
and so on

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



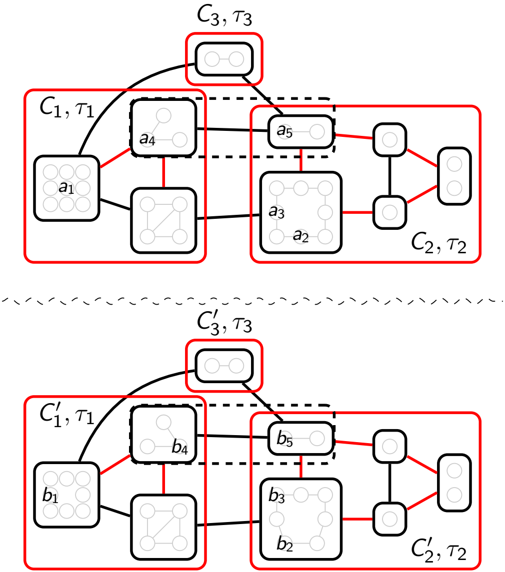
and so on

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



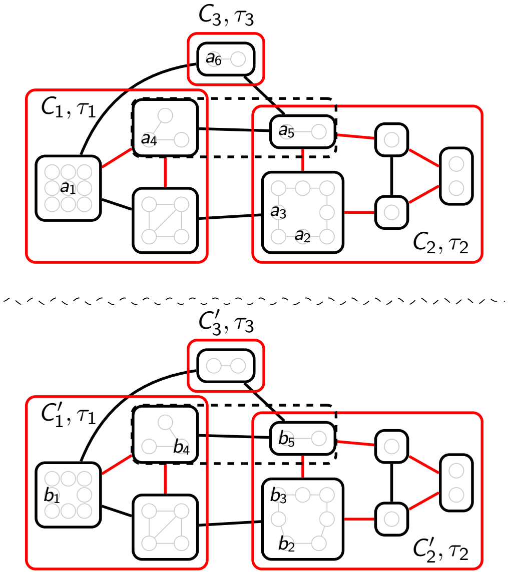
and so on

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



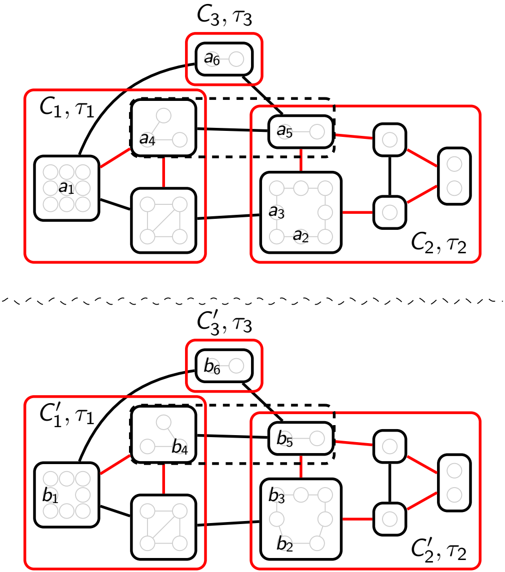
and so on

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



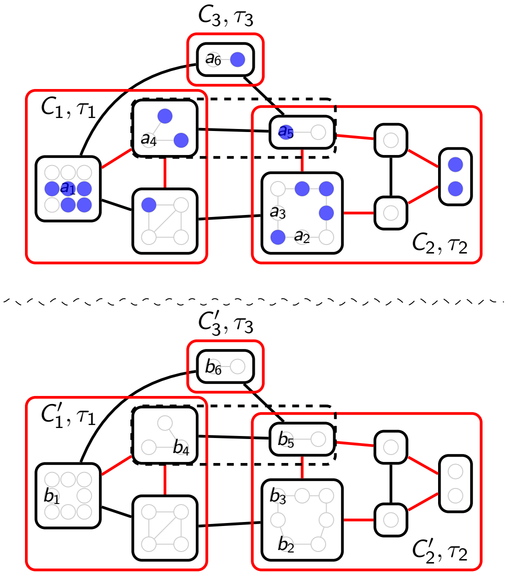
and so on

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



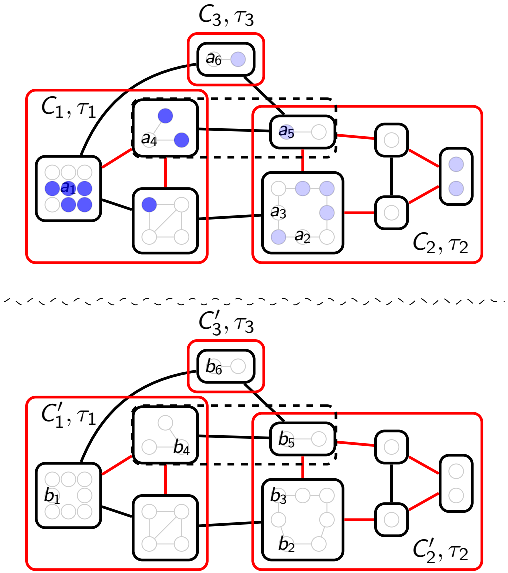
and so on

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



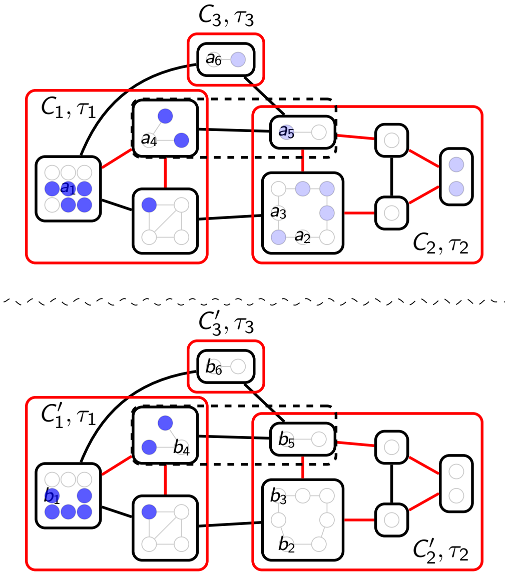
If Spoiler plays a set, Duplicator looks at the intersection with C_1,

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



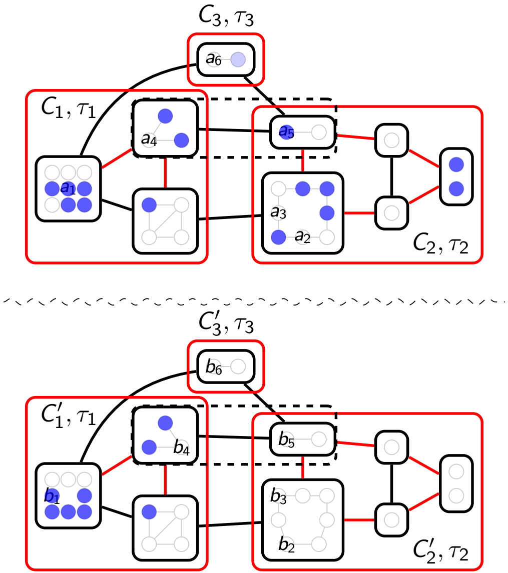
If Spoiler plays a set, Duplicator looks at the intersection with C_1,

