Introduction to Parameterized Complexity

Ignasi Sau

CNRS, LIRMM, Université de Montpellier, France

Seminário
UFC, Fortaleza
Fevereiro 2020

Outline of the talk

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

Next section is...

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

Next subsection is...

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

Crucial notion in complexity theory: NP-completeness

P: problems that can be solved in polynomial time.

NP: problems for which a solution can be verified in polynomial time.

Crucial notion in complexity theory: NP-completeness

P: problems that can be solved in polynomial time.

NP: problems for which a solution can be verified in polynomial time.

- Conjecture: P $\neq$ NP.

Crucial notion in complexity theory: NP-completeness

P: problems that can be solved in polynomial time.

NP: problems for which a solution can be verified in polynomial time.

- Conjecture: P $\neq$ NP.
- Cook-Levin Theorem (1971): the SAT problem is NP-complete: unless P $=$ NP, it cannot be solved in polynomial time.

Crucial notion in complexity theory: NP-completeness

P: problems that can be solved in polynomial time.

NP: problems for which a solution can be verified in polynomial time.

- Conjecture: P $\neq$ NP.
- Cook-Levin Theorem (1971): the SAT problem is NP-complete: unless P $=$ NP, it cannot be solved in polynomial time.
- Karp (1972): list of 21 *important* NP-complete problems.

Crucial notion in complexity theory: NP-completeness

P: problems that can be solved in polynomial time.

NP: problems for which a solution can be verified in polynomial time.

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

Crucial notion in complexity theory: NP-completeness

P: problems that can be solved in polynomial time.

NP: problems for which a solution can be verified in polynomial time.

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

No algorithm solves all instances optimally in polynomial time.

Typical approaches to cope with intractability

NP-hard: no algorithm solves all instances optimally in polynomial time.

Typical approaches to cope with intractability

NP-hard: no algorithm solves all instances optimally in polynomial time.

- Approximation algorithms: In polynomial time, find solutions that are "close" to the optimal ones.

Typical approaches to cope with intractability

NP-hard: no algorithm solves all instances optimally in polynomial time.

- Approximation algorithms: In polynomial time, find solutions that are "close" to the optimal ones.
- Moderately exponential-time algorithms: Solve the problem in exponential time, but reasonably "fast" (1.15^n vs 2^n).

Typical approaches to cope with intractability

NP-hard: no algorithm solves all instances optimally in polynomial time.

- Approximation algorithms: In polynomial time, find solutions that are "close" to the optimal ones.
- Moderately exponential-time algorithms: Solve the problem in exponential time, but reasonably "fast" (1.15^n vs 2^n).
- Heuristics: Produce "good" solutions, but with no guarantee.

Typical approaches to cope with intractability

NP-hard: no algorithm solves all instances optimally in polynomial time.

- Approximation algorithms: In polynomial time, find solutions that are "close" to the optimal ones.
- Moderately exponential-time algorithms: Solve the problem in exponential time, but reasonably "fast" (1.15^n vs 2^n).
- Heuristics: Produce "good" solutions, but with no guarantee.
- Parameterized complexity: Topic of this talk...

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:

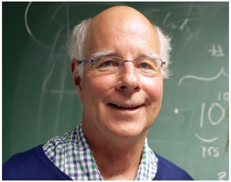
- VLSI design: the number of circuit layers is usually ≤ 10.
- Computational biology: Real instances of DNA chain reconstruction usually have treewidth ≤ 11.
- Robotics: Number of degrees of freedom in motion planning problems ≤ 10.
- Compilers: Checking compatibility of type declarations is hard, but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

$\boxed{\text{Idea}}$ Measure the complexity of an algorithm in terms of the input size and an additional integer 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

In a parameterized problem, an instance is a pair (x, k), where

- x is a typical input (in our setting, a graph).
- k is a positive integer called the parameter.

Parameterized problems

In a parameterized problem, an instance is a pair (x, k), where

- x is a typical input (in our setting, a graph).
- k is a positive integer called the parameter.

Examples of parameterized problems on graphs, with an instance (G, k):

Parameterized problems

In a parameterized problem, an instance is a pair (x, k), where

- x is a typical input (in our setting, a graph).
- k is a positive integer called the parameter.

Examples of parameterized problems on graphs, with an instance (G, k):

1. k-Vertex Cover: Does G contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?

Parameterized problems

In a parameterized problem, an instance is a pair (x, k), where

- x is a typical input (in our setting, a graph).
- k is a positive integer called the parameter.

Examples of parameterized problems on graphs, with an instance (G, k):

1. k-VERTEX COVER: Does G contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?

2. k-CLIQUE: Does G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise adjacent vertices?

Parameterized problems

In a parameterized problem, an instance is a pair (x, k), where

- x is a typical input (in our setting, a graph).
- k is a positive integer called the parameter.

Examples of parameterized problems on graphs, with an instance (G, k):

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

Parameterized problems

In a parameterized problem, an instance is a pair (x, k), where

- x is a typical input (in our setting, a graph).
- k is a positive integer called the parameter.

Examples of parameterized problems on graphs, with an instance (G, k):

1. k-VERTEX COVER: Does G contain a set $S \subseteq V(G)$, with $|S| \leq k$, containing at least an endpoint of every edge?
2. k-CLIQUE: Does G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise adjacent vertices?
3. VERTEX k-COLORING: Can $V(G)$ be colored with $\leq k$ colors, so that adjacent vertices get different colors?

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

They behave quite differently...

1. k-Vertex Cover: solvable in time $2^k \cdot n^2$

2. k-Clique: solvable in time $k^2 \cdot n^k$

3. Vertex k-Coloring: NP-hard for every fixed $k \geq 3$

They behave quite differently...

1. k-Vertex Cover: solvable in time $2^k \cdot n^2 \quad = \quad \boxed{f(k) \cdot n^{\mathcal{O}(1)}}$

2. k-Clique: solvable in time $k^2 \cdot n^k \quad = \quad \boxed{f(k) \cdot n^{g(k)}}$

3. Vertex k-Coloring: NP-hard for every fixed $k \geq 3$

They behave quite differently...

1. k-Vertex Cover: solvable in time $2^k \cdot n^2 \quad = \quad \boxed{f(k) \cdot n^{\mathcal{O}(1)}}$

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

2. k-Clique: solvable in time $k^2 \cdot n^k \quad = \quad \boxed{f(k) \cdot n^{g(k)}}$

3. Vertex k-Coloring: NP-hard for every fixed $k \geq 3$

They behave quite differently...

1. k-Vertex Cover: solvable in time $2^k \cdot n^2 \quad = \quad \boxed{f(k) \cdot n^{\mathcal{O}(1)}}$

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

2. k-Clique: solvable in time $k^2 \cdot n^k \quad = \quad \boxed{f(k) \cdot n^{g(k)}}$

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

3. Vertex k-Coloring: NP-hard for every fixed $k \geq 3$

They behave quite differently...

1. k-Vertex Cover: solvable in time $2^k \cdot n^2 \quad = \quad \boxed{f(k) \cdot n^{\mathcal{O}(1)}}$

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

2. k-Clique: solvable in time $k^2 \cdot n^k \quad = \quad \boxed{f(k) \cdot n^{g(k)}}$

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

3. Vertex k-Coloring: NP-hard for every fixed $k \geq 3$

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

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

So far, nobody has managed to find an FPT algorithm for k-Clique.

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

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

So far, nobody has managed to find an FPT algorithm for k-Clique.

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

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

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

How to transfer hardness among parameterized problems?

Let A, B be two parameterized problems.

How to transfer hardness among parameterized problems?

Let A, B 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

How to transfer hardness among parameterized problems?

