Algorithmic aspects of minor-closed graph classes

Ignasi Sau

LIRMM, Université de Montpellier, CNRS, France

Escuela de Ciencias Informáticas (ECI)
UBA, Buenos Aires, July 24-28, 2023

Outline of this course

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
4. Bidimensionality
5. Irrelevant vertex technique
6. Application to hitting minors
7. Kernelization (?)

Outline of this course (more precise)

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Evaluación de este curso

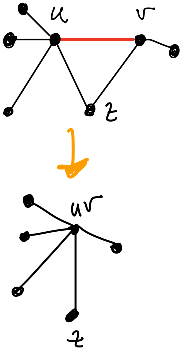
- En los slides, hay $\sim$ 20 preguntas, indicadas con (why?)
- El último día de curso, voy a elegir 12 o 13 de ellas, y podréis elegir 10 entre ellas para responderlas por escrito.
- Todos los slides están disponibles en `www.lirmm.fr/~sau/talks/ECI-2023-Ignasi.pdf`.
- Se podrán traer los slides en un ordenador, y apuntes.

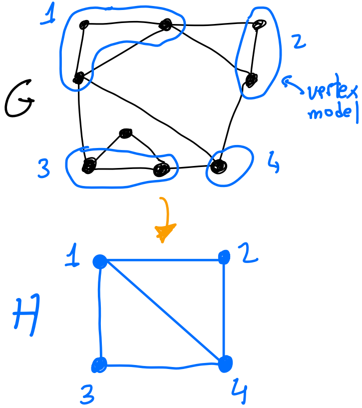
Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Graph minors

A graph H is a minor of a graph G, denoted by $H \leqslant_m G$, if H can be obtained by a subgraph of G by contracting edges.





Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.
- Subgraphs of series-parallel graphs (why?).

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.
- Subgraphs of series-parallel graphs (why?).
- Planar graphs (why?).

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.
- Subgraphs of series-parallel graphs (why?).
- Planar graphs (why?).
- Graphs embeddable in a fixed surface.

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.
- Subgraphs of series-parallel graphs (why?).
- Planar graphs (why?).
- Graphs embeddable in a fixed surface.
- Linklessly embeddable graphs.

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.
- Subgraphs of series-parallel graphs (why?).
- Planar graphs (why?).
- Graphs embeddable in a fixed surface.
- Linklessly embeddable graphs.
- Knotlessly embeddable graphs.

Minor-closed graph classes

A graph class $\mathcal{C}$ is minor-closed (or closed under minors) if

$$G \in \mathcal{C} \;\Rightarrow\; H \in \mathcal{C} \text{ for every } H \leqslant_m G.$$

Examples of minor-closed graph classes:

- Independent sets.
- Forests.
- Subgraphs of series-parallel graphs (why?).
- Planar graphs (why?).
- Graphs embeddable in a fixed surface.
- Linklessly embeddable graphs.
- Knotlessly embeddable graphs.
- ...

Characterizing a graph class by excluded minors

Let $\mathcal{F}$ be a (possibly infinite) family of graphs. We define $\text{exc}(\mathcal{F})$ as the class of all graphs that do not contain any of the graphs in $\mathcal{F}$ as a minor.

Characterizing a graph class by excluded minors

Let $\mathcal{F}$ be a (possibly infinite) family of graphs. We define $\mathrm{exc}(\mathcal{F})$ as the class of all graphs that do not contain any of the graphs in $\mathcal{F}$ as a minor.

Easy: for every family $\mathcal{F}$, the class $\mathrm{exc}(\mathcal{F})$ is minor-closed (why?).

Characterizing a graph class by excluded minors

Let $\mathcal{F}$ be a (possibly infinite) family of graphs. We define $\text{exc}(\mathcal{F})$ as the class of all graphs that do not contain any of the graphs in $\mathcal{F}$ as a minor.

Easy: for every family $\mathcal{F}$, the class $\text{exc}(\mathcal{F})$ is minor-closed (why?).

We say that $\mathcal{F}$ characterizes $\text{exc}(\mathcal{F})$ by excluded minors.

Characterizing a graph class by excluded minors

Let $\mathcal{F}$ be a (possibly infinite) family of graphs. We define $\text{exc}(\mathcal{F})$ as the class of all graphs that do not contain any of the graphs in $\mathcal{F}$ as a minor.

Easy: for every family $\mathcal{F}$, the class $\text{exc}(\mathcal{F})$ is minor-closed (why?).

We say that $\mathcal{F}$ characterizes $\text{exc}(\mathcal{F})$ by excluded minors.

Conversely, every minor-closed graph class $\mathcal{C}$ can be characterized by excluded minors:

> List all the graphs $\mathcal{F}_{\mathcal{C}} := \{G_1, G_2, \ldots\}$ that do not belong to $\mathcal{C}$, and then $\mathcal{C} = \text{exc}(\mathcal{F}_{\mathcal{C}})$.

Characterizing a graph class by excluded minors

Let $\mathcal{F}$ be a (possibly infinite) family of graphs. We define $\text{exc}(\mathcal{F})$ as the class of all graphs that do not contain any of the graphs in $\mathcal{F}$ as a minor.

Easy: for every family $\mathcal{F}$, the class $\text{exc}(\mathcal{F})$ is minor-closed (why?).

We say that $\mathcal{F}$ characterizes $\text{exc}(\mathcal{F})$ by excluded minors.

Conversely, every minor-closed graph class $\mathcal{C}$ can be characterized by excluded minors:

> List all the graphs $\mathcal{F}_{\mathcal{C}} := \{G_1, G_2, \dots\}$ that do not belong to $\mathcal{C}$, and then $\mathcal{C} = \text{exc}(\mathcal{F}_{\mathcal{C}})$.

Note that, in general, this list $\mathcal{F}_{\mathcal{C}} = \{G_1, G_2, \dots\}$ may be infinite.

Examples for some minor-closed classes

- If $\mathcal{C}$ = independent sets, then $\mathcal{C} =$

Examples for some minor-closed classes

- If $\mathcal{C}$ = independent sets, then $\mathcal{C} = \text{exc}(K_2)$.

Examples for some minor-closed classes

- If $\mathcal{C}$ = independent sets, then $\mathcal{C}$ =exc(K_2).
- If $\mathcal{C}$ = forests, then

Examples for some minor-closed classes

- If $\mathcal{C}$ = independent sets, then $\mathcal{C}$ =exc(K_2).
- If $\mathcal{C}$ = forests, then $\mathcal{C}$ = exc(K_3).

Examples for some minor-closed classes

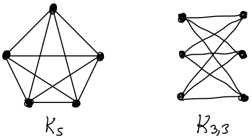
- If $\mathcal{C}$ = independent sets, then $\mathcal{C} = \text{exc}(K_2)$.
- If $\mathcal{C}$ = forests, then $\mathcal{C} = \text{exc}(K_3)$.
- If $\mathcal{C}$ = series-parallel graphs, then $\mathcal{C} = \text{exc}(K_4)$.

Examples for some minor-closed classes

- If $\mathcal{C}$ = independent sets, then $\mathcal{C}$ =exc(K_2).
- If $\mathcal{C}$ = forests, then $\mathcal{C}$ = exc(K_3).
- If $\mathcal{C}$ = series-parallel graphs, then $\mathcal{C}$ = exc(K_4).
- If $\mathcal{C}$ = outerplanar graphs, then $\mathcal{C}$ = exc(K_4, $K_{2,3}$).

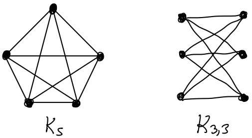
Examples for some minor-closed classes

- If $\mathcal{C}$ = independent sets, then $\mathcal{C} = \text{exc}(K_2)$.
- If $\mathcal{C}$ = forests, then $\mathcal{C} = \text{exc}(K_3)$.
- If $\mathcal{C}$ = series-parallel graphs, then $\mathcal{C} = \text{exc}(K_4)$.
- If $\mathcal{C}$ = outerplanar graphs, then $\mathcal{C} = \text{exc}(K_4, K_{2,3})$.
- If $\mathcal{C}$ = planar graphs, then $\mathcal{C} = \text{exc}(K_5, K_{3,3})$. [Kuratowski. 1930]



Examples for some minor-closed classes

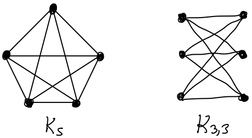
- If $\mathcal{C}$ = independent sets, then $\mathcal{C}$ =exc(K_2).
- If $\mathcal{C}$ = forests, then $\mathcal{C}$ = exc(K_3).
- If $\mathcal{C}$ = series-parallel graphs, then $\mathcal{C}$ = exc(K_4).
- If $\mathcal{C}$ = outerplanar graphs, then $\mathcal{C}$ = exc$(K_4, K_{2,3})$.
- If $\mathcal{C}$ = planar graphs, then $\mathcal{C}$ = exc$(K_5, K_{3,3})$. [Kuratowski. 1930]



- If $\mathcal{C}$ = graphs embeddable in the projective plane, then $|\mathcal{F}_{\mathcal{C}}| = 35$.

Examples for some minor-closed classes

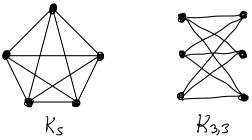
- If $\mathcal{C}$ = independent sets, then $\mathcal{C}$ =exc(K_2).
- If $\mathcal{C}$ = forests, then $\mathcal{C}$ = exc(K_3).
- If $\mathcal{C}$ = series-parallel graphs, then $\mathcal{C}$ = exc(K_4).
- If $\mathcal{C}$ = outerplanar graphs, then $\mathcal{C}$ = exc($K_4, K_{2,3}$).
- If $\mathcal{C}$ = planar graphs, then $\mathcal{C}$ = exc($K_5, K_{3,3}$). [Kuratowski. 1930]



- If $\mathcal{C}$ = graphs embeddable in the projective plane, then $|\mathcal{F}_\mathcal{C}| = 35$.
- If $\mathcal{C}$ = graphs embeddable in a fixed non-orientable surface, then $\mathcal{F}_\mathcal{C}$ is finite. [Archdeacon, Huneke. 1989]

Examples for some minor-closed classes

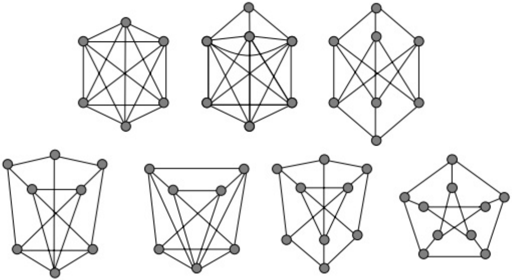
- If $\mathcal{C}$ = independent sets, then $\mathcal{C}$ =exc(K_2).
- If $\mathcal{C}$ = forests, then $\mathcal{C}$ = exc(K_3).
- If $\mathcal{C}$ = series-parallel graphs, then $\mathcal{C}$ = exc(K_4).
- If $\mathcal{C}$ = outerplanar graphs, then $\mathcal{C}$ = exc($K_4, K_{2,3}$).
- If $\mathcal{C}$ = planar graphs, then $\mathcal{C}$ = exc($K_5, K_{3,3}$). [Kuratowski. 1930]



- If $\mathcal{C}$ = graphs embeddable in the projective plane, then $|\mathcal{F}_\mathcal{C}| = 35$.
- If $\mathcal{C}$ = graphs embeddable in a fixed non-orientable surface, then $\mathcal{F}_\mathcal{C}$ is finite. [Archdeacon, Huneke. 1989]
- If $\mathcal{C}$ = graphs embeddable in a fixed orientable surface, then $\mathcal{F}_\mathcal{C}$ is finite. [Robertson, Seymour. 1990]

A last example

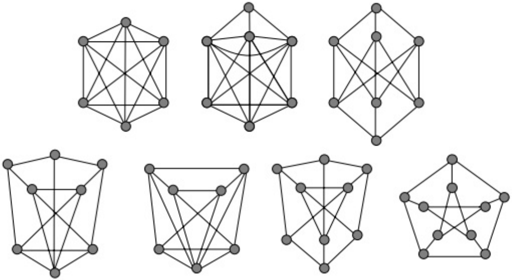
If $\mathcal{C}$ = linklessly embeddable graphs, then $\mathcal{F}_{\mathcal{C}} =$



[Robertson, Seymour. 1990]

A last example

If $\mathcal{C}$ = linklessly embeddable graphs, then $\mathcal{F}_{\mathcal{C}} =$



[Robertson, Seymour. 1990]

$\mathcal{F}_{\mathcal{C}}$ seems to get complicated... but always finite!

