Fast algorithms parameterized by treewidth

ESIGMA meeting

Paris, May 31-June 1, 2018

Ignasi Sau

CNRS, LIRMM, Université de Montpellier

Outline of the talk

1. Area of research: parameterized complexity
2. FPT algorithms parameterized by treewidth
3. A possible line of research

Next section is...

1. Area of research: parameterized complexity
2. FPT algorithms parameterized by treewidth
3. A possible line of research

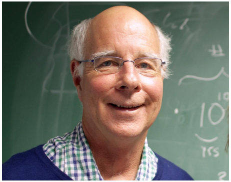
The area of parameterized complexity

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 $\mathsf{P} = \mathsf{NP}$, they cannot be solved in polynomial time.

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 $\mathsf{P} = \mathsf{NP}$, they cannot be solved in polynomial time.
- But, are all NP-hard problems (or instances) equally hard?

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.

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

$$f(k) \cdot n^{O(1)}, \quad \text{for some function } f.$$

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 Δ of G.

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 Δ 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 Δ 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 $\mathrm{tw}(G)$.

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 Δ 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 $\mathrm{tw}(G)$.

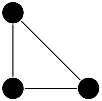
  This problem is also FPT...

Next section is...

1. Area of research: parameterized complexity
2. FPT algorithms parameterized by treewidth
3. A possible line of research

Treewidth via k-trees

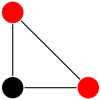
Example of a 2-tree:



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

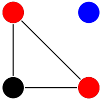
Example of a 2-tree:



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.

Treewidth via k-trees

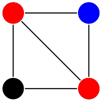
Example of a 2-tree:



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

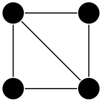
Example of a 2-tree:



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.

Treewidth via k-trees

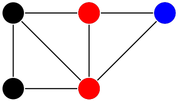
Example of a 2-tree:



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

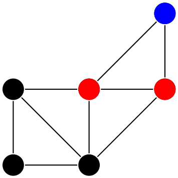
Example of a 2-tree:



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.

Treewidth via k-trees

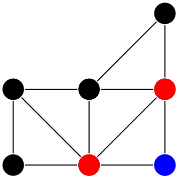
Example of a 2-tree:



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

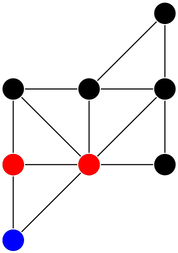
Example of a 2-tree:



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.

Treewidth via k-trees

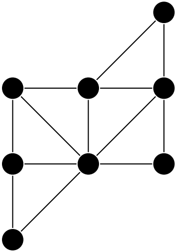
Example of a 2-tree:



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

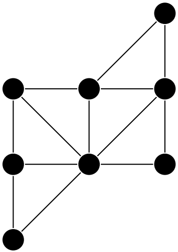
Example of a 2-tree:



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.

Treewidth via k-trees

Example of a 2-tree:



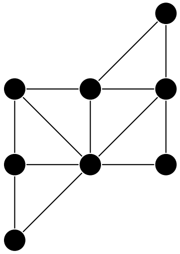
A *k*-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

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...

Courcelle's theorem

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

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

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

Courcelle's theorem

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.

Courcelle's theorem

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

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

Theorem (Courcelle, 1990)

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

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...

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!

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!

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

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!

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

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!

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

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

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!

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

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

This is a very active area in parameterized complexity.

Lower bounds on the running times of FPT algorithms

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

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 $(2-\varepsilon)^n$

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

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 $2^{o(n)}$

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

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

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

Bounds for problems parameterized by treewidth

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

Bounds for problems parameterized by treewidth

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:

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

Bounds for problems parameterized by treewidth

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:

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

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

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

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

The revolution of single-exponential algorithms

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

This was false!!

Cut&Count: [Cygan, Nederlof, Pilipczuk², 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 $2^{O(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{O(1)}$ were optimal for connectivity problems.

This was false!!

Cut&Count: [Cygan, Nederlof, Pilipczuk², van Rooij, Wojtaszczyk. 2011]
Randomized single-exponential algorithms for connectivity problems.