Let A, B 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 $g : \mathbb{N} \to \mathbb{N}$.

How to transfer hardness among parameterized problems?

Let A, B 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 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 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 $g : \mathbb{N} \to \mathbb{N}$.

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

$\boxed{\text{Idea}}$ 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

Kernelization

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

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

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

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 $g : \mathbb{N} \to \mathbb{N}$.

Kernelization

$\boxed{\text{Idea}}$ 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

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 $g : \mathbb{N} \rightarrow \mathbb{N}$.

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:

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

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? $\boxed{\text{NO!}}$

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

Parameterized problem L

k-Clique

k-Vertex Cover

k-Path

Vertex k-Coloring

k-Clique

k-Vertex Cover XP

k-Path

para-NP-hard

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

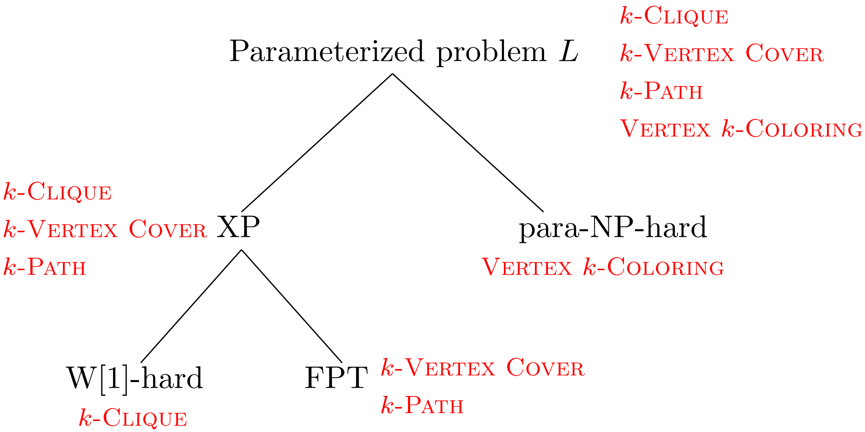
para-NP-hard

Vertex k-Coloring

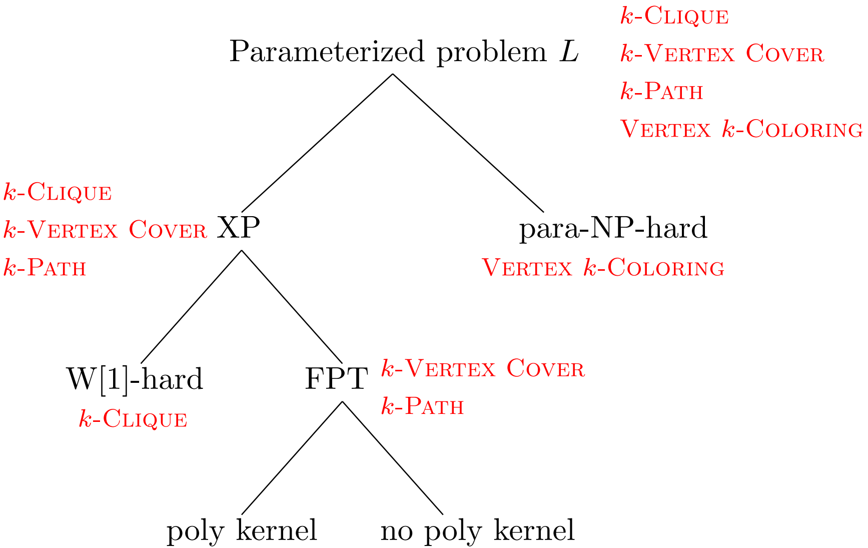
W[1]-hard

FPT

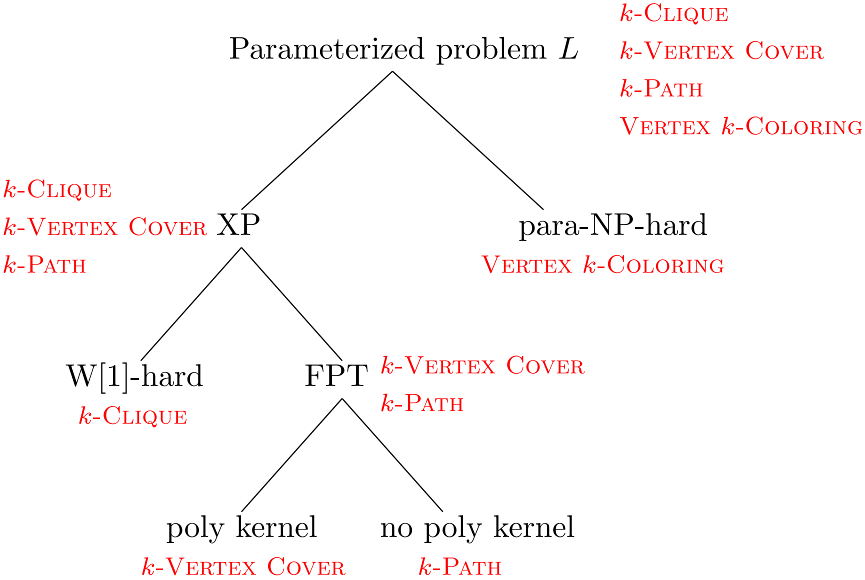
Typical approach to deal with a parameterized problem



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

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

- Finding a k-clique parameterized by k: W[1]-hard.

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

- Finding a k-clique parameterized by k: W[1]-hard.
- Finding a k-clique parameterized by Δ:

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

- Finding a k-clique parameterized by k: W[1]-hard.
- Finding a k-clique parameterized by Δ: FPT.

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

- Finding a k-clique parameterized by k: W[1]-hard.
- Finding a k-clique parameterized by Δ: FPT.

There are mainly two types of parameters:

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

- Finding a k-clique parameterized by k: W[1]-hard.
- Finding a k-clique parameterized by Δ: FPT.

There are mainly two types of parameters:

- Parameters considering characteristics of the desired solution:
  typically, the size of the solution we are looking for.

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

- Finding a k-clique parameterized by k: W[1]-hard.
- Finding a k-clique parameterized by Δ: FPT.

There are mainly two types of parameters:

- Parameters considering characteristics of the desired solution:
  typically, the size of the solution we are looking for.
- Parameters considering structural characteristics of the input graph:
  maximum degree,

The choice of the parameter is crucial!

Every choice of the parameter defines a different parameterized problem:

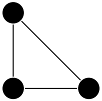
- Finding a k-clique parameterized by k: W[1]-hard.
- Finding a k-clique parameterized by Δ: FPT.

There are mainly two types of parameters:

- Parameters considering characteristics of the desired solution: typically, the size of the solution we are looking for.
- Parameters considering structural characteristics of the input graph: maximum degree, or treewidth.

Treewidth via k-trees

Example of a 2-tree:

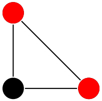


[Figure by Julien Baste]

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

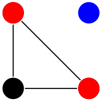
Example of a 2-tree:



[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.

Example of a 2-tree:

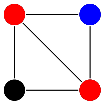


[Figure by Julien Baste]

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

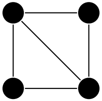
Example of a 2-tree:



[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.