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



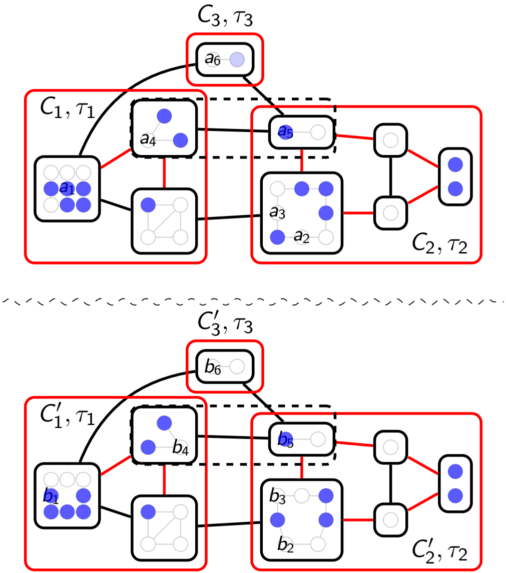
calls her winning strategy in C'_1

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



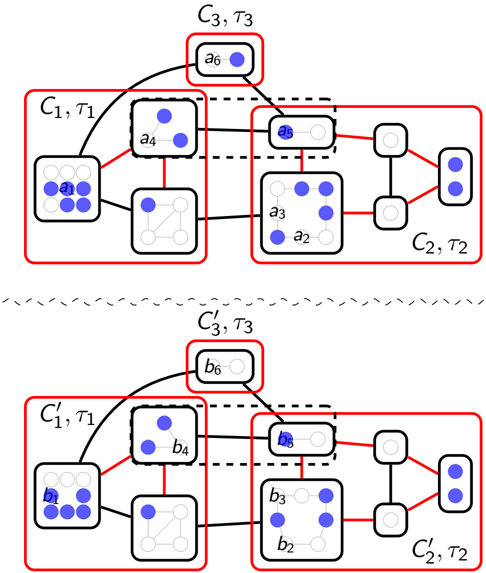
same for the other components

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



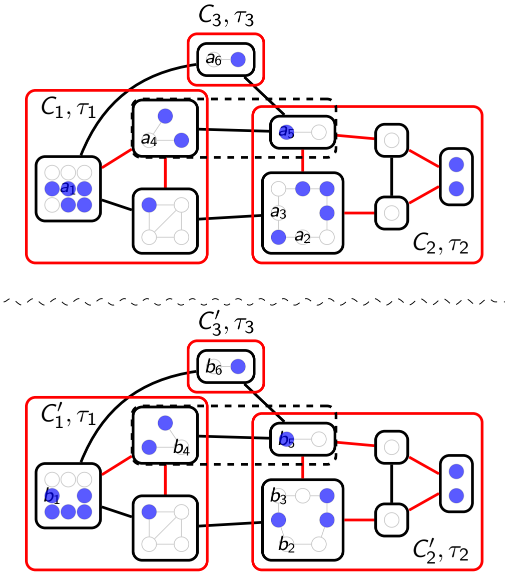
same for the other components

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



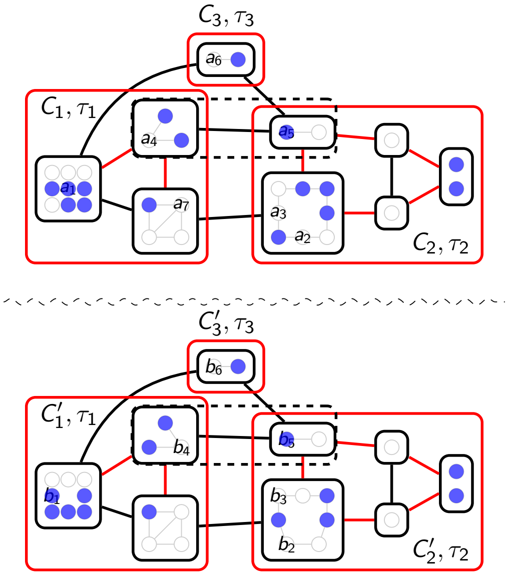
same for the other components

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



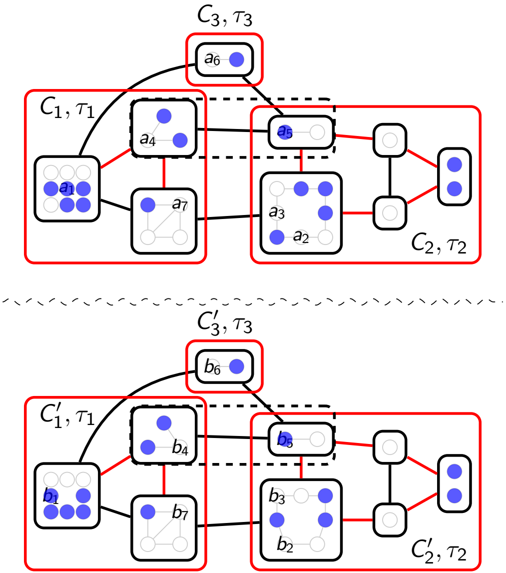
and plays the union

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



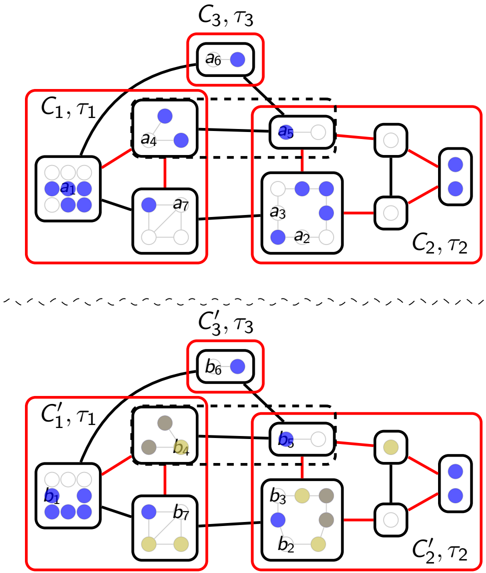
that fully defines the winning strategy of Duplicator

