Model Checking Disjoint-Paths Logic
on Topological-Minor-Free Graph Classes

Giannos Stamoulis

LIRMM, Univ Montpellier, CNRS, Montpellier, France

Joint work with

Petr A. Golovach
Department of Informatics, University of Bergen, Norway

Dimitrios M. Thilikos
LIRMM, Univ Montpellier, CNRS, Montpellier, France

Nicole Schirrmacher
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Alexandre Vigny
University of Bremen, Germany

Seminaire Groupe Boreal, 16/05/2023

Algorithmic Meta-Theorems

Algorithmic Meta-Theorems (AMTs):

General conditions that imply the automatic derivation of efficient algorithms.

Algorithmic Meta-Theorems (AMTs):

General conditions that imply the automatic derivation of efficient algorithms.

"Algorithms that output algorithms"

Algorithmic Meta-Theorems (AMTs):

General conditions that imply the automatic derivation of efficient algorithms.

"Algorithms that output algorithms"

- Provide uniform explanation why problems are tractable.
- Establish general algorithmic techniques for solving them.

Algorithmic Meta-Theorems (AMTs):

General conditions that imply the automatic derivation of efficient algorithms.

"Algorithms that output algorithms"

- Provide uniform explanation why problems are tractable.
- Establish general algorithmic techniques for solving them.

Form of an **AMT**:

"All *problems* definable in a certain *logic* on a certain class of *structures* can be solved *efficiently*."

Algorithmic Meta-Theorems (AMTs):

General conditions that imply the automatic derivation of efficient algorithms.

"Algorithms that output algorithms"

- Provide uniform explanation why problems are tractable.
- Establish general algorithmic techniques for solving them.

Form of an **AMT**:

"All *problems* definable in a certain **logic** on a certain class of *structures* can be solved *efficiently*."

First-**O**rder Logic **(FO)**:

First-**O**rder Logic **(FO)**:

First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$

First-**O**rder Logic **(FO)**:

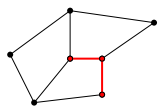
First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$

- *(Induced) Subgraph Containment.*

Does G contain H as a subgraph?

$\exists x \exists y \exists z \left(\mathbf{adj}(x, y) \wedge \mathbf{adj}(y, z) \wedge \neg \mathbf{adj}(x, z) \right)$



First-**O**rder Logic **(FO)**:

First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$

- *(Induced) Subgraph Containment.*

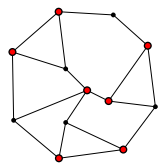
Does G contain H as a subgraph?

$\exists x \exists y \exists z \left(\mathbf{adj}(x, y) \wedge \mathbf{adj}(y, z) \wedge \neg \mathbf{adj}(x, z) \right)$

- *Vertex Cover.*

Does G contain a set S of k vertices that intersects all edges of G?

$\exists v_1 \ldots \exists v_k \quad \forall x \; \forall y \left(\mathbf{adj}(x, y) \rightarrow \bigvee_{i \in \{1, \ldots, k\}} (v_i = x \vee v_i = y) \right)$



First-**O**rder Logic **(FO)**:

First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$

- *(Induced) Subgraph Containment.*

Does G contain H as a subgraph?

$\exists x \exists y \exists z \left(\mathbf{adj}(x, y) \wedge \mathbf{adj}(y, z) \wedge \neg \mathbf{adj}(x, z) \right)$

- *Vertex Cover.*

Does G contain a set S of k vertices that intersects all edges of G?

$\exists v_1 \ldots \exists v_k \quad \forall x \; \forall y \left(\mathbf{adj}(x, y) \rightarrow \bigvee_{i \in \{1, \ldots, k\}} (v_i = x \vee v_i = y) \right)$

First-**O**rder Logic **(FO)**:

First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$

- *(Induced) Subgraph Containment.*

Does G contain H as a subgraph?

$\exists x \exists y \exists z \left(\mathbf{adj}(x, y) \wedge \mathbf{adj}(y, z) \wedge \neg \mathbf{adj}(x, z) \right)$

- *Vertex Cover.*

Does G contain a set S of k vertices that intersects all edges of G?

$\exists v_1 \ldots \exists v_k \quad \forall x \; \forall y \left(\mathbf{adj}(x, y) \rightarrow \bigvee_{i \in \{1, \ldots, k\}} (v_i = x \vee v_i = y) \right)$

- *Feedback Vertex Set.*

First-**O**rder Logic **(FO)**:

First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$

- *(Induced) Subgraph Containment.*

Does G contain H as a subgraph?

$\exists x \exists y \exists z \left(\mathbf{adj}(x, y) \wedge \mathbf{adj}(y, z) \wedge \neg \mathbf{adj}(x, z) \right)$

- *Vertex Cover.*

Does G contain a set S of k vertices that intersects all edges of G?

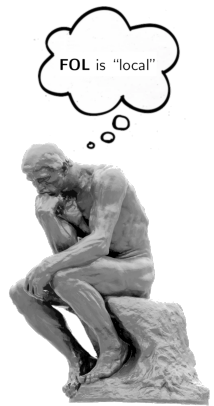
$\exists v_1 \ldots \exists v_k \quad \forall x \; \forall y \left(\mathbf{adj}(x, y) \rightarrow \bigvee_{i \in \{1, \ldots, k\}} (v_i = x \vee v_i = y) \right)$

- *Feedback Vertex Set.* ✗

First-**O**rder Logic **(FO)**:

First-order variables vertices $x, y, \ldots$

equality predicate	$x = y$	Combined with $\wedge$, $\vee$, $\neg$
adjacency predicate	$\mathbf{adj}(x, y)$	and quantification $\exists x$, $\forall x$



- *(Induced) Subgraph Containment.*

Does G contain H as a subgraph?

$\exists x \exists y \exists z \left(\mathbf{adj}(x, y) \wedge \mathbf{adj}(y, z) \wedge \neg \mathbf{adj}(x, z) \right)$

- *Vertex Cover.*

Does G contain a set S of k vertices that intersects all edges of G?

$\exists v_1 \ldots \exists v_k \quad \forall x \; \forall y \left(\mathbf{adj}(x, y) \rightarrow \bigvee_{i \in \{1, \ldots, k\}} (v_i = x \vee v_i = y) \right)$

- *Feedback Vertex Set.* ✗

Monadic **S**econd-**O**rder Logic (**MSO**):

Extend **FO** by allowing quantification over sets of vertices $\forall X, \exists X$.

Monadic **S**econd-**O**rder Logic (**MSO**):

Extend **FO** by allowing quantification over sets of vertices $\forall X, \exists X$.

- 3-*colorability*.

$$\exists V_1 \exists V_2 \exists V_3 \Big(\big(\forall x \, (x \in V_1 \vee x \in V_2 \vee x \in V_3) \big) \wedge \big(\forall x \forall y \, (x, y \in V_1) \vee (x, y \in V_2) \vee (x, y \in V_3) \implies \neg \mathbf{adj}(x, y) \big) \Big)$$

Monadic **S**econd-**O**rder Logic (**MSO**):

Extend **FO** by allowing quantification over sets of vertices $\forall X, \exists X$.

- 3-*colorability*.

$$\exists V_1 \exists V_2 \exists V_3 \left(\left(\forall x \, (x \in V_1 \vee x \in V_2 \vee x \in V_3) \right) \wedge \left(\forall x \forall y \, (x, y \in V_1) \vee (x, y \in V_2) \vee (x, y \in V_3) \implies \neg \mathbf{adj}(x, y) \right) \right)$$

- *Connectivity*.

$$\forall x \; \forall y \; \neg \left(\exists V \;\; \left(x \in V \;\wedge\; y \notin V \;\wedge \left(\forall u \; \forall v \; \mathbf{adj}(u, v) \implies (u \in V \Leftrightarrow v \in V) \right) \right) \right)$$

Model-checking

Complexity of model-checking.

Complexity of model-checking.

FO:
We can test whether an **FO**-formula φ is satisfied on a graph G ($G \models \varphi$), in time $|G|^{\mathcal{O}(|\varphi|)}$.

- Assuming ETH, we cannot test $G \models \varphi$ in time $|G|^{o(|\varphi|)}$ for general graphs.

Complexity of model-checking.

FO:
We can test whether an **FO**-formula φ is satisfied on a graph G ($G \models \varphi$), in time $|G|^{\mathcal{O}(|\varphi|)}$.

- Assuming ETH, we cannot test $G \models \varphi$ in time $|G|^{o(|\varphi|)}$ for general graphs.

MSO:
3-Colorability is NP-complete.

- Assuming $P \neq NP$, we cannot test $G \models \varphi$ in time $|G|^{f(|\varphi|)}$ for any function f.

Complexity of model-checking.

FO:
We can test whether an **FO**-formula φ is satisfied on a graph G ($G \models \varphi$), in time $|G|^{\mathcal{O}(|\varphi|)}$.

- Assuming ETH, we cannot test $G \models \varphi$ in time $|G|^{o(|\varphi|)}$ for general graphs.

MSO:
3-Colorability is NP-complete.

- Assuming $P \neq NP$, we cannot test $G \models \varphi$ in time $|G|^{f(|\varphi|)}$ for any function f.

- Our notion of efficiency:

Fixed-Parameter Tractability, i.e., algorithms running in time $f(|\varphi|) \cdot |G|^{\mathcal{O}(1)}$.