Wagner's conjecture

Conjecture (Wagner. 1970)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

Wagner's conjecture... now Robertson-Seymour's theorem

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \text{exc}(\mathcal{F}_{\mathcal{C}})$.

Reformulations

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

Reformulations

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

Note that for every minor-closed graph class $\mathcal{C}$, the set of minor-minimal graphs not in $\mathcal{C}$ is unique (why?): it is denoted by $\mathrm{obs}(\mathcal{C})$ (obstruction set).

Reformulations

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \text{exc}(\mathcal{F}_{\mathcal{C}})$.

Note that for every minor-closed graph class $\mathcal{C}$, the set of minor-minimal graphs not in $\mathcal{C}$ is unique (why?): it is denoted by $\text{obs}(\mathcal{C})$ (obstruction set).

Equivalent: For every minor-closed graph class $\mathcal{C}$, $\text{obs}(\mathcal{C})$ is finite.

Reformulations

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \text{exc}(\mathcal{F}_{\mathcal{C}})$.

Note that for every minor-closed graph class $\mathcal{C}$, the set of minor-minimal graphs not in $\mathcal{C}$ is unique (why?): it is denoted by $\text{obs}(\mathcal{C})$ (obstruction set).

Equivalent: For every minor-closed graph class $\mathcal{C}$, $\text{obs}(\mathcal{C})$ is finite.

Yet equivalent: Every infinite set $\{G_1, G_2, \ldots\}$ of finite graphs contains two graphs such that one is a minor of the other (there is no infinite antichain).

Well-quasi orders

A partially ordered set (poset) is a set P with a partial binary relation $\leq$:

1. Reflexive: $a \leq a$.
2. Antisymmetric: if $a \leq b$ and $b \leq a$, then $a = b$.
3. Transitive: if $a \leq b$ and $b \leq c$, then $a \leq c$.

Well-quasi orders

A partially ordered set (poset) is a set P with a partial binary relation $\leq$:

1. Reflexive: $a \leq a$.
2. Antisymmetric: if $a \leq b$ and $b \leq a$, then $a = b$.
3. Transitive: if $a \leq b$ and $b \leq c$, then $a \leq c$.

A poset $(P, \leq)$ is well-quasi-ordered (wqo) if every infinite sequence $(x_1, x_2, \ldots)$ has two elements x_i and x_j such that $i < j$ and $x_i \leq x_j$.

Well-quasi orders

A partially ordered set (poset) is a set P with a partial binary relation $\leq$:

1. Reflexive: $a \leq a$.
2. Antisymmetric: if $a \leq b$ and $b \leq a$, then $a = b$.
3. Transitive: if $a \leq b$ and $b \leq c$, then $a \leq c$.

A poset $(P, \leq)$ is well-quasi-ordered (wqo) if every infinite sequence $(x_1, x_2, \ldots)$ has two elements x_i and x_j such that $i < j$ and $x_i \leq x_j$.

Equivalent (why?): $(P, \leq)$ contains neither an infinite descending chain nor an infinite antichain (i.e., set of pairwise incomparable elements).

Well-quasi orders

A partially ordered set (poset) is a set P with a partial binary relation $\leq$:

1. Reflexive: $a \leq a$.
2. Antisymmetric: if $a \leq b$ and $b \leq a$, then $a = b$.
3. Transitive: if $a \leq b$ and $b \leq c$, then $a \leq c$.

A poset $(P, \leq)$ is well-quasi-ordered (wqo) if every infinite sequence $(x_1, x_2, \ldots)$ has two elements x_i and x_j such that $i < j$ and $x_i \leq x_j$.

Equivalent (why?): $(P, \leq)$ contains neither an infinite descending chain nor an infinite antichain (i.e., set of pairwise incomparable elements).

In the case of graph minors: there is no infinite descending chain (why?), so wqo $\Leftrightarrow$ no infinite antichain.

Well-quasi orders

A partially ordered set (poset) is a set P with a partial binary relation $\leq$:

1. Reflexive: $a \leq a$.
2. Antisymmetric: if $a \leq b$ and $b \leq a$, then $a = b$.
3. Transitive: if $a \leq b$ and $b \leq c$, then $a \leq c$.

A poset $(P, \leq)$ is well-quasi-ordered (wqo) if every infinite sequence $(x_1, x_2, \ldots)$ has two elements x_i and x_j such that $i < j$ and $x_i \leq x_j$.

Equivalent (why?): $(P, \leq)$ contains neither an infinite descending chain nor an infinite antichain (i.e., set of pairwise incomparable elements).

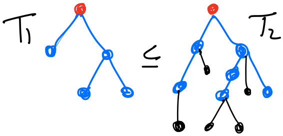
In the case of graph minors: there is no infinite descending chain (why?), so wqo $\Leftrightarrow$ no infinite antichain.

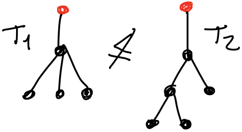
R&S theorem: Finite graphs are wqo with respect to the minor relation.

Illustrative example: rooted trees

Let T_1 and T_2 be two finite rooted trees.

Def: $T_1 \leq T_2$ if there is a subdivision of T_1 that occurs as a rooted subgraph of T_2 (the root of T_1 is not necessarily mapped to the root of T_2).

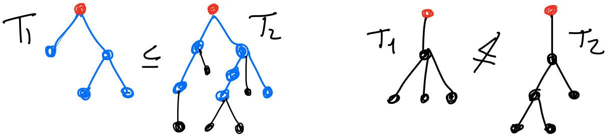




Illustrative example: rooted trees

Let T_1 and T_2 be two finite rooted trees.

Def: $T_1 \leq T_2$ if there is a subdivision of T_1 that occurs as a rooted subgraph of T_2 (the root of T_1 is not necessarily mapped to the root of T_2).

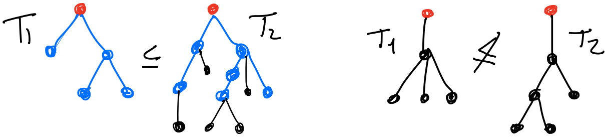


Conjecture (Vázsonyi. 1937)

Finite rooted trees are wqo with respect to the relation $\leq$.

Illustrative example: rooted trees

Let T_1 and T_2 be two finite rooted trees.
Def: $T_1 \leq T_2$ if there is a subdivision of T_1 that occurs as a rooted subgraph of T_2 (the root of T_1 is not necessarily mapped to the root of T_2).



Conjecture (Vázsonyi. 1937)

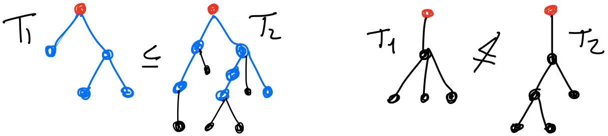
Finite rooted trees are wqo with respect to the relation $\leq$.

Proved independently by: [Kruskal. 1960]

[Tarkowski. 1960]

Illustrative example: rooted trees

Let T_1 and T_2 be two finite rooted trees.
Def: $T_1 \leq T_2$ if there is a subdivision of T_1 that occurs as a rooted subgraph of T_2 (the root of T_1 is not necessarily mapped to the root of T_2).



Conjecture (Vázsonyi. 1937)

Finite rooted trees are wqo with respect to the relation $\leq$.

Proved independently by: [Kruskal. 1960]

[Tarkowski. 1960]

We will now see a simple proof by [Nash-Williams. 1963]

By contradiction, suppose that there is a bad infinite sequence:
$(T_1, T_2, \ldots)$ of rooted trees with no $i < j$ such that $T_i \leq T_j$.

By contradiction, suppose that there is a bad infinite sequence:
$(T_1, T_2, \ldots)$ of rooted trees with no $i < j$ such that $T_i \leq T_j$.

We choose the bad sequence in this particular way:

- Choose T_1 as a smallest tree that can start a bad sequence.

By contradiction, suppose that there is a bad infinite sequence: $(T_1, T_2, \ldots)$ of rooted trees with no $i < j$ such that $T_i \leq T_j$.

We choose the bad sequence in this particular way:

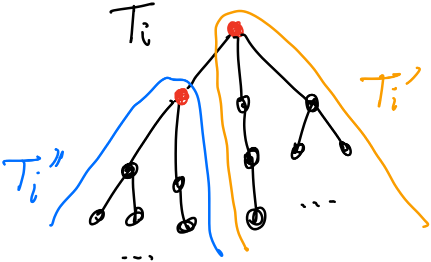
- Choose T_1 as a smallest tree that can start a bad sequence.
- For every $k > 1$, choose T_k as a smallest tree which occurs as the k-th element of a bad sequence starting with $(T_1, \ldots, T_{k-1})$.

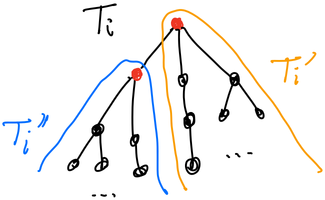
By contradiction, suppose that there is a bad infinite sequence:
$(T_1, T_2, \ldots)$ of rooted trees with no $i < j$ such that $T_i \leq T_j$.

We choose the bad sequence in this particular way:

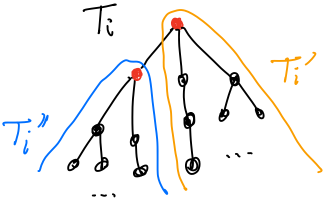
- Choose T_1 as a smallest tree that can start a bad sequence.
- For every $k > 1$, choose T_k as a smallest tree which occurs as the k-th element of a bad sequence starting with $(T_1, \ldots, T_{k-1})$.

For $k \geq 1$:
Let T_i' be the tree obtained from T_i by deleting any branch from the root.
Let T_i'' be the deleted branch (rooted at a child of the root of T_i).



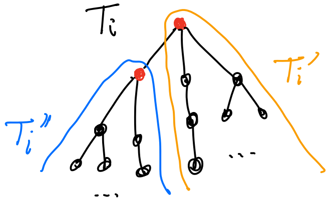


Claim: the sequence $(T_1', T_2', \ldots)$ cannot contain a bad subsequence.