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



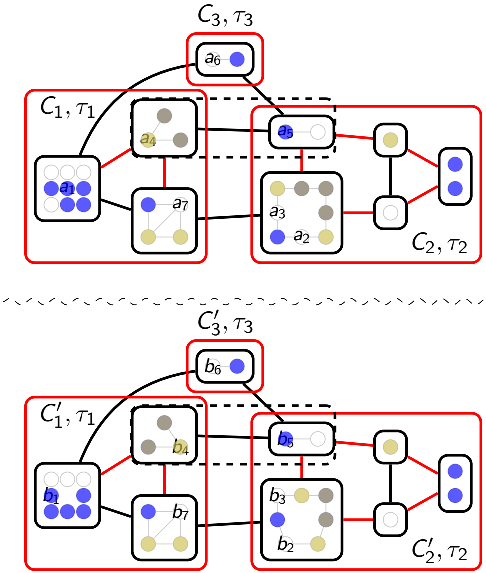
that fully defines the winning strategy of Duplicator

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



that fully defines the winning strategy of Duplicator

Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game



that fully defines the winning strategy of Duplicator

Turning it into a uniform algorithm

Reminder:

- #non-equivalent partitioned sentences of rank k: $f(d,k)$
- #rank-k partitioned types bounded by $g(d,k) = 2^{f(d,k)}$

For each newly observed type τ,

- keep a representative $(H,\mathcal{P})_\tau$ on at most $(d+1)^{g(d,k)}$ vertices
- determine the $0,1$-vector of satisfied sentences on $(H,\mathcal{P})_\tau$
- record the value of $F(\tau_1,\ldots,\tau_{d'},B,X,Y)$ for future uses

Turning it into a uniform algorithm

Reminder:

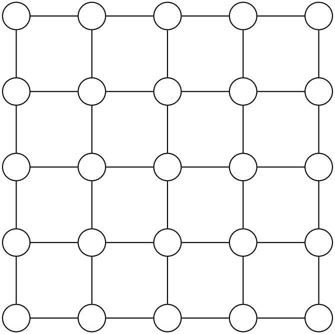
- $\#$non-equivalent partitioned sentences of rank k: $f(d,k)$
- $\#$rank-k partitioned types bounded by $g(d,k) = 2^{f(d,k)}$

For each newly observed type τ,

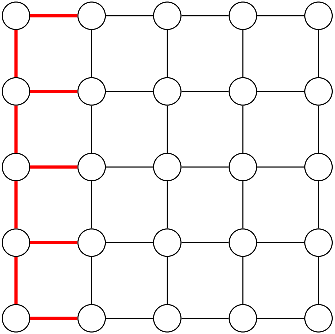
- keep a representative $(H,\mathcal{P})_\tau$ on at most $(d+1)^{g(d,k)}$ vertices
- determine the $0,1$-vector of satisfied sentences on $(H,\mathcal{P})_\tau$
- record the value of $F(\tau_1,\ldots,\tau_{d'},B,X,Y)$ for future uses

To decide $G \models \varphi$, look at position φ in the $0,1$-vector of $\mathrm{tp}_k^{\mathsf{MSO}}(G)$

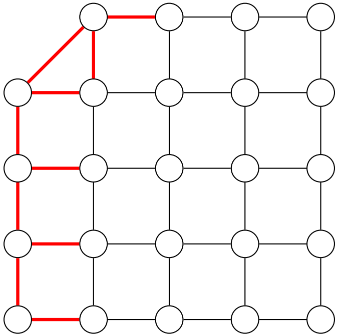
Twin-width is more general than the classic widths



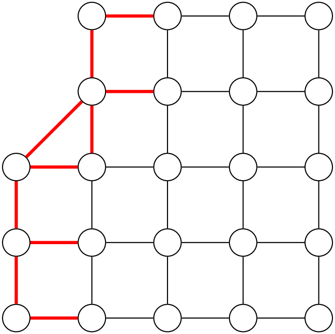
Twin-width is more general than the classic widths



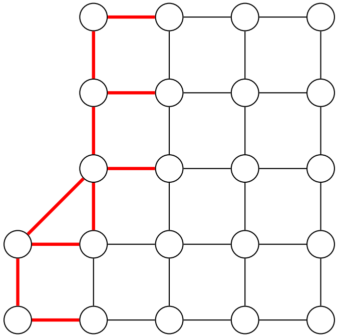
Twin-width is more general than the classic widths



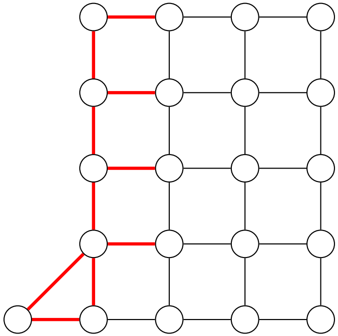
Twin-width is more general than the classic widths



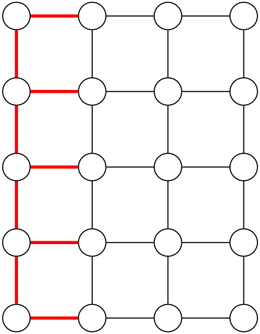
Twin-width is more general than the classic widths