Form of an **AMT**:

"All *problems* definable in a certain *logic* on a certain class of *structures* can be solved *efficiently*."

Form of an **AMT**:

"All *problems* definable in a certain *logic* on a certain class of **structures** can be solved *efficiently*."

Form of an **AMT**:

"All *problems* definable in a certain *logic* on a certain class of **structures** can be solved *efficiently*."

Every $\mathbf{MSO}_2$ property can be tested in linear time on graphs of bounded treewidth.
[Courcelle, 1990]

Form of an **AMT**:

"All *problems* definable in a certain *logic* on a certain class of **structures** can be solved *efficiently*."

Every $\mathbf{MSO}_2$ property can be tested in linear time on graphs of bounded treewidth.
[Courcelle, 1990]

Every **FO** property can be tested in (almost) linear time on nowhere dense graph classes.
[Grohe, Kreutzer, & Siebertz, 2017]

(C)MSO$_1$: bounded cliquewidth. [Courcelle, Makowski, & Rotics, 2000] + [Oum & Seymour, 2006]

FO:

Sparse classes:

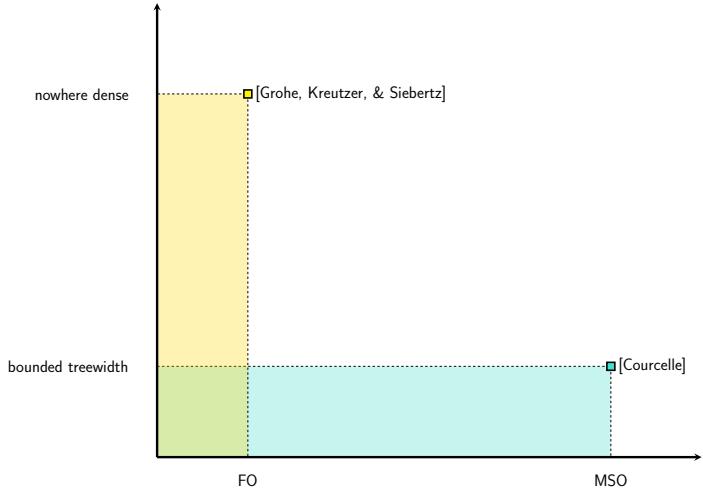
locally bounded treewidth [Frick & Grohe, 2001]
excluding a minor [Flum & Grohe, 2001]
bounded degree [Seese, 1996] locally excluding a minor [Dawar, Grohe, & Kreutzer, 2007]
bounded expansion [Dvořák, Kráľ, & Thomas, 2011]
nowhere dense [Grohe, Kreutzer, & Siebertz, 2017]

bounded twinwidth* [Bonnet, Kim, Thomassé, & Watrigant, 2022]
* given an identification sequence with the input.

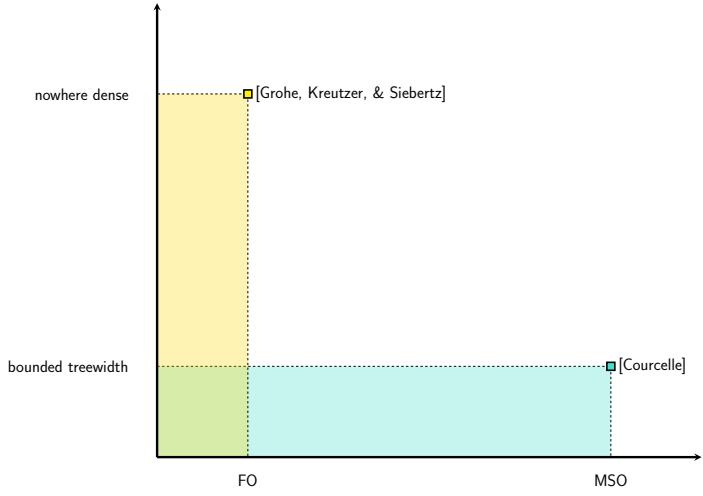
Dense classes:

structurally bounded degree [Gajarský, Hliněný, Lokshtanov, Obdržálek, & Ramanujan, 2016]
structurally bounded expansion [Gajarský, Kreutzer, Nešetřil, Ossona de Mendez, Pilipczuk, Siebertz, & Toruńczyk, 2018]
structurally nowhere dense [Dreier, Mählmann, Siebertz, 2023]

structurally bounded local cliquewidth [Bonnet, Dreier, Gajarský, Kreutzer, Mählmann, Simon, & Toruńczyk, 2022]

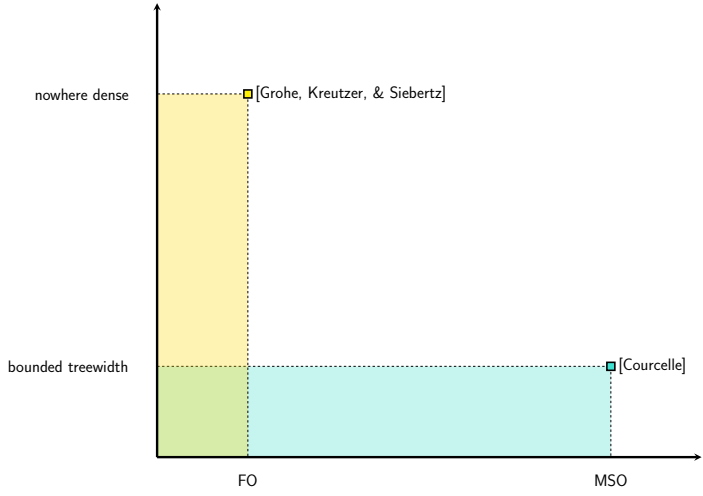


Assuming ETH:



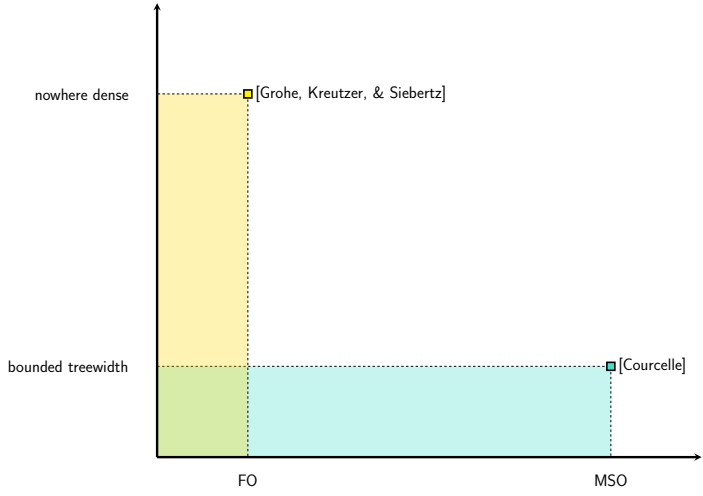
Assuming ETH:

- No FPT algorithm for **MSO** on unbounded treewidth classes.

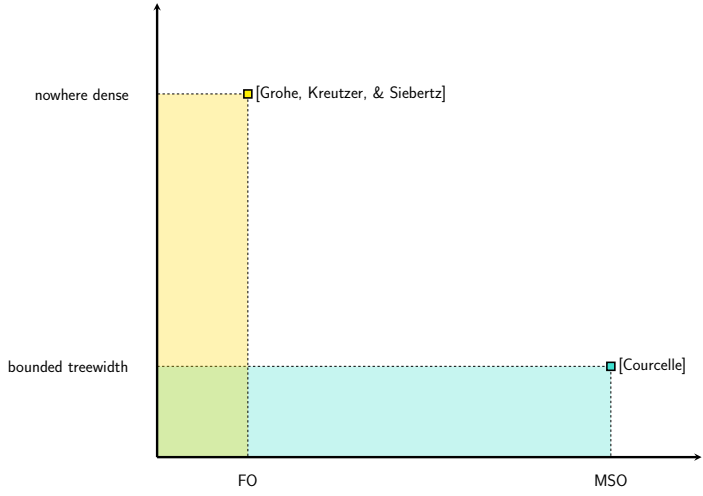


Assuming ETH:

- No FPT algorithm for **MSO** on unbounded treewidth classes.
- No FPT algorithm for **FO** on subgraph-closed somewhere dense classes.

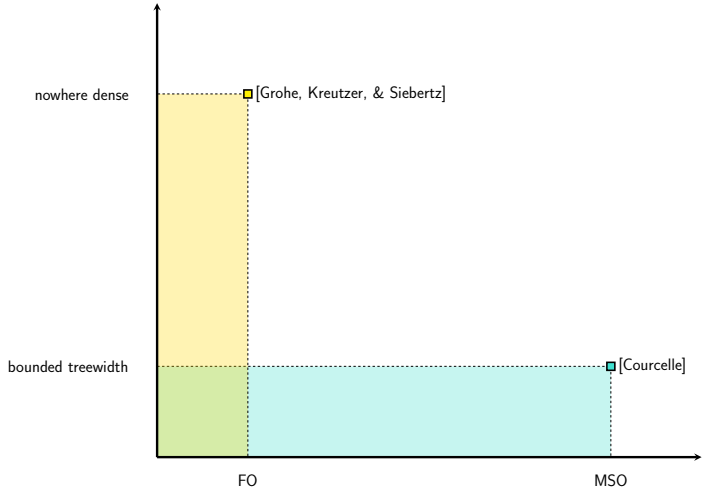


What to do next?



