Efficient algorithms on graphs of bounded treewidth

Ignasi Sau

CNRS, LIRMM, Université de Montpellier

JGA 2018, Grenoble, France
14-16 novembre 2018

Outline of the talk

1. Introduction
   - Parameterized complexity
   - Treewidth
2. FPT algorithms parameterized by treewidth
3. The $\mathcal{F}$-Deletion problem
4. Conclusions

Next section is...

Next subsection is...

1. Introduction
   - Parameterized complexity
   - Treewidth
2. FPT algorithms parameterized by treewidth
3. The $\mathcal{F}$-Deletion problem
4. Conclusions

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.
- Karp (1972): list of 21 *important* NP-complete problems.
- Nowadays, literally thousands of problems are known to be NP-hard: unless P = NP, they cannot be solved in polynomial time.

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.
- Karp (1972): list of 21 *important* NP-complete problems.
- Nowadays, literally thousands of problems are known to be NP-hard: unless $\mathsf{P} = \mathsf{NP}$, they cannot be solved in polynomial time.
- But what does it mean for a problem to be NP-hard?

  No algorithm solves all instances optimally in polynomial time.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

- VLSI design: the number of circuit layers is usually ≤ 10.
- Computational biology: Real instances of DNA chain reconstruction usually have treewidth ≤ 11.
- Robotics: Number of degrees of freedom in motion planning problems ≤ 10.
- Compilers: Checking compatibility of type declarations is hard, but usually the depth of type declarations is ≤ 10.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

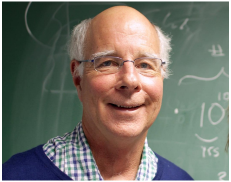
- VLSI design: the number of circuit layers is usually ≤ 10.
- Computational biology: Real instances of DNA chain reconstruction usually have treewidth ≤ 11.
- Robotics: Number of degrees of freedom in motion planning problems ≤ 10.
- Compilers: Checking compatibility of type declarations is hard, but usually the depth of type declarations is ≤ 10.

Message In many applications, not only the total size of the instance matters, but also the value of an additional parameter.

The area of parameterized complexity

Idea Measure the complexity of an algorithm in terms of the input size and an additional parameter.

This theory started in the late 80's, by Downey and Fellows:



Today, it is a well-established area with hundreds of articles published every year in the most prestigious TCS journals and conferences.

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

- k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?
- k-Clique: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise adjacent vertices?
- Vertex k-Coloring: Can the vertices of a graph be colored with $\leq k$ colors, so that any two adjacent vertices get different colors?

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

- k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?
- k-Clique: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise adjacent vertices?
- Vertex k-Coloring: Can the vertices of a graph be colored with $\leq k$ colors, so that any two adjacent vertices get different colors?

These three problems are NP-hard, but are they equally hard?

They behave quite differently...

- k-Vertex Cover: Solvable in time $\mathcal{O}(2^k \cdot (m+n))$
- k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k)$
- Vertex k-Coloring: NP-hard for fixed $k = 3$.

They behave quite differently...

- k-Vertex Cover: Solvable in time $\mathcal{O}(2^k \cdot (m+n)) = f(k) \cdot n^{\mathcal{O}(1)}$.

- k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

- Vertex k-Coloring: NP-hard for fixed $k = 3$.

They behave quite differently...

- k-Vertex Cover: Solvable in time $\mathcal{O}(2^k \cdot (m+n)) = f(k) \cdot n^{\mathcal{O}(1)}$.

  $\boxed{\text{The problem is FPT}}$ (fixed-parameter tractable)

- k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

- Vertex k-Coloring: NP-hard for fixed $k = 3$.

They behave quite differently...

- k-Vertex Cover: Solvable in time $\mathcal{O}(2^k \cdot (m+n)) = f(k) \cdot n^{\mathcal{O}(1)}$.

  $\boxed{\text{The problem is FPT}}$ (fixed-parameter tractable)

- k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

  $\boxed{\text{The problem is XP}}$ (slice-wise polynomial)

- Vertex k-Coloring: NP-hard for fixed $k = 3$.

They behave quite differently...

- k-Vertex Cover: Solvable in time $\mathcal{O}(2^k \cdot (m+n)) = f(k) \cdot n^{\mathcal{O}(1)}$.

  $\boxed{\text{The problem is FPT}}$ (fixed-parameter tractable)

- k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

  $\boxed{\text{The problem is XP}}$ (slice-wise polynomial)

- Vertex k-Coloring: NP-hard for fixed $k = 3$.

  $\boxed{\text{The problem is para-NP-hard}}$

Why k-Clique may not be FPT?

k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-Clique may not be FPT?

k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-Clique may not be FPT?

Why k-Clique may not be FPT?

k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-Clique may not be FPT?

So far, nobody has managed to find an FPT algorithm.

(also, nobody has found a poly-time algorithm for 3-SAT)

Why k-Clique may not be FPT?

k-Clique: Solvable in time $\mathcal{O}(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-Clique may not be FPT?

So far, nobody has managed to find an FPT algorithm.

(also, nobody has found a poly-time algorithm for 3-SAT)

Working hypothesis of parameterized complexity: $\boxed{k\text{-Clique is not FPT}}$

(in classical complexity: 3-SAT cannot be solved in poly-time)

How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from A to B is an algorithm such that:

$$\boxed{\text{Instance } (x, k) \text{ of } A} \xrightarrow{\text{time } f(k) \cdot |x|^{\mathcal{O}(1)}} \boxed{\text{Instance } (x', k') \text{ of } B}$$

How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from A to B is an algorithm such that:

$\boxed{\text{Instance } (x, k) \text{ of } A}$ $\xrightarrow{\text{time } f(k) \cdot |x|^{\mathcal{O}(1)}}$ $\boxed{\text{Instance } (x', k') \text{ of } B}$

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of B.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from A to B is an algorithm such that:

Instance (x, k) of A $\xrightarrow{\text{time } f(k) \cdot |x|^{\mathcal{O}(1)}}$ Instance (x', k') of B

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of B.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

W[1]-hard problem: $\exists$ parameterized reduction from k-CLIQUE to it.

W[2]-hard problem: $\exists$ param. reduction from k-DOMINATING SET to it.

How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from A to B is an algorithm such that:

Instance (x, k) of A $\xrightarrow{\text{time } f(k) \cdot |x|^{\mathcal{O}(1)}}$ Instance (x', k') of B

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of B.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

W[1]-hard problem: $\exists$ parameterized reduction from k-CLIQUE to it.

W[2]-hard problem: $\exists$ param. reduction from k-DOMINATING SET to it.

W[i]-hard: strong evidence of not being FPT.

How to transfer hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems.

A parameterized reduction from A to B is an algorithm such that:

Instance (x, k) of A $\xrightarrow{\text{time } f(k) \cdot |x|^{\mathcal{O}(1)}}$ Instance (x', k') of B

1. (x, k) is a YES-instance of A $\Leftrightarrow$ (x', k') is a YES-instance of B.
2. $k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

W[1]-hard problem: $\exists$ parameterized reduction from k-CLIQUE to it.

W[2]-hard problem: $\exists$ param. reduction from k-DOMINATING SET to it.

W[i]-hard: strong evidence of not being FPT. Hypothesis: $\boxed{\text{FPT} \neq \text{W[1]}}$

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

Kernelization

Idea polynomial-time preprocessing.

A kernel for a parameterized problem A is an algorithm such that:

Instance (x, k) of A $\xrightarrow{\text{polynomial time}}$ Instance (x', k') of A

Kernelization

Idea: polynomial-time preprocessing.

A kernel for a parameterized problem A is an algorithm such that:

Instance (x, k) of A → polynomial time → Instance (x', k') of A

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of A.
2. $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

A kernel for a parameterized problem A is an algorithm such that:

Instance (x, k) of A → polynomial time → Instance (x', k') of A

1. (x, k) is a Yes-instance of A $\Leftrightarrow$ (x', k') is a Yes-instance of A.
2. $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

The function g is called the size of the kernel.

If g is a polynomial (linear), then we have a polynomial (linear) kernel.

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

A kernel for a parameterized problem A is an algorithm such that:

$\boxed{\text{Instance } (x, k) \text{ of } A}$ polynomial time $\Rightarrow$ $\boxed{\text{Instance } (x', k') \text{ of } A}$

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of A.
2. $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

The function g is called the size of the kernel.

If g is a polynomial (linear), then we have a polynomial (linear) kernel.

Fact: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels?

Fact: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels?

Do all FPT problems admit polynomial kernels?

Fact: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels? **NO!**

Theorem (Bodlaender, Downey, Fellows, Hermelin. 2009)

Deciding whether a graph has a PATH *with $\geq k$ vertices is* FPT *but does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/poly.