Twin-width is more general than the classic widths

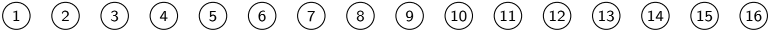


Twin-width is more general than the classic widths

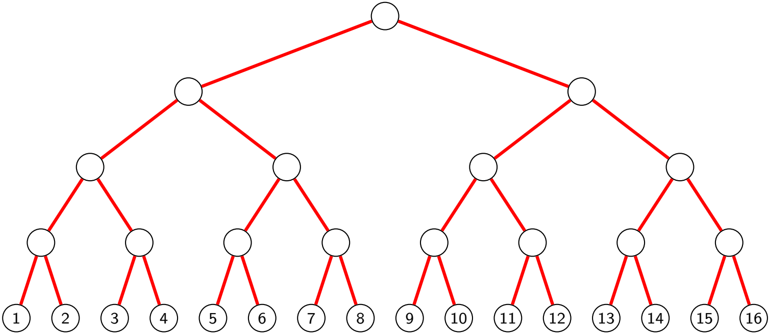


4-sequence for planar grids, but unbounded cliquewidth

($\geqslant 2\log n$)-subdivisions have twin-width at most 4

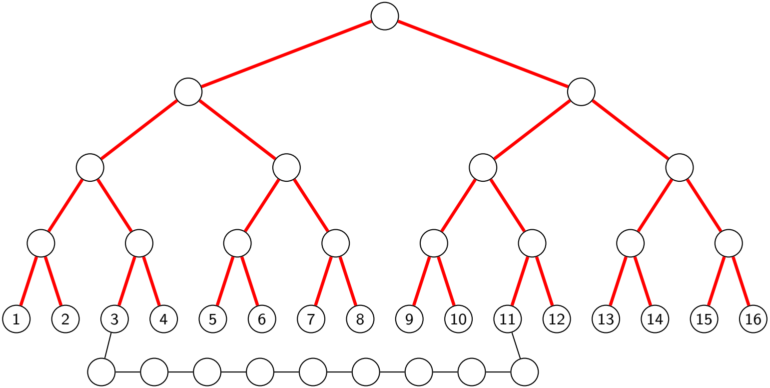


$(\geqslant 2 \log n)$-subdivisions have twin-width at most 4



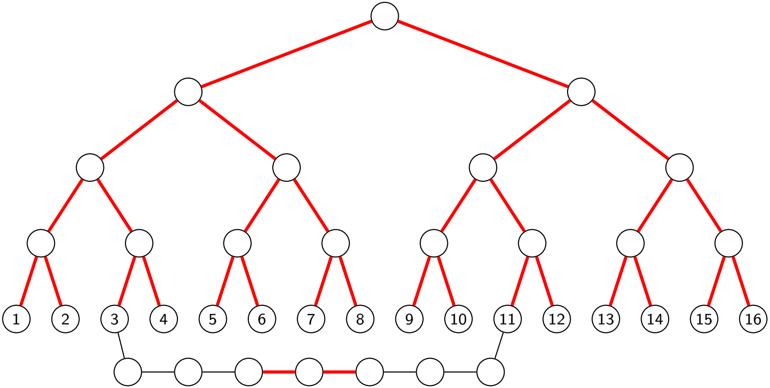
Add a red full binary tree whose leaves are the vertex set

($\geqslant 2\log n$)-subdivisions have twin-width at most 4



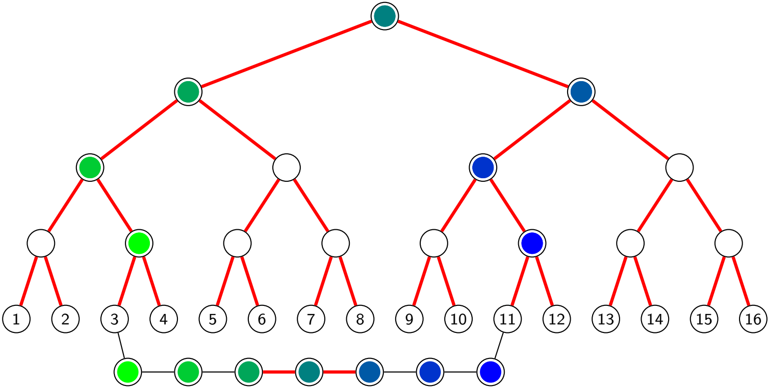
Take any subdivided edge

$(\geqslant 2 \log n)$-subdivisions have twin-width at most 4



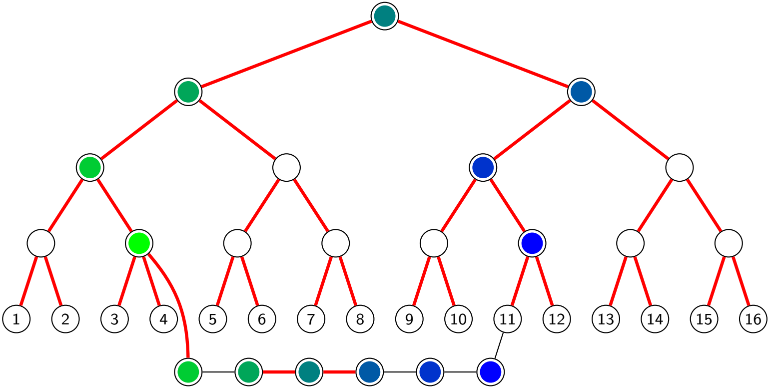
Shorten it to the length of the path in the red tree