Example of a 2-tree:

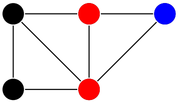


[Figure by Julien Baste]

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:

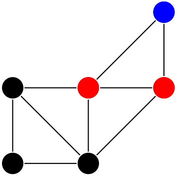


[Figure by Julien Baste]

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:

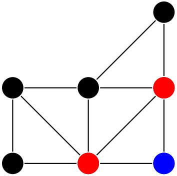


[Figure by Julien Baste]

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:

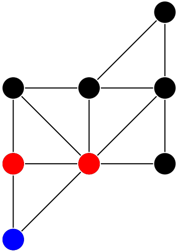


[Figure by Julien Baste]

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:

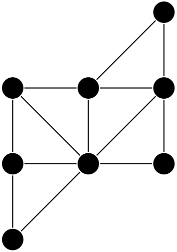


[Figure by Julien Baste]

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:

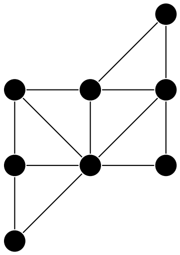


[Figure by Julien Baste]

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:



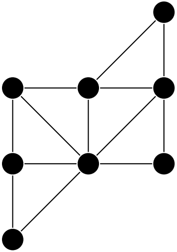
[Figure by Julien Baste]

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]

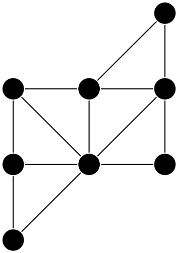
A k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

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.

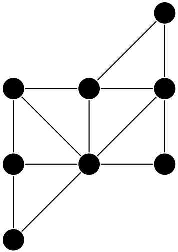
A partial k-tree is a subgraph of a k-tree.

Treewidth of a graph G, denoted $\text{tw}(G)$: 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:

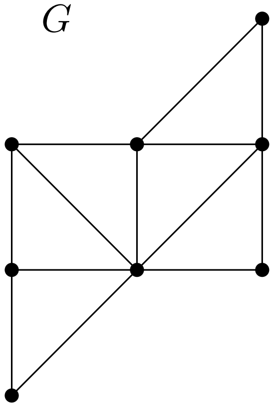
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \ \ \forall t \in V(T)$ (bags),

  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:

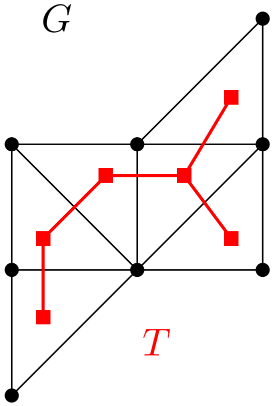
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  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:

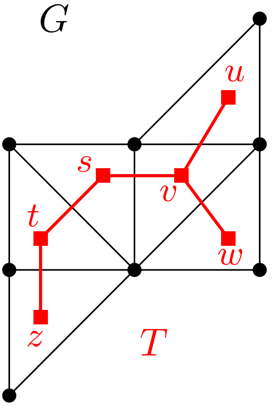
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  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:

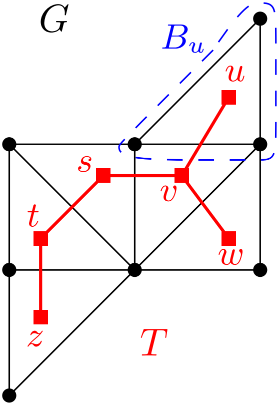
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \ \ \forall t \in V(T)$ (bags),

  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:

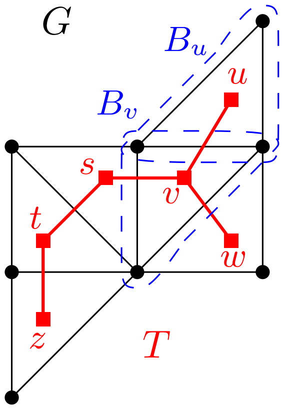
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G)\ \ \forall t \in V(T)$ (bags),

  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),

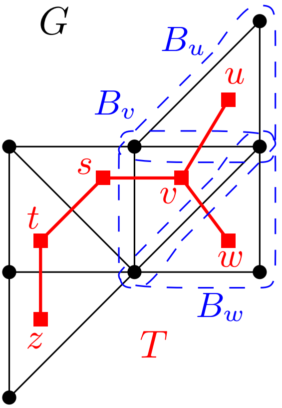
  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G), \exists t \in V(T)$ 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:

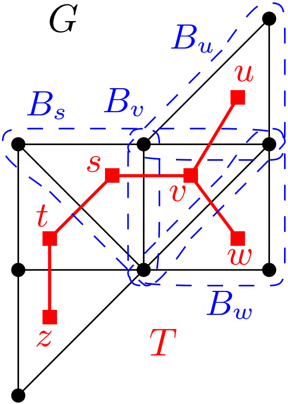
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  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),

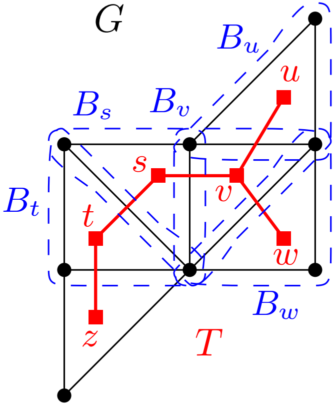
  satisfying the following:

  - $\bigcup_{t \in V(T)} B_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ 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:

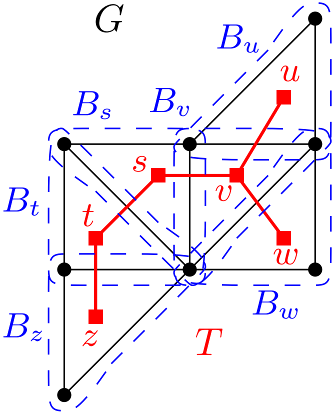
  pair $(T, \{B_t \mid t \in V(T)\})$, where
  T is a tree, and
  $B_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  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...

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

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

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.

Treewidth behaves very well algorithmically

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

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

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]

   - 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!

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

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

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!

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

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.

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!

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

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

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

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

[Impagliazzo, Paturi. 1999]

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]

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

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]

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

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]

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

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 $2^{o(\sqrt{k})} \cdot n^{O(1)}$.

Dynamic programming on tree decompositions

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

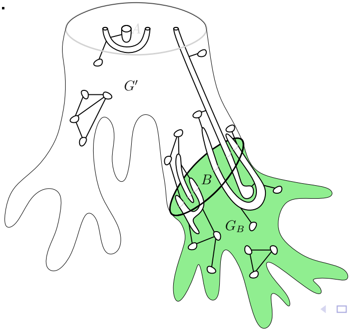
Dynamic programming on tree decompositions

- 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

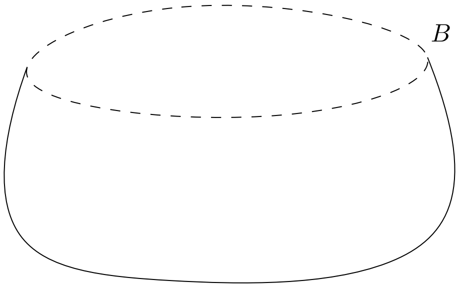
- Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.
- Starting from the leaves of the tree decomposition, a set of appropriately defined partial solutions is computed recursively until the root, where a global solution is obtained.
-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



[Figure by Valentin Garnero]

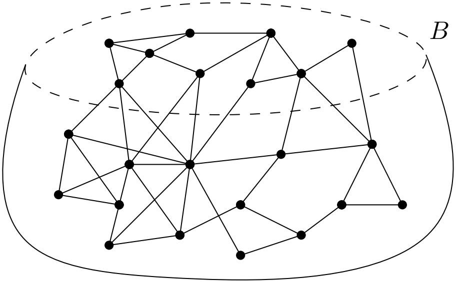
Two behaviors for problems parameterized by treewidth