Typical approach to deal with a parameterized problem

Parameterized problem L

k-Clique

k-Vertex Cover

k-Path

Vertex k-Coloring

Typical approach to deal with a parameterized problem

Parameterized problem L

XP

para-NP-hard

k-Clique

k-Vertex Cover

k-Path

Vertex k-Coloring

Typical approach to deal with a parameterized problem

k-Clique
k-Vertex Cover
k-Path
Vertex k-Coloring

Parameterized problem L

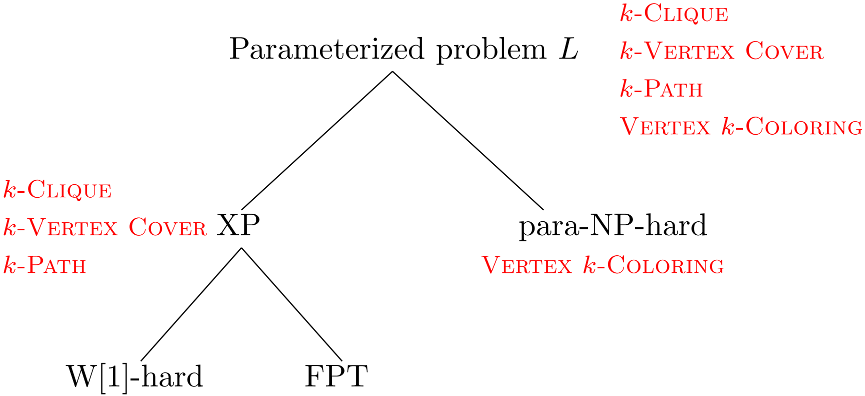
k-Clique
k-Vertex Cover
k-Path

XP

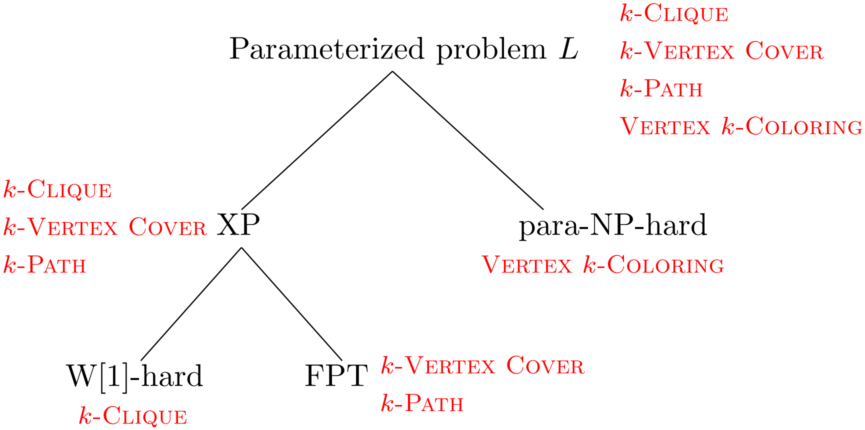
para-NP-hard

Vertex k-Coloring

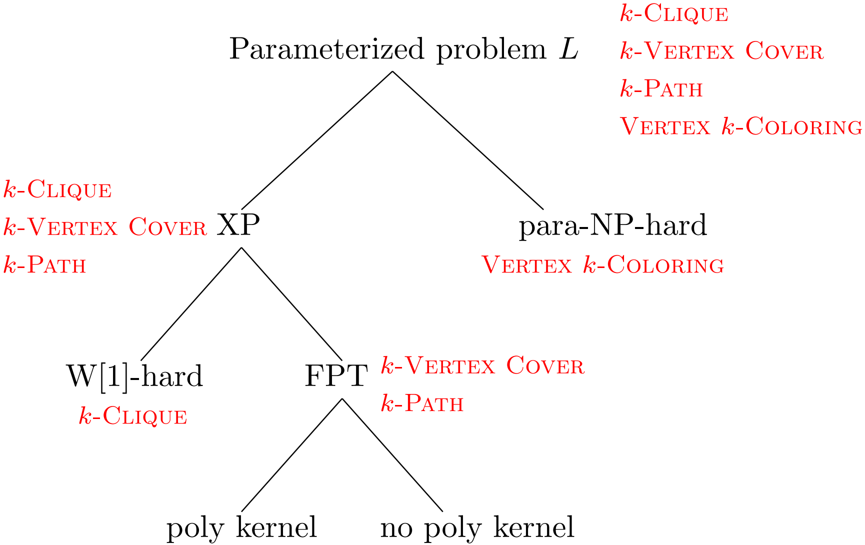
Typical approach to deal with a parameterized problem



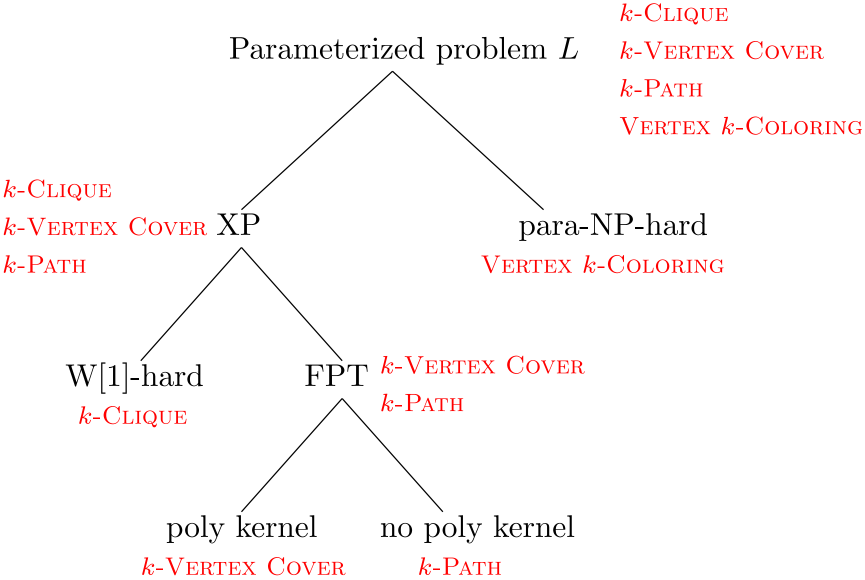
Typical approach to deal with a parameterized problem



Typical approach to deal with a parameterized problem



Typical approach to deal with a parameterized problem

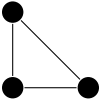


Next subsection is...

1. Introduction
   - Parameterized complexity
   - Treewidth
2. FPT algorithms parameterized by treewidth
3. The $\mathcal{F}$-DELETION problem
4. Conclusions

Treewidth via k-trees

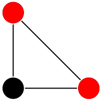
Example of a 2-tree:



[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Example of a 2-tree:

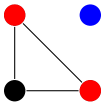


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

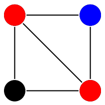


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

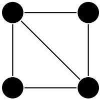


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

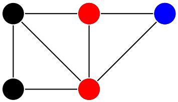


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

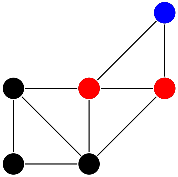


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

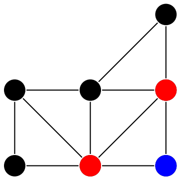


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

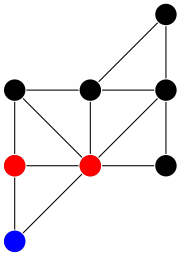


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

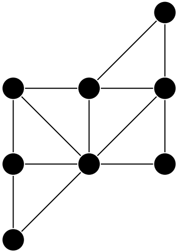


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:

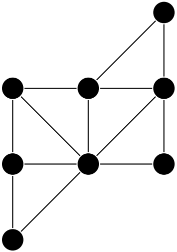


[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

Treewidth via k-trees

Example of a 2-tree:



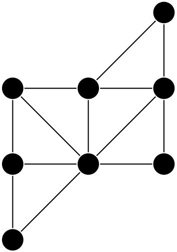
[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

A partial k-tree is a subgraph of a k-tree.

Treewidth via k-trees

Example of a 2-tree:



[Figure by Julien Baste]

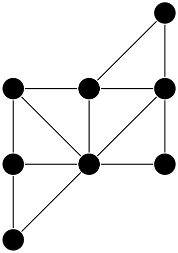
A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

A partial k-tree is a subgraph of a k-tree.

Treewidth of a graph G, denoted $\mathrm{tw}(G)$: smallest integer k such that G is a partial k-tree.

Treewidth via k-trees

Example of a 2-tree:



[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

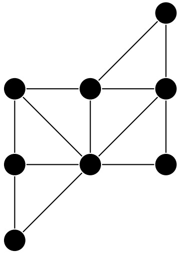
A partial k-tree is a subgraph of a k-tree.

Treewidth of a graph G, denoted $\mathrm{tw}(G)$: smallest integer k such that G is a partial k-tree.

Invariant that measures the topological resemblance of a graph to a tree.

Treewidth via k-trees

Example of a 2-tree:



[Figure by Julien Baste]

A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

A partial k-tree is a subgraph of a k-tree.

Treewidth of a graph G, denoted $\mathrm{tw}(G)$: smallest integer k such that G is a partial k-tree.

Invariant that measures the topological resemblance of a graph to a tree.

Construction suggests the notion of tree decomposition: small separators.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where

  T is a tree, and

  $B_t \subseteq V(G) \ \ \forall t \in V(T)$ (bags),

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  satisfying the following:

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:

  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G:
  minimum width of a tree decomposition of G.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \ \ \forall t \in V(T)$ (bags),

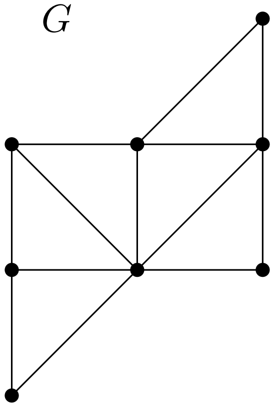
  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:

  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G:
  minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

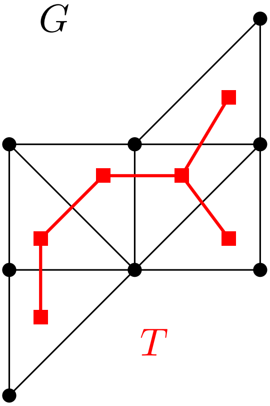
  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:

  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

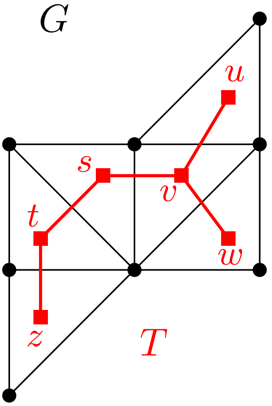
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

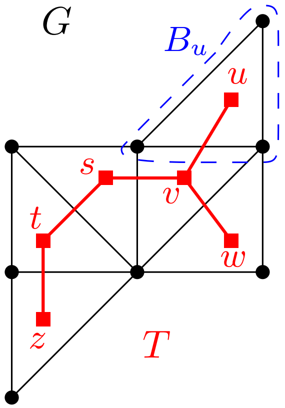
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u,v\} \in E(G)$, $\exists t \in V(T)$ with $\{u,v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G:
  minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

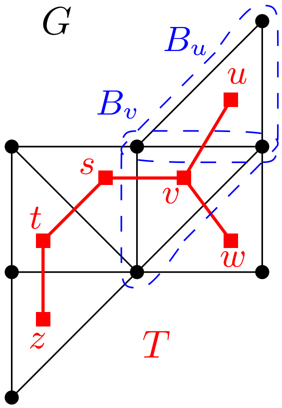
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G:
  minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

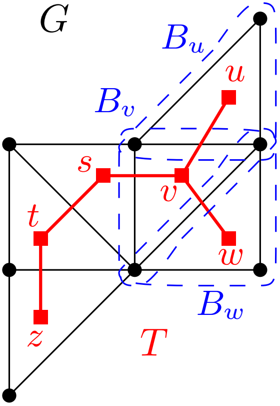
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G), \exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