$(\geqslant 2 \log n)$-subdivisions have twin-width at most 4



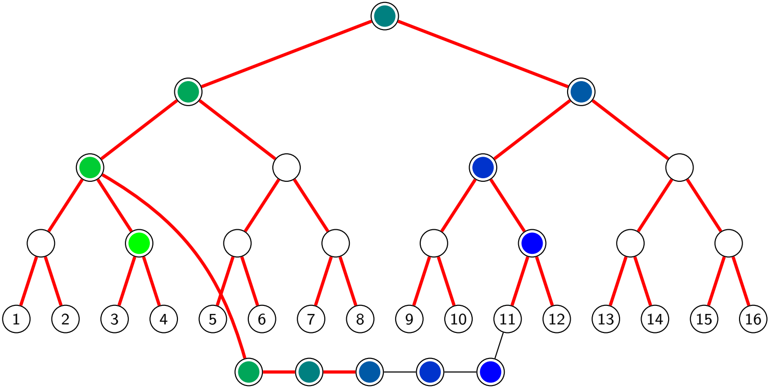
Zip the subdivided edge in the tree

($\geqslant 2\log n$)-subdivisions have twin-width at most 4



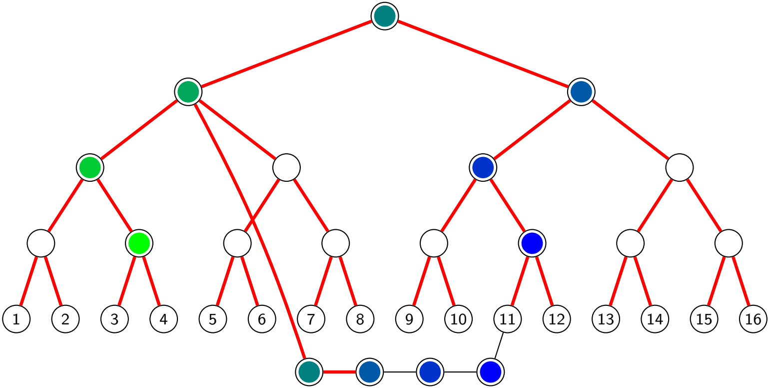
Zip the subdivided edge in the tree

($\geqslant 2 \log n$)-subdivisions have twin-width at most 4



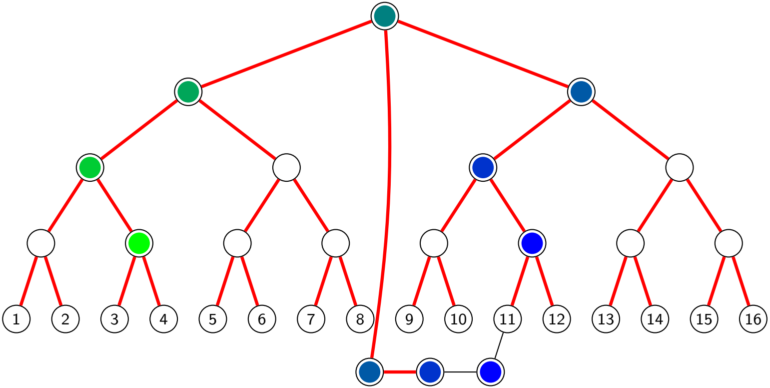
Zip the subdivided edge in the tree

($\geqslant 2\log n$)-subdivisions have twin-width at most 4



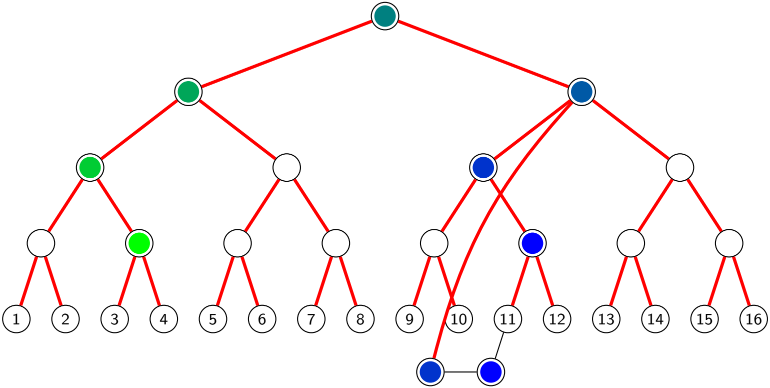
Zip the subdivided edge in the tree

$(\geqslant 2\log n)$-subdivisions have twin-width at most 4



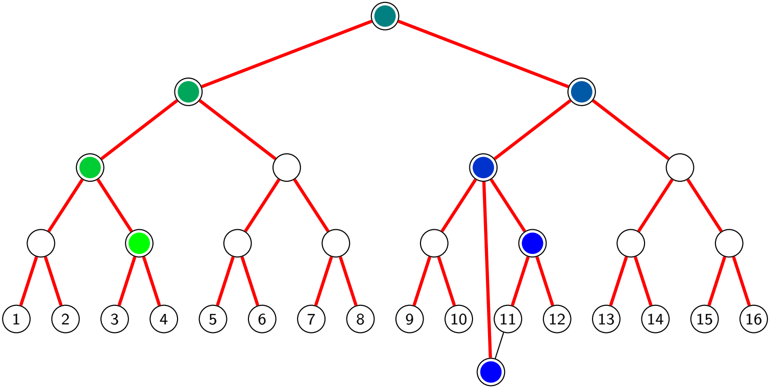
Zip the subdivided edge in the tree

($\geqslant 2 \log n$)-subdivisions have twin-width at most 4



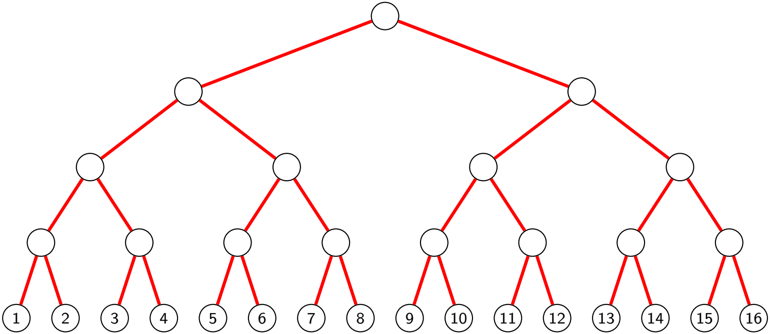
Zip the subdivided edge in the tree

($\geqslant 2 \log n$)-subdivisions have twin-width at most 4



Zip the subdivided edge in the tree

($\geqslant 2 \log n$)-subdivisions have twin-width at most 4



Move to the next subdivided edge also of unbounded cliquewidth

Theorem

The following classes have bounded twin-width, and $O(1)$-sequences can be computed in polynomial time.

- *Bounded rank-width, and even, boolean-width graphs,*
- *every hereditary proper subclass of permutation graphs,*
- *posets of bounded antichain size (seen as digraphs),*
- *unit interval graphs,*
- *K_t-minor free graphs,*
- *map graphs,*
- *subgraphs of d-dimensional grids,*
- *K_t-free unit d-dimensional ball graphs,*
- *$\Omega(\log n)$-subdivisions of all the n-vertex graphs,*
- *cubic expanders defined by iterative random 2-lifts from K_4,*
- *strong products of two bounded twin-width classes, one with bounded degree, etc.*

Theorem

The following classes have bounded twin-width, and $O(1)$-sequences can be computed in polynomial time.

- *Bounded rank-width, and even, boolean-width graphs,*
- *every hereditary proper subclass of permutation graphs,*
- *posets of bounded antichain size (seen as digraphs),*
- *unit interval graphs,*
- *K_t-minor free graphs,*
- *map graphs,*
- *subgraphs of d-dimensional grids,*
- *K_t-free unit d-dimensional ball graphs,*
- *$\Omega(\log n)$-subdivisions of all the n-vertex graphs,*
- *cubic expanders defined by iterative random 2-lifts from K_4,*
- *strong products of two bounded twin-width classes, one with bounded degree, etc.*

Can we solve problems faster, given an $O(1)$-sequence?

k-Independent Set given a $d = O(1)$-sequence

d-sequence: $G = G_n, G_{n-1}, \ldots, G_2, G_1 = K_1$

Algorithm: **For every connected subset D of size at most k of the red graph of every G_i, store in $T[D, i]$ one largest independent set in $G\langle D \rangle$ intersecting every vertex of D.**

k-Independent Set given a $d = O(1)$-sequence

d-sequence: $G = G_n, G_{n-1}, \ldots, G_2, G_1 = K_1$

Algorithm: **For every connected subset D of size at most k of the red graph of every G_i, store in $T[D, i]$ one largest independent set in $G\langle D \rangle$ intersecting every vertex of D.**

Initialization: $T[\{v\}, n] = \{v\}$

End: $T[\{V(G)\}, 1] =$ IS of size at least k or largest IS in G

Running time: $d^{2k}n^2$ red connected subgraphs,
actually only $d^{2k}n = 2^{O_d(k)}n$ updates

k-Independent Set given a $d = O(1)$-sequence

d-sequence: $G = G_n, G_{n-1}, \ldots, G_2, G_1 = K_1$

Algorithm: **For every connected subset D of size at most k of the red graph of every G_i, store in $T[D, i]$ one largest independent set in $G\langle D \rangle$ intersecting every vertex of D.**

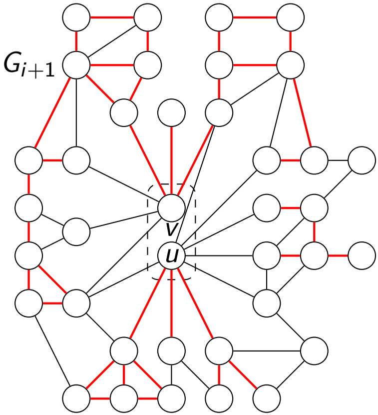
Initialization: $T[\{v\}, n] = \{v\}$

End: $T[\{V(G)\}, 1] =$ IS of size at least k or largest IS in G

Running time: $d^{2k}n^2$ red connected subgraphs,
actually only $d^{2k}n = 2^{O_d(k)}n$ updates