What to do next?

- *Meta-algorithmics* of **FO**: *counting predicates*, *transitive-closure operators*, *fixed-point operators*, *successor-invariant formulas*, *FO-interpretability*,...

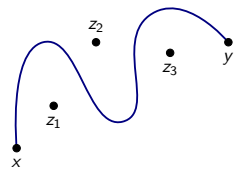


What to do next?

- *Meta-algorithmics* of **FO**: *counting predicates*, *transitive-closure operators*, *fixed-point operators*, *successor-invariant formulas*, *FO-interpretability*,...
- **Consider logics between FO and MSO**.

Separator Logic & (Scattered) Disjoint-Paths Logic

Separator logic (FO+conn):
[Schirrmacher, Siebertz, & Vigny, 2021]
[Bojańczyk, 2021]



Additional predicate $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$:

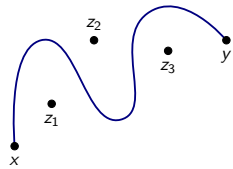
There is an (x, y)-path that avoids $z_1, \ldots, z_k$.

Can express:
k-connectivity. "$\forall x \forall y \forall z_1 \ldots \forall z_{k-1} \mathrm{conn}_{k-1}(x, y, z_1, \ldots, z_{k-1})$"
Feedback Vertex Set.

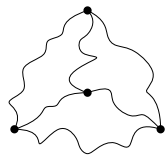
Separator logic (FO+conn):
[Schirrmacher, Siebertz, & Vigny, 2021]
[Bojańczyk, 2021]

Additional predicate $\mathsf{conn}_k(x, y, z_1, \ldots, z_k)$:

There is an (x, y)-path that avoids $z_1, \ldots, z_k$.



Can express:
k-connectivity. "$\forall x \forall y \forall z_1 \ldots \forall z_{k-1} \mathsf{conn}_{k-1}(x, y, z_1, \ldots, z_{k-1})$"
Feedback Vertex Set.

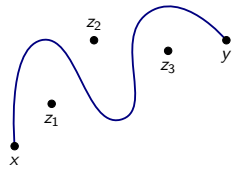


Cannot express:
Planarity, (Topological) Minor Containment.

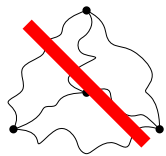
Separator logic (FO+conn):
[Schirrmacher, Siebertz, & Vigny, 2021]
[Bojańczyk, 2021]

Additional predicate $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$:

There is an (x, y)-path that avoids $z_1, \ldots, z_k$.

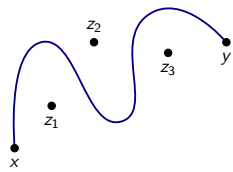


Can express:
k-connectivity. "$\forall x \forall y \forall z_1 \ldots \forall z_{k-1} \mathrm{conn}_{k-1}(x, y, z_1, \ldots, z_{k-1})$"
Feedback Vertex Set.



Cannot express:
Planarity, (Topological) Minor Containment.

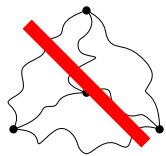
Separator logic (FO+conn):
[Schirrmacher, Siebertz, & Vigny, 2021]
[Bojańczyk, 2021]



Additional predicate $\mathrm{conn}_k(x, y, z_1, \ldots, z_k)$:

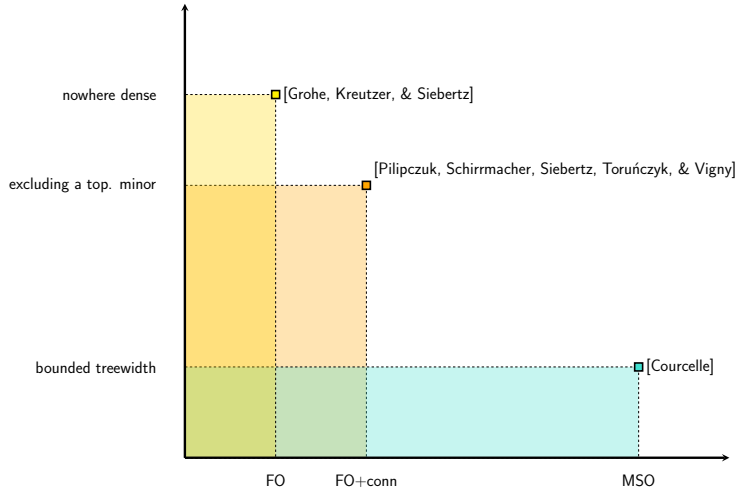
There is an (x, y)-path that avoids $z_1, \ldots, z_k$.

Can express:
k-connectivity. "$\forall x \forall y \forall z_1 \ldots \forall z_{k-1} \mathrm{conn}_{k-1}(x, y, z_1, \ldots, z_{k-1})$"
Feedback Vertex Set.



Cannot express:
Planarity, (Topological) Minor Containment.

$$\mathrm{FO} \subseteq \mathrm{FO{+}conn} \subseteq \mathrm{MSO}$$



Every FO+conn property can be tested in cubic time on graphs excluding a topological minor*.
[Pilipczuk, Schirrmacher, Siebertz, Toruńczyk, & Vigny, 2022]

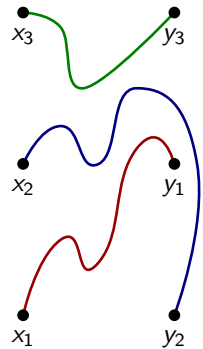
- Model checking FO+conn on graphs **not** excluding a top. minor: as hard as FO on general graphs.

FO+DP

[Schirrmacher, Siebertz, & Vigny, 2021]

Additional predicate $\mathsf{dp}_k(x_1, y_1, \ldots, x_k, y_k)$:

There are pairwise vertex-disjoint paths between x_i and y_i, for every $i \in \{1, \ldots, k\}$.



FO+DP

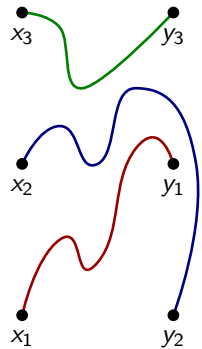
[Schirrmacher, Siebertz, & Vigny, 2021]

Additional predicate $\mathrm{dp}_k(x_1, y_1, \ldots, x_k, y_k)$:

There are pairwise vertex-disjoint paths between x_i and y_i, for every $i \in \{1, \ldots, k\}$.

Can express:
(Topological) Minor Containment.

Cannot express:
Bipartiteness.

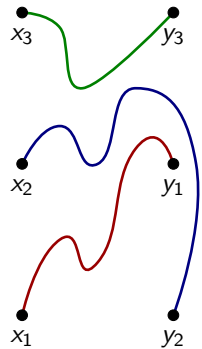


FO+DP

[Schirrmacher, Siebertz, & Vigny, 2021]

Additional predicate $\mathsf{dp}_k(x_1, y_1, \ldots, x_k, y_k)$:

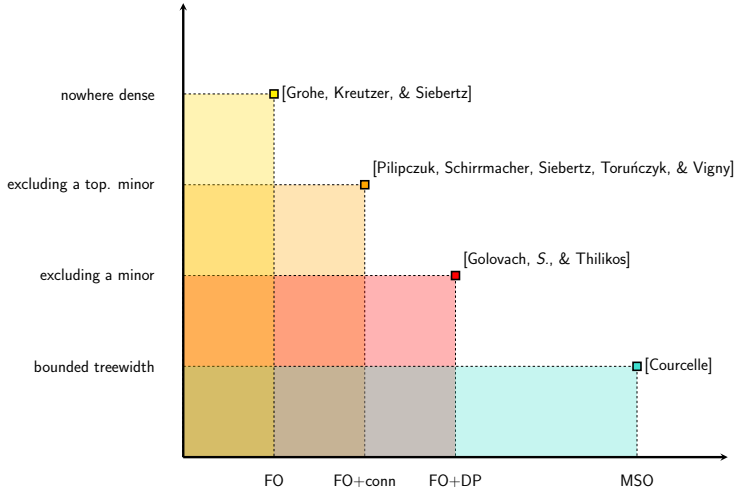
There are pairwise vertex-disjoint paths between x_i and y_i, for every $i \in \{1, \ldots, k\}$.



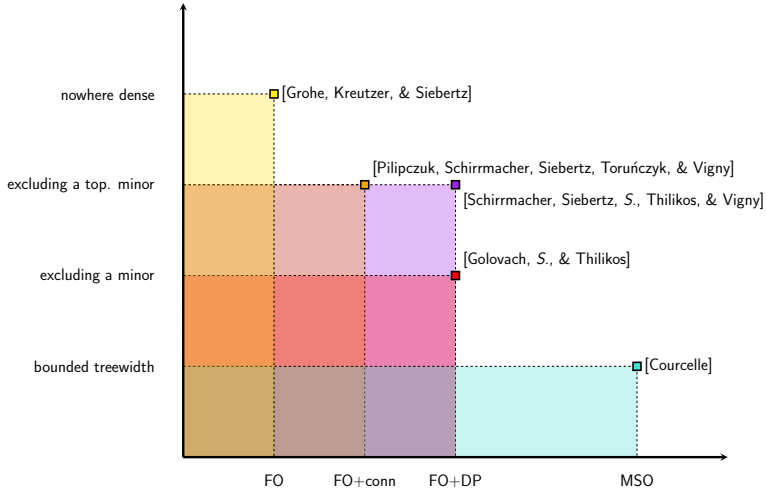
Can express:
(Topological) Minor Containment.