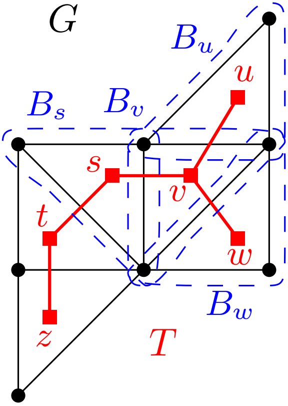
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G), \exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G:
  minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

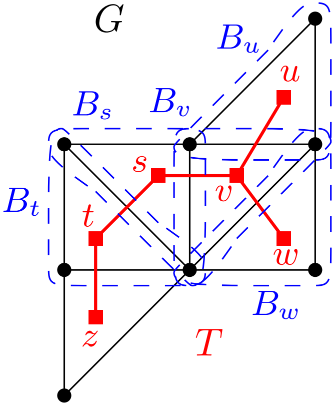
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:
  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G:
  minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

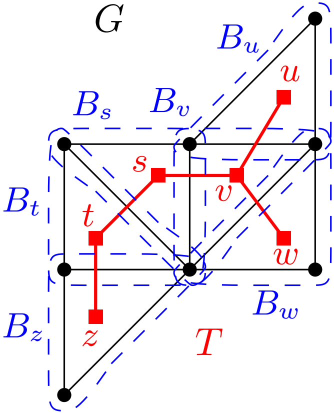
  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq B_t$.
  - $\forall v \in V(G)$, bags containing v define a connected subtree of T.

- Width of a tree decomposition:

  $\max_{t \in V(T)} |B_t| - 1$.

- Treewidth of a graph G: minimum width of a tree decomposition of G.



Why treewidth?

Treewidth is important for (at least) 3 different reasons:

Why treewidth?

Treewidth is important for (at least) 3 different reasons:

1. Treewidth is a fundamental combinatorial tool in graph theory: key role in the Graph Minors project of Robertson and Seymour.

Why treewidth?

Treewidth is important for (at least) 3 different reasons:

1. Treewidth is a fundamental combinatorial tool in graph theory: key role in the Graph Minors project of Robertson and Seymour.
2. Treewidth behaves very well algorithmically, and algorithms parameterized by treewidth appear very often in FPT algorithms.

Why treewidth?

Treewidth is important for (at least) 3 different reasons:

1. Treewidth is a fundamental combinatorial tool in graph theory: key role in the Graph Minors project of Robertson and Seymour.
2. Treewidth behaves very well algorithmically, and algorithms parameterized by treewidth appear very often in FPT algorithms.
3. In many practical scenarios, it turns out that the treewidth of the associated graph is small (programming languages, road networks, ...).

Next section is...

Treewidth behaves very well algorithmically

Treewidth behaves very well algorithmically

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S)$: $[\ \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G)\ ]$

Treewidth behaves very well algorithmically

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S)$: $[\ \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G)\ ]$

Theorem (Courcelle. 1990)

Every problem expressible in MSOL can be solved in time $f(\text{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

In parameterized complexity: FPT parameterized by treewidth.

Treewidth behaves very well algorithmically

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S)$: $[\ \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G)\ ]$

Theorem (Courcelle. 1990)

Every problem expressible in MSOL can be solved in time $f(\mathrm{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

In parameterized complexity: FPT parameterized by treewidth.

Examples: VERTEX COVER, DOMINATING SET, HAMILTONIAN CYCLE, CLIQUE, INDEPENDENT SET, k-COLORING for fixed k, ...

Only good news?

Only good news?

1. Are all “natural” graph problems FPT parameterized by treewidth?

Only good news?

1. Are all "natural" graph problems FPT parameterized by treewidth?

   The vast majority, but not all of them:

   - LIST COLORING is W[1]-hard parameterized by treewidth.

     [Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]

Only good news?

1. Are all "natural" graph problems FPT parameterized by treewidth?

   The vast majority, but not all of them:

   - List Coloring is W[1]-hard parameterized by treewidth.

     [Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]

   - Some problems involving weights or colors are even NP-hard on graphs of constant treewidth (even on trees!).

Only good news?

1. Are all “natural” graph problems FPT parameterized by treewidth?

   The vast majority, but not all of them:

   - List Coloring is W[1]-hard parameterized by treewidth.

     [Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]

   - Some problems involving weights or colors are even NP-hard on graphs of constant treewidth (even on trees!).

2. For the problems that are FPT parameterized by treewidth, what about the existence of polynomial kernels?

Only good news?

1. Are all "natural" graph problems FPT parameterized by treewidth?

   The vast majority, but not all of them:

   - List Coloring is W[1]-hard parameterized by treewidth.

     [Fellows, Fomin, Lokshtanov, Rosamond, Saurabh, Szeider, Thomassen. 2007]

   - Some problems involving weights or colors are even NP-hard on graphs of constant treewidth (even on trees!).

2. For the problems that are FPT parameterized by treewidth, what about the existence of polynomial kernels?

   Most natural problems (Vertex Cover, Dominating Set, ...) do not admit polynomial kernels parameterized by treewidth.

Is it enough to prove that a problem is FPT?

Typically, Courcelle's theorem allows to prove that a problem is FPT...

$$f(\mathrm{tw}) \cdot n^{\mathcal{O}(1)}$$

Is it enough to prove that a problem is FPT?

Typically, Courcelle's theorem allows to prove that a problem is FPT...
... but the running time can (and must) be huge!

$$f(\mathrm{tw}) \cdot n^{\mathcal{O}(1)} = 2^{3^{4^{5^{6^{7^{8^{\mathrm{tw}}}}}}}} \cdot n^{\mathcal{O}(1)}$$

Is it enough to prove that a problem is FPT?

Typically, Courcelle's theorem allows to prove that a problem is FPT...

... but the running time can (and must) be huge!

$$f(\mathrm{tw}) \cdot n^{\mathcal{O}(1)} = 2^{3^{4^{5^{6^{7^{8^{\mathrm{tw}}}}}}}} \cdot n^{\mathcal{O}(1)}$$

$\boxed{\text{Major goal}}$ find the smallest possible function $f(\mathrm{tw})$.

This is a very active area in parameterized complexity.

Is it enough to prove that a problem is FPT?

Typically, Courcelle's theorem allows to prove that a problem is FPT...

... but the running time can (and must) be huge!

$$f(\mathrm{tw}) \cdot n^{\mathcal{O}(1)} = 2^{3^{4^{5^{6^{7^{8^{\mathrm{tw}}}}}}}} \cdot n^{\mathcal{O}(1)}$$

$\boxed{\text{Major goal}}$ find the smallest possible function $f(\mathrm{tw})$.

This is a very active area in parameterized complexity.

Remark: Algorithms parameterized by treewidth appear very often as a "black box" in all kinds of parameterized algorithms.

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$?

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

[Impagliazzo, Paturi. 1999]

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

$\boxed{\text{ETH: The 3-SAT problem on } n \text{ variables cannot be solved in time } 2^{o(n)}}$

$\boxed{\text{SETH: The SAT problem on } n \text{ variables cannot be solved in time } (2-\varepsilon)^n}$

[Impagliazzo, Paturi. 1999]

$$\text{SETH} \Rightarrow \text{ETH}$$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

[Impagliazzo, Paturi. 1999]

$$\text{SETH} \Rightarrow \text{ETH} \Rightarrow \text{FPT} \neq \text{W}[1]$$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

[Impagliazzo, Paturi. 1999]

$$\text{SETH} \Rightarrow \text{ETH} \Rightarrow \mathsf{FPT} \neq \mathsf{W[1]} \Rightarrow \mathsf{P} \neq \mathsf{NP}$$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

  Is it possible to obtain an FPT algorithm in time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

[Impagliazzo, Paturi. 1999]

$$\text{SETH} \Rightarrow \text{ETH} \Rightarrow \text{FPT} \neq \text{W[1]} \Rightarrow \text{P} \neq \text{NP}$$

Typical statements:

ETH $\Rightarrow$ k-Vertex Cover cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$.

ETH $\Rightarrow$ Planar k-Vertex Cover cannot in time $2^{o(\sqrt{k})} \cdot n^{O(1)}$.

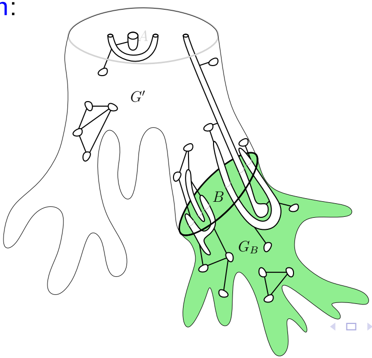
- Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.

Dynamic programming on tree decompositions

- Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.
- Starting from the leaves of the tree decomposition, a set of appropriately defined partial solutions is computed recursively until the root, where a global solution is obtained.

Dynamic programming on tree decompositions

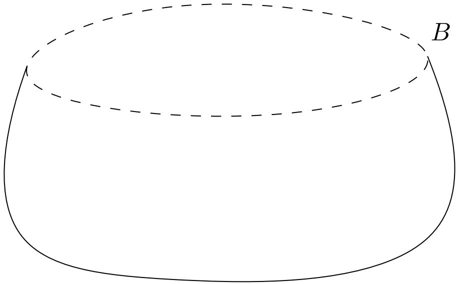
- Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.
- Starting from the leaves of the tree decomposition, a set of appropriately defined partial solutions is computed recursively until the root, where a global solution is obtained.
- The way that these partial solutions are defined depends on each particular problem:



[Figure by Valentin Garnero]

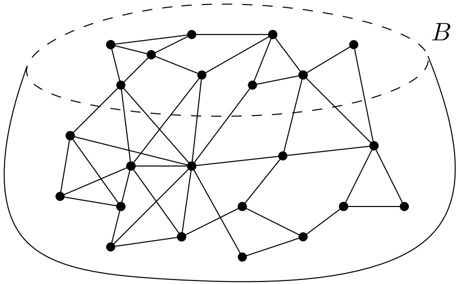
Two behaviors for problems parameterized by treewidth

Local problems: Vertex Cover, Dominating Set, Clique, Independent Set, q-Coloring for fixed q.