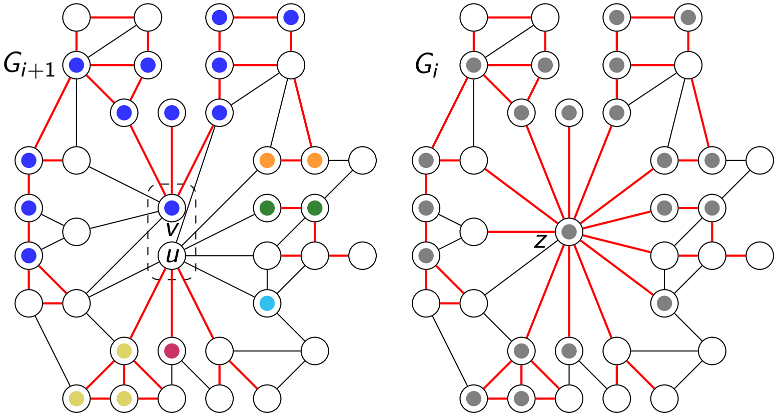
How to compute $T[D, i]$ from all the $T[D', i+1]$?

k-Independent Set: Update of partial solutions



Best partial solution inhabiting •?

k-Independent Set: Update of partial solutions



3 unions of $\leqslant d+2$ red connected subgraphs to consider in G_{i+1} with u, or v, or both

FO model checking on graphs of bounded twin-width

The previous algorithm generalizes to:

Theorem (B., Kim, Thomassé, Watrigant '20)

FO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ on graphs G given with a d-sequence.

FO model checking on graphs of bounded twin-width

The previous algorithm generalizes to:

Theorem (B., Kim, Thomassé, Watrigant '20)

FO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ *on graphs* G *given with a* d*-sequence.*

Add **Gaifman's locality of FO** to our MSO model checking algorithm

FO model checking on graphs of bounded twin-width

The previous algorithm generalizes to:

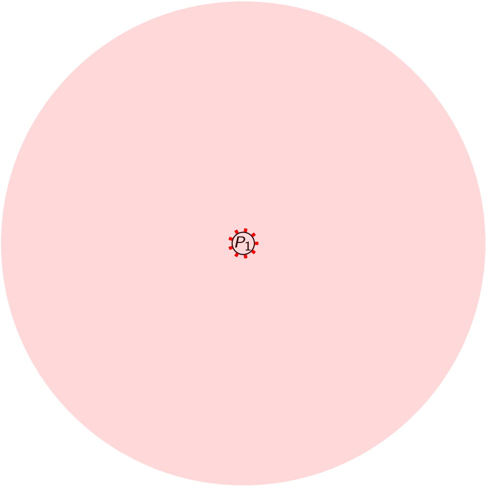
Theorem (B., Kim, Thomassé, Watrigant '20)

FO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ on graphs G given with a d-sequence.

Add **Gaifman's locality of FO** to our MSO model checking algorithm

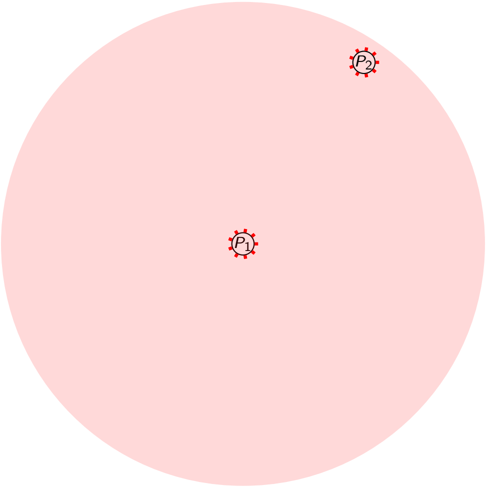
Thank you for your attention!

Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...

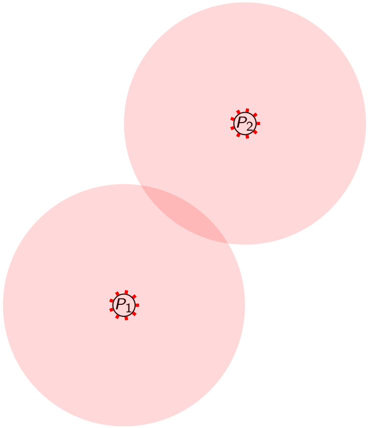
Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...

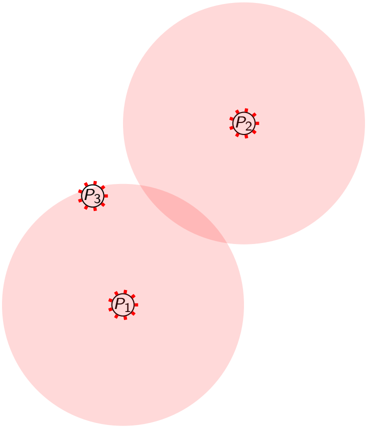
P_2 is at distance at most 2^{k-2} from $\{P_1\}$ in $(G, \mathcal{P}_i)$

Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...
P_2 is at distance at most 2^{k-2} from $\{P_1\}$ in $(G, \mathcal{P}_i)$

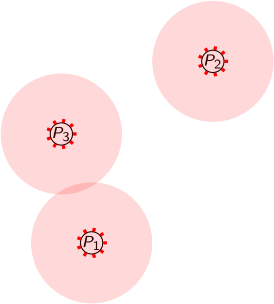
Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...

P_3 is at distance at most 2^{k-3} from $\{P_1, P_2\}$ in $(G, \mathcal{P}_i)$

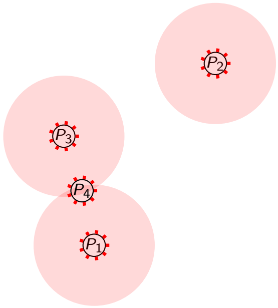
Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...

P_3 is at distance at most 2^{k-3} from $\{P_1, P_2\}$ in $(G, \mathcal{P}_i)$

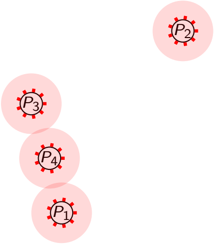
Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...

P_4 is at distance at most 2^{k-4} from $\{P_1, P_2, P_3\}$ in $(G, \mathcal{P}_i)$

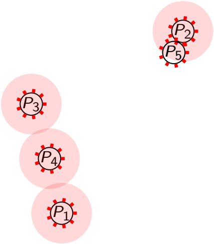
Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...

