Fast algorithms parameterized by treewidth

ESIGMA meeting
Paris, May 31-June 1, 2018

Ignasi Sau
CNRS, LIRMM, Université de Montpellier
1. Area of research: parameterized complexity

2. FPT algorithms parameterized by treewidth

3. A possible line of research
Area of research: parameterized complexity

FPT algorithms parameterized by treewidth

A possible line of research
The area of parameterized complexity

**Idea** Measure the complexity of an algorithm in terms of the **input size** and an **additional parameter**.
The area of parameterized complexity

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

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

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

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

Karp (1972): list of 21 important NP-complete problems.

Nowadays, literally thousands of problems are known to be NP-hard: unless $P = NP$, they cannot be solved in polynomial time.

But, are all NP-hard problems (or instances) equally hard?
given an NP-hard problem with input size $n$, fix one parameter $k$ of the input to see whether the problem gets more “tractable”.

**Example:** the size of a **Vertex Cover**.
Parameterized complexity in one slide

- **Idea**: given an NP-hard problem with input size $n$, fix one parameter $k$ of the input to see whether the problem gets more “tractable”.

  **Example**: the size of a Vertex Cover.

- Given a (NP-hard) problem with input of size $n$ and a parameter $k$, a fixed-parameter tractable (FPT) algorithm runs in time

  $$f(k) \cdot n^{O(1)},$$

  for some function $f$. 
Examples of parameterized problems

- Decide whether a graph $G$ has a vertex cover of size at most $k$.

- Decide whether a graph $G$ has a clique of size at least $k$.
Examples of parameterized problems

- Decide whether a graph $G$ has a **vertex cover** of size at most $k$.
  
  This problem is **FPT**.

- Decide whether a graph $G$ has a **clique** of size at least $k$.
  
  This problem is probably **not FPT**: it is $W[1]$-hard.
Examples of parameterized problems

- Decide whether a graph $G$ has a vertex cover of size at most $k$. This problem is FPT.

- Decide whether a graph $G$ has a clique of size at least $k$. This problem is probably not FPT: it is W[1]-hard.

- Decide whether a graph $G$ has a clique of size at least $k$, parameterized by the maximum degree $\Delta$ of $G$. This problem is also FPT...
Examples of parameterized problems

- Decide whether a graph $G$ has a vertex cover of size at most $k$.
  This problem is FPT.

- Decide whether a graph $G$ has a clique of size at least $k$.
  This problem is probably not FPT: it is W[1]-hard.

- Decide whether a graph $G$ has a clique of size at least $k$, parameterized by the maximum degree $\Delta$ of $G$.
  This problem is FPT.
Examples of parameterized problems

- Decide whether a graph $G$ has a vertex cover of size at most $k$. This problem is FPT.
- Decide whether a graph $G$ has a clique of size at least $k$. This problem is probably not FPT: it is W[1]-hard.
- Decide whether a graph $G$ has a clique of size at least $k$, parameterized by the maximum degree $\Delta$ of $G$. This problem is FPT.
- Decide whether a graph $G$ has a clique of size at least $k$, parameterized by the treewidth of $G$, denoted $\text{tw}(G)$. This problem is also FPT...
Examples of parameterized problems

- Decide whether a graph $G$ has a *vertex cover* of size at most $k$.
  This problem is FPT.

- Decide whether a graph $G$ has a *clique* of size at least $k$.
  This problem is probably not FPT: it is W[1]-hard.

- Decide whether a graph $G$ has a *clique* of size at least $k$, parameterized by the maximum degree $\Delta$ of $G$.
  This problem is FPT.

- Decide whether a graph $G$ has a *clique* of size at least $k$, parameterized by the treewidth of $G$, denoted $\text{tw}(G)$.
  This problem is also FPT...
1. Area of research: parameterized complexity

2. FPT algorithms parameterized by treewidth

3. A possible line of research
Treewidth via $k$-trees

Example of a 2-tree: 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: 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 \textit{k-tree} is a graph that can be built starting from a \((k + 1)\)-clique and then iteratively adding a vertex connected to a \textit{k-clique}.