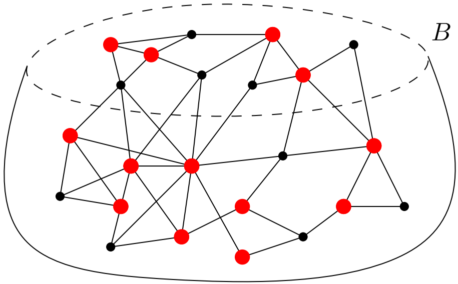
Two behaviors for problems parameterized by treewidth

Local problems VERTEX COVER, DOMINATING SET, CLIQUE, INDEPENDENT SET, q-COLORING for fixed q.



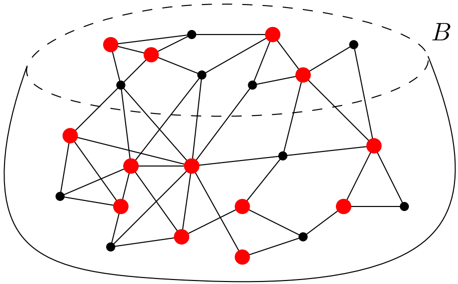
Two behaviors for problems parameterized by treewidth

Local problems: VERTEX COVER, DOMINATING SET, CLIQUE, INDEPENDENT SET, q-COLORING for fixed q.



Two behaviors for problems parameterized by treewidth

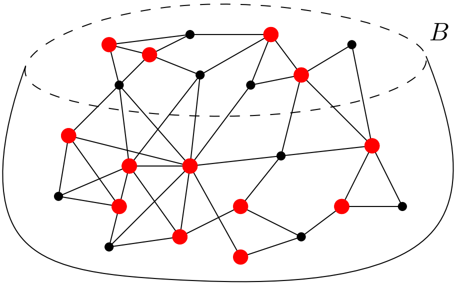
Local problems VERTEX COVER, DOMINATING SET, CLIQUE, INDEPENDENT SET, q-COLORING for fixed q.



- It is sufficient to store, for each bag B, the subset of vertices of B that belong to a partial solution: $\boxed{2^{\mathrm{tw}} \text{ choices}}$

Two behaviors for problems parameterized by treewidth

Local problems: VERTEX COVER, DOMINATING SET, CLIQUE, INDEPENDENT SET, q-COLORING for fixed q.

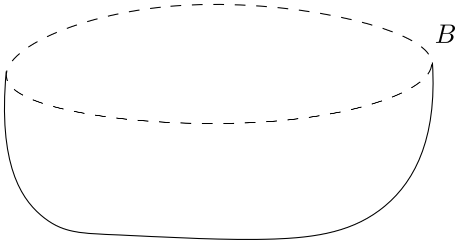


- It is sufficient to store, for each bag B, the subset of vertices of B that belong to a partial solution: 2^{tw} choices
- The "natural" DP algorithms lead to (optimal) single-exponential algorithms:

$$2^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}.$$

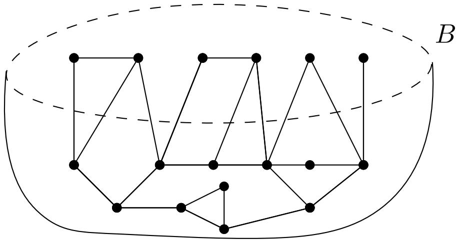
Connectivity problems seem to be more complicated...

$\boxed{\text{Connectivity problems}}$ Hamiltonian Cycle, Longest Path, Steiner Tree, Connected Vertex Cover.



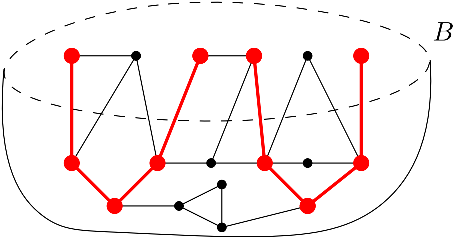
Connectivity problems seem to be more complicated...

Connectivity problems HAMILTONIAN CYCLE, LONGEST CYCLE, STEINER TREE, CONNECTED VERTEX COVER.



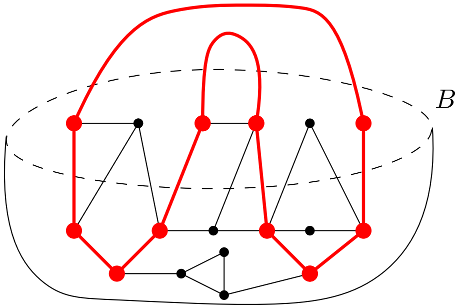
Connectivity problems seem to be more complicated...

Connectivity problems: HAMILTONIAN CYCLE, LONGEST CYCLE, STEINER TREE, CONNECTED VERTEX COVER.



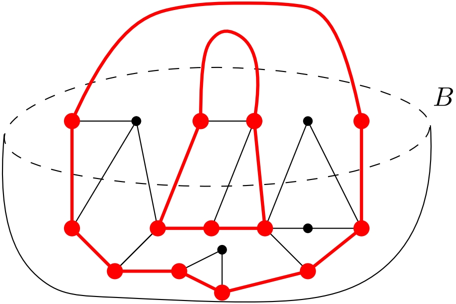
Connectivity problems seem to be more complicated...

Connectivity problems | Hamiltonian Cycle, Longest Cycle, Steiner Tree, Connected Vertex Cover.



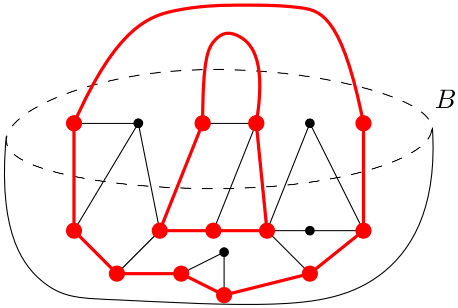
Connectivity problems seem to be more complicated...

Connectivity problems: Hamiltonian Cycle, Longest Cycle, Steiner Tree, Connected Vertex Cover.



Connectivity problems seem to be more complicated...

Connectivity problems HAMILTONIAN CYCLE, LONGEST CYCLE, STEINER TREE, CONNECTED VERTEX COVER.

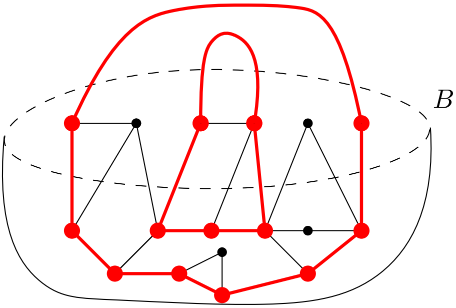


- Now it is not sufficient to store the subset of vertices of B that belong to a partial solution, but also how they are matched:

$\boxed{2^{\mathcal{O}(\text{tw} \log \text{tw})} \text{ choices}}$

Connectivity problems seem to be more complicated...

Connectivity problems HAMILTONIAN CYCLE, LONGEST CYCLE, STEINER TREE, CONNECTED VERTEX COVER.



- Now it is not sufficient to store the subset of vertices of B that belong to a partial solution, but also how they are matched:

  $\boxed{2^{\mathcal{O}(\text{tw} \log \text{tw})} \text{ choices}}$

- The "natural" DP algorithms provide only time $2^{\mathcal{O}(\text{tw} \cdot \log \text{tw})} \cdot n^{\mathcal{O}(1)}$.

Two types of behavior

There seem to be two behaviors for problems parameterized by treewidth:

- Local problems:

  $$\boxed{2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}}$$

  Vertex Cover, Dominating Set, ...

- Connectivity problems:

  $$\boxed{2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}}$$

  Longest Path, Steiner Tree, ...

Single-exponential algorithms on sparse graphs

On $\boxed{\text{topologically structured}}$ graphs (planar, surfaces, minor-free), it is possible to solve connectivity problems in time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$:

- Planar graphs: [Dorn, Penninkx, Bodlaender, Fomin. 2005]
- Graphs on surfaces: [Dorn, Fomin, Thilikos. 2006] [Rué, S., Thilikos. 2010]
- Minor-free graphs: [Dorn, Fomin, Thilikos. 2008] [Rué, S., Thilikos. 2012]

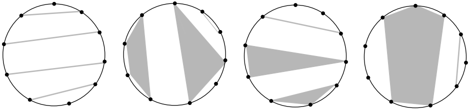
Single-exponential algorithms on sparse graphs

On $\boxed{\text{topologically structured}}$ graphs (planar, surfaces, minor-free), it is possible to solve connectivity problems in time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$:

- Planar graphs: [Dorn, Penninkx, Bodlaender, Fomin. 2005]
- Graphs on surfaces: [Dorn, Fomin, Thilikos. 2006] [Rué, S., Thilikos. 2010]
- Minor-free graphs: [Dorn, Fomin, Thilikos. 2008] [Rué, S., Thilikos. 2012]

$\boxed{\text{Main idea}}$ special type of decomposition with nice topological properties:

partial solutions $\Longleftrightarrow$ non-crossing partitions



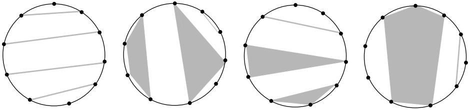
Single-exponential algorithms on sparse graphs

On $\boxed{\text{topologically structured}}$ graphs (planar, surfaces, minor-free), it is possible to solve connectivity problems in time $2^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}$:

- Planar graphs: [Dorn, Penninkx, Bodlaender, Fomin. 2005]
- Graphs on surfaces: [Dorn, Fomin, Thilikos. 2006] [Rué, S., Thilikos. 2010]
- Minor-free graphs: [Dorn, Fomin, Thilikos. 2008] [Rué, S., Thilikos. 2012]

$\boxed{\text{Main idea}}$ special type of decomposition with nice topological properties:

partial solutions $\Longleftrightarrow$ non-crossing partitions



$$\text{CN}(k) \; = \; \frac{1}{k+1}\binom{2k}{k} \; \sim \; \frac{4^k}{\sqrt{\pi}k^{3/2}} \; \leq \; 4^k.$$

The revolution of single-exponential algorithms

It was believed that, except on sparse graphs (planar, surfaces), algorithms in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ were optimal for connectivity problems.

The revolution of single-exponential algorithms

It was believed that, except on sparse graphs (planar, surfaces), algorithms in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ were optimal for connectivity problems.