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



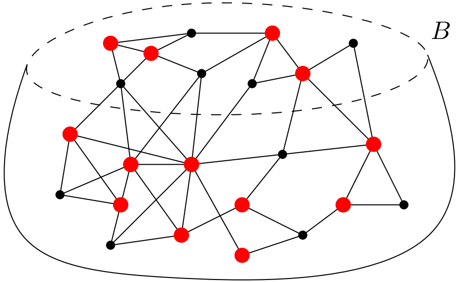
Two behaviors for problems parameterized by treewidth

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



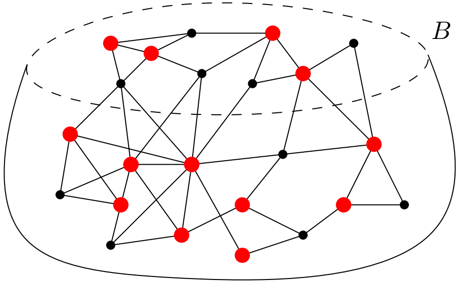
Two behaviors for problems parameterized by treewidth

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



Two behaviors for problems parameterized by treewidth

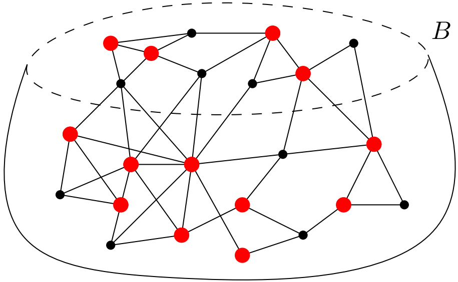
Local problems VERTEX COVER, DOMINATING SET, CLIQUE, INDEPENDENT SET, q-COLORING for fixed q.



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

Two behaviors for problems parameterized by treewidth

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

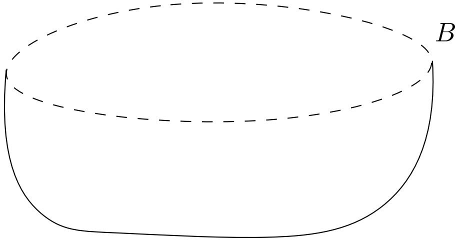


- It is sufficient to store, for each bag B, the subset of vertices of B that belong to a partial solution: 2^{tw} choices
- The "natural" DP algorithms lead to (optimal) single-exponential algorithms:

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

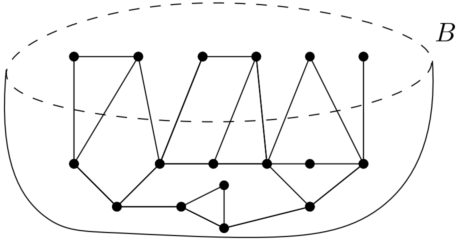
Connectivity problems seem to be more complicated...

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



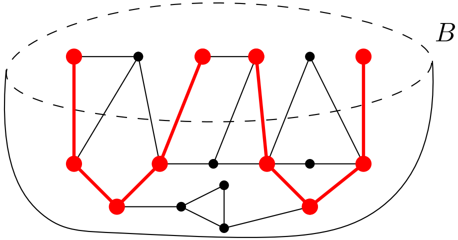
Connectivity problems seem to be more complicated...

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



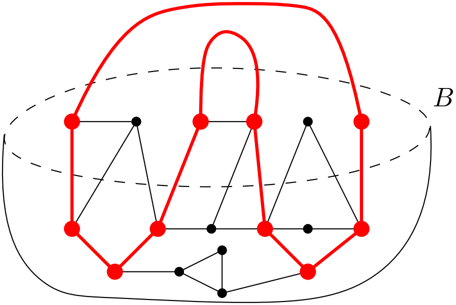
Connectivity problems seem to be more complicated...

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



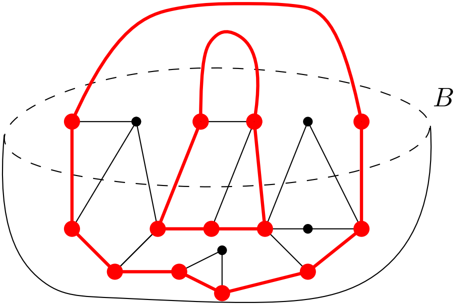
Connectivity problems seem to be more complicated...

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



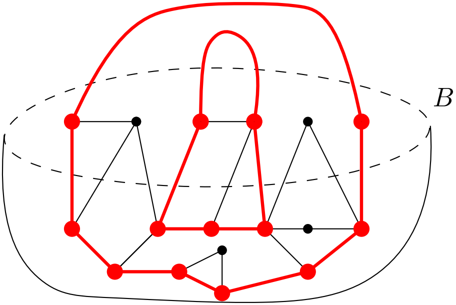
Connectivity problems seem to be more complicated...

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



Connectivity problems seem to be more complicated...

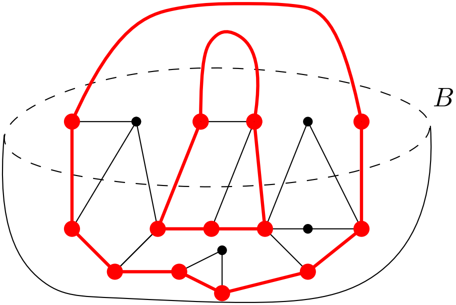
Connectivity problems HAMILTONIAN CYCLE, LONGEST CYCLE, STEINER TREE, CONNECTED VERTEX COVER.



- It is not sufficient to store the subset of vertices of B that belong to a partial solution, but also how they are matched (Bell number):

Connectivity problems seem to be more complicated...

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

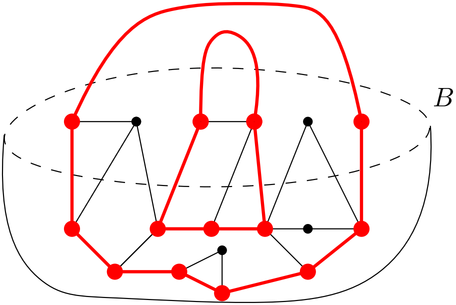


- It is not sufficient to store the subset of vertices of B that belong to a partial solution, but also how they are matched (Bell number):

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

Connectivity problems seem to be more complicated...

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



- It is not sufficient to store the subset of vertices of B that belong to a partial solution, but also how they are matched (Bell number):

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

- 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, ...

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.

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

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).

The revolution of single-exponential algorithms

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

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

Cut&Count technique: [Cygan, Nederlof, Pilipczuk², 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]

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

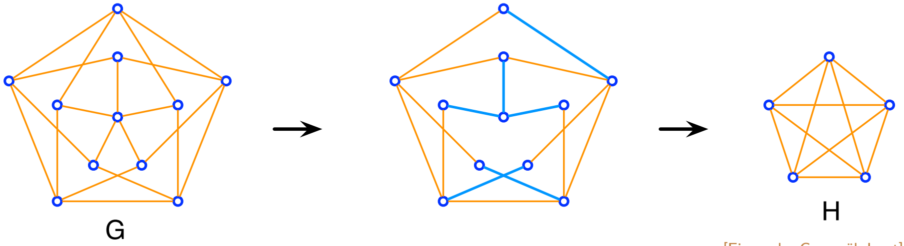
Next section is...

1. Introduction
   - Parameterized complexity
   - Treewidth

2. FPT algorithms parameterized by treewidth

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

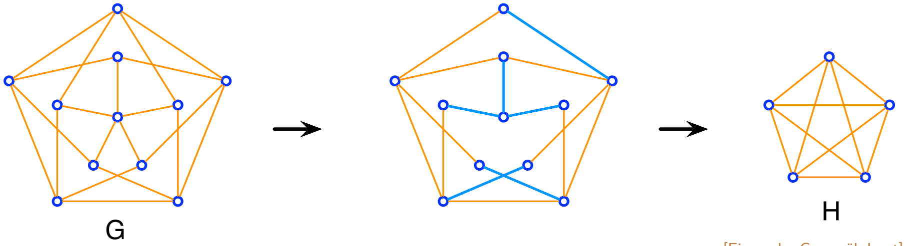
Minors and topological minors



[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.