Example of a 2-tree:
Treewidth via $k$-trees

Example of a 2-tree: 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: 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 $k$-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:
Treewidth via $k$-trees

Example of a 2-tree: 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 $k$-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:
Treewidth via \( k \)-trees

Example of a 2-tree:

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 $k$-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:**

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 \( G \): smallest integer \( k \) such that \( G \) is a partial \( k \)-tree.
Treewidth via \( k \)-trees

Example of a 2-tree:

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 \( G \): smallest integer \( k \) such that \( G \) is a partial \( k \)-tree.

**Treewidth:**
Invariant that measures the topological resemblance of a graph to a tree.
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. In many practical scenarios, it turns out that the treewidth of the associated graph is small (programming languages, road networks, ...).
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. In many practical scenarios, it turns out that the treewidth of the associated graph is small (programming languages, road networks, ...).

3. Treewidth behaves very well algorithmically...
Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

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

Theorem (Courcelle, 1990)
Every problem expressible in MSOL can be solved in time \( f(tw) \cdot n \) on graphs on \( n \) vertices and treewidth at most \( tw \).

Examples:
Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, \( k \)-Coloring for fixed \( k \), ...
Courcelle’s theorem

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

Example: \( \text{DomSet}(S) : [ \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G) ] \)
Courcelle’s theorem

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

Example: $\text{DomSet}(S) : \left( \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G) \right)$

Theorem (Courcelle, 1990)

Every problem expressible in MSOL can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$. 
Courcelle’s theorem

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

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

Theorem (Courcelle, 1990)
Every problem expressible in MSOL can be solved in time $f(tw) \cdot n$ on graphs on $n$ vertices and treewidth at most $tw$.

Examples: Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, $k$-Coloring for fixed $k$, ...
Not all problems are FPT parameterized by treewidth!

Are all “natural” graph problems FPT parameterized by treewidth?
Not all problems are FPT parameterized by treewidth!

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.
Not all problems are FPT parameterized by treewidth!

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.

- Some problems involving weights or colors are even \(NP\)-hard on graphs of constant treewidth (or trees!).
Is it enough to prove that a problem is FPT?

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

\[ f(tw) \cdot n^{O(1)} \]
Typically, Courcelle’s theorem allows to prove that a problem is FPT...

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

$$f(tw) \cdot n^{O(1)} = 2^{345678tw} \cdot n^{O(1)}$$
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!

\[ f(tw) \cdot n^{O(1)} = 2^{345678^{tw}} \cdot n^{O(1)} \]

Major goal: find the smallest possible function \( f(tw) \).
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!

\[ f(tw) \cdot n^{O(1)} = 2^{345678^{tw}} \cdot n^{O(1)} \]

Major goal: find the smallest possible function \( f(tw) \).

This is a very active area in parameterized complexity.
Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

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

**ETH**: The 3-SAT problem on $n$ variables cannot be solved in time $2^{o(n)}$.
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

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 $(2 - \varepsilon)^n$.
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

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 $(2 - \varepsilon)^n$.

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

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

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

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 $(2 - \varepsilon)^n$.

SETH $\Rightarrow$ ETH $\Rightarrow$ FPT $\neq$ W[1]
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

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 $(2 - \varepsilon)^n$.

$\text{SETH} \Rightarrow \text{ETH} \Rightarrow \text{FPT} \neq \text{W}[1] \Rightarrow \text{P} \neq \text{NP}$
Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

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 $(2 - \varepsilon)^n$.

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

Typical statements:
- ETH $\Rightarrow$ $k$-VERTEX COVER cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$. 

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

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 $(2 - \varepsilon)^n$.

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

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 be solved in time $2^{o(\sqrt{k})} \cdot n^{O(1)}$. 
Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.
Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.

For many problems, like Vertex Cover or Dominating Set, the “natural” DP algorithms lead to (optimal) single-exponential algorithms:

\[ \mathcal{O}(2^{O(tw)} \cdot n^{O(1)}). \]
Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.