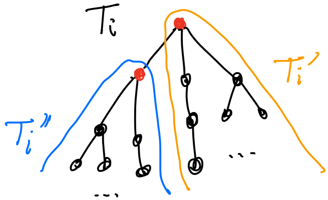
Claim: the sequence $(T_1', T_2', \ldots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T_{i_1}', T_{i_2}', \ldots)$ be a bad subsequence.



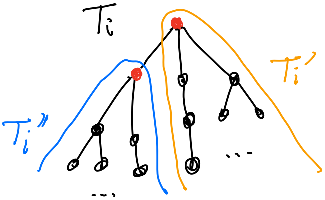
Claim: the sequence $(T_1', T_2', \ldots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T_{i_1}', T_{i_2}', \ldots)$ be a bad subsequence. Then $(T_1, \ldots, T_{i_1-1}, T_{i_1}', T_{i_2}', \ldots)$ is bad



Claim: the sequence $(T_1', T_2', \dots)$ cannot contain a bad subsequence.

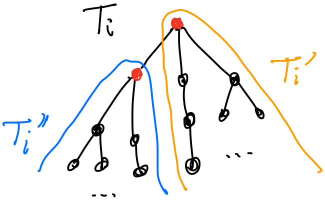
Proof: Suppose it does, and let $(T_{i_1}', T_{i_2}', \dots)$ be a bad subsequence. Then $(T_1, \dots, T_{i_1-1}, T_{i_1}', T_{i_2}', \dots)$ is bad... but T_{i_1}' is smaller than T_{i_1}. $\square$



Claim: the sequence $(T'_1, T'_2, \ldots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T'_{i_1}, T'_{i_2}, \ldots)$ be a bad subsequence. Then $(T_1, \ldots, T_{i_1-1}, T'_{i_1}, T'_{i_2}, \ldots)$ is bad... but T'_{i_1} is smaller than T_{i_1}. $\square$

It follows (why? hard! Uses Ramsey) that $(T'_1, T'_2, \ldots)$ contains an infinite increasing subsequence $T'_{j_1} \leq T'_{j_2} \leq \ldots$

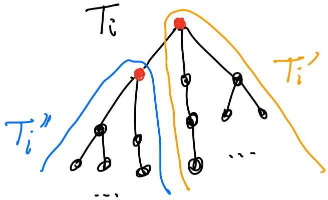


Claim: the sequence $(T_1', T_2', \ldots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T_{i_1}', T_{i_2}', \ldots)$ be a bad subsequence. Then $(T_1, \ldots, T_{i_1 - 1}, T_{i_1}', T_{i_2}', \ldots)$ is bad... but T_{i_1}' is smaller than T_{i_1}. $\square$

It follows (why? hard! Uses Ramsey) that $(T_1', T_2', \ldots)$ contains an infinite increasing subsequence $T_{j_1}' \leq T_{j_2}' \leq \ldots$

Claim: the sequence $(T_{j_1}'', T_{j_2}'', \ldots)$ cannot be bad (why?).



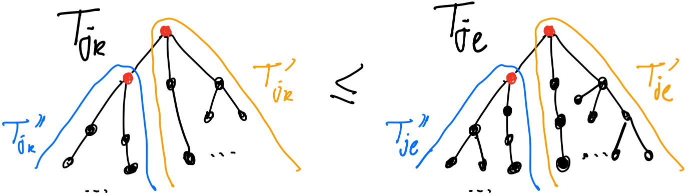
Claim: the sequence $(T_1', T_2', \dots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T_{i_1}', T_{i_2}', \dots)$ be a bad subsequence. Then $(T_1, \dots, T_{i_1 - 1}, T_{i_1}', T_{i_2}', \dots)$ is bad... but T_{i_1}' is smaller than T_{i_1}. $\square$

It follows (why? hard! Uses Ramsey) that $(T_1', T_2', \dots)$ contains an infinite increasing subsequence $T_{j_1}' \leq T_{j_2}' \leq \dots$

Claim: the sequence $(T_{j_1}'', T_{j_2}'', \dots)$ cannot be bad (why?).

There exist $k < \ell$ such that $T_{j_k}'' \leq T_{j_\ell}''$



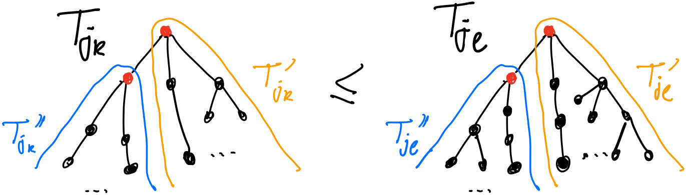
Claim: the sequence $(T'_1, T'_2, \ldots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T'_{i_1}, T'_{i_2}, \ldots)$ be a bad subsequence. Then $(T_1, \ldots, T_{i_1-1}, T'_{i_1}, T'_{i_2}, \ldots)$ is bad... but T'_{i_1} is smaller than T_{i_1}. □

It follows (why? hard! Uses Ramsey) that $(T'_1, T'_2, \ldots)$ contains an infinite increasing subsequence $T'_{j_1} \leq T'_{j_2} \leq \ldots$

Claim: the sequence $(T''_{j_1}, T''_{j_2}, \ldots)$ cannot be bad (why?).

There exist $k < \ell$ such that $T''_{j_k} \leq T''_{j_\ell}$



Claim: the sequence $(T'_1, T'_2, \ldots)$ cannot contain a bad subsequence.

Proof: Suppose it does, and let $(T'_{i_1}, T'_{i_2}, \ldots)$ be a bad subsequence. Then $(T_1, \ldots, T_{i_1-1}, T'_{i_1}, T'_{i_2}, \ldots)$ is bad... but T'_{i_1} is smaller than T_{i_1}. $\square$

It follows (why? hard! Uses Ramsey) that $(T'_1, T'_2, \ldots)$ contains an infinite increasing subsequence $T'_{j_1} \leq T'_{j_2} \leq \ldots$

Claim: the sequence $(T''_{j_1}, T''_{j_2}, \ldots)$ cannot be bad (why?).

There exist $k < \ell$ such that $T''_{j_k} \leq T''_{j_\ell} \;\Rightarrow\; T_{j_k} \leq T_{j_\ell}$, contradiction to bad!

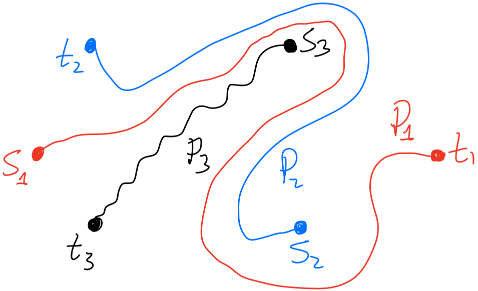
A notion strongly linked to graph minors

A notion strongly linked to graph minors

DISJOINT PATHS

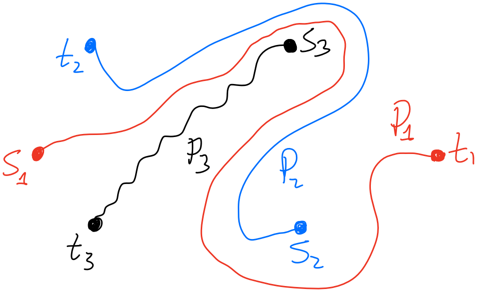
Input: a graph G and $2k$ vertices $s_1, \dots, s_k, t_1, \dots, t_k$.

Question: does G contain k vertex-disjoint paths $P_1, \dots, P_k$ such that P_i connects s_i to t_i?



A notion strongly linked to graph minors

DISJOINT PATHS
Input: a graph G and $2k$ vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?



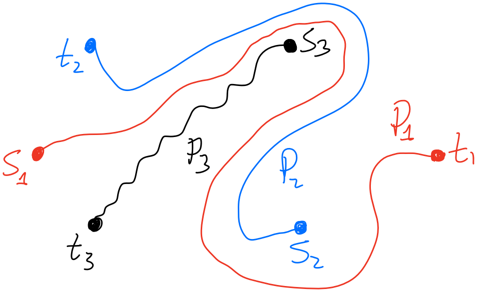
Much stronger than k vertex-disjoint paths from $s_1, \ldots, s_k$ to $t_1, \ldots, t_k$.

A notion strongly linked to graph minors

Disjoint Paths

Input: a graph G and $2k$ vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$.

Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?



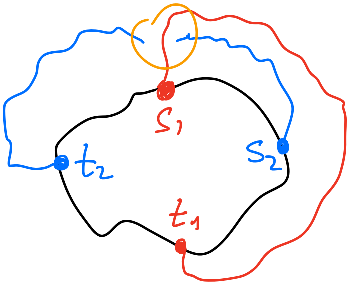
Much stronger than k vertex-disjoint paths from $s_1, \ldots, s_k$ to $t_1, \ldots, t_k$.

A graph G is k-linked if every instance of Disjoint Paths in G with k pairs is positive.

Topology appears naturally in linkages

Theorem (Thomassen and Seymour. 1980)

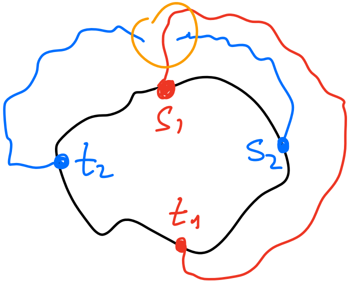
Let G be a 4-connected graph and $s_1, s_2, t_1, t_2 \in V(G)$. Then (s_1, s_2) and (t_1, t_2) are linked unless G is planar and s_1, s_2, t_1, t_2 are on the boundary of the same face, in this cyclic order.



Topology appears naturally in linkages

Theorem (Thomassen and Seymour. 1980)

Let G be a 4-connected graph and $s_1, s_2, t_1, t_2 \in V(G)$. Then (s_1, s_2) and (t_1, t_2) are linked unless G is planar and s_1, s_2, t_1, t_2 are on the boundary of the same face, in this cyclic order.



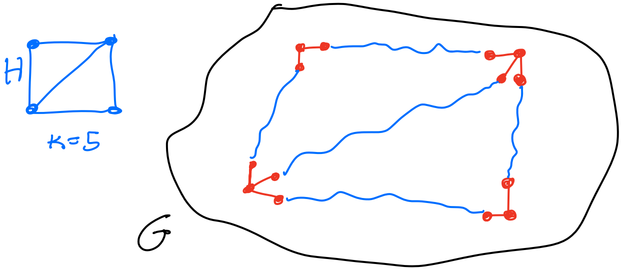
A combinatorial condition (linkage) is translated to a purely topological one (embedding).

Why linkages are useful for finding graph minors?

Let H be a graph with $|E(H)| = k$ and G be a k-linked graph.

Why linkages are useful for finding graph minors?

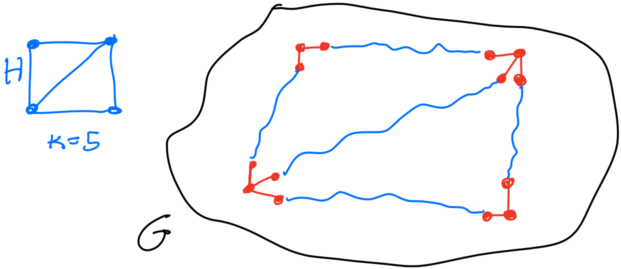
Let H be a graph with $|E(H)| = k$ and G be a k-linked graph.