$\boxed{\text{This was false!!}}$

Cut&Count technique: [Cygan, Nederlof, Pilipczuk^2, van Rooij, Wojtaszczyk. 2011]

Randomized single-exponential algorithms for connectivity problems.

The revolution of single-exponential algorithms

It was believed that, except on sparse graphs (planar, surfaces), algorithms in time $2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ were optimal for connectivity problems.

This was false!!

Cut&Count technique: [Cygan, Nederlof, Pilipczuk^2, van Rooij, Wojtaszczyk. 2011]

Randomized single-exponential algorithms for connectivity problems.

1. Relax the connectivity requirement by considering a set of cuts that contain the relevant (connected) solutions.
2. Count modulo 2 the number of cuts, because the non-connected solutions will cancel out. By assigning random weights to the vertices/edges, guarantee that w.h.p. the optimal solution is unique (Isolation Lemma).

The revolution of single-exponential algorithms

It was believed that, except on sparse graphs (planar, surfaces), algorithms in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ were optimal for connectivity problems.

$\boxed{\text{This was false!!}}$

Cut&Count technique: [Cygan, Nederlof, Pilipczuk^2, van Rooij, Wojtaszczyk. 2011]

Randomized single-exponential algorithms for connectivity problems.

1. Relax the connectivity requirement by considering a set of cuts that contain the relevant (connected) solutions.
2. Count modulo 2 the number of cuts, because the non-connected solutions will cancel out. By assigning random weights to the vertices/edges, guarantee that w.h.p. the optimal solution is unique (Isolation Lemma).

Deterministic algorithms with algebraic tricks: [Bodlaender, Cygan, Kratsch, Nederlof. 2013]

Representative sets in matroids: [Fomin, Lokshtanov, Saurabh. 2014]

End of the story?

Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth?

End of the story?

Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth?

$\boxed{\text{No!}}$

CYCLE PACKING: find the maximum number of vertex-disjoint cycles.

End of the story?

Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth?

$\boxed{\text{No!}}$

Cycle Packing: find the maximum number of vertex-disjoint cycles.

An algorithm in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ is optimal under the ETH.

[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]

End of the story?

Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth?

$\boxed{\text{No!}}$

Cycle Packing: find the maximum number of vertex-disjoint cycles.

An algorithm in time $2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ is optimal under the ETH.

[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]

This reduction uses a framework introduced by [Lokshtanov, Marx, Saurabh. 2011]

End of the story?

Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth?

$\boxed{\text{No!}}$

Cycle Packing: find the maximum number of vertex-disjoint cycles.

An algorithm in time $2^{\mathcal{O}(\text{tw} \cdot \log \text{tw})} \cdot n^{\mathcal{O}(1)}$ is optimal under the ETH.

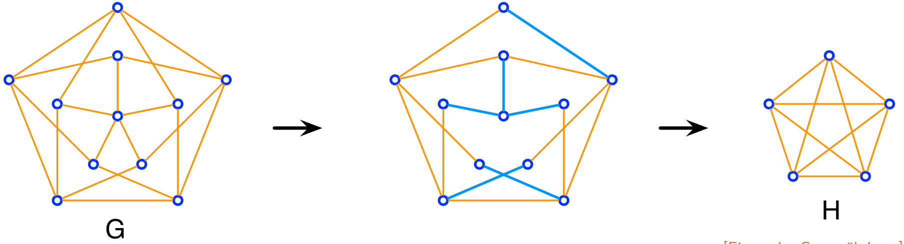
[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]

This reduction uses a framework introduced by [Lokshtanov, Marx, Saurabh. 2011]

There are other examples of such problems...

Next section is...

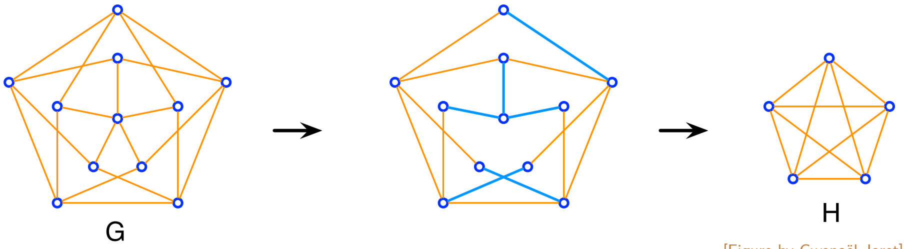
Minors and topological minors



[Figure by Gwenaël Joret]

- H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.

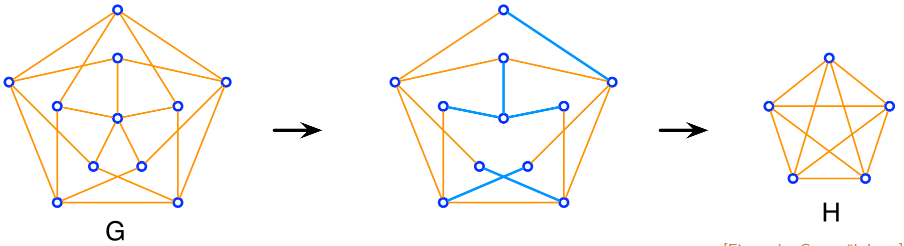
Minors and topological minors



[Figure by Gwenaël Joret]

- H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.
- H is a topological minor of G if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of deg ≤ 2.

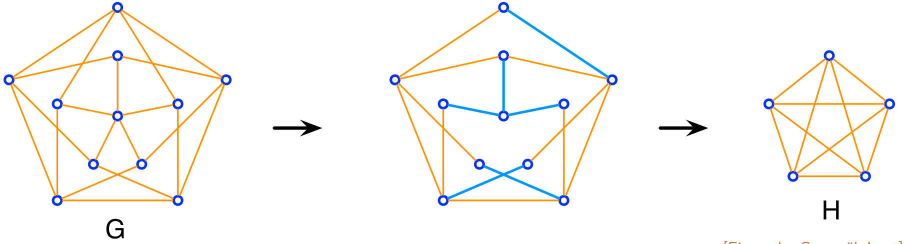
Minors and topological minors



[Figure by Gwenaël Joret]

- H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.
- H is a topological minor of G if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of deg ≤ 2.
- Therefore: $\boxed{H \text{ topological minor of } G \Rightarrow H \text{ minor of } G}$

Minors and topological minors



[Figure by Gwenaël Joret]

- H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.
- H is a topological minor of G if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of deg ≤ 2.
- Therefore: $\boxed{H \text{ topological minor of } G \not\Leftarrow H \text{ minor of } G}$

The $\mathcal{F}$-M-Deletion problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

The $\mathcal{F}$-M-Deletion problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-Deletion

Input: A graph G and an integer k.

Parameter: The treewidth tw of G.

Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

The $\mathcal{F}$-M-DELETION problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-DELETION	
Input:	A graph G and an integer k.
Parameter:	The treewidth tw of G.
Question:	Does G contain a set $S \subseteq V(G)$ with $\lvert S\rvert \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.

The $\mathcal{F}$-M-Deletion problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-Deletion	
Input:	A graph G and an integer k.
Parameter:	The treewidth tw of G.
Question:	Does G contain a set $S \subseteq V(G)$ with $\lvert S \rvert \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: Vertex Cover.
  Easily solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

The $\mathcal{F}$-M-Deletion problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-Deletion	
Input:	A graph G and an integer k.
Parameter:	The treewidth tw of G.
Question:	Does G contain a set $S \subseteq V(G)$ with $\lvert S\rvert \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: Vertex Cover.
  Easily solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{C_3\}$: Feedback Vertex Set.

The $\mathcal{F}$-M-DELETION problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-DELETION	
Input:	A graph G and an integer k.
Parameter:	The treewidth $\mathbf{tw}$ of G.
Question:	Does G contain a set $S \subseteq V(G)$ with $\lvert S \rvert \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.
  Easily solvable in time $2^{\Theta(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$.

- $\mathcal{F} = \{C_3\}$: FEEDBACK VERTEX SET.
  "Hardly" solvable in time $2^{\Theta(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$. [Cut&Count. 2011]

The $\mathcal{F}$-M-DELETION problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-DELETION	
Input:	A graph G and an integer k.
Parameter:	The treewidth tw of G.
Question:	Does G contain a set $S \subseteq V(G)$ with $\|S\| \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.
  Easily solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{C_3\}$: FEEDBACK VERTEX SET.
  "Hardly" solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Cut&Count. 2011]
- $\mathcal{F} = \{K_5, K_{3,3}\}$: VERTEX PLANARIZATION.

The $\mathcal{F}$-M-DELETION problem

Let $\mathcal{F}$ be a fixed finite collection of graphs.

$\mathcal{F}$-M-DELETION	
Input:	A graph G and an integer k.
Parameter:	The treewidth tw of G.
Question:	Does G contain a set $S \subseteq V(G)$ with $\|S\| \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.
  Easily solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{C_3\}$: FEEDBACK VERTEX SET.
  "Hardly" solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Cut&Count. 2011]
- $\mathcal{F} = \{K_5, K_{3,3}\}$: VERTEX PLANARIZATION.
  Solvable in time $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Jansen, Lokshtanov, Saurabh. 2014 + Pilipczuk. 2015]

Covering topological minors

$\mathcal{F}$-M-DELETION

Input: A graph G and an integer k.

Parameter: The treewidth tw of G.

Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a minor?

Covering topological minors

$\mathcal{F}$-M-DELETION

Input: A graph G and an integer k.

Parameter: The treewidth tw of G.

Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a minor?

$\mathcal{F}$-TM-DELETION

Input: A graph G and an integer k.

Parameter: The treewidth tw of G.

Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a topol. minor?

Covering topological minors

$\mathcal{F}$-M-Deletion
Input: A graph G and an integer k.
Parameter: The treewidth tw of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a minor?

$\mathcal{F}$-TM-Deletion
Input: A graph G and an integer k.
Parameter: The treewidth tw of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a topol. minor?

- Both problems are NP-hard if $\mathcal{F}$ contains some edge. [Lewis, Yannakakis. 1980]

Covering topological minors

$\mathcal{F}$-M-DELETION
Input: A graph G and an integer k.
Parameter: The treewidth tw of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a minor?

$\mathcal{F}$-TM-DELETION
Input: A graph G and an integer k.
Parameter: The treewidth tw of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a topol. minor?