Cannot express:
Bipartiteness.

$$\mathsf{FO} \subseteq \mathsf{FO{+}conn} \subseteq \mathsf{FO{+}DP} \subseteq \mathsf{MSO}$$



Every FO+DP property can be tested in quadratic time on graphs excluding a minor*.
[Golovach, *S.*, & Thilikos, 2023]



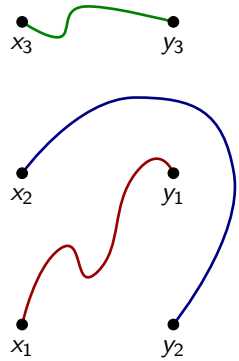
Every FO+DP property can be tested in cubic time on graphs excluding a topological minor.
[Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

FO+SDP

Scattered disjoint paths predicates:

s-$\mathrm{dp}_k(x_1, y_1, \ldots, x_k, y_k)$

There are pairwise vertex-disjoint paths
between x_i and y_i, for every $i \in \{1, \ldots, k\}$
s.t. no two vertices of two distinct paths are within distance $\leq s$.

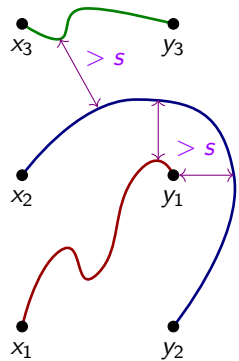


FO+SDP

Scattered disjoint paths predicates:

s-$\mathrm{dp}_k(x_1, y_1, \ldots, x_k, y_k)$

There are pairwise vertex-disjoint paths
between x_i and y_i, for every $i \in \{1, \ldots, k\}$
s.t. no two vertices of two distinct paths are within distance $\leq s$.

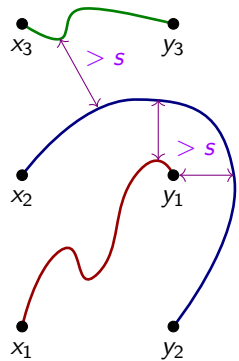


FO+SDP

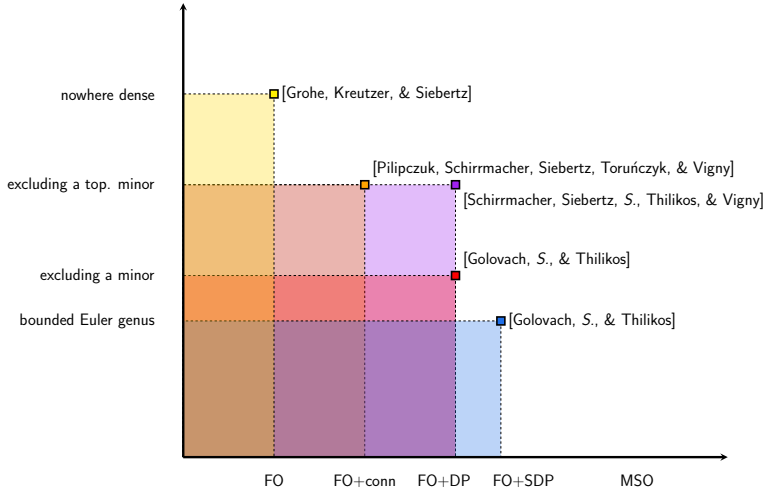
Scattered disjoint paths predicates:

s-$\mathrm{dp}_k(x_1, y_1, \ldots, x_k, y_k)$

There are pairwise vertex-disjoint paths
between x_i and y_i, for every $i \in \{1, \ldots, k\}$
s.t. no two vertices of two distinct paths are within distance $\leq s$.



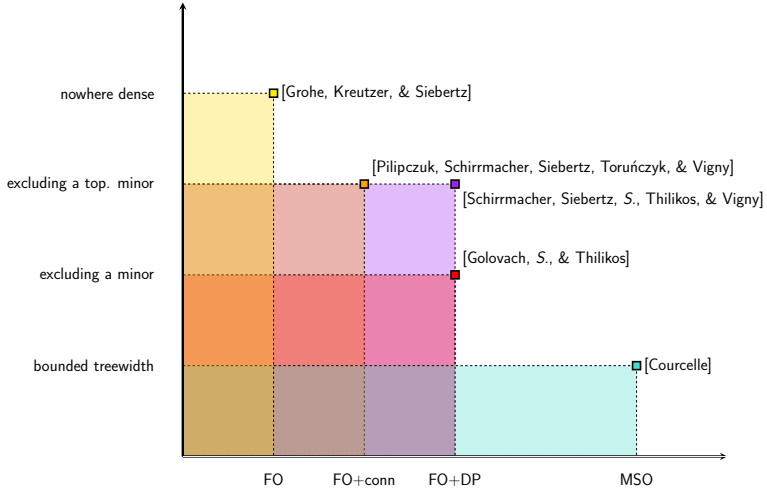
$\mathrm{dp}_k(x_1, y_1, \ldots, x_k, y_k) = 0\text{-}\mathrm{dp}_k(x_1, y_1, \ldots, x_k, y_k)$



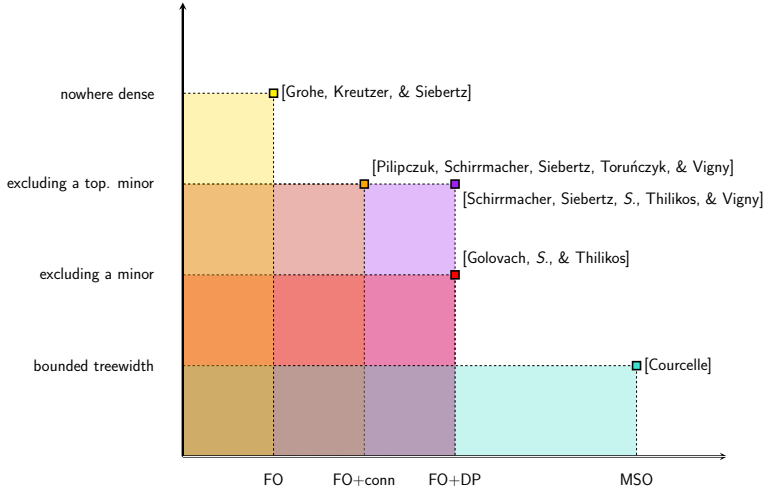
Every FO+SDP property can be tested in quadratic time on graphs of bounded Euler genus.
[Golovach, *S.*, & Thilikos, 2023]

More logics?

More logics?

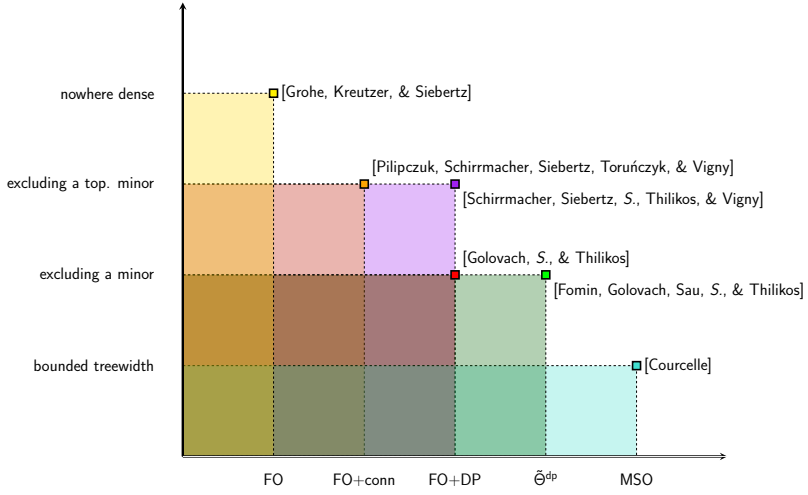

nowhere dense
excluding a top. minor
excluding a minor
bounded treewidth
[Grohe, Kreutzer, & Siebertz]
[Pilipczuk, Schirrmacher, Siebertz, Toruńczyk, & Vigny]
[Schirrmacher, Siebertz, S., Thilikos, & Vigny]
[Golovach, S., & Thilikos]
[Courcelle]
FO
FO+conn
FO+DP
MSO

More logics?



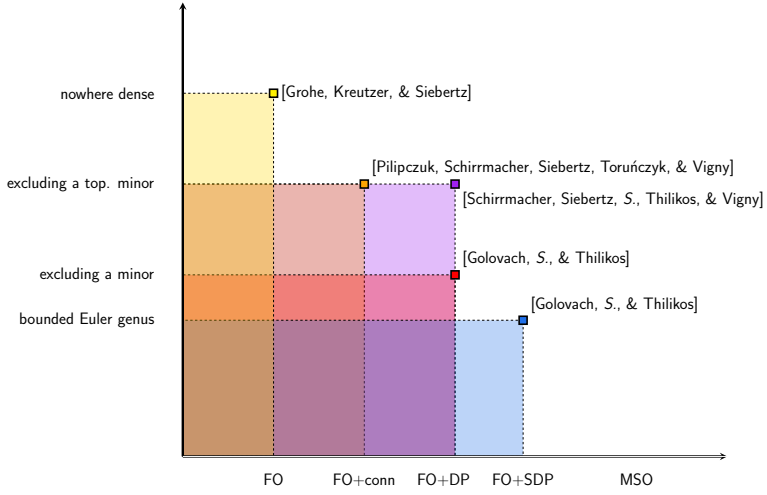
Compound logic(s) $\tilde{\Theta}^{(s)dp} = \begin{cases} \text{sentences expressing the recursive removal of MSO-expressible} \\ \text{and bounded treewidth modulators to a FO+(S)DP property.} \end{cases}$

More logics?