Then we can easily find H as a minor in G!

Why linkages are useful for finding graph minors?

Let H be a graph with $|E(H)| = k$ and G be a k-linked graph.



Then we can easily find H as a minor in G!

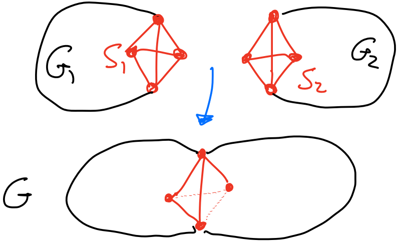
Idea: if the goal is to decide whether $H \leq_m G$, if G is k-linked, then "yes".
Otherwise, we may exploit a topological obstruction to k-linkedness...

Another crucial notion: treewidth

Let G_1 and G_2 be two graphs, and let $S_i \subseteq V(G_i)$ be a k-clique.

Another crucial notion: treewidth

Let G_1 and G_2 be two graphs, and let $S_i \subseteq V(G_i)$ be a k-clique.

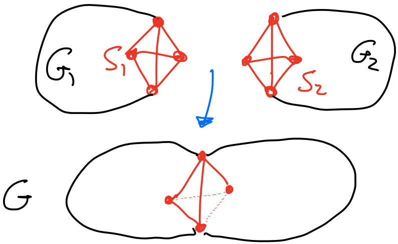


Let G be obtained by identifying S_1 with S_2 and deleting some (possibly none, possibly all) edges between the vertices in $S_1 = S_2$.

We say that G is a k-clique-sum of G_1 and G_2.

Another crucial notion: treewidth

Let G_1 and G_2 be two graphs, and let $S_i \subseteq V(G_i)$ be a k-clique.



Let G be obtained by identifying S_1 with S_2 and deleting some (possibly none, possibly all) edges between the vertices in $S_1 = S_2$.

We say that G is a k-clique-sum of G_1 and G_2.

We say that a graph G has treewidth at most k if it can be obtained by repeatedly taking a k-clique-sum with a graph on at most $k+1$ vertices.

Structure of minor-free graphs

Let H be a fixed graph. Recall that $\text{exc}(H)$ is the class of all graphs that do not contain H as a minor.

Structure of minor-free graphs

Let H be a fixed graph. Recall that $\mathrm{exc}(H)$ is the class of all graphs that do not contain H as a minor.

What is the typical structure of a graph $G \in \mathrm{exc}(H)$?

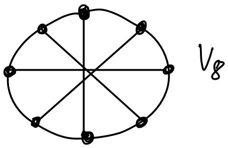
Structure of minor-free graphs

Let H be a fixed graph. Recall that $\mathrm{exc}(H)$ is the class of all graphs that do not contain H as a minor.

What is the typical structure of a graph $G \in \mathrm{exc}(H)$?

Theorem (Wagner. 1937)

A graph $G \in \mathrm{exc}(K_5)$ if and only if it can be obtained by 0-, 1-, 2- and 3-clique-sums from planar graphs and V_8.



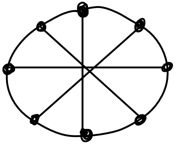
Structure of minor-free graphs

Let H be a fixed graph. Recall that $\mathrm{exc}(H)$ is the class of all graphs that do not contain H as a minor.

What is the typical structure of a graph $G \in \mathrm{exc}(H)$?

Theorem (Wagner. 1937)

A graph $G \in \mathrm{exc}(K_5)$ if and only if it can be obtained by 0-, 1-, 2- and 3-clique-sums from planar graphs and V_8.



V_8

Paradigm: we find "pieces" that exclude K_5 for topological reasons (planarity), add some exceptions (V_8), and then define rules (clique-sums) that preserve being K_5-minor-free.

An intermediate case: excluding a planar graph

Let H be a fixed planar graph.

What is the structure of a graph $G \in \mathrm{exc}(H)$?

An intermediate case: excluding a planar graph

Let H be a fixed planar graph.

What is the structure of a graph $G \in \mathrm{exc}(H)$?

Theorem (Robertson, Seymour. 1986)

For every planar graph H there is an integer $t(H) > 0$ such that every graph in $\mathrm{exc}(H)$ has treewidth at most $t(H)$.

An intermediate case: excluding a planar graph

Let H be a fixed planar graph.

What is the structure of a graph $G \in \mathrm{exc}(H)$?

Theorem (Robertson, Seymour. 1986)

For every planar graph H there is an integer $t(H) > 0$ such that every graph in $\mathrm{exc}(H)$ has treewidth at most $t(H)$.

Thus, every graph in $\mathrm{exc}(H)$ can be built by "gluing" bounded-sized graphs in a tree-like structure ($t(H)$-clique-sums).

An intermediate case: excluding a planar graph

Let H be a fixed planar graph.

What is the structure of a graph $G \in \text{exc}(H)$?

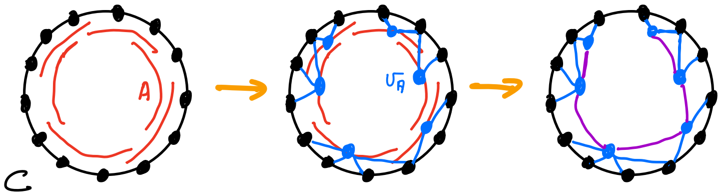
Theorem (Robertson, Seymour. 1986)

For every planar graph H there is an integer $t(H) > 0$ such that every graph in $\text{exc}(H)$ has treewidth at most $t(H)$.

Thus, every graph in $\text{exc}(H)$ can be built by "gluing" bounded-sized graphs in a tree-like structure ($t(H)$-clique-sums).

Note: this is an approximate characterization (i.e., not "iff").

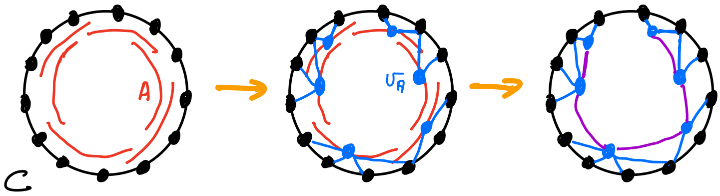
Vortices



Adding a vortex of depth h to a cycle C:

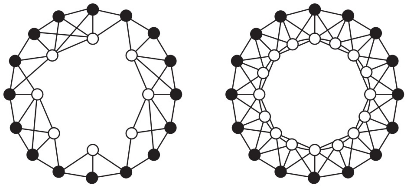
- Select arcs on C so that each vertex is contained in at most h arcs.
- For each arc A, create a vertex v_A.
- Connect v_A to some vertices on the arc A.
- connect any pair (v_A, v_B) for which A and B have a common vertex.

Vortices



Adding a vortex of depth h to a cycle C:

- Select arcs on C so that each vertex is contained in at most h arcs.
- For each arc A, create a vertex v_A.
- Connect v_A to some vertices on the arc A.
- connect any pair (v_A, v_B) for which A and B have a common vertex.



Structure theorem

Theorem (Robertson, Seymour. 1999)

For every graph H there is an integer $h > 0$ such that every graph in $\mathrm{exc}(H)$ can be (efficiently) constructed in the following way:

Structure theorem

Theorem (Robertson, Seymour. 1999)

For every graph H there is an integer $h > 0$ such that every graph in $\mathrm{exc}(H)$ can be (efficiently) constructed in the following way:

1. *Start with a graph G embedded in a connected closed surface Σ with genus at most h so that each face is homeomorphic with an open disc.*

Structure theorem

Theorem (Robertson, Seymour. 1999)

For every graph H there is an integer $h > 0$ such that every graph in $\mathrm{exc}(H)$ can be (efficiently) constructed in the following way:

1. *Start with a graph G embedded in a connected closed surface Σ with genus at most h so that each face is homeomorphic with an open disc.*
2. *Select at most h faces of G and add a vortex of depth at most h to each of them.*

Structure theorem

Theorem (Robertson, Seymour. 1999)

For every graph H there is an integer $h > 0$ such that every graph in $\mathrm{exc}(H)$ can be (efficiently) constructed in the following way:

1. *Start with a graph G embedded in a connected closed surface Σ with genus at most h so that each face is homeomorphic with an open disc.*
2. *Select at most h faces of G and add a vortex of depth at most h to each of them.*
3. *Create at most h new vertices (apices) and connect them to the other vertices arbitrarily.*

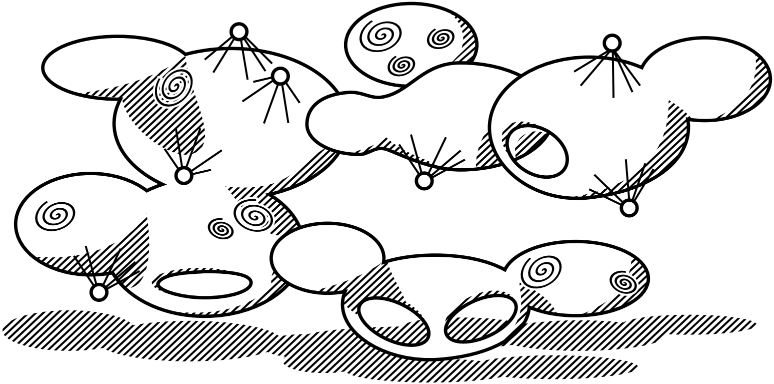
Structure theorem

Theorem (Robertson, Seymour. 1999)

For every graph H there is an integer $h > 0$ such that every graph in $\mathrm{exc}(H)$ can be (efficiently) constructed in the following way:

1. *Start with a graph G embedded in a connected closed surface Σ with genus at most h so that each face is homeomorphic with an open disc.*
2. *Select at most h faces of G and add a vortex of depth at most h to each of them.*
3. *Create at most h new vertices (apices) and connect them to the other vertices arbitrarily.*
4. *Repeatedly construct the h-clique-sum of the current graph with another graph constructed using steps 1-2-3 above.*

A visualization of an H-minor-free graph



[Figure by Felix Riedl]

Sketch of sketch of sketch of proof of Wagner's conjecture

Let's try to mimic the proof for rooted trees by Nash-Williams:

Sketch of sketch of sketch of proof of Wagner's conjecture

By contradiction, suppose that there is a bad infinite sequence: $(G_1, G_2, \ldots)$ of graphs with no $i < j$ such that $G_i \leq_{\mathrm{m}} G_j$.

Sketch of sketch of sketch of proof of Wagner's conjecture

By contradiction, suppose that there is a bad infinite sequence:
$(G_1, G_2, \ldots)$ of graphs with no $i < j$ such that $G_i \leq_{\mathrm{m}} G_j$.

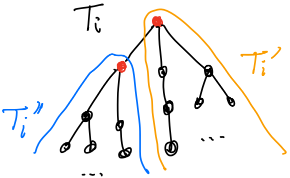
Again, choose $(G_1, G_2, \ldots)$ so that G_i is a minimal continuation.

Sketch of sketch of sketch of proof of Wagner's conjecture

By contradiction, suppose that there is a bad infinite sequence: $(G_1, G_2, \ldots)$ of graphs with no $i < j$ such that $G_i \leq_m G_j$.