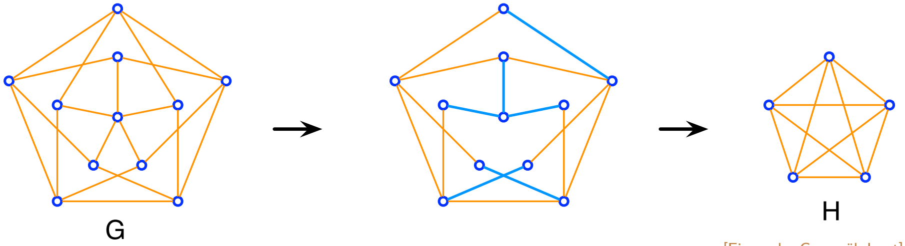
Minors and topological minors



[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



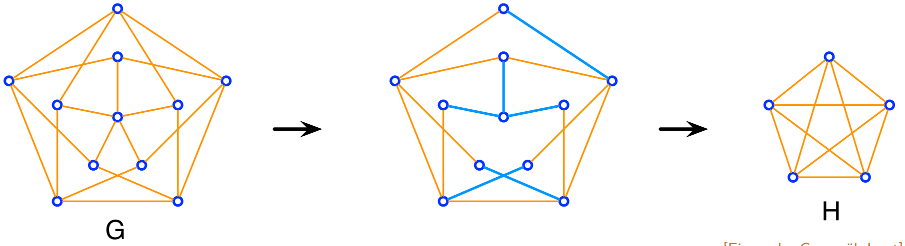
[Figure by Gwenaël Joret]

- 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 $\|S\| \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: Vertex Cover.

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

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 $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{K_3\}$: 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 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 $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{K_3\}$: Feedback Vertex Set.
  "Hardly" solvable in time $2^{\Theta(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [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 $\mathbf{tw}$ 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 $2^{\Theta(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{K_3\}$: Feedback Vertex Set.
  "Hardly" solvable in time $2^{\Theta(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$. [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 $\mathbf{tw}$ 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 $2^{\Theta(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{K_3\}$: Feedback Vertex Set.
  "Hardly" solvable in time $2^{\Theta(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$. [Cut&Count. 2011]
- $\mathcal{F} = \{K_5, K_{3,3}\}$: Vertex Planarization.
  Solvable in time $2^{\Theta(\mathbf{tw} \cdot \log \mathbf{tw})} \cdot n^{\mathcal{O}(1)}$. [Jansen, Lokshtanov, Saurabh. 2014 + Pilipczuk. 2015]

Covering topological minors

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 graph in $\mathcal{F}$ as a minor?

Covering topological minors

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 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

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 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's Theorem.

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

Objective

Determine, for every fixed $\mathcal{F}$, the (asymptotically) smallest function $f_{\mathcal{F}}$ such that $\mathcal{F}$-M-Deletion/$\mathcal{F}$-TM-Deletion can be solved in time

$$f_{\mathcal{F}}(\mathsf{tw}) \cdot n^{\mathcal{O}(1)}$$

on n-vertex graphs.

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

Objective

Determine, for every fixed $\mathcal{F}$, the (asymptotically) smallest function $f_{\mathcal{F}}$ such that $\mathcal{F}$-M-DELETION/$\mathcal{F}$-TM-DELETION can be solved in time

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

on n-vertex graphs.

- We do not want to optimize the degree of the polynomial factor.
- We do not want to optimize the constants.
- Our hardness results hold under the ETH.

Summary of our results: arXiv 1704.07284+1907.04442

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

Summary of our results: arXiv 1704.07284+1907.04442

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

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

- G planar: $\mathcal{F}$-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.)

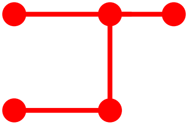
- $\mathcal{F}$: $\mathcal{F}$-M/TM-Deletion not in time $2^{o(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$
  unless the ETH fails, even if G planar.

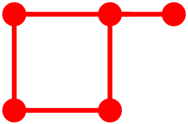
- $\mathcal{F} = \{H\}$, H connected: complete tight dichotomy...

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

A dichotomy for hitting a connected minor

A dichotomy for hitting a connected minor

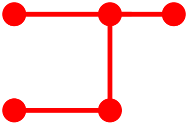


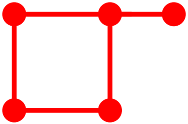


Theorem

Let H be a connected graph.

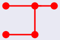
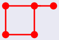
A dichotomy for hitting a connected minor





Theorem

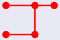
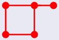
Let H *be a connected graph.*
The $\{H\}$-M-DELETION *problem is solvable in time*

- $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$, *if* $H \preceq_c$  *or* $H \preceq_c$ .

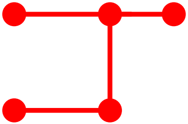
A dichotomy for hitting a connected minor

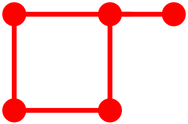
Theorem

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

- $2^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(1)}$, *if* $H \preceq_c$  *or* $H \preceq_c$ .
- $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$, *otherwise.*

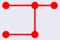
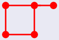
A dichotomy for hitting a connected minor





Theorem

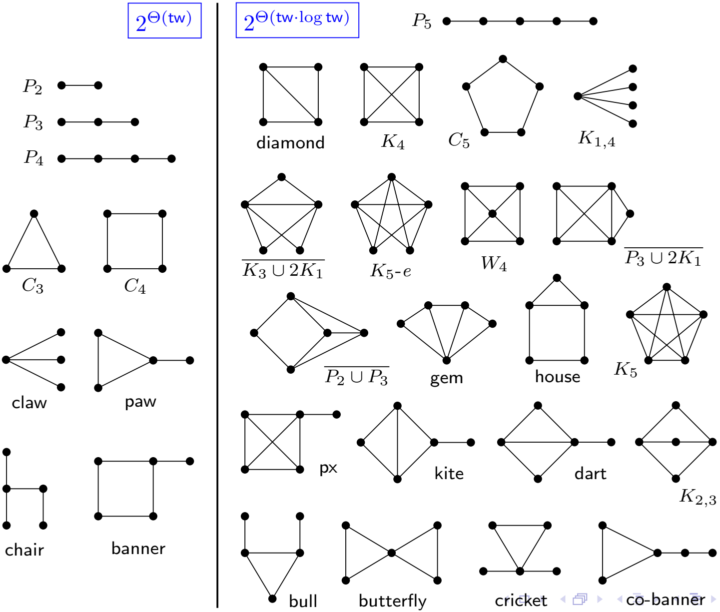
Let H *be a connected graph.*
The $\{H\}$-M-DELETION *problem is solvable in time*

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

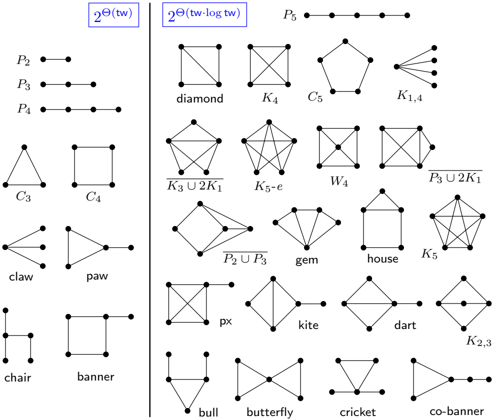
In both cases, the running time is asymptotically optimal under the ETH.