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

The revolution of single-exponential algorithms

It was believed that, except on sparse graphs (planar, surfaces), algorithms in time $2^{O(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{O(1)}$ 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]

End of the story?

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? 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? No!

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

An algorithm in time $2^{O(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{O(1)}$ is optimal under the ETH.
[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]

End of the story?

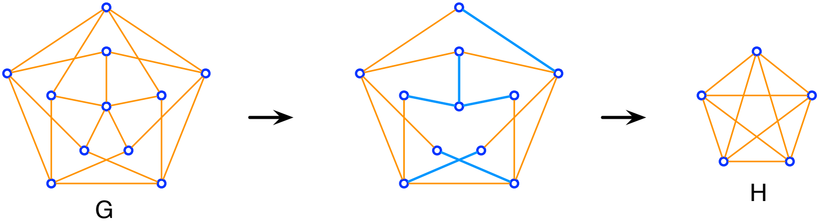
Do all connectivity problems admit single-exponential algorithms (on general graphs) parameterized by treewidth? No!

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

An algorithm in time $2^{O(\mathsf{tw} \cdot \log \mathsf{tw})} \cdot n^{O(1)}$ is optimal under the ETH.
[Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]

There are other examples of such problems...

Graph minors



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

The $\mathcal{F}$-Deletion problem

Let $\mathcal{F}$ be a 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 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 $\mathcal{F}$-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 $\mathcal{F}$-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 $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.

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

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 $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.
- $\mathcal{F} = \{C_3\}$: 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 $\mathcal{F}$-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 $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.
- $\mathcal{F} = \{C_3\}$: 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 problem is solvable in time $2^{\Theta(\text{tw} \cdot \log \text{tw})} \cdot n^{O(1)}$.

The $\mathcal{F}$-DELETION problem

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

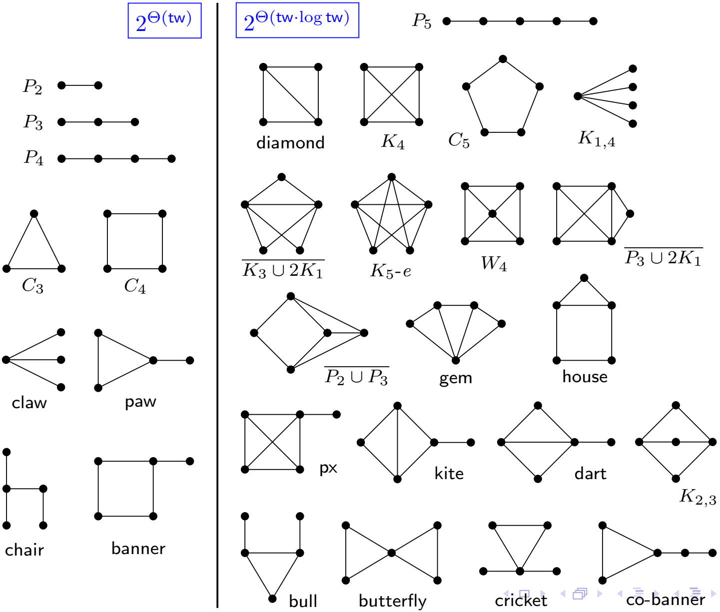
$\mathcal{F}$-DELETION
Input: A graph G and an integer k.
Parameter: The treewidth tw of G.
Question: Does G contain a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G - S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.
  The problem is easily solvable in time $2^{\Theta(\text{tw})} \cdot n^{O(1)}$.
- $\mathcal{F} = \{C_3\}$: 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 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]

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



Next section is...

Treewidth in ESIGMA

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.

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.

Strategy for a fixed problem:

1. Is the problem FPT 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.

Strategy for a fixed problem:

1. Is the problem FPT parameterized by treewidth?
   If it is not, end of the story.

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.

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 $f(\text{tw})$ so that the problem is solvable in time $f(\text{tw}) \cdot n^{O(1)}$, assuming the ETH or the SETH.

A “democratic” state (like Spain) should not have political prisoners, right?