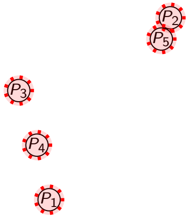
P_4 is at distance at most 2^{k-4} from $\{P_1, P_2, P_3\}$ in $(G, \mathcal{P}_i)$

Local tuple of parts



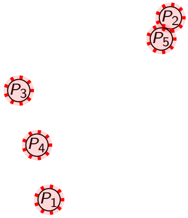
$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...
P_q is at distance at most 2^{k-q} from $\{P_1, \ldots, P_{q-1}\}$ in $(G, \mathcal{P}_i)$

Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...
P_q is at distance at most 2^{k-q} from $\{P_1, ..., P_{q-1}\}$ in $(G, \mathcal{P}_i)$

Local tuple of parts



$(P_1, P_2, \ldots, P_q)$ is *local around* P_1 if...
P_q is at distance at most 2^{k-q} from $\{P_1, ..., P_{q-1}\}$ in $(G, \mathcal{P}_i)$

Partitioned local sentences and types

A prenex sentence is *partitioned local around* X in $(G, \mathcal{P}_i)$ if of the form $Qx_1 \in X \; Qx_2 \in P_2 \; \ldots \; Qx_k \in P_k \; \psi(x_1, \ldots, x_k)$ with

- ψ is quantifier-free, and
- $(X, P_2, \ldots, P_k)$ local around X in $(G, \mathcal{P}_i)$.

Partitioned local sentences and types

A prenex sentence is *partitioned local around* X in $(G, \mathcal{P}_i)$ if of the form $Qx_1 \in X\ Qx_2 \in P_2\ \ldots\ Qx_k \in P_k\ \psi(x_1, \ldots, x_k)$ with

- ψ is quantifier-free, and
- $(X, P_2, \ldots, P_k)$ local around X in $(G, \mathcal{P}_i)$.

And the corresponding types:

$$\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \{\varphi : \mathsf{qr}(\varphi) \leqslant k,$$

$$\varphi \text{ is partitioned local around } X \text{ in } (G, \mathcal{P}_i),$$

$$(G, \mathcal{P}_i) \models \varphi\}.$$

Partitioned local sentences/types in $(G, \mathcal{P}_n)$ and $(G, \mathcal{P}_1)$

Initialization of the dynamic programming

In $(G, \mathcal{P}_n = \{\{v\} : v \in V(G)\})$: for every $v \in V(G)$,
$Qx_1 \in \{v\}\ Qx_2 \in \{v\}\ \ldots\ Qx_k \in \{v\}\ \psi \equiv \psi(v, v, \ldots, v)$

Partitioned local types are easy to compute in $(G, \mathcal{P}_n)$

Partitioned local sentences/types in $(G, \mathcal{P}_n)$ and $(G, \mathcal{P}_1)$

Initialization of the dynamic programming

In $(G, \mathcal{P}_n = \{\{v\} : v \in V(G)\})$: for every $v \in V(G)$,
$Qx_1 \in \{v\}\ Qx_2 \in \{v\}\ \ldots\ Qx_k \in \{v\}\ \psi \equiv \psi(v, v, \ldots, v)$

Partitioned local types are easy to compute in $(G, \mathcal{P}_n)$

Output of the dynamic programming

In $(G, \mathcal{P}_1 = \{V(G)\})$:
$Qx_1 \in V(G)\ Qx_2 \in V(G)\ \ldots\ Qx_k \in V(G)\ \psi \equiv$ classic sentences

The partitioned local type in $(G, \mathcal{P}_1)$ coincides with the type of G

Partitioned local types give the partitioned types

Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \mathsf{ltp}_k^{\mathsf{FO}}(G', \mathcal{P}'_i, f(X))$

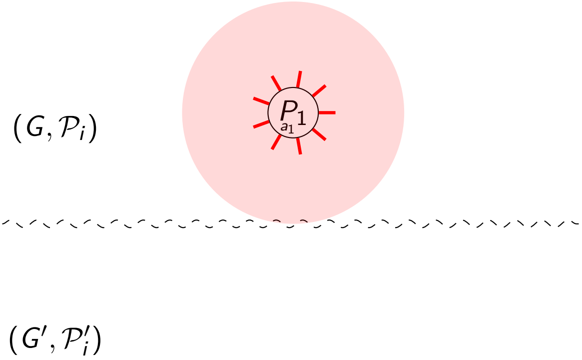
$(G, \mathcal{P}_i)$

$(G', \mathcal{P}'_i)$

Local strategies win the global game

Partitioned local types give the partitioned types

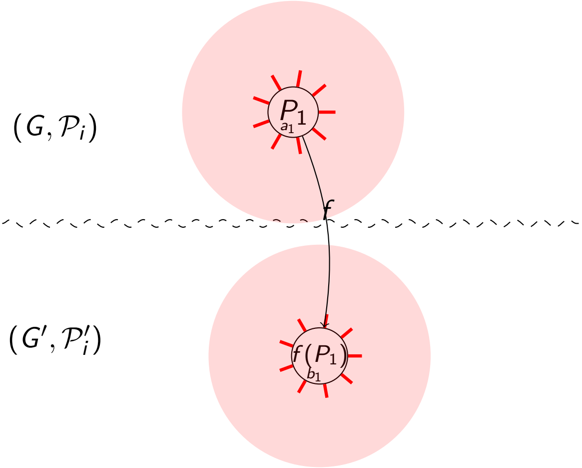
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \mathsf{ltp}_k^{\mathsf{FO}}(G', \mathcal{P}'_i, f(X))$



Say, Spoiler plays in P_1

Partitioned local types give the partitioned types

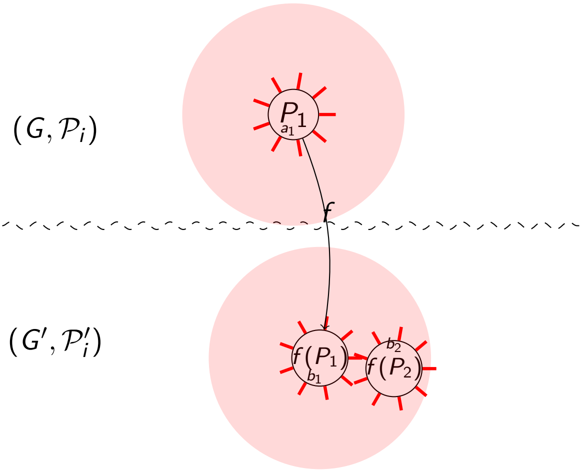
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \mathsf{ltp}_k^{\mathsf{FO}}(G', \mathcal{P}'_i, f(X))$



Duplicator answers in $f(P_1)$ following the local game around P_1

Partitioned local types give the partitioned types

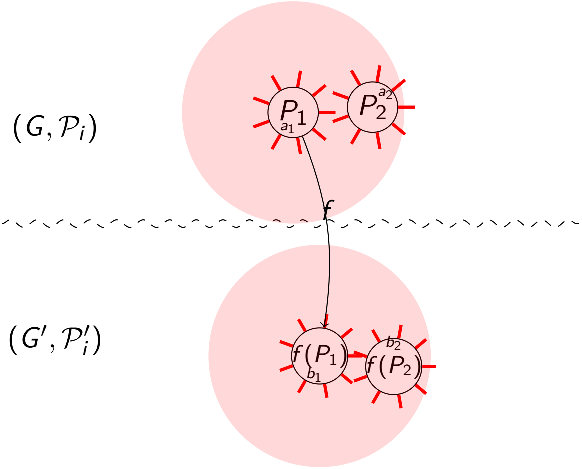
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \mathsf{ltp}_k^{\mathsf{FO}}(G', \mathcal{P}'_i, f(X))$