Again, choose $(G_1, G_2, \ldots)$ so that G_i is a minimal continuation.

For trees, we decomposed each T_i into T_i' and T_i''... but now??

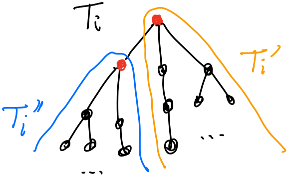


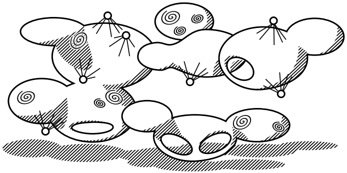
Sketch of sketch of sketch of proof of Wagner's conjecture

By contradiction, suppose that there is a bad infinite sequence:
$(G_1, G_2, \ldots)$ of graphs with no $i < j$ such that $G_i \leq_m G_j$.

Again, choose $(G_1, G_2, \ldots)$ so that G_i is a minimal continuation.

For trees, we decomposed each T_i into T_i' and T_i''... but now??





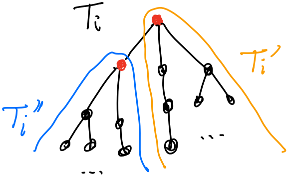
Every G_i with $i \geq 2$ is G_1-minor-free $\rightsquigarrow$ structure theorem of R&S!

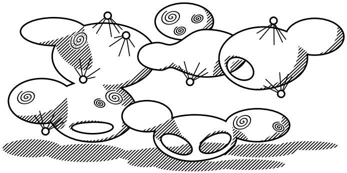
Sketch of sketch of sketch of proof of Wagner's conjecture

By contradiction, suppose that there is a bad infinite sequence:
$(G_1, G_2, \ldots)$ of graphs with no $i < j$ such that $G_i \leq_{\mathrm{m}} G_j$.

Again, choose $(G_1, G_2, \ldots)$ so that G_i is a minimal continuation.

For trees, we decomposed each T_i into T_i' and T_i''... but now??





Every G_i with $i \geq 2$ is G_1-minor-free $\rightsquigarrow$ structure theorem of R&S!

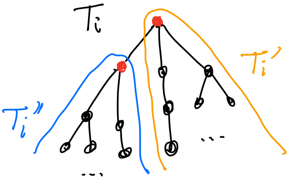
- If G_1 is planar, every G_i has bounded treewidth: similar to trees.

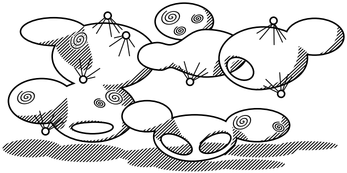
Sketch of sketch of sketch of proof of Wagner's conjecture

By contradiction, suppose that there is a bad infinite sequence: $(G_1, G_2, \dots)$ of graphs with no $i < j$ such that $G_i \leq_{\mathrm{m}} G_j$.

Again, choose $(G_1, G_2, \dots)$ so that G_i is a minimal continuation.

For trees, we decomposed each T_i into T_i' and T_i''... but now??





Every G_i with $i \geq 2$ is G_1-minor-free $\rightsquigarrow$ structure theorem of R&S!

- If G_1 is planar, every G_i has bounded treewidth: similar to trees.
- Otherwise, by the structure theorem: similar to "extended" surfaces (with apices and vortices), glued in a tree-like way.

Some algorithmic consequences

DISJOINT PATHS

Input: an n-vertex graph G and vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$.

Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Some algorithmic consequences

DISJOINT PATHS
Input: an n-vertex graph G and vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Theorem (Robertson, Seymour. 1995)

The DISJOINT PATHS *problem can be solved in time* $f(k) \cdot n^3$.

Some algorithmic consequences

DISJOINT PATHS
Input: an n-vertex graph G and vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Theorem (Robertson, Seymour. 1995)

The DISJOINT PATHS *problem can be solved in time* $f(k) \cdot n^3$.

Improved to $f(k) \cdot n^2$. [Kawarabayash, Kobayashi, Reed. 2012]

Some algorithmic consequences

DISJOINT PATHS
Input: an n-vertex graph G and vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Theorem (Robertson, Seymour. 1995)

The DISJOINT PATHS *problem can be solved in time* $f(k) \cdot n^3$.

Improved to $f(k) \cdot n^2$. [Kawarabayash, Kobayashi, Reed. 2012]

Corollary

For an n-vertex graph G and an h-vertex graph H, testing whether $H \leq_m G$ can be done in time $f(h) \cdot n^2$.

More algorithmic consequences

Corollary

For an n-vertex graph G and an h-vertex graph H, testing whether $H \leq_{\mathrm{m}} G$ can be done in time $f(h) \cdot n^2$.

More algorithmic consequences

Corollary

For an n-vertex graph G and an h-vertex graph H, testing whether $H \leq_{\mathrm{m}} G$ can be done in time $f(h) \cdot n^2$.

Recall:

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

More algorithmic consequences

Corollary

For an n-vertex graph G and an h-vertex graph H, testing whether $H \leq_{\mathrm{m}} G$ can be done in time $f(h) \cdot n^2$.

Recall:

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

Corollary

Every minor-closed property can be tested in quadratic time.

More algorithmic consequences

Corollary

For an n-vertex graph G and an h-vertex graph H, testing whether $H \leq_{\mathrm{m}} G$ can be done in time $f(h) \cdot n^2$.

Recall:

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

Corollary

Every minor-closed property can be tested in quadratic time.

Proof: check $H \leq_{\mathrm{m}} G$ for every graph H in the finite set $\mathcal{F}_{\mathcal{C}}$. □

More algorithmic consequences

Corollary

For an n-vertex graph G and an h-vertex graph H, testing whether $H \leq_{\mathrm{m}} G$ can be done in time $f(h) \cdot n^2$.

Recall:

Theorem (Robertson, Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, there exists a finite set of graphs $\mathcal{F}_{\mathcal{C}}$ such that $\mathcal{C} = \mathrm{exc}(\mathcal{F}_{\mathcal{C}})$.

Corollary

Every minor-closed property can be tested in quadratic time.

Proof: check $H \leq_{\mathrm{m}} G$ for every graph H in the finite set $\mathcal{F}_{\mathcal{C}}$. □

This says that there $\boxed{\text{exists}}$ an algorithm... no idea how to $\boxed{\text{construct}}$ it!!

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation?

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor:** $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor:** $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor:** $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]

A few words on other containment relations

★ **Minor**: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor**: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure?

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ Topological minor: $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor:** $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ **Topological minor:** $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

1. Graphs are WQO w.r.t. the topological minor relation?

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ Topological minor: $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

1. Graphs are WQO w.r.t. the topological minor relation? NO! (why?)

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor:** $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ **Topological minor:** $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

1. Graphs are WQO w.r.t. the topological minor relation? NO! (why?)
2. Topological Minor Testing is FPT when param. by $|V(H)|$?

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ Topological minor: $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

1. Graphs are WQO w.r.t. the topological minor relation? NO! (why?)
2. Topological Minor Testing is FPT when param. by $|V(H)|$?
   YES! [Grohe, Kawarabayashi, Marx, Wollan. 2011]

A few words on other containment relations

★ **Minor:** $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ **Contraction minor:** $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ **Topological minor:** $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

1. Graphs are WQO w.r.t. the topological minor relation? NO! (why?)
2. Topological Minor Testing is FPT when param. by $|V(H)|$?
   YES! [Grohe, Kawarabayashi, Marx, Wollan. 2011]
3. Nice structure?

A few words on other containment relations

★ Minor: $H \preceq_m G$ if H can be obtained from a subgraph of G by contracting edges.

1. Graphs are WQO w.r.t. the minor relation.
2. Minor Testing is FPT when parameterized by $|V(H)|$.
3. H-minor-free graphs have a nice structure.

★ Contraction minor: $H \preceq_{cm} G$ if H can be obtained from G by contracting edges.

1. Graphs are WQO w.r.t. the contraction minor relation? NO! (why?)
2. Contraction Minor Testing is FPT when param. by $|V(H)|$?
   NO! NP-hard already for $|V(H)| \leq 4$. [Brouwer and Veldman. 1987]
3. Nice structure? Not really: They contain cliques, chordal graphs...

★ Topological minor: $H \preceq_{tp} G$ if H can be obtained from a subgraph of G by contracting edges with at least one endpoint of degree ≤ 2.

1. Graphs are WQO w.r.t. the topological minor relation? NO! (why?)
2. Topological Minor Testing is FPT when param. by $|V(H)|$?
   YES! [Grohe, Kawarabayashi, Marx, Wollan. 2011]
3. Nice structure? YES! [Grohe and Marx. 2012]

Structure of sparse graphs

H-topological-minor-free

$\cup$

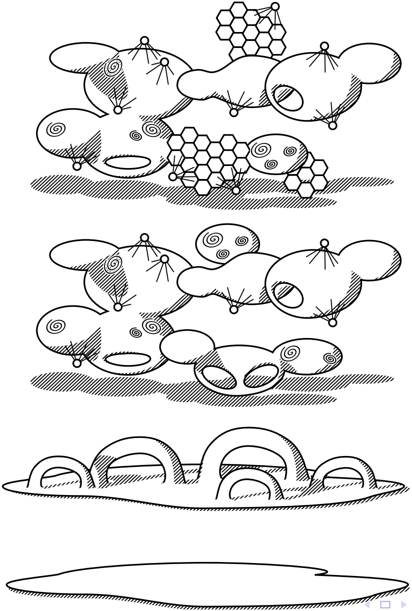
H-minor-free

$\cup$

bounded genus

$\cup$

planar



[Figure by Felix Riedl]

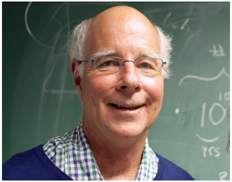
Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Parameterized complexity in a nutshell

$\boxed{\text{Idea}}$ 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 and very active area.

Parameterized problems

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

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

Parameterized problems

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

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

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

Parameterized problems

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

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

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

These three problems are NP-hard, but are they $\boxed{\text{equally}}$ hard?

They behave quite differently...

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

They behave quite differently...

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

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

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

They behave quite differently...

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

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

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

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

They behave quite differently...

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

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

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

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

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

They behave quite differently...

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

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

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

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

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

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

skip

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

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

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

Why k-Clique may not be FPT?

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

Why k-Clique may not be FPT?

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

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

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

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

How to transfer hardness among parameterized problems?

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

How to transfer hardness among parameterized problems?

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

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

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

How to transfer hardness among parameterized problems?

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

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

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

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

How to transfer hardness among parameterized problems?

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

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

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

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

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

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

How to transfer hardness among parameterized problems?

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

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

Instance (x, k) of A → 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 $g : \mathbb{N} \rightarrow \mathbb{N}$.

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

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

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

How to transfer hardness among parameterized problems?

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

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

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

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 (more later!)

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

Kernelization (more later!)

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

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

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

Kernelization (more later!)

$\boxed{\text{Idea}}$ 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 $g : \mathbb{N} \to \mathbb{N}$.

Kernelization (more later!)

Idea polynomial-time preprocessing.

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

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

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

The function g is called the size of the kernel.

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

Kernelization (more later!)

$\boxed{\text{Idea}}$ 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 $g : \mathbb{N} \rightarrow \mathbb{N}$.