skip

Complexity of hitting a single connected minor H

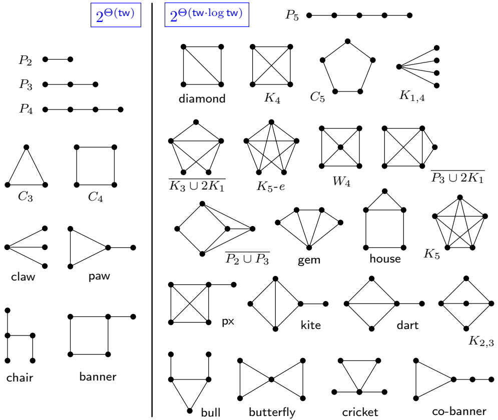


A compact statement for a single connected graph

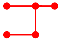


All these cases can be succinctly described as follows:

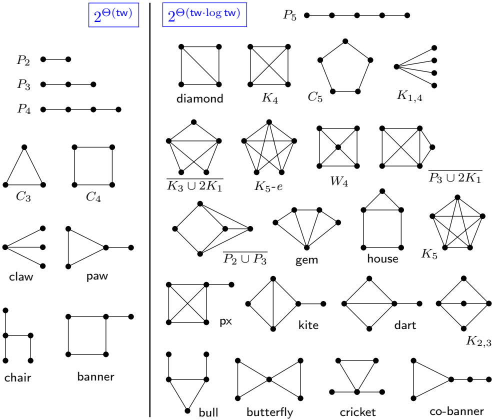
A compact statement for a single connected graph



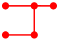
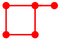
All these cases can be succinctly described as follows:

- All graphs on the left are contractions of  or 

A compact statement for a single connected graph



All these cases can be succinctly described as follows:

- All graphs on the left are contractions of  or 
- All graphs on the right are not contractions of  or 

We have three types of results

We have three types of results

1. $\boxed{\text{General algorithms}}$

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

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
   - $2^{o(\mathrm{tw})}$ is "easy".
   - $2^{o(\mathrm{tw} \cdot \log \mathrm{tw})}$ is much more involved and we get ideas from:

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

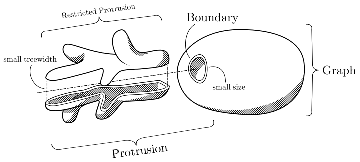
Some ideas of the general algorithms

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

Some ideas of the general algorithms

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

We build on the machinery of boundaried graphs and representatives:



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

[Fomin, Lokshtanov, Saurabh, Thilikos. 2010]

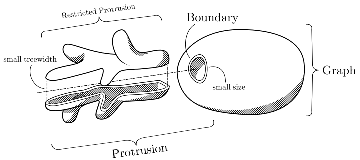
[Kim, Langer, Paul, Reidl, Rossmanith, S., Sikdar. 2013]

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

Some ideas of the general algorithms

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

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]

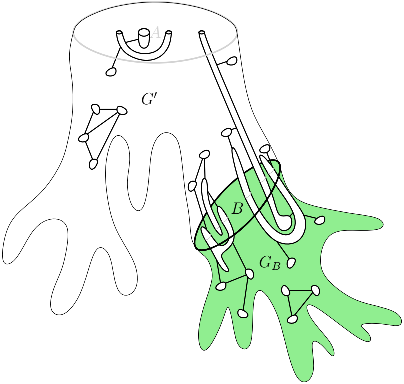
- Every $\mathcal{F}$: time $2^{\mathcal{O}(\text{tw}\cdot\log \text{tw})} \cdot n^{\mathcal{O}(1)}$.

  Extra: Bidimensionality, irrelevant vertices, protrusion decompositions...

skip

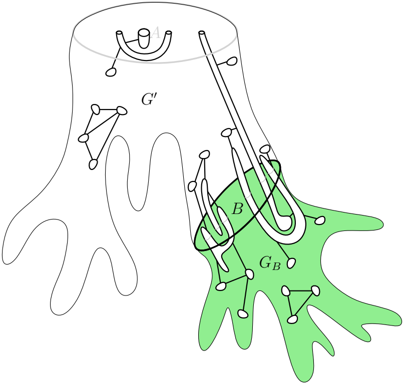
Algorithm for a general collection $\mathcal{F}$

- We see G as a t-boundaried graph.



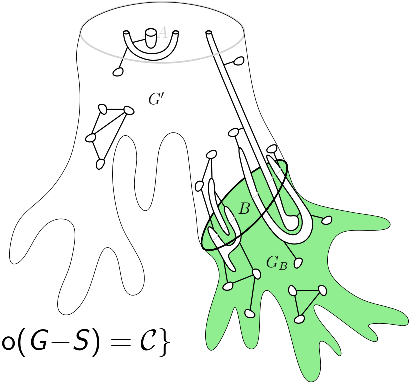
Algorithm for a general collection $\mathcal{F}$

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



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

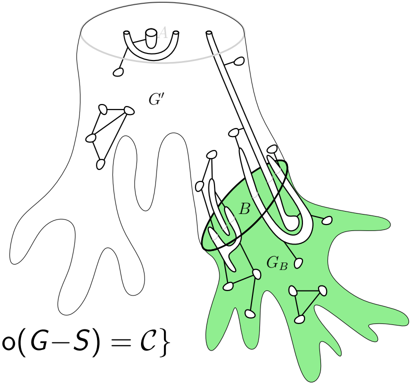
- We see G as a t-boundaried graph.
- folio of G: set of all its $\mathcal{F}$-minor-free minors, up to size $\mathcal{O}_{\mathcal{F}}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:



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

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 $\mathcal{O}_{\mathcal{F}}(t)$.



- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:

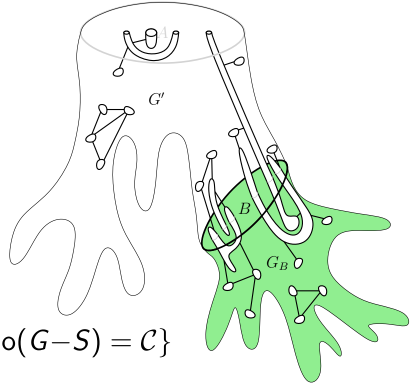
$$\mathbf{p}(G, \mathcal{C}) = \min\{|S| \ : \ S \subseteq V(G) \wedge \text{folio}(G - S) = \mathcal{C}\}$$

- For every t-boundaried graph G,

$$|\text{folio}(G)| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$$

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 $\mathcal{O}_{\mathcal{F}}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:



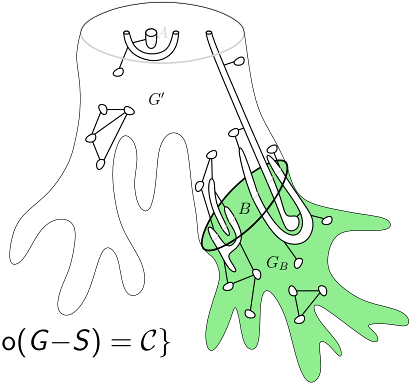
$$\mathbf{p}(G, \mathcal{C}) = \min\{|S| : S \subseteq V(G) \wedge \text{folio}(G - S) = \mathcal{C}\}$$

- For every t-boundaried graph G,

$$|\text{folio}(G)| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$$

- The number of distinct folios is $2^{2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}}$.