- Both problems are NP-hard if $\mathcal{F}$ contains some edge. [Lewis, Yannakakis. 1980]
- FPT by Courcelle (or by Graph Minors theory).

Covering topological minors

$\mathcal{F}$-M-DELETION
Input: A graph G and an integer k.
Parameter: The treewidth $\mathbf{tw}$ of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a minor?

$\mathcal{F}$-TM-DELETION
Input: A graph G and an integer k.
Parameter: The treewidth $\mathbf{tw}$ of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any graph in $\mathcal{F}$ as a topol. minor?

- Both problems are NP-hard if $\mathcal{F}$ contains some edge. [Lewis, Yannakakis. 1980]
- FPT by Courcelle (or by Graph Minors theory).
- $\boxed{\text{Goal}}$ find smallest function $f_{\mathcal{F}}$ s.t. $\exists$ algo in time $f_{\mathcal{F}}(\mathbf{tw}) \cdot n^{\mathcal{O}(1)}$.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

[1] Connected collection $\mathcal{F}$: all the graphs are connected.
[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion in time $2^{2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$.

[1] Connected collection $\mathcal{F}$: all the graphs are connected.

[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-DELETION in time $2^{2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected[1] + planar[2]: $\mathcal{F}$-M-DELETION in time $2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.

[1] Connected collection $\mathcal{F}$: all the graphs are connected.

[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion in time $2^{2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected[1] + planar[2]: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.

[1] Connected collection $\mathcal{F}$: all the graphs are connected.

[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion in time $2^{2^{\mathcal{O}(\mathsf{tw}\cdot\log \mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected[1] + planar[2]: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathsf{tw}\cdot\log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.

(For $\mathcal{F}$-TM-Deletion we need: $\mathcal{F}$ contains a subcubic planar graph.)

[1] Connected collection $\mathcal{F}$: all the graphs are connected.

[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion in time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.

- $\mathcal{F}$ connected[1] + planar[2]: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

- G planar + $\mathcal{F}$ connected: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

  (For $\mathcal{F}$-TM-Deletion we need: $\mathcal{F}$ contains a subcubic planar graph.)

- $\mathcal{F}$ connected: $\mathcal{F}$-M/TM-Deletion not in time $2^{o(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ unless the ETH fails, even if G planar.

---

[1] Connected collection $\mathcal{F}$: all the graphs are connected.

[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion in time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected[1] + planar[2]: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

  (For $\mathcal{F}$-TM-Deletion we need: $\mathcal{F}$ contains a subcubic planar graph.)
- $\mathcal{F}$ connected: $\mathcal{F}$-M/TM-Deletion not in time $2^{o(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$
  unless the ETH fails, even if G planar.
- $\mathcal{F} = \{H\}$, H planar + connected:

[1] Connected collection $\mathcal{F}$: all the graphs are connected.

[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

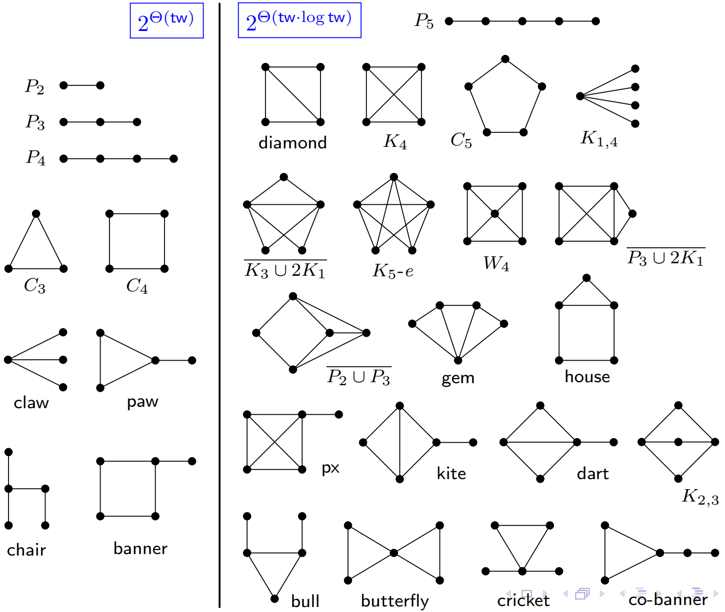
Work with Julien Baste and Dimitrios M. Thilikos (2016-)

- For every $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion in time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected[1] + planar[2]: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathrm{tw}\cdot\log\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

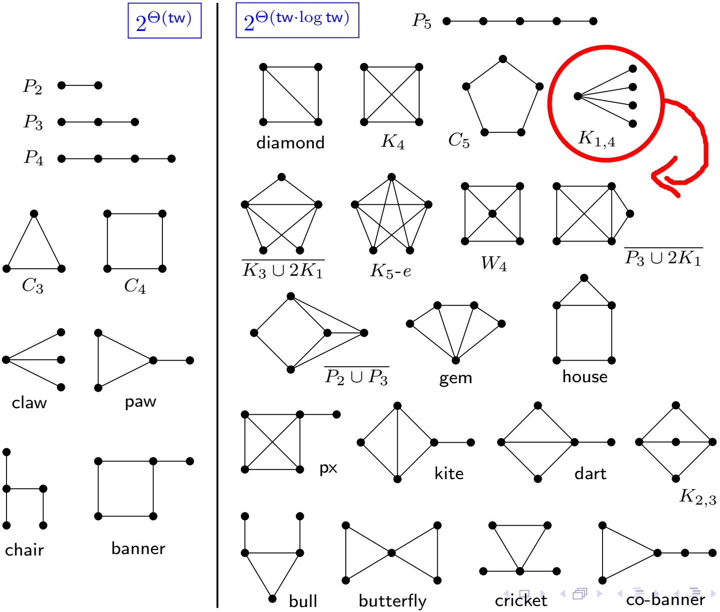
  (For $\mathcal{F}$-TM-Deletion we need: $\mathcal{F}$ contains a subcubic planar graph.)
- $\mathcal{F}$ connected: $\mathcal{F}$-M/TM-Deletion not in time $2^{o(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ unless the ETH fails, even if G planar.
- $\mathcal{F} = \{H\}$, H planar + connected: tight dichotomy (next slide).

[1] Connected collection $\mathcal{F}$: all the graphs are connected.
[2] Planar collection $\mathcal{F}$: contains at least one planar graph.

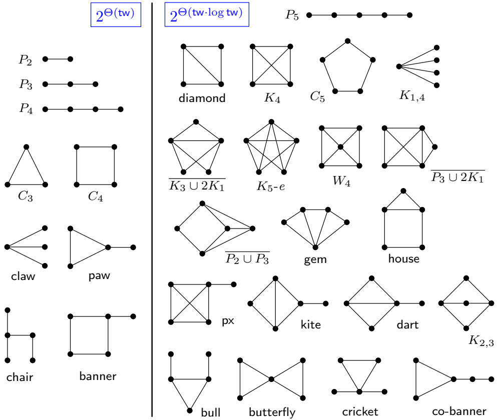
Complexity of hitting small planar minors H



For topological minors, there is only one change

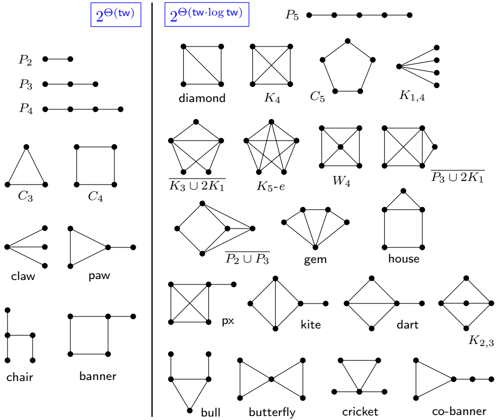


A compact statement for small planar minors

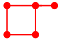


All these cases can be succinctly described as follows:

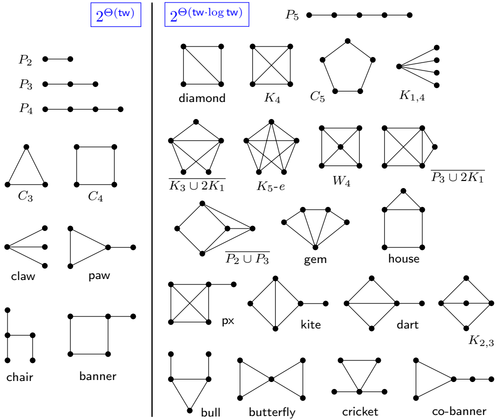
A compact statement for small planar minors



All these cases can be succinctly described as follows:

- All the graphs on the left are minors of 

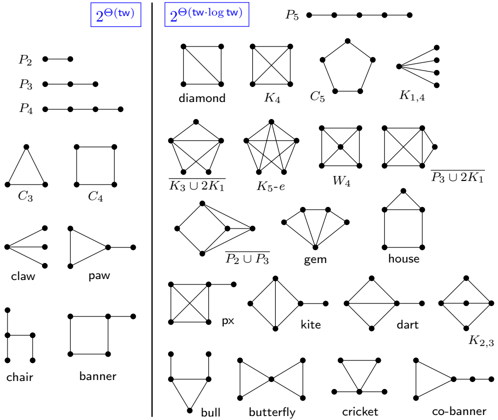
A compact statement for small planar minors



All these cases can be succinctly described as follows:

- All the graphs on the left are minors of 
- All the graphs on the right are not minors of 

A compact statement for small planar minors



All these cases can be succinctly described as follows:

- All the graphs on the left are minors of 
- All the graphs on the right are not minors of  ... except P_5.

A dichotomy for hitting connected planar minors

We can prove that any connected planar H with $|V(H)| \geq 6$ is "hard".

A dichotomy for hitting connected planar minors

We can prove that any connected planar H with $|V(H)| \geq 6$ is "hard".

Theorem (Baste, S., Thilikos. 2018)

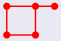
Let H be a connected planar graph.

A dichotomy for hitting connected planar minors

We can prove that any connected planar H with $|V(H)| \geq 6$ is "hard".

Theorem (Baste, S., Thilikos. 2018)

Let H be a connected planar graph.
The $\{H\}$-M-DELETION problem is solvable in time

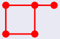
- $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$, *if* $H \preceq_{\mathrm{m}}$  *and* $H \neq P_5$.