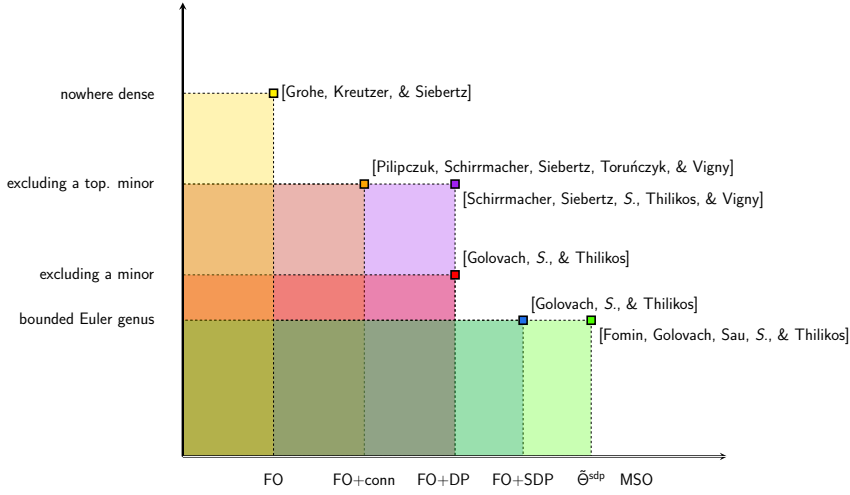
Compound logic(s) $\tilde{\Theta}^{(s)dp} = \begin{cases} \text{sentences expressing the recursive removal of MSO-expressible} \\ \text{and bounded treewidth modulators to a FO+(S)DP property.} \end{cases}$

More logics?



Compound logic(s) $\tilde{\Theta}^{(s)dp} = \begin{cases} \text{sentences expressing the recursive removal of MSO-expressible} \\ \text{and bounded treewidth modulators to a FO+(S)DP property.} \end{cases}$

More logics?

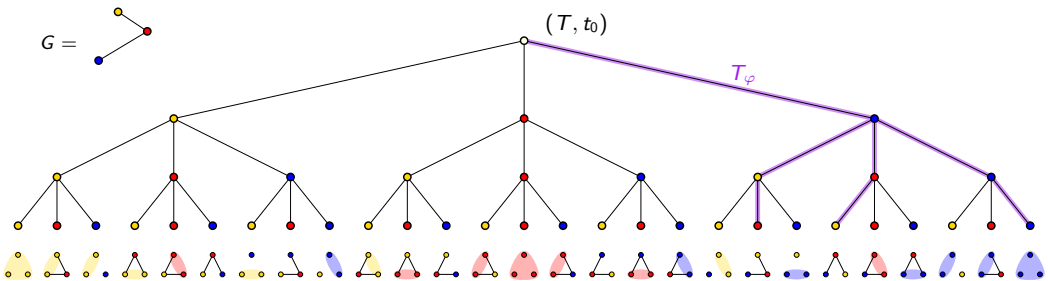


Compound logic(s) $\tilde{\Theta}^{(s)dp} = \begin{cases} \text{sentences expressing the recursive removal of MSO-expressible} \\ \text{and bounded treewidth modulators to a FO+(S)DP property.} \end{cases}$

Discussion of the proof

Model checking game [Hinttikka, 1982].
a.k.a. *game trees*, *evaluation trees*, *morphism trees*,...

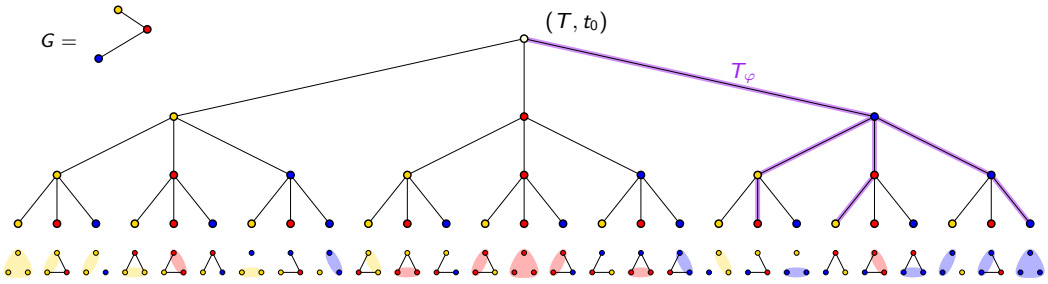
Example:



$$\varphi = \exists x_1 \, \forall x_2 \, \exists x_3 \, (x_1 = x_2 \vee E(x_2, x_3)).$$

Model checking game [Hinttikka, 1982].
a.k.a. *game trees*, *evaluation trees*, *morphism trees*,...

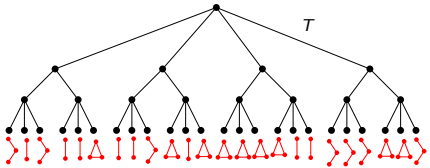
Example:



$$\varphi = \exists x_1 \; \forall x_2 \; \exists x_3 \; (x_1 = x_2 \vee E(x_2, x_3)).$$

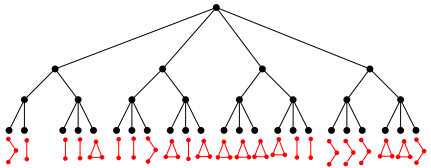
- For a graph of size n and a formula with q quantifiers, the tree is of size n^q.

One can reduce T:



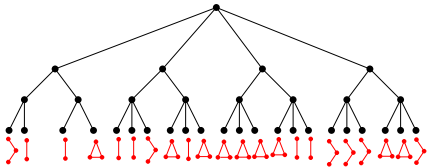
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



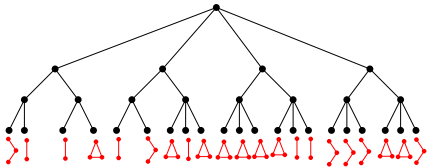
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



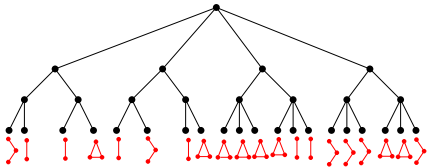
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



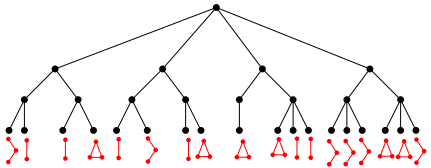
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



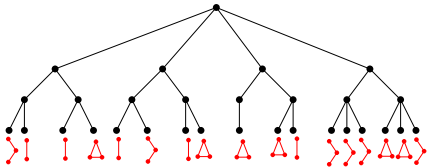
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



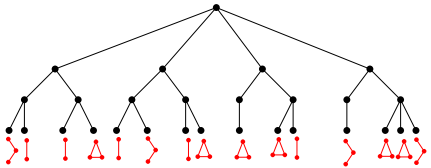
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



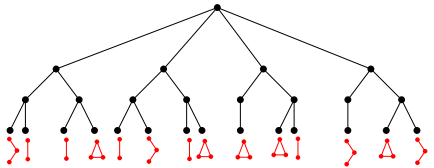
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



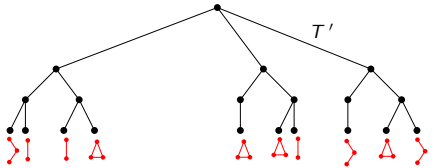
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



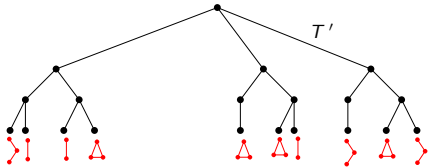
- We can "crop" some branches of T to get an "equivalent" T'.

One can reduce T:



- We can "crop" some branches of T to get an "equivalent" T'.

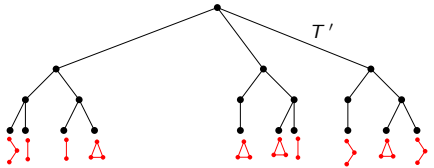
One can reduce T:



- We can "crop" some branches of T to get an "equivalent" T'.

More formally:

One can reduce T:



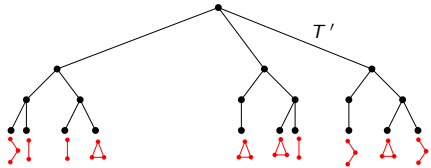
- We can "crop" some branches of T to get an "equivalent" T'.

More formally:

- Fix $r \in \mathbb{N}$. For every $\bar{v} \in V(G)^r$,

$$\textbf{pattern}^0(G, \bar{v}) = \text{ set of all atomic formulas that are true for } \bar{v} \text{ in } G.$$

One can reduce T:



- We can "crop" some branches of T to get an "equivalent" T'.