For many problems, like Vertex Cover or Dominating Set, the “natural” DP algorithms lead to (optimal) single-exponential algorithms:

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

But for the so-called connectivity problems, like Longest Path or Steiner Tree, the “natural” DP algorithms provide only time

$$2^{O(tw \log tw)} \cdot n^{O(1)}.$$
The revolution of single-exponential algorithms

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

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

This was false!!

**Cut&Count:** [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 $2^{O(tw \cdot \log tw)} \cdot n^{O(1)}$ were optimal for connectivity problems.

This was false!!

Cut&Count: [Cygan, Nederlof, Pilipczuk\(^2\), van Rooij, Wojtaszczyk. 2011]
Randomized single-exponential algorithms for connectivity problems.

Deterministic algorithms: [Bodlaender, Cygan, Kratsch, Nederlof. 2013]
It was believed that, except on dense graphs (planar, surfaces), algorithms in time $2^{O(tw \cdot \log tw)} \cdot n^{O(1)}$ were optimal for connectivity problems.

This was false!!

**Cut&Count:** [Cygan, Nederlof, Pilipczuk$^2$, van Rooij, Wojtaszczyk. 2011]
Randomized single-exponential algorithms for connectivity problems.

**Deterministic algorithms:** [Bodlaender, Cygan, Kratsch, Nederlof. 2013]

**Representative sets in matroids:** [Fomin, Lokshtanov, Saurabh. 2014]
Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth?

No!

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

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

[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011] There are other examples of such problems...
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? \textbf{No!}

\textbf{Cycle Packing:} find the maximum number of vertex-disjoint cycles.
Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth? **No!**

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

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

[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]
Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth? **No!**

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

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

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

There are other examples of such problems...
$H$ is a **minor** of a graph $G$ if $H$ can be obtained from a **subgraph** of $G$ by contracting edges.
The \textit{F-Deletion} problem

Let $\mathcal{F}$ be a \textit{fixed finite collection} of graphs.
The $\mathcal{F}$-Deletion problem

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

**$\mathcal{F}$-Deletion**

**Input:** A graph $G$ and an integer $k$.

**Parameter:** The treewidth $\text{tw}$ 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 \( F \)-Deletion problem

Let \( F \) be a fixed finite collection of graphs.

\( F \)-Deletion

Input: A graph \( G \) and an integer \( k \).

Parameter: The treewidth \( \text{tw} \) 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 \( F \) as a minor?

- \( F = \{ K_2 \} \): Vertex Cover.

With Dimitrios M. Thilikos and Julien Baste we proved the following... [arXiv:1704.07284. 2018]
The $\mathcal{F}$-Deletion problem

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

$\mathcal{F}$-Deletion

**Input:** A graph $G$ and an integer $k$.

**Parameter:** The treewidth $\text{tw}$ 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 problem is easily solvable in time $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.  

With Dimitrios M. Thilikos and Julien Baste we proved the following...
The \( \mathcal{F} \)-Deletion problem

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

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

- \( \mathcal{F} = \{K_2\} \): \textit{Vertex Cover}.
  The problem is easily solvable in time \( 2^{\Theta(tw)} \cdot n^{O(1)} \).

- \( \mathcal{F} = \{C_3\} \): \textit{Feedback Vertex Set}.
The $\mathcal{F}$-Deletion problem

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

**$\mathcal{F}$-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 problem is easily solvable in time $2^{Θ(tw)} \cdot n^{O(1)}$.

- $\mathcal{F} = \{C_3\}$: **Feedback Vertex Set**.
  The problem is “hardly” solvable in time $2^{Θ(tw)} \cdot n^{O(1)}$.
The \( \mathcal{F} \)-Deletion problem

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

\[ \mathcal{F}-\text{Deletion} \]

**Input:** A graph \( G \) and an integer \( k \).

**Parameter:** The treewidth \( \text{tw} \) 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 problem is easily solvable in time \( 2^{\Theta(\text{tw})} \cdot n^{O(1)} \).