The function g is called the size of the kernel.

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

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

XP

k-Clique
k-Vertex Cover
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

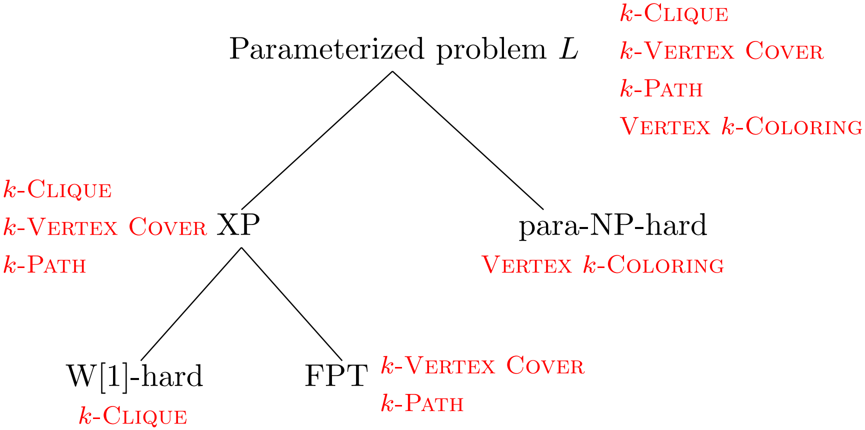
k-Clique
k-Vertex Cover XP
k-Path

para-NP-hard
Vertex k-Coloring

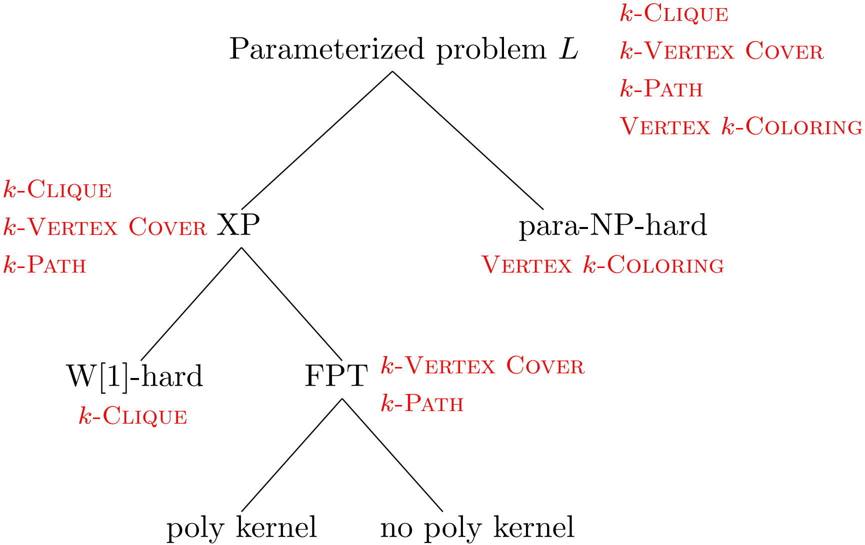
W[1]-hard

FPT

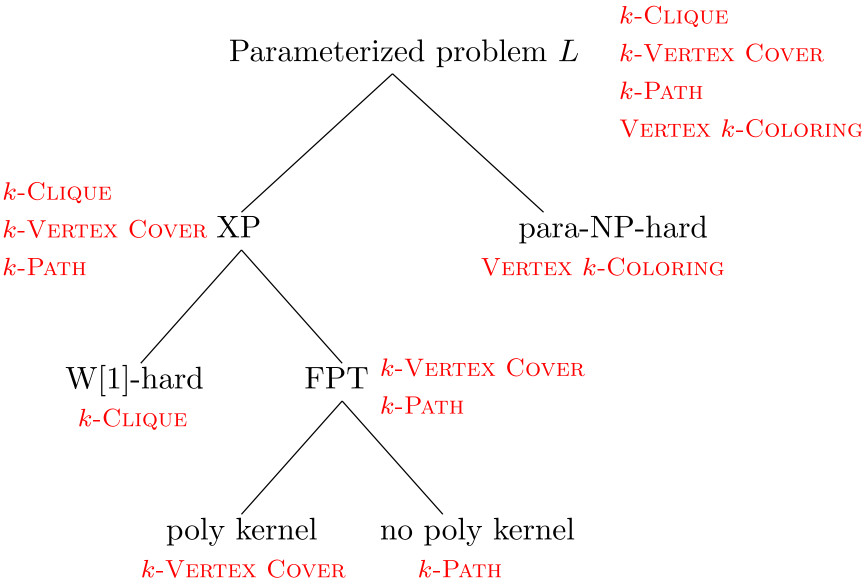
Typical approach to deal with a parameterized problem



Typical approach to deal with a parameterized problem



Typical approach to deal with a parameterized problem



Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

The multiples origins of treewidth

- 1972: Bertelè and Brioschi (dimension).
- 1976: Halin (S-functions of graphs).
- 1984: Arnborg and Proskurowski (partial k-trees).
- 1984: Robertson and Seymour (treewidth).

A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.

A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:

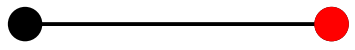
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



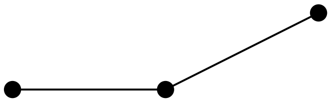
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



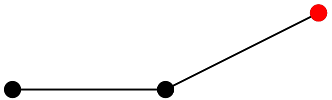
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



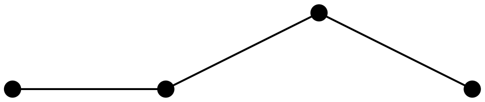
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



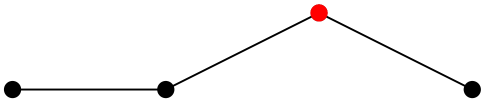
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



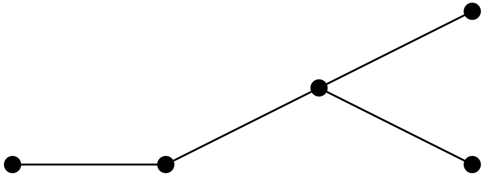
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



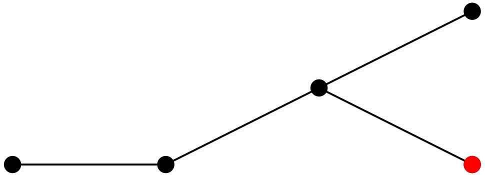
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



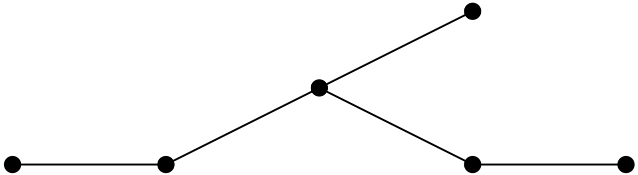
A measure of the similarity with a tree

Treewidth measures the (topological) similarity of a graph with a tree.

Natural candidates:

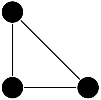
- Number of cycles.
- Vertex-deletion distance to a forest (feedback vertex set number).

Generalization based on the following property of trees:



Treewidth via k-trees

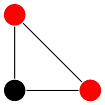
Example of a 2-tree:



[Figure by Julien Baste]

For $k \geq 1$, 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:



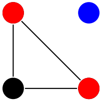
[Figure by Julien Baste]

For $k \geq 1$, 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

For $k \geq 1$, 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.

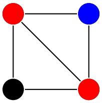
Example of a 2-tree:



[Figure by Julien Baste]

Treewidth via k-trees

Example of a 2-tree:



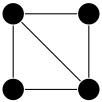
[Figure by Julien Baste]

For $k \geq 1$, 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

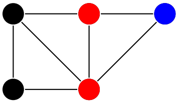
Example of a 2-tree:

For $k \geq 1$, 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.



[Figure by Julien Baste]

Example of a 2-tree:

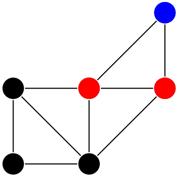


[Figure by Julien Baste]

For $k \geq 1$, 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:

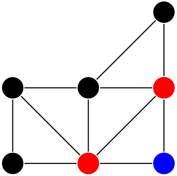


[Figure by Julien Baste]

For $k \geq 1$, 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:

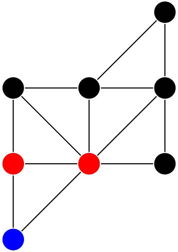


[Figure by Julien Baste]

For $k \geq 1$, 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:

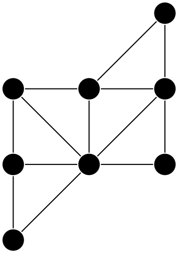


[Figure by Julien Baste]

For $k \geq 1$, 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:

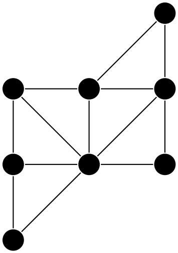


[Figure by Julien Baste]

For $k \geq 1$, 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:



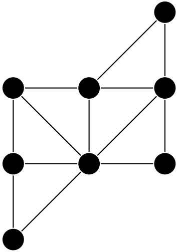
[Figure by Julien Baste]

For $k \geq 1$, 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]

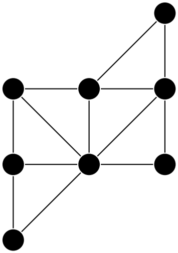
For $k \geq 1$, a k-tree is a graph that can be built starting from a $(k+1)$-clique and then iteratively adding a vertex connected to a k-clique.

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]

For $k \geq 1$, 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.

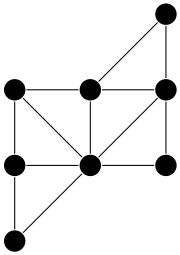
A partial k-tree is a subgraph of a k-tree.

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 forest.

Treewidth via k-trees

Example of a 2-tree:



[Figure by Julien Baste]

For $k \geq 1$, 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 forest.

Construction suggests the notion of tree decomposition: small separators.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

  satisfying the following:

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

  satisfying the following:

  - $\bigcup_{t \in V(T)} X_t = V(G)$,
  - $\forall \{u, v\} \in E(G)$, $\exists t \in V(T)$ with $\{u, v\} \subseteq X_t$.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

  satisfying the following:

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

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

  satisfying the following:

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

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

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

  satisfying the following:

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

- Width of a tree decomposition:
  $\max_{t \in V(T)} |X_t| - 1$.
- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.