More formally:

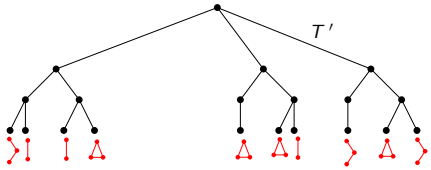
- Fix $r \in \mathbb{N}$. For every $\bar{v} \in V(G)^r$,

$$\textbf{pattern}^0(G, \bar{v}) = \text{ set of all atomic formulas that are true for } \bar{v} \text{ in } G.$$

- For each $i \in [r-1]$ and every $\bar{v} \in V(G)^{r-i}$,

$$\textbf{pattern}^i(G, \bar{v}) = \left\{\textbf{pattern}^{i-1}(G, \bar{v}u) \mid u \in V(G)\right\}$$

One can reduce T:



- We can "crop" some branches of T to get an "equivalent" T'.

More formally:

- Fix $r \in \mathbb{N}$. For every $\bar{v} \in V(G)^r$,

$$\mathbf{pattern}^0(G, \bar{v}) = \text{ set of all atomic formulas that are true for } \bar{v} \text{ in } G.$$

- For each $i \in [r-1]$ and every $\bar{v} \in V(G)^{r-i}$,

$$\mathbf{pattern}^i(G, \bar{v}) = \left\{\mathbf{pattern}^{i-1}(G, \bar{v}u) \mid u \in V(G)\right\}$$

- $\mathbf{pattern}^r(G) = \left\{\mathbf{pattern}^{r-1}(G, v) \mid v \in V(G)\right\}$.

Compute $\mathbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

Compute $\mathbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the $\mathbf{pattern}^r(G)$?

Compute $\mathbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the $\mathbf{pattern}^r(G)$?

Compute $\mathbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the $\mathbf{pattern}^r(G)$?

**** Atomic formulas are $\mathcal{L}$-definable $\implies$ $\mathbf{pattern}^r$ is $\mathcal{L}$-definable ****

Compute $\textbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the pattern$^r(G)$?

**** Atomic formulas are $\mathcal{L}$-definable $\implies$* **pattern^r** *is $\mathcal{L}$-definable ****

- In **FO**: apply [Grohe, Kreutzer, & Siebertz] if G is sparse.

Compute $\mathbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the $\mathbf{pattern}^r(G)$?

**** Atomic formulas are $\mathcal{L}$-definable $\implies$ $\mathbf{pattern}^r$ is $\mathcal{L}$-definable ****

- In **FO**: apply [Grohe, Kreutzer, & Siebertz] if G is sparse.
- In **MSO**: apply [Courcelle] if G has bounded treewidth.

Compute $\textbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the pattern$^r(G)$?

**** Atomic formulas are $\mathcal{L}$-definable $\implies$* **pattern^r** *is $\mathcal{L}$-definable ****

- In **FO**: apply [Grohe, Kreutzer, & Siebertz] if G is sparse.
- In **MSO**: apply [Courcelle] if G has bounded treewidth.
- In **FO+DP** ?

Compute $\mathbf{pattern}^r(G) \implies$ can decide whether $G \models \varphi$, for every φ with r quantifiers.

How to compute the $\mathbf{pattern}^r(G)$?

**** Atomic formulas are $\mathcal{L}$-definable $\implies$ $\mathbf{pattern}^r$ is $\mathcal{L}$-definable ****

- In **FO**: apply [Grohe, Kreutzer, & Siebertz] if G is sparse.
- In **MSO**: apply [Courcelle] if G has bounded treewidth.
- In **FO+DP** ? If G has **large treewidth** ?

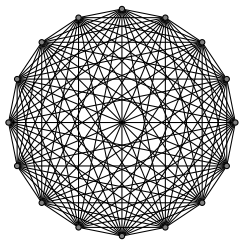
Large treewidth

Flat Wall Theorem

[Robertson & Seymour, Graph Minors XIII, 1995]
[Kawarabayashi, Thomas, & Wollan, 2018]
[Sau, *S.*, & Thilikos, 2022]

Large clique minor

Large *"flat" grid-like* induced subgraph

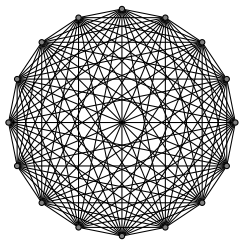


Large treewidth

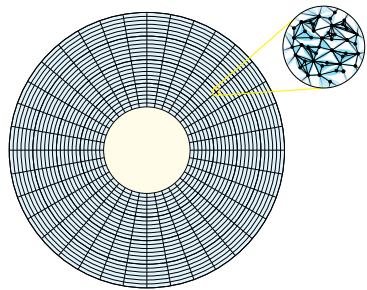
Flat Wall Theorem

[Robertson & Seymour, Graph Minors XIII, 1995]
[Kawarabayashi, Thomas, & Wollan, 2018]
[Sau, *S.*, & Thilikos, 2022]

Large clique minor



Large *"flat" grid-like* induced subgraph

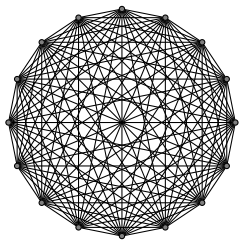


Large treewidth

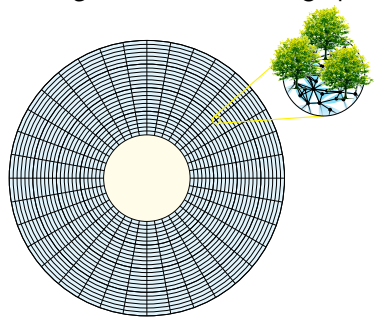
Flat Wall Theorem

[Robertson & Seymour, Graph Minors XIII, 1995]
[Kawarabayashi, Thomas, & Wollan, 2018]
[Sau, *S.*, & Thilikos, 2022]

Large clique minor



Large *"flat" grid-like* induced subgraph

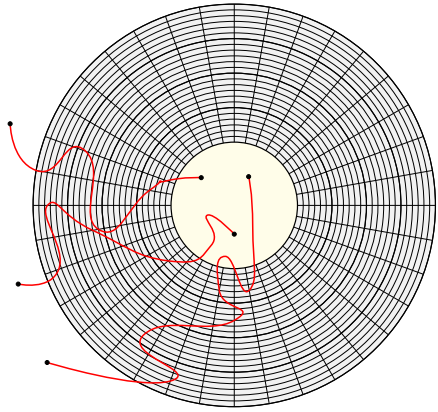


How this helps to compute the pattern?

Compute partial patterns!

How this helps to compute the pattern?

Compute partial patterns!



How this helps to compute the pattern?

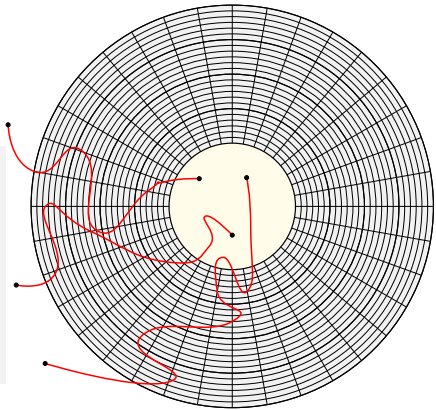
Compute partial patterns!

Linkage Combing Lemma

[Golovach, S., & Thilikos, *Combing a linkage in an annulus*, 2022]

There is a function $f : \mathbb{N} \to \mathbb{N}$ such that if
- G is a partially disk-embedded graph,
- $(\mathcal{C}, \mathcal{P})$ is a disk-embedded railed annulus of size $f(k)$, and
- L is an annulus-avoiding linkage of size $\leq k$,

then there is an equivalent linkage L' that traverses the middle cycle of $\mathcal{C}$ through $\mathcal{P}$.



How this helps to compute the pattern?

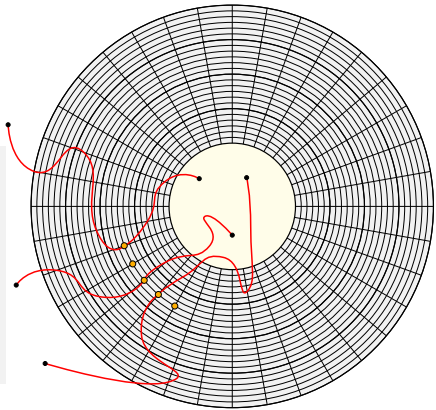
Compute partial patterns!

Linkage Combing Lemma

[Golovach, S., & Thilikos, *Combing a linkage in an annulus*, 2022]

There is a function $f : \mathbb{N} \to \mathbb{N}$ such that if
- G is a partially disk-embedded graph,
- $(\mathcal{C}, \mathcal{P})$ is a disk-embedded railed annulus of size $f(k)$, and
- L is an annulus-avoiding linkage of size $\leq k$,

then there is an equivalent linkage L' that traverses the middle cycle of $\mathcal{C}$ through $\mathcal{P}$.



How this helps to compute the pattern?