- \( \mathcal{F} = \{ C_3 \} \): **Feedback Vertex Set**.
  The problem is “hardly” solvable in time \( 2^{\Theta(\text{tw})} \cdot n^{O(1)} \).

- \( \mathcal{F} = \{ K_5, K_{3,3} \} \): **Vertex Planarization**.

The \textbf{F-Deletion} problem

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

\textbf{F-Deletion}\
\textbf{Input:} A graph $G$ and an integer $k$. \\
\textbf{Parameter:} The treewidth $\text{tw}$ of $G$. \\
\textbf{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?

- $\mathcal{F} = \{K_2\}$: \textsc{Vertex Cover}. \\
  The problem is easily solvable in time $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.

- $\mathcal{F} = \{C_3\}$: \textsc{Feedback Vertex Set}. \\
  The problem is “hardly” solvable in time $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.

- $\mathcal{F} = \{K_5, K_{3,3}\}$: \textsc{Vertex Planarization}. \\
  The problem is solvable in time $2^{\Theta(\text{tw} \cdot \log \text{tw})} \cdot n^{O(1)}$. 

With Dimitrios M. Thilikos and Julien Baste we proved the following... [arXiv:1704.07284. 2018]
The $\mathcal{F}$-Deletion problem

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

\textbf{$\mathcal{F}$-Deletion}

\textbf{Input:} A graph $G$ and an integer $k$.

\textbf{Parameter:} The treewidth $\text{tw}$ of $G$.

\textbf{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?

- $\mathcal{F} = \{K_2\}$: \textsc{Vertex Cover}.
  The problem is easily solvable in time $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.

- $\mathcal{F} = \{C_3\}$: \textsc{Feedback Vertex Set}.
  The problem is “hardly” solvable in time $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.

- $\mathcal{F} = \{K_5, K_{3,3}\}$: \textsc{Vertex Planarization}.
  The problem is solvable in time $2^{\Theta(\text{tw} \cdot \log \text{tw})} \cdot n^{O(1)}$.

With Dimitrios M. Thilikos and Julien Baste we proved the following...

Complexity of $\{H\}$-$\text{DELETION}$ for small planar graphs $H$

Classification of the complexity of $\{H\}$-$\text{DELETION}$ for all connected simple planar graphs $H$ with $|V(H)| \leq 5$ and $|E(H)| \geq 1$: for the 9 graphs on the left (resp. 20 graphs on the right), the problem is solvable in time $2^{\Theta(tw)}$ (resp. $2^{\Theta(tw \cdot \log tw)}$). For $\{H\}$-$\text{TM-DELETION}$, $K_1,4$ should be on the left.
Area of research: parameterized complexity

FPT algorithms parameterized by treewidth

A possible line of research
Study the parameterized complexity of graph mining problems parameterized by treewidth.
Treewidth in ESIGMA

Study the parameterized complexity of graph mining problems parameterized by treewidth.

Examples:

- **Edge Clique Cover.**
- Any of the problems mentioned so far in the talks.
Study the parameterized complexity of graph mining problems parameterized by treewidth.

Examples:
- **Edge Clique Cover.**
- Any of the problems mentioned so far in the talks.

**Strategy** for a fixed problem:
1. Is the problem **FPT** parameterized by **treewidth**?
Study the parameterized complexity of graph mining problems parameterized by treewidth.

Examples:

- **Edge Clique Cover**.
- Any of the problems mentioned so far in the talks.

**Strategy** for a fixed problem:

1. Is the problem **FPT** parameterized by **treewidth**? If it is not, end of the story.
Study the parameterized complexity of graph mining problems parameterized by treewidth.

**Examples:**

- **Edge Clique Cover.**
- Any of the problems mentioned so far in the talks.

**Strategy** for a fixed problem:

1. Is the problem **FPT** parameterized by treewidth? If it is not, end of the story.
2. If it is, try to find the smallest function $f(tw)$ so that the problem is solvable in time $f(tw) \cdot n^{O(1)}$, assuming the ETH or the SETH.
A “democratic” state (like Spain) should not have political prisoners, right?