A dichotomy for hitting connected planar minors

We can prove that any connected planar H with $|V(H)| \geq 6$ is "hard".

Theorem (Baste, S., Thilikos. 2018)

Let H be a connected planar graph.
The $\{H\}$-M-Deletion problem is solvable in time

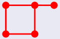
- $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$, *if* $H \preceq_{\mathsf{m}}$  *and* $H \neq P_5$.
- $2^{\mathcal{O}(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$, *otherwise.*

A dichotomy for hitting connected planar minors

We can prove that any connected planar H with $|V(H)| \geq 6$ is "hard".

Theorem (Baste, S., Thilikos. 2018)

Let H be a connected planar graph.
The $\{H\}$-M-DELETION problem is solvable in time

- $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$, *if* $H \preceq_{\mathrm{m}}$  *and* $H \neq P_5$.
- $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$, *otherwise.*

In both cases, the running time is asymptotically optimal under the ETH.

skip

We have three types of results

We have three types of results

1. General algorithms
   - For every $\mathcal{F}$: time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
   - $\mathcal{F}$ connected + planar: time $2^{\mathcal{O}(\mathrm{tw}\cdot\log\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
   - G planar + $\mathcal{F}$ connected: time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

We have three types of results

1. General algorithms

- For every $\mathcal{F}$: time $2^{2^{\mathcal{O}(\text{tw} \cdot \log \text{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected + planar: time $2^{\mathcal{O}(\text{tw} \cdot \log \text{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: time $2^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}$.

2. Ad-hoc single-exponential algorithms

- Some use "typical" dynamic programming.
- Some use the rank-based approach. [Bodlaender, Cygan, Kratsch, Nederlof. 2013]

We have three types of results

1. General algorithms
   - For every $\mathcal{F}$: time $2^{2^{\mathcal{O}(\text{tw} \cdot \log \text{tw})}} \cdot n^{\mathcal{O}(1)}$.
   - $\mathcal{F}$ connected + planar: time $2^{\mathcal{O}(\text{tw} \cdot \log \text{tw})} \cdot n^{\mathcal{O}(1)}$.
   - G planar + $\mathcal{F}$ connected: time $2^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}$.
2. Ad-hoc single-exponential algorithms
   - Some use "typical" dynamic programming.
   - Some use the rank-based approach. [Bodlaender, Cygan, Kratsch, Nederlof. 2013]
3. Lower bounds under the ETH
   - $2^{o(\text{tw})}$ is "easy".
   - $2^{o(\text{tw} \cdot \log \text{tw})}$ is much more involved and we get ideas from:

     [Lokshtanov, Marx, Saurabh. 2011] [Marcin Pilipczuk. 2017] [Bonnet, Brettell, Kwon, Marx. 2017]

skip

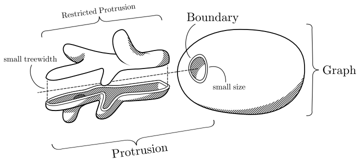
Some ideas of the general algorithms

- For every $\mathcal{F}$: time $2^{2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected + planar: time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

Some ideas of the general algorithms

- For every $\mathcal{F}$: time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F}$ connected + planar: time $2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- G planar + $\mathcal{F}$ connected: time $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

We build on the machinery of boundaried graphs and representatives:



[Bodlaender, Fomin, Lokshtanov, Penninkx, Saurabh, Thilikos. 2009]

[Fomin, Lokshtanov, Saurabh, Thilikos. 2010]

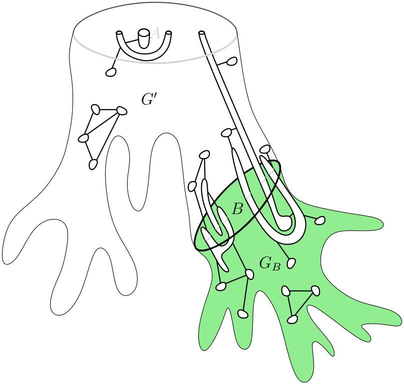
[Kim, Langer, Paul, Reidl, Rossmanith, S., Sikdar. 2013]

[Garnero, Paul, S., Thilikos. 2014]

skip

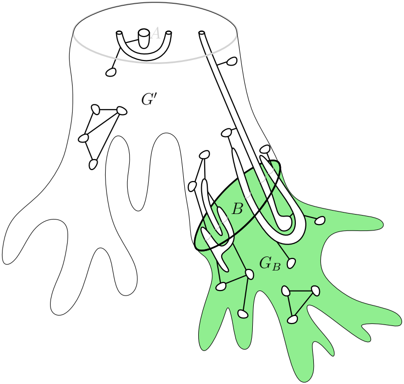
Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.



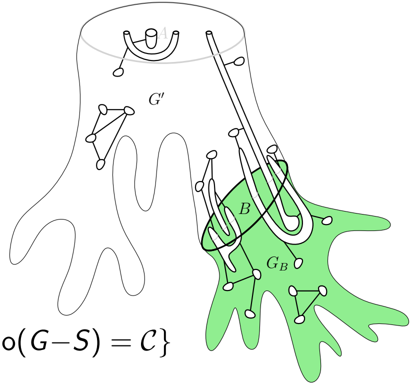
Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.
- folio of G: set of all its $\mathcal{F}$-minor-free minors, up to size $\mathcal{O}(t)$.



Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.
- folio of G: set of all its $\mathcal{F}$-minor-free minors, up to size $\mathcal{O}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:

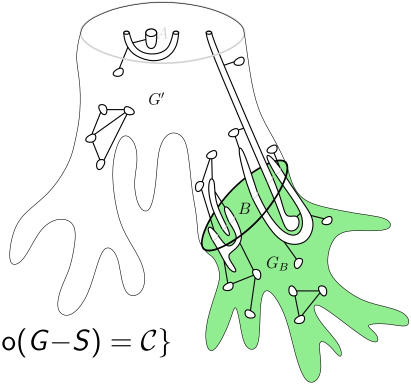


$$\mathbf{p}(G,\mathcal{C}) = \min\{|S| \ : \ S \subseteq V(G) \wedge \text{folio}(G-S) = \mathcal{C}\}$$

Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.
- folio of G: set of all its $\mathcal{F}$-minor-free minors, up to size $\mathcal{O}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:

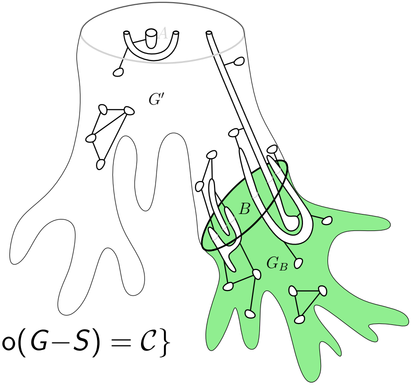
$$\mathbf{p}(G,\mathcal{C}) = \min\{|S| \ : \ S \subseteq V(G) \wedge \text{folio}(G-S) = \mathcal{C}\}$$



- For every t-boundaried graph G, $|\text{folio}(G)| = 2^{\mathcal{O}_{\mathcal{F}}(t \log t)}$.

Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.
- folio of G: set of all its $\mathcal{F}$-minor-free minors, up to size $\mathcal{O}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:

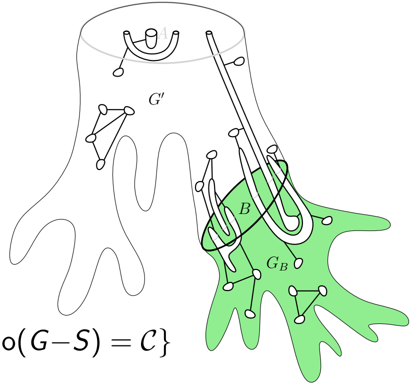


$$\mathbf{p}(G, \mathcal{C}) = \min\{|S| \;:\; S \subseteq V(G) \wedge \text{folio}(G - S) = \mathcal{C}\}$$

- For every t-boundaried graph G, $|\text{folio}(G)| = 2^{\mathcal{O}_{\mathcal{F}}(t \log t)}$.
- The number of distinct folios is $2^{2^{\mathcal{O}_{\mathcal{F}}(t \log t)}}$.

Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.
- folio of G: set of all its $\mathcal{F}$-minor-free minors, up to size $\mathcal{O}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:

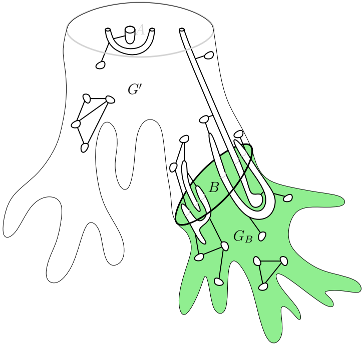


$$\mathbf{p}(G, \mathcal{C}) = \min\{|S| \ : \ S \subseteq V(G) \wedge \text{folio}(G - S) = \mathcal{C}\}$$

- For every t-boundaried graph G, $|\text{folio}(G)| = 2^{\mathcal{O}_{\mathcal{F}}(t \log t)}$.
- The number of distinct folios is $2^{2^{\mathcal{O}_{\mathcal{F}}(t \log t)}}$.
- This gives an algorithm running in time $2^{2^{\mathcal{O}_{\mathcal{F}}(\text{tw} \cdot \log \text{tw})}} \cdot n^{\mathcal{O}(1)}$.

skip

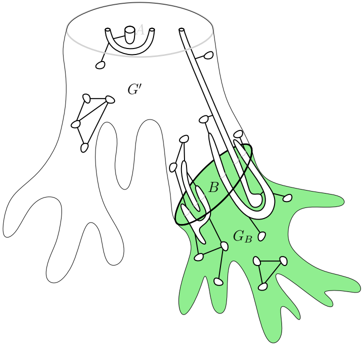
Algorithm for a connected and planar collection $\mathcal{F}$



Algorithm for a connected and planar collection $\mathcal{F}$

- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

$$G_1 \equiv^{(\mathcal{F},t)} G_2 \quad \text{if } \forall G' \in \mathcal{B}^t,$$
$$\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2.$$

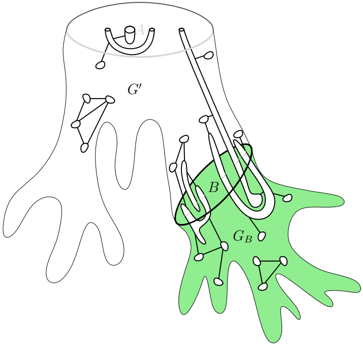


Algorithm for a connected and planar collection $\mathcal{F}$

- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

$$G_1 \equiv^{(\mathcal{F},t)} G_2 \quad \text{if } \forall G' \in \mathcal{B}^t,$$
$$\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2.$$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.