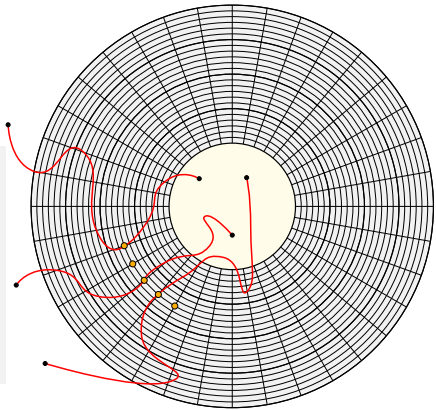
Compute partial patterns!

Linkage Combing Lemma

[Golovach, S., & Thilikos, *Combing a linkage in an annulus*, 2022]

There is a function $f : \mathbb{N} \to \mathbb{N}$ such that if
- G is a partially disk-embedded graph,
- $(\mathcal{C}, \mathcal{P})$ is a disk-embedded railed annulus of size $f(k)$, and
- L is an annulus-avoiding linkage of size $\leq k$,

then there is an equivalent linkage L' that traverses the middle cycle of $\mathcal{C}$ through $\mathcal{P}$.



- Completely **control** how paths enter the inner part of the *annulus*.

How this helps to compute the pattern?

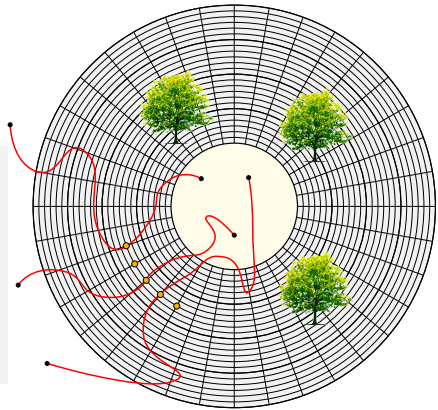
Compute partial patterns!

Linkage Combing Lemma

[Golovach, S., & Thilikos, *Combing a linkage in an annulus*, 2022]

There is a function $f : \mathbb{N} \to \mathbb{N}$ such that if
- G is a partially disk-embedded graph,
- $(\mathcal{C}, \mathcal{P})$ is a disk-embedded railed annulus of size $f(k)$, and
- L is an annulus-avoiding linkage of size $\leq k$,

then there is an equivalent linkage L' that traverses the middle cycle of $\mathcal{C}$ through $\mathcal{P}$.



- Completely **control** how paths enter the inner part of the *annulus*.
- We work in a part of small treewidth: can compute partial patterns.

How this helps to compute the pattern?

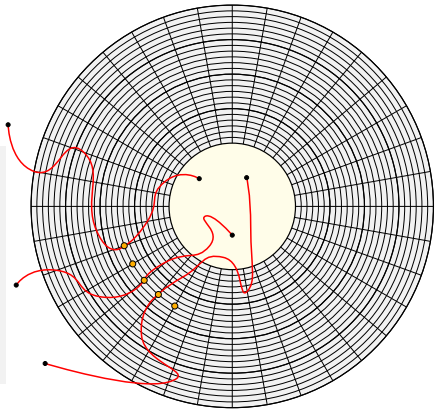
Compute partial patterns!

Linkage Combing Lemma

[Golovach, S., & Thilikos, *Combing a linkage in an annulus*, 2022]

There is a function $f : \mathbb{N} \to \mathbb{N}$ such that if
- G is a partially disk-embedded graph,
- $(\mathcal{C}, \mathcal{P})$ is a disk-embedded railed annulus of size $f(k)$, and
- L is an annulus-avoiding linkage of size $\leq k$,

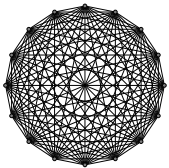
then there is an equivalent linkage L' that traverses the middle cycle of $\mathcal{C}$ through $\mathcal{P}$.



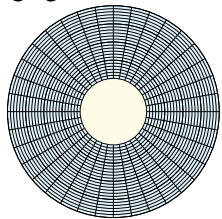
- Completely **control** how paths enter the inner part of the *annulus*.
- We work in a part of small treewidth: can compute partial patterns.
- Same partial patterns $\underset{\textbf{Combing}}{\Longrightarrow}$ same (global) patterns.

Large treewidth

Large clique minor



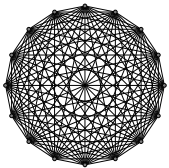
Large *grid-like "flat" structure*



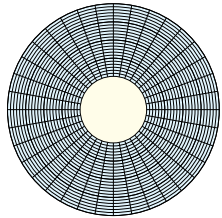
- Compute partial patterns in *bounded treewidth part*.

Large treewidth

Large clique minor



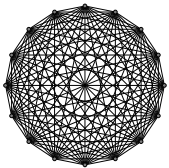
Large *grid-like "flat" structure*



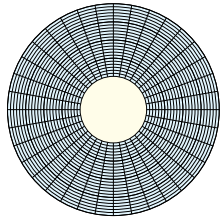
- Compute partial patterns in *bounded treewidth part*.
- Same partial patterns $\underset{\text{Combing}}{\Longrightarrow}$ same (global) patterns

Large treewidth

Large clique minor



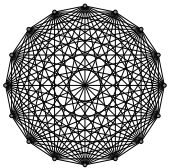
Large *grid-like "flat" structure*



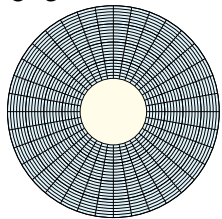
- Compute partial patterns in *bounded treewidth part*.
- Same partial patterns $\xrightarrow[\text{Combing}]{}$ same (global) patterns
- **Remove pattern-duplicates** and reduce the instance.

Large treewidth

Large clique minor



Large *grid-like "flat" structure*

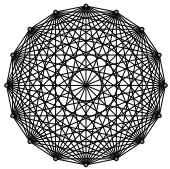


- Compute partial patterns in *bounded treewidth part*.
- Same partial patterns $\xrightarrow[\text{Combing}]{}$ same (global) patterns
- **Remove pattern-duplicates** and reduce the instance.

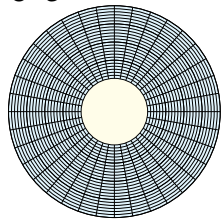
 $\Longrightarrow$ get same-pattern graph of *bd. treewidth*.

Large treewidth

Large clique minor



Large *grid-like "flat" structure*



- Compute partial patterns in *bounded treewidth part*.
- Same partial patterns $\overset{\text{Combing}}{\Longrightarrow}$ same (global) patterns
- **Remove pattern-duplicates** and reduce the instance.

 $\Longrightarrow$ get same-pattern graph of *bd. treewidth*.

[Golovach, *S.*, Thilikos, 2023]

Large clique minor case:

Large clique minor case:

- (q, k)-*unbreakable* graph G (*mesure of inseparabilty*):

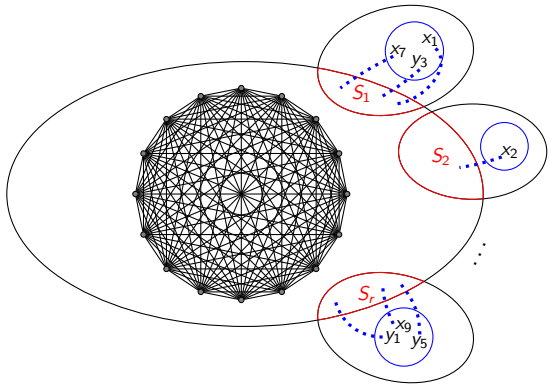
For every separation (A, B) of G of order at most k, either $|A| \leq q$ or $|B| \leq q$.

Large clique minor case:

- (q, k)-*unbreakable* graph G (*mesure of inseparabilty*):

For every separation (A, B) of G of order at most k, either $|A| \leq q$ or $|B| \leq q$.

- On graphs with large clique minors that are *unbreakable* :



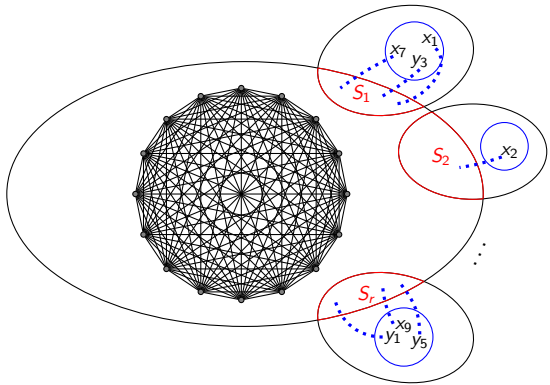
Large clique minor case:

- (q, k)-*unbreakable* graph G (*mesure of inseparabilty*):

For every separation (A, B) of G of order at most k, either $|A| \leq q$ or $|B| \leq q$.

- On graphs with large clique minors that are *unbreakable* :

Long disjoint paths always exist



Large clique minor case:

- (q, k)-*unbreakable* graph G (*mesure of inseparabilty*):

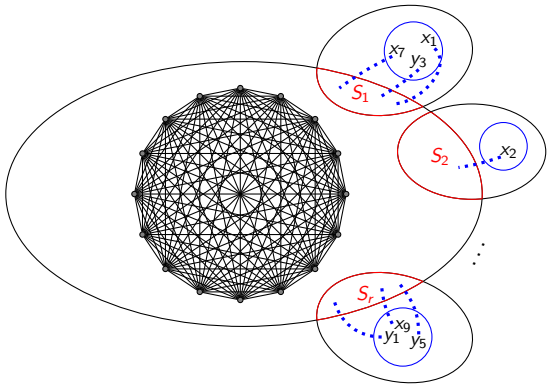
For every separation (A, B) of G of order at most k, either $|A| \leq q$ or $|B| \leq q$.