Now when Spoiler plays close to P_1 or $f(P_1)$

Partitioned local types give the partitioned types

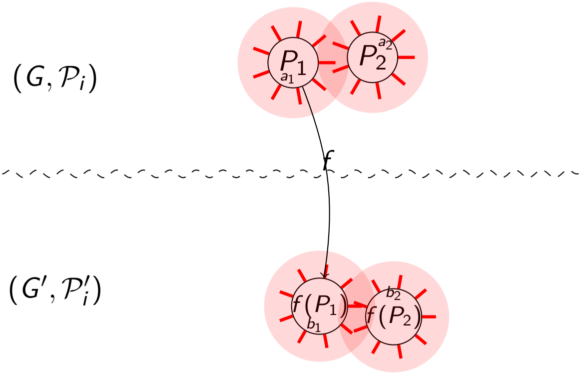
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathrm{ltp}_k^{\mathrm{FO}}(G, \mathcal{P}_i, X) = \mathrm{ltp}_k^{\mathrm{FO}}(G', \mathcal{P}'_i, f(X))$



Duplicator follows the winning local strategy

Partitioned local types give the partitioned types

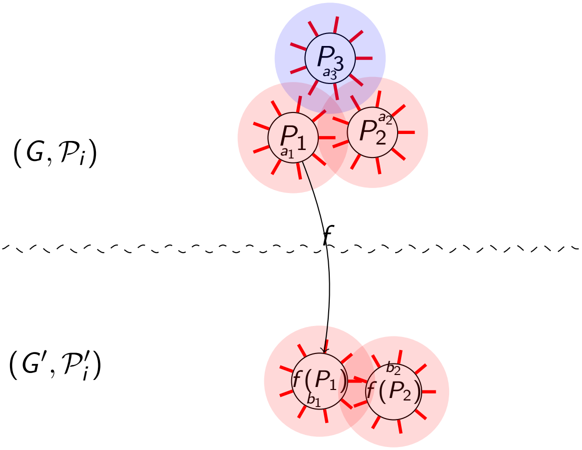
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathrm{ltp}_k^{\mathrm{FO}}(G, \mathcal{P}_i, X) = \mathrm{ltp}_k^{\mathrm{FO}}(G', \mathcal{P}'_i, f(X))$



Duplicator follows the winning local strategy

Partitioned local types give the partitioned types

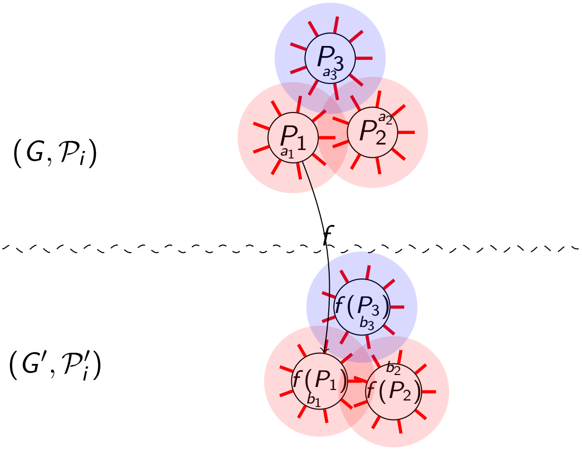
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \mathsf{ltp}_k^{\mathsf{FO}}(G', \mathcal{P}'_i, f(X))$



If Spoiler plays too far

Partitioned local types give the partitioned types

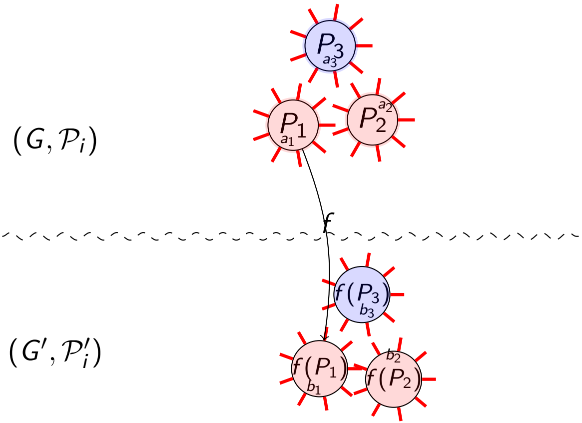
Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathsf{ltp}_k^{\mathsf{FO}}(G, \mathcal{P}_i, X) = \mathsf{ltp}_k^{\mathsf{FO}}(G', \mathcal{P}'_i, f(X))$



Duplicator starts a new local game around that new part

Partitioned local types give the partitioned types

Isom. $f : \mathcal{P}_i \to \mathcal{P}'_i$ with $\mathrm{ltp}_k^{\mathrm{FO}}(G, \mathcal{P}_i, X) = \mathrm{ltp}_k^{\mathrm{FO}}(G', \mathcal{P}'_i, f(X))$



Duplicator starts a new local game around that new part

Concluding as in the MSO model checking algorithm

$(G, \mathcal{P}_{i+1}) \rightsquigarrow (G, \mathcal{P}_i)$: X and Y are merged in Z

Partitioned local types around P

- only needs an update if P is at distance at most 2^{k-1} from Z

Concluding as in the MSO model checking algorithm

$(G, \mathcal{P}_{i+1}) \rightsquigarrow (G, \mathcal{P}_i)$: X and Y are merged in Z

Partitioned local types around P

- only needs an update if P is at distance at most 2^{k-1} from Z
- update only involves parts at distance at most 2^{k-1} from P

Concluding as in the MSO model checking algorithm

$(G, \mathcal{P}_{i+1}) \rightsquigarrow (G, \mathcal{P}_i)$: X and Y are merged in Z

Partitioned local types around P

- only needs an update if P is at distance at most 2^{k-1} from Z
- update only involves parts at distance at most 2^{k-1} from P
- hence at most d^{2^k} parts: conclude like MSO model checking

Concluding as in the MSO model checking algorithm

$(G, \mathcal{P}_{i+1}) \rightsquigarrow (G, \mathcal{P}_i)$: X and Y are merged in Z

Partitioned local types around P

- only needs an update if P is at distance at most 2^{k-1} from Z
- update only involves parts at distance at most 2^{k-1} from P
- hence at most d^{2^k} parts: conclude like MSO model checking

Each contraction: $O_{d,k}(1) = O(d^{2^k})$ updates in $O_{d,k}(1) = f(d, k)$
Total time: $O_{d,k}(n)$