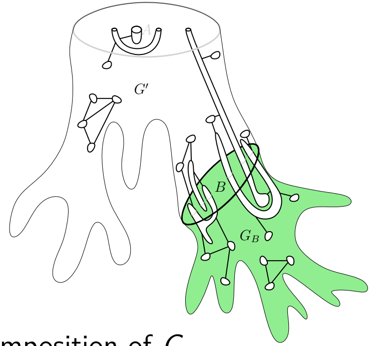


Algorithm for a connected and planar collection $\mathcal{F}$

- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

  $G_1 \equiv^{(\mathcal{F},t)} G_2$ if $\forall G' \in \mathcal{B}^t$,
  $\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2$.

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.



- We compute, using DP over a tree decomposition of G, the following parameter for every representative R:

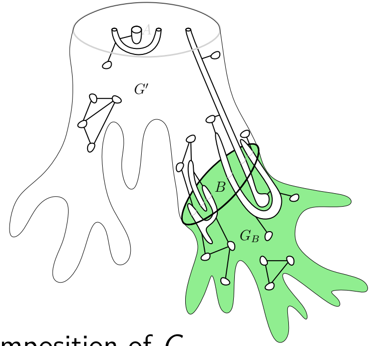
$$\mathbf{p}(G, R) = \min\{|S| \ : \ S \subseteq V(G) \ \wedge \ \mathrm{rep}_{\mathcal{F},t}(G - S) = R\}$$

Algorithm for a connected and planar collection $\mathcal{F}$

- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

$$G_1 \equiv^{(\mathcal{F},t)} G_2 \quad \text{if } \forall G' \in \mathcal{B}^t,$$
$$\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2.$$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.



- We compute, using DP over a tree decomposition of G, the following parameter for every representative R:

$$\mathbf{p}(G,R) = \min\{|S| \ : \ S \subseteq V(G) \ \wedge \ \mathrm{rep}_{\mathcal{F},t}(G - S) = R\}$$

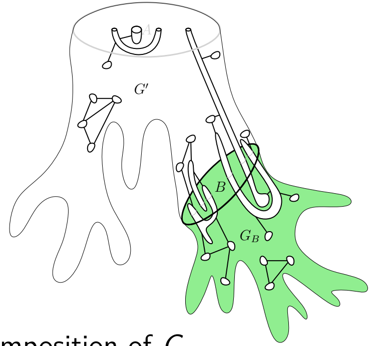
- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$.

Algorithm for a connected and planar collection $\mathcal{F}$

- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

$$G_1 \equiv^{(\mathcal{F},t)} G_2 \quad \text{if } \forall G' \in \mathcal{B}^t,$$
$$\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2.$$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.

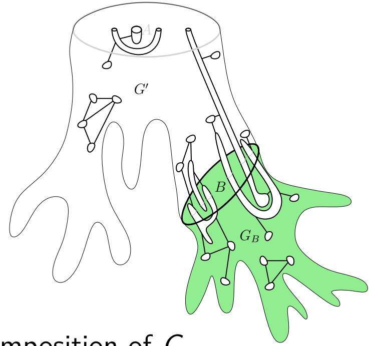


- We compute, using DP over a tree decomposition of G, the following parameter for every representative R:

$$\mathbf{p}(G,R) = \min\{|S| \ : \ S \subseteq V(G) \ \wedge \ \mathrm{rep}_{\mathcal{F},t}(G-S) = R\}$$

- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$.
  $\#$ labeled graphs of size $\leq t$ and tw $\leq h$ is $2^{\mathcal{O}_h(t \cdot \log t)}$. [Baste, Noy, S. 2017]

Algorithm for a connected and planar collection $\mathcal{F}$



- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

  $$G_1 \equiv^{(\mathcal{F},t)} G_2 \quad \text{if } \forall G' \in \mathcal{B}^t,$$
  $$\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2.$$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.

- We compute, using DP over a tree decomposition of G, the following parameter for every representative R:

  $$\mathbf{p}(G,R) = \min\{|S| \ : \ S \subseteq V(G) \ \wedge \ \mathrm{rep}_{\mathcal{F},t}(G-S) = R\}$$

- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$.
  $\#$ labeled graphs of size $\leq t$ and tw $\leq h$ is $2^{\mathcal{O}_h(t \cdot \log t)}$. [Baste, Noy, S. 2017]

- This gives an algorithm running in time $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.

Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.
- We use a sphere-cut decomposition of the input planar graph G.

  [Seymour, Thomas. 1994] [Dorn, Penninkx, Bodlaender, Fomin. 2010]

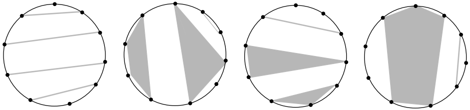
Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.

- We use a sphere-cut decomposition of the input planar graph G.

  [Seymour, Thomas. 1994] [Dorn, Penninkx, Bodlaender, Fomin. 2010]

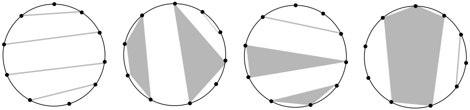
- Nice topological properties: each separator corresponds to a noose.



Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.
- We use a sphere-cut decomposition of the input planar graph G.

  [Seymour, Thomas. 1994] [Dorn, Penninkx, Bodlaender, Fomin. 2010]
- Nice topological properties: each separator corresponds to a noose.



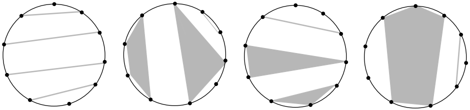
- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t)}$.

  Number of planar triangulations on t vertices is $2^{\mathcal{O}(t)}$. [Tutte. 1962]

Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.
- We use a sphere-cut decomposition of the input planar graph G.

  [Seymour, Thomas. 1994] [Dorn, Penninkx, Bodlaender, Fomin. 2010]
- Nice topological properties: each separator corresponds to a noose.



- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t)}$.

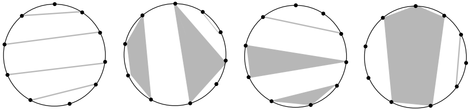
  Number of planar triangulations on t vertices is $2^{\mathcal{O}(t)}$. [Tutte. 1962]
- This gives an algorithm running in time $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.
- We use a sphere-cut decomposition of the input planar graph G.

  [Seymour, Thomas. 1994] [Dorn, Penninkx, Bodlaender, Fomin. 2010]

- Nice topological properties: each separator corresponds to a noose.



- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t)}$.
  Number of planar triangulations on t vertices is $2^{\mathcal{O}(t)}$. [Tutte. 1962]
- This gives an algorithm running in time $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- We can extend this algorithm to input graphs G embedded in arbitrary surfaces by using surface-cut decompositions. [Rué, S., Thilikos. 2014]

Next section is...

1. Introduction
   - Parameterized complexity
   - Treewidth

2. FPT algorithms parameterized by treewidth

3. The $\mathcal{F}$-Deletion problem

4. Conclusions

What's next about $\mathcal{F}$-DELETION?

What's next about $\mathcal{F}$-DELETION?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-DELETION for every family $\mathcal{F}$

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.
- Dichotomy for $\{H\}$-TM-Deletion when H planar + connected.

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.
- Dichotomy for $\{H\}$-TM-Deletion when H planar + connected.
- Only "missing" connected graph on at most 5 vertices: K_5.
  We think that $\{K_5\}$-Deletion is solvable in time $2^{\Theta(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.

- Dichotomy for $\{H\}$-TM-Deletion when H planar + connected.

- Only "missing" connected graph on at most 5 vertices: K_5.
  We think that $\{K_5\}$-Deletion is solvable in time $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.

- We do not even know if there exists some $\mathcal{F}$ such that $\mathcal{F}$-Deletion cannot be solved in time $2^{o(\mathsf{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.

- Dichotomy for $\{H\}$-TM-Deletion when H planar + connected.

- Only "missing" connected graph on at most 5 vertices: K_5.
  We think that $\{K_5\}$-Deletion is solvable in time $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.

- We do not even know if there exists some $\mathcal{F}$ such that $\mathcal{F}$-Deletion cannot be solved in time $2^{o(\mathsf{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.

  Deletion to genus at most g: $2^{\mathcal{O}_g(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.
- Dichotomy for $\{H\}$-TM-Deletion when H planar + connected.
- Only "missing" connected graph on at most 5 vertices: K_5.
We think that $\{K_5\}$-Deletion is solvable in time $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- We do not even know if there exists some $\mathcal{F}$ such that $\mathcal{F}$-Deletion cannot be solved in time $2^{o(\mathrm{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.

  Deletion to genus at most g: $2^{\mathcal{O}_g(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]
- $\boxed{\text{Conjecture}}$ For every connected family $\mathcal{F}$, the $\mathcal{F}$-Deletion problem is solvable in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

What's next about $\mathcal{F}$-Deletion?

- Ultimate goal: classify the (asymptotically) tight complexity of $\mathcal{F}$-Deletion for every family $\mathcal{F}$... we are still far from it.
- Dichotomy for $\{H\}$-TM-Deletion when H planar + connected.
- Only "missing" connected graph on at most 5 vertices: K_5.
  We think that $\{K_5\}$-Deletion is solvable in time $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- We do not even know if there exists some $\mathcal{F}$ such that $\mathcal{F}$-Deletion cannot be solved in time $2^{o(\mathrm{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.

  Deletion to genus at most g: $2^{\mathcal{O}_g(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]
- $\boxed{\text{Conjecture}}$ For every connected family $\mathcal{F}$, the $\mathcal{F}$-Deletion problem is solvable in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- Consider families $\mathcal{F}$ containing disconnected graphs.

Gràcies!

LLIBERTAT
PRESOS POLÍTICS

FREEDOM FOR ALL CATALAN POLITICAL PRISONERS IN SPAIN