- On graphs with large clique minors that are *unbreakable* :

Long disjoint paths always exist

$\Downarrow$

Suffices to check only for *short* paths



Large clique minor case:

- (q, k)-*unbreakable* graph G (*mesure of inseparabilty*):

For every separation (A, B) of G of order at most k, either $|A| \leq q$ or $|B| \leq q$.

- On graphs with large clique minors that are *unbreakable* :

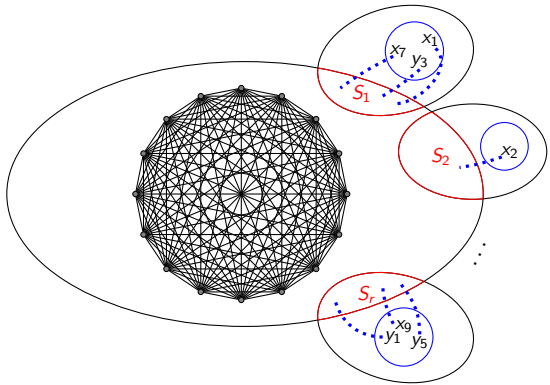
Long disjoint paths always exist

$\Downarrow$

Suffices to check only for *short* paths

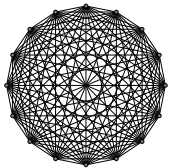
$\Downarrow$

FO+DP collapses to FO



Large treewidth

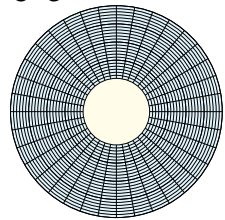
Large clique minor



- Extra assumption: unbreakability.
- FO+DP collapses to FO.
- Using FO-machinery, compute pattern.

[Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

Large *grid-like "flat" structure*



- Compute partial patterns in *bounded treewidth part*.
- Same partial patterns $\overset{\text{Combing}}{\Longrightarrow}$ same (global) patterns
- **Remove pattern-duplicates** and reduce the instance.

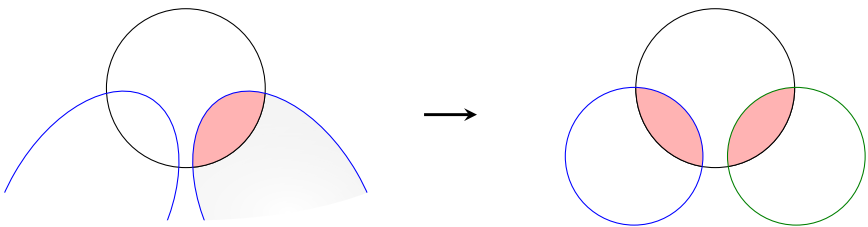
 $\Longrightarrow$ get same-pattern graph of *bd. treewidth*.

[Golovach, *S.*, Thilikos, 2023]

Wrapping up the proof. [Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

Wrapping up the proof. [Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

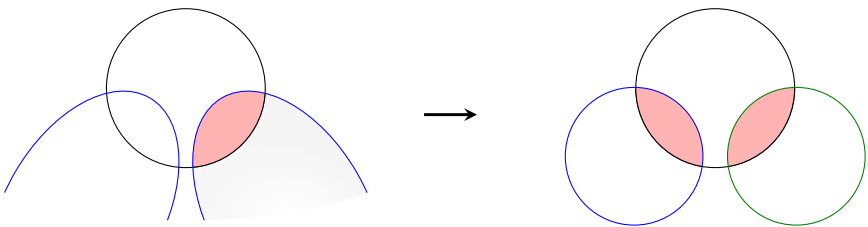
Dynamic programming on a tree-decomposition in unbreakable bags.
[Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh, 2014]



Wrapping up the proof. [Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

Dynamic programming on a tree-decomposition in unbreakable bags.
[Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh, 2014]

Goal: For every node, compute pattern of the graph corresponding to the subtree.

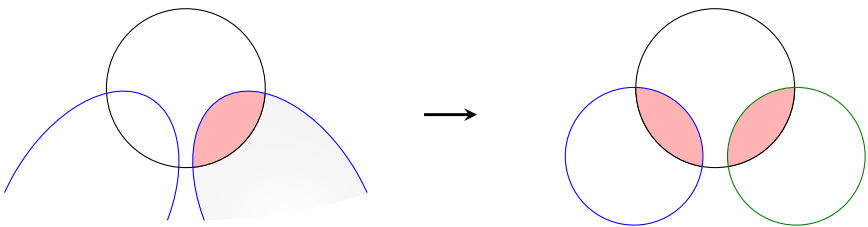


Wrapping up the proof. [Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

Dynamic programming on a tree-decomposition in unbreakable bags.
[Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh, 2014]

Goal: For every node, compute pattern of the graph corresponding to the subtree.

- For bags without large minors: [Golovach, *S.*, Thilikos, 2023]

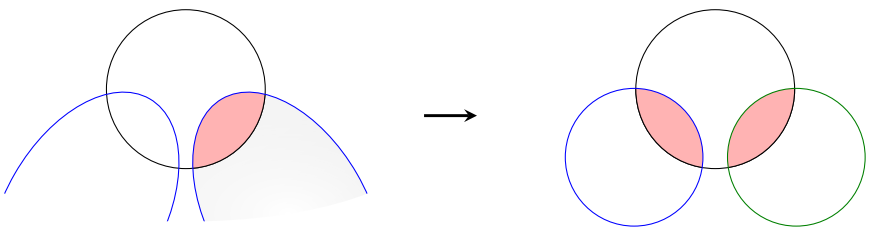


Wrapping up the proof. [Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

Dynamic programming on a tree-decomposition in unbreakable bags.
[Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh, 2014]

Goal: For every node, compute pattern of the graph corresponding to the subtree.

- For bags without large minors: [Golovach, *S.*, Thilikos, 2023]
- For bags with large minors: unbreakability $\implies$ FO+DP collapses to FO $\implies$ FO-machinery



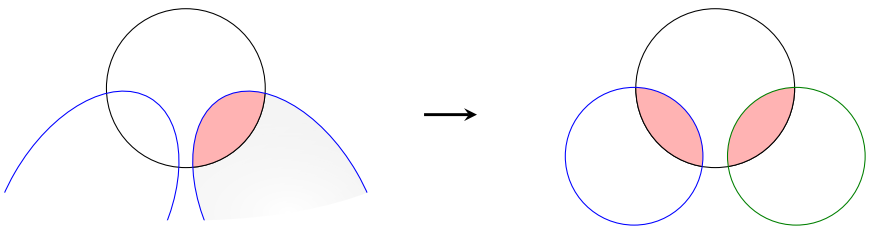
Wrapping up the proof. [Schirrmacher, Siebertz, *S.*, Thilikos, & Vigny, 2023+]

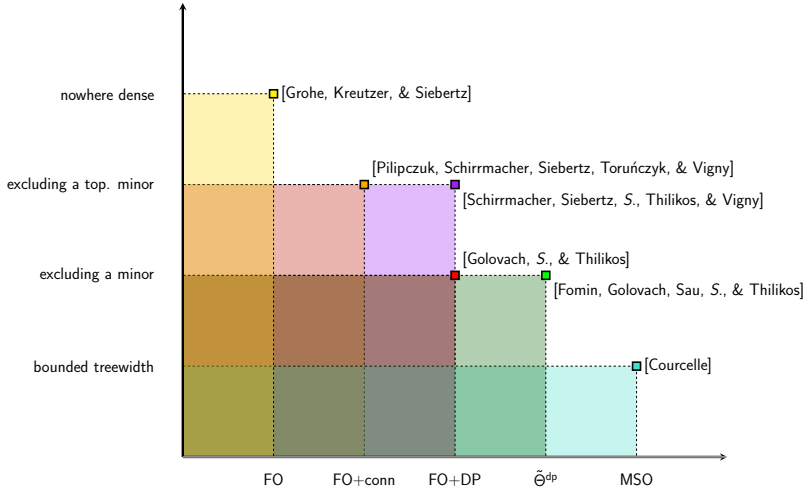
Dynamic programming on a tree-decomposition in unbreakable bags.
[Cygan, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh, 2014]

Goal: For every node, compute pattern of the graph corresponding to the subtree.

- For bags without large minors: [Golovach, *S.*, Thilikos, 2023]
- For bags with large minors: unbreakability $\implies$ FO+DP collapses to FO $\implies$ FO-machinery

- Combine solutions bottom-up: Possible **only if the input graph excludes some topological minor.**




nowhere dense
excluding a top. minor
excluding a minor
bounded treewidth
[Grohe, Kreutzer, & Siebertz]
[Pilipczuk, Schirrmacher, Siebertz, Toruńczyk, & Vigny]
[Schirrmacher, Siebertz, S., Thilikos, & Vigny]
[Golovach, S., & Thilikos]
[Fomin, Golovach, Sau, S., & Thilikos]
[Courcelle]
FO
FO+conn
FO+DP
$\tilde{\Theta}^{\mathrm{dp}}$
MSO

Thank you!