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 $\mathcal{O}_{\mathcal{F}}(t)$.
- We compute, using DP over a tree decomposition of G, the following parameter for every folio $\mathcal{C}$:

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

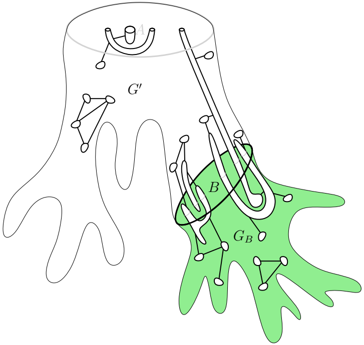
- For every t-boundaried graph G,

$$|\text{folio}(G)| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$$

- The number of distinct folios is $2^{2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}}$.
- This gives an algorithm running in time $2^{2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw} \cdot \log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.

skip

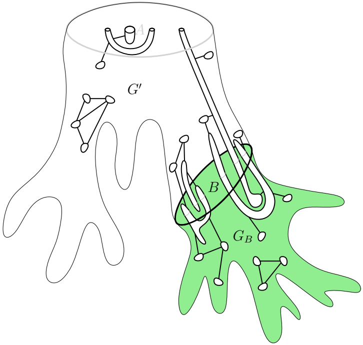
Algorithm for a planar collection $\mathcal{F}$



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

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

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

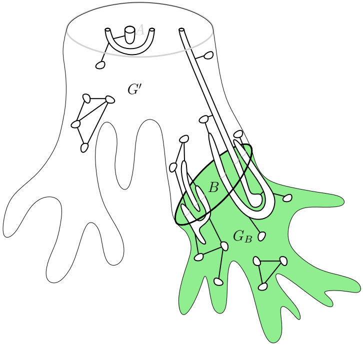


Algorithm for a planar collection $\mathcal{F}$

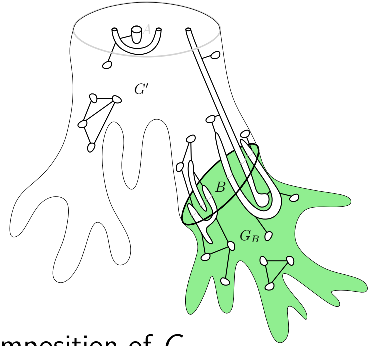
- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

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

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



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



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

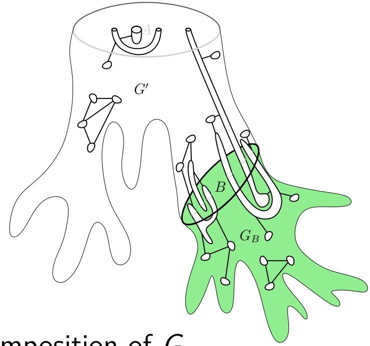
$$G_1 \equiv^{(\mathcal{F},t)} G_2 \quad \text{if } \forall G' \in \mathcal{B}^t,$$
$$\mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_1 \iff \mathcal{F} \preceq_{\mathrm{m}} G' \oplus G_2.$$

- $\mathcal{R}^{(\mathcal{F},t)}$: 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:

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

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



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

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

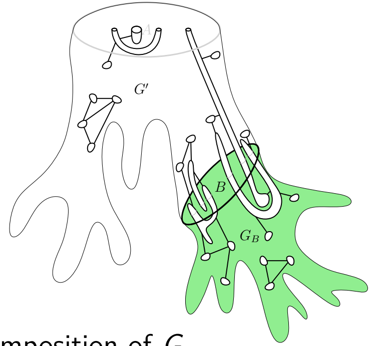
- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.

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

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

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

Algorithm for a 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 $\equiv^{(\mathcal{F},t)}$.

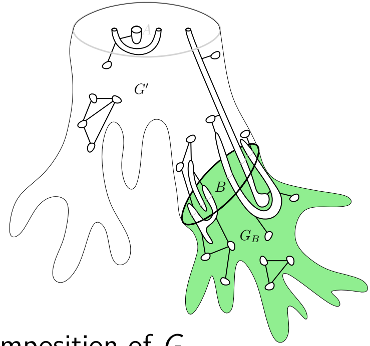
- We compute, using DP over a tree decomposition of G, the following parameter for every representative R:

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

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

  $\#$ 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 planar collection $\mathcal{F}$



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

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

- $\mathcal{R}^{(\mathcal{F},t)}$: 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 $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t\cdot\log t)}$. $\boxed{\text{Planarity!}}$

  # 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 $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

skip

Algorithm for any collection $\mathcal{F}$

Algorithm for any collection $\mathcal{F}$

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

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,

$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,

$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$

- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem:

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem: As R is $\mathcal{F}$-minor-free, if $\mathrm{tw}(R \setminus B) > c_{\mathcal{F}}$,

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem: As R is $\mathcal{F}$-minor-free, if $\mathrm{tw}(R \setminus B) > c_{\mathcal{F}}$, $R \setminus B$ contains a large flat wall,

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem: As R is $\mathcal{F}$-minor-free, if $\mathrm{tw}(R \setminus B) > c_{\mathcal{F}}$, $R \setminus B$ contains a large flat wall, where we can find an irrelevant vertex.

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,

$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$

- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem: As R is $\mathcal{F}$-minor-free, if $\text{tw}(R \setminus B) > c_{\mathcal{F}}$, $R \setminus B$ contains a large flat wall, where we can find an irrelevant vertex.
- R has a treewidth modulator of size $\mathcal{O}(t)$ containing its boundary B.

Algorithm for any collection $\mathcal{F}$

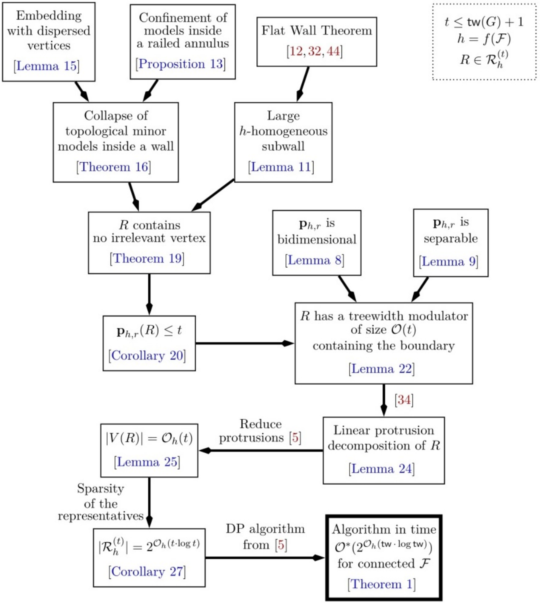
- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem: As R is $\mathcal{F}$-minor-free, if $\mathrm{tw}(R \setminus B) > c_{\mathcal{F}}$, $R \setminus B$ contains a large flat wall, where we can find an irrelevant vertex.
- R has a treewidth modulator of size $\mathcal{O}(t)$ containing its boundary B.
- We can then find a linear protrusion decomposition of R.

Algorithm for any collection $\mathcal{F}$

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- We are done: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)}$ and the same DP works!
- Flat Wall Theorem: As R is $\mathcal{F}$-minor-free, if $\mathrm{tw}(R \setminus B) > c_{\mathcal{F}}$, $R \setminus B$ contains a large flat wall, where we can find an irrelevant vertex.
- R has a treewidth modulator of size $\mathcal{O}(t)$ containing its boundary B.
- We can then find a linear protrusion decomposition of R.
- By applying protrusion reduction, we obtain that $|V(R)| = \mathcal{O}_{\mathcal{F}}(t)$.

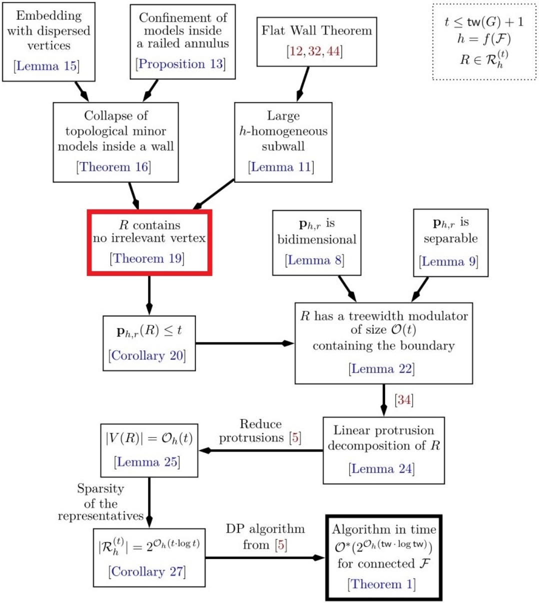
Algorithm for any collection $\mathcal{F}$

Algorithm for any collection $\mathcal{F}$



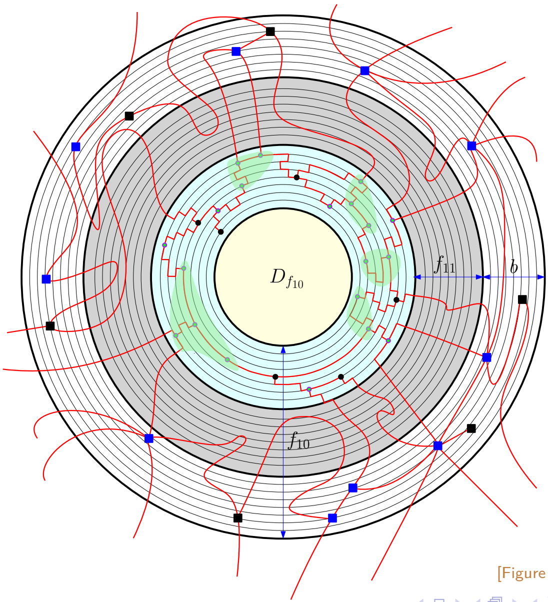
skip

Algorithm for any collection $\mathcal{F}$



Hard part: finding an irrelevant vertex inside a flat wall

Hard part: finding an irrelevant vertex inside a flat wall



skip

[Figure by Dimitrios M. Thilikos]

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

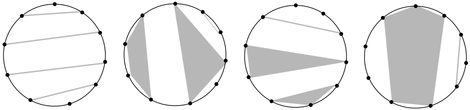
- $\boxed{\text{Idea}}$ get an improved bound on $|\mathcal{R}^{(\mathcal{F},t)}|$.
- We use a sphere-cut decomposition of the input planar graph G.

[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.

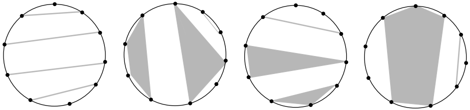


Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ 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.



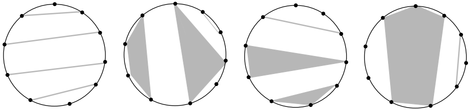
- The number of representatives is $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(t)}$.

  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 $2^{\mathcal{O}_{\mathcal{F}}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$.

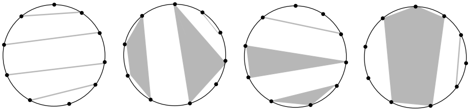
Algorithm when the input graph G is planar

- $\boxed{\text{Idea}}$ 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. skip [Rué, S., Thilikos. 2014]

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

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-DELETION and $\mathcal{F}$-TM-DELETION for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
    - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathsf{tw})}$ or $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})}$?

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathrm{tw})}$ or $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})}$?
  - Lower bounds for families $\mathcal{F}$ containing disconnected graphs.

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathsf{tw})}$ or $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})}$?
  - Lower bounds for families $\mathcal{F}$ containing disconnected graphs.
    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?

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-DELETION and $\mathcal{F}$-TM-DELETION for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathsf{tw})}$ or $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})}$?
  - Lower bounds for families $\mathcal{F}$ containing disconnected graphs.
    Deletion to genus at most g: $2^{\mathcal{O}_g(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]
- Concerning the $\boxed{\text{topological minor}}$ version:

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

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathrm{tw})}$ or $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})}$?
  - Lower bounds for families $\mathcal{F}$ containing disconnected graphs. Deletion to genus at most g: $2^{\mathcal{O}_g(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]
- Concerning the $\boxed{\text{topological minor}}$ version:
  - Dichotomy for $\{H\}$-TM-Deletion when H connected (+planar).

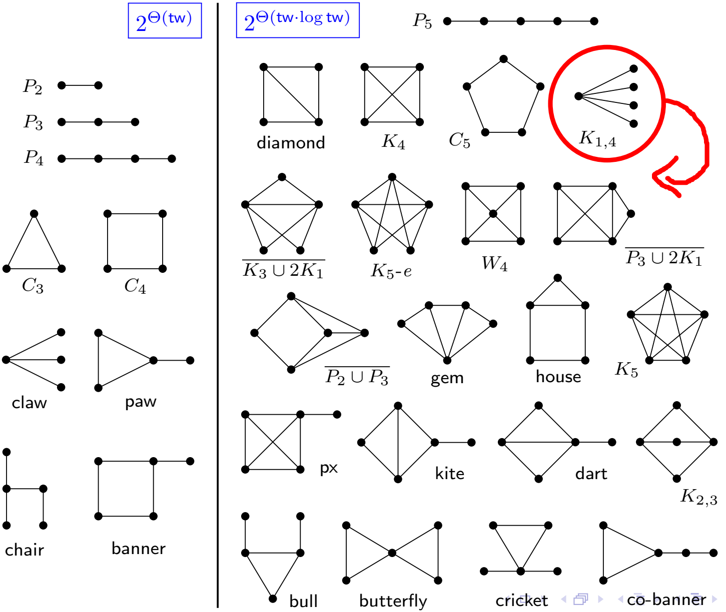
What's next about $\mathcal{F}$-Deletion?

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-Deletion and $\mathcal{F}$-TM-Deletion for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathsf{tw})}$ or $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})}$?
  - Lower bounds for families $\mathcal{F}$ containing disconnected graphs.
    Deletion to genus at most g: $2^{\mathcal{O}_g(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]
- Concerning the $\boxed{\text{topological minor}}$ version:
  - Dichotomy for $\{H\}$-TM-Deletion when H connected (+planar).
  - We do not know if there exists some $\mathcal{F}$ such that $\mathcal{F}$-TM-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?

- $\boxed{\text{Goal}}$ classify the (asymptotically) tight complexity of $\mathcal{F}$-M-DELETION and $\mathcal{F}$-TM-DELETION for every family $\mathcal{F}$.
- Concerning the $\boxed{\text{minor}}$ version:
  - We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
  - Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathrm{tw})}$ or $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})}$?
  - Lower bounds for families $\mathcal{F}$ containing disconnected graphs. Deletion to genus at most g: $2^{\mathcal{O}_g(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Pilipczuk. 2017]
- Concerning the $\boxed{\text{topological minor}}$ version:
  - Dichotomy for $\{H\}$-TM-DELETION when H connected (+planar).
  - We do not know if there exists some $\mathcal{F}$ such that $\mathcal{F}$-TM-DELETION cannot be solved in time $2^{o(\mathrm{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.
  - $\boxed{\text{Conjecture}}$ For every family $\mathcal{F}$, the $\mathcal{F}$-TM-DELETION problem is solvable in time $2^{\mathcal{O}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.

For topological minors, there is (at least) one change



Gràcies!