An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

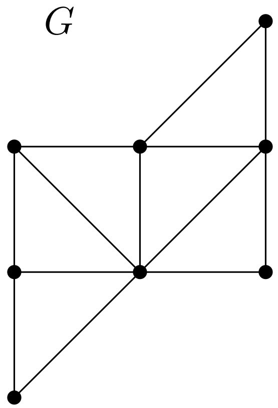
  satisfying the following:

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

- Width of a tree decomposition:

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

- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

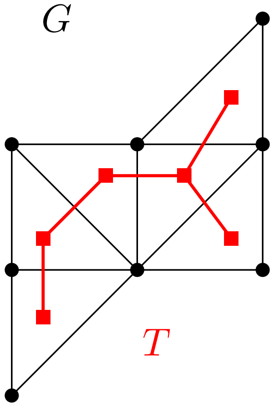
  satisfying the following:

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

- Width of a tree decomposition:

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

- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

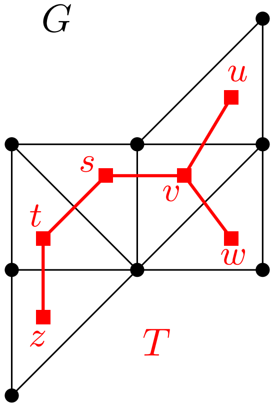
- Tree decomposition of a graph G:

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

  satisfying the following:

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

- Width of a tree decomposition:
  $\max_{t \in V(T)} |X_t| - 1$.
- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

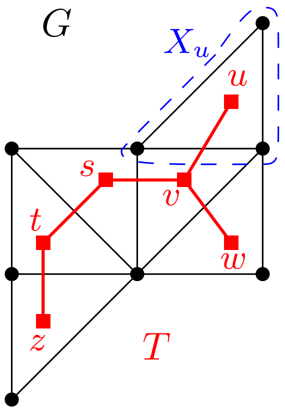
  pair $(T, \{X_t \mid t \in V(T)\})$, where
  T is a tree, and
  $X_t \subseteq V(G) \;\; \forall t \in V(T)$ (bags),

  satisfying the following:

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

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

- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

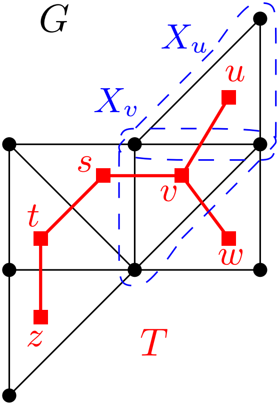
- Tree decomposition of a graph G:

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

  satisfying the following:

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

- Width of a tree decomposition: $\max_{t \in V(T)} |X_t| - 1$.
- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

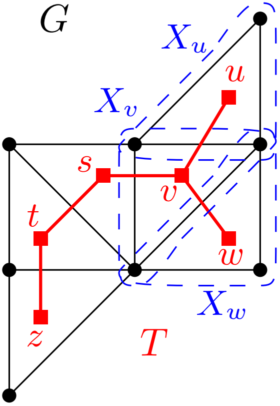
  satisfying the following:

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

- Width of a tree decomposition:

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

- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

- Tree decomposition of a graph G:

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

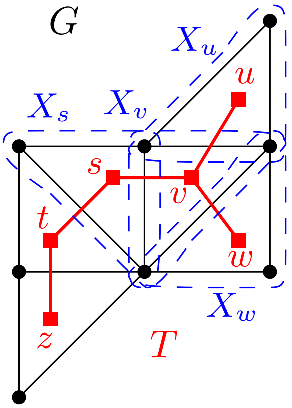
  satisfying the following:

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

- Width of a tree decomposition:

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

- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

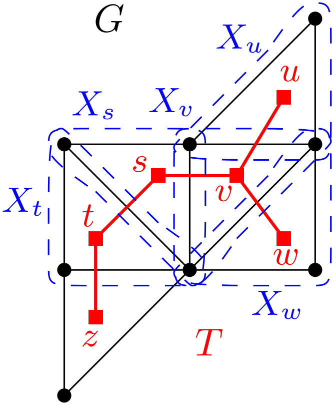
- Tree decomposition of a graph G:

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

  satisfying the following:

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

- Width of a tree decomposition:
  $\max_{t \in V(T)} |X_t| - 1$.
- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



An equivalent (and more common) definition of treewidth

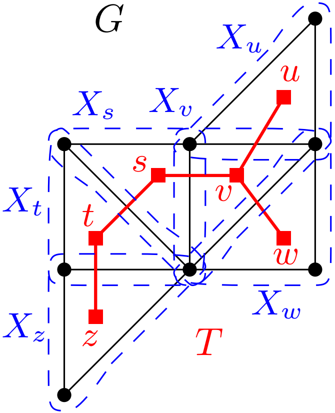
- Tree decomposition of a graph G:

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

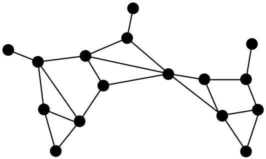
  satisfying the following:

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

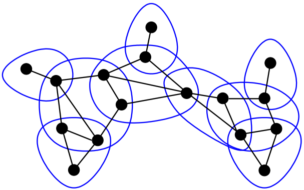
- Width of a tree decomposition:
  $\max_{t \in V(T)} |X_t| - 1$.
- Treewidth of a graph G, $\mathrm{tw}(G)$: minimum width of a tree decomposition of G.



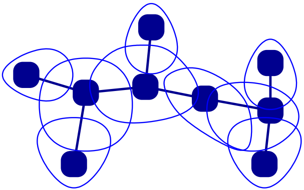
Treewidth measures the tree-likeness of a graph



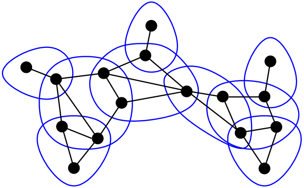
Treewidth measures the tree-likeness of a graph



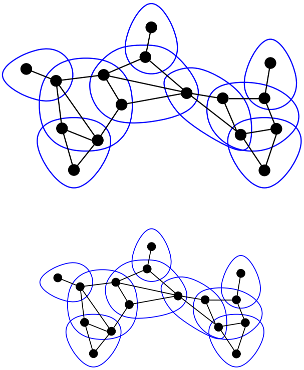
Treewidth measures the tree-likeness of a graph



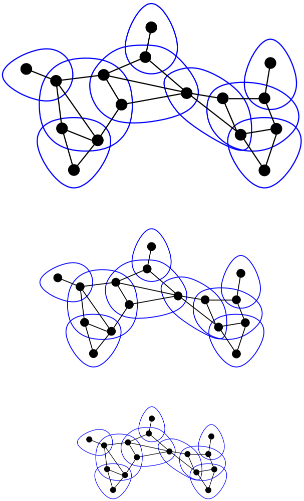
Treewidth measures the tree-likeness of a graph



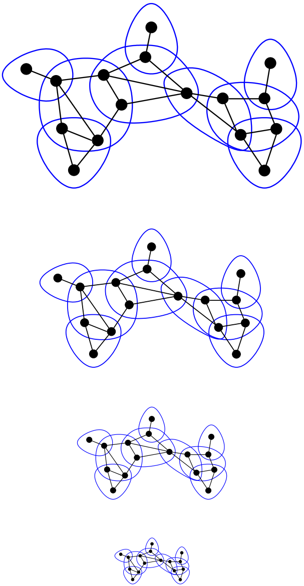
Treewidth measures the tree-likeness of a graph



Treewidth measures the tree-likeness of a graph



Treewidth measures the tree-likeness of a graph



Every bag of a tree decomposition is a separator

Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

Every bag of a tree decomposition is a separator

Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

- For every $t \in V(T)$, X_t is a separator in G.

Every bag of a tree decomposition is a separator

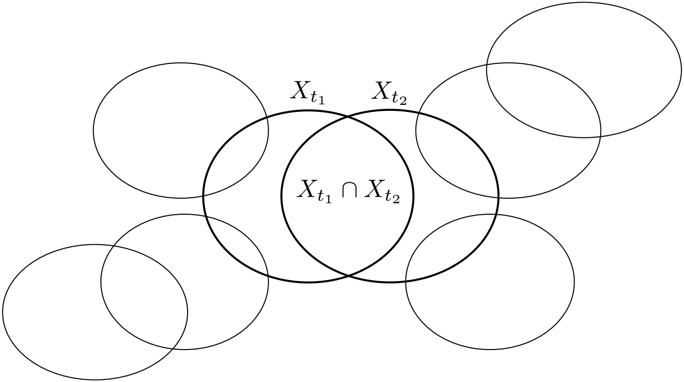
Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.

Every bag of a tree decomposition is a separator

Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

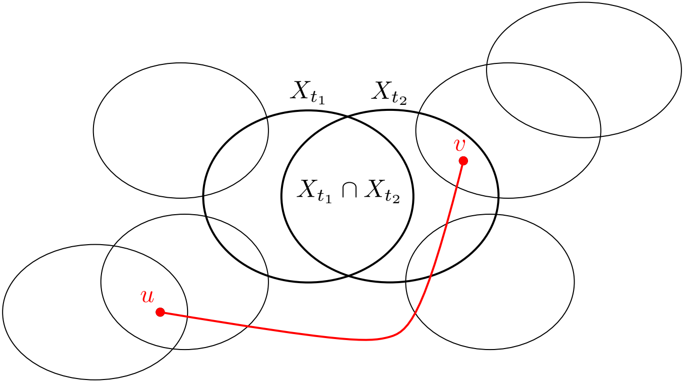
- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.



Every bag of a tree decomposition is a separator

Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

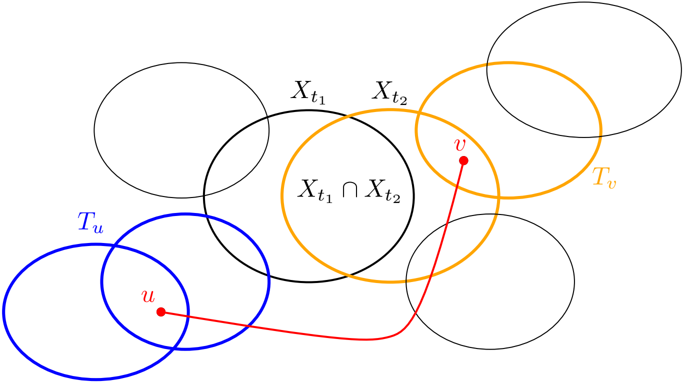
- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.



Every bag of a tree decomposition is a separator

Let $(T, \mathcal{X} = \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.



Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique.

Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t.

Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

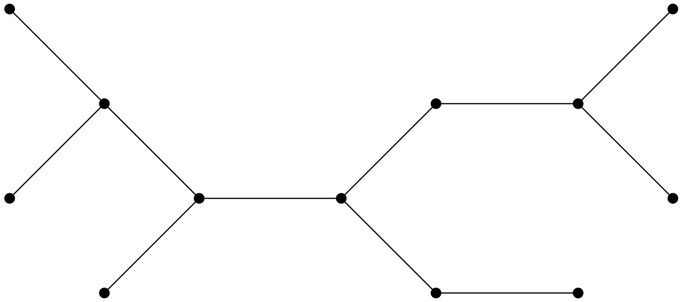
Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:

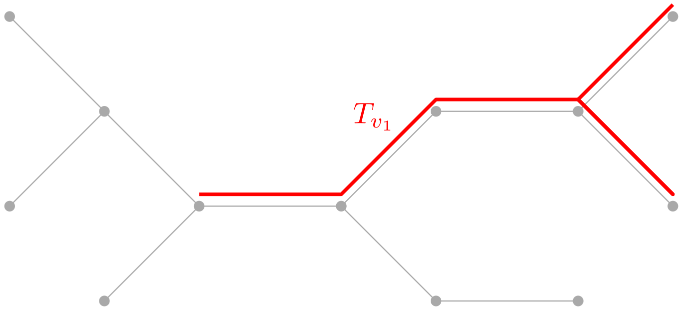


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:

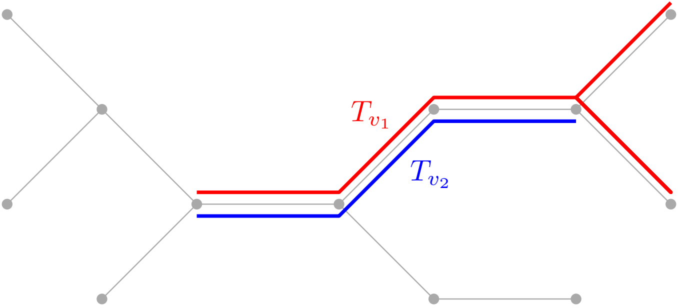


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \dots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \dots, v_{t-1}\}$:

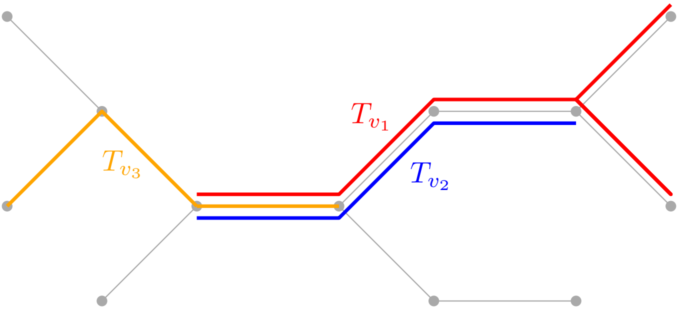


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:

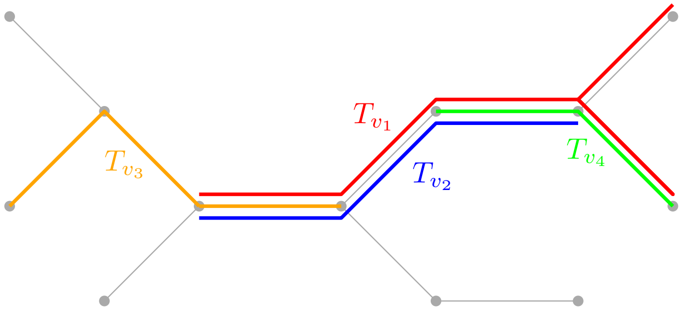


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:

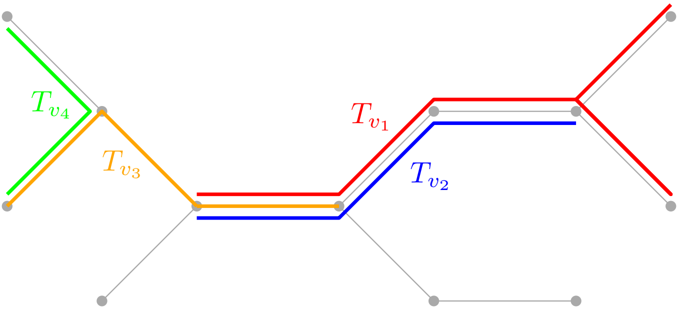


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:

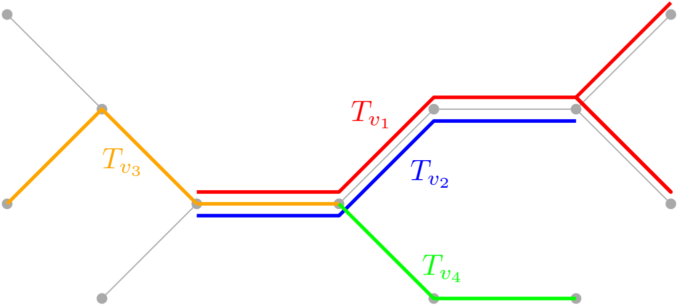


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:

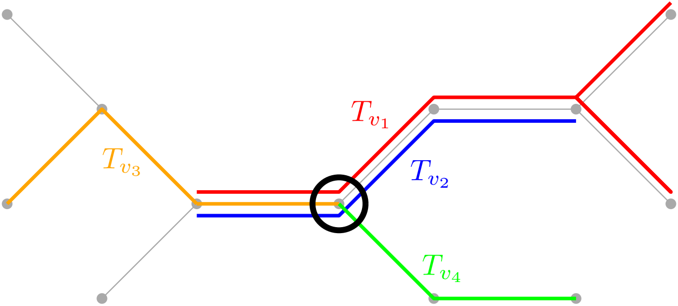


Every clique is entirely contained in some bag

Let G be a graph, $(T, \mathcal{X})$ be a tree decomposition of G, and let $K \subseteq V(G)$ be a clique. Then there exists a bag $X_t \in \mathcal{X}$ such that $K \subseteq X_t$.

Let $K = \{v_1, \ldots, v_t\}$. Proof by induction on t. True for $t \leq 2$.

Consider the subtrees in $(T, \mathcal{X})$ corresponding to vertices $\{v_1, \ldots, v_{t-1}\}$:



Examples

- If F is a forest, then $\mathrm{tw}(F) = 1$. (why?)

Examples

- If F is a forest, then $\mathrm{tw}(F) = 1$. (why?)
- If C is a cycle, then $\mathrm{tw}(C) = 2$. (why?)

Examples

- If F is a forest, then $\mathrm{tw}(F) = 1$. (why?)
- If C is a cycle, then $\mathrm{tw}(C) = 2$. (why?)
- If K_n is the clique on n vertices, then $\mathrm{tw}(K_n) = n - 1$.

Examples

- If F is a forest, then $\mathrm{tw}(F) = 1$. (why?)
- If C is a cycle, then $\mathrm{tw}(C) = 2$. (why?)
- If K_n is the clique on n vertices, then $\mathrm{tw}(K_n) = n - 1$.
- If $K_{a,b}$ is the complete bipartite graph with parts of sizes a and b, then $\mathrm{tw}(K_{a,b}) = \min\{a, b\}$. (why?)

Examples

- If F is a forest, then $\text{tw}(F) = 1$. (why?)
- If C is a cycle, then $\text{tw}(C) = 2$. (why?)
- If K_n is the clique on n vertices, then $\text{tw}(K_n) = n - 1$.
- If $K_{a,b}$ is the complete bipartite graph with parts of sizes a and b, then $\text{tw}(K_{a,b}) = \min\{a, b\}$. (why?)
- If G is an outerplanar graph, or a series-parallel graph, then $\text{tw}(G) \leq 2$. (why?)

Examples

- If F is a forest, then $\mathrm{tw}(F) = 1$. (why?)
- If C is a cycle, then $\mathrm{tw}(C) = 2$. (why?)
- If K_n is the clique on n vertices, then $\mathrm{tw}(K_n) = n - 1$.
- If $K_{a,b}$ is the complete bipartite graph with parts of sizes a and b, then $\mathrm{tw}(K_{a,b}) = \min\{a, b\}$. (why?)
- If G is an outerplanar graph, or a series-parallel graph, then $\mathrm{tw}(G) \leq 2$. (why?)
- If G is a planar graph on n vertices, then $\mathrm{tw}(G) = \mathcal{O}(\sqrt{n})$.

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 subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Brambles

How to provide a lower bound on the treewidth of a graph?

Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in G from A to B.

Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in G from A to B.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.

Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in G from A to B.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.

A bramble in a graph G is a family $\mathcal{B}$ of pairwise touching connected vertex sets of G.

Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in G from A to B.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.

A bramble in a graph G is a family $\mathcal{B}$ of pairwise touching connected vertex sets of G.

The order of a bramble $\mathcal{B}$ in a graph G is the minimum size of a vertex set $S \subseteq V(G)$ intersecting all the sets in $\mathcal{B}$.

Brambles

How to provide a lower bound on the treewidth of a graph?

Two sets $A, B \subseteq V(G)$ touch if either $A \cap B \neq \emptyset$ or there is an edge in G from A to B.

A set $S \subseteq V(G)$ is connected if $G[S]$ is connected.

A bramble in a graph G is a family $\mathcal{B}$ of pairwise touching connected vertex sets of G.

The order of a bramble $\mathcal{B}$ in a graph G is the minimum size of a vertex set $S \subseteq V(G)$ intersecting all the sets in $\mathcal{B}$.

Theorem (Robertson and Seymour. 1993)

For every $k \geq 0$ and graph G, the treewidth of G is at least k if and only if G contains a bramble of order at least $k+1$.

Another dual notion to treewidth: linkedness

[slides borrowed from Christophe Paul]

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are separable if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

Another dual notion to treewidth: linkedness

[slides borrowed from Christophe Paul]

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are separable if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

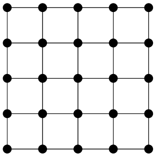
- For $k \geq 1$, a set $X \subseteq V(G)$ is k-well-linked if $|X| \geq k$ and $\forall\ Y, Z \subseteq X$, $|Y| = |Z| \leq k$, Y and Z are not separable.

Another dual notion to treewidth: linkedness

[slides borrowed from Christophe Paul]

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are separable if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

- For $k \geq 1$, a set $X \subseteq V(G)$ is k-well-linked if $|X| \geq k$ and $\forall\ Y, Z \subseteq X$, $|Y| = |Z| \leq k$, Y and Z are not separable.



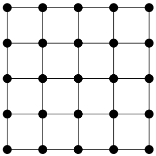
The perimeter of the $(k \times k)$-grid is k-well-linked (why?)

Another dual notion to treewidth: linkedness

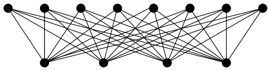
[slides borrowed from Christophe Paul]

- Two sets $Y, Z \subseteq V(G)$, with $|Y| = |Z|$, are separable if there is a set $S \subseteq V(G)$ with $|S| < |Y|$ and such that $G - S$ contains no path between $Y \setminus S$ and $Z \setminus S$.

- For $k \geq 1$, a set $X \subseteq V(G)$ is k-well-linked if $|X| \geq k$ and $\forall\ Y, Z \subseteq X$, $|Y| = |Z| \leq k$, Y and Z are not separable.



The perimeter of the $(k \times k)$-grid is k-well-linked (why?)



$K_{2k,k}$ is k-well-linked (why?)

Highly linked graphs have large treewidth

Lemma

If G contains a $(k+1)$-well-linked set X with $|X| \geqslant 3k$, then $\mathrm{tw}(G) \geq k$.

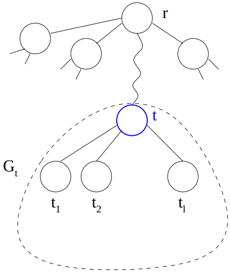
skip

Lemma

If G contains a $(k+1)$-well-linked set X with $|X| \geqslant 3k$, then $\mathrm{tw}(G) \geq k$.

skip

Contradiction: Consider a tree decomposition of G of width $< k$.



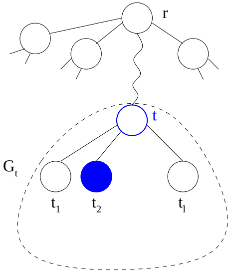
Let t be a "lowest" node with $|V_t \cap X| > 2k$.

Lemma

If G contains a $(k+1)$-well-linked set X with $|X| \geqslant 3k$, then $\mathrm{tw}(G) \geq k$.

skip

Contradiction: Consider a tree decomposition of G of width $< k$.



Let t be a "lowest" node with $|V_t \cap X| > 2k$.

If $\exists i \in [\ell]$ such that $|V_{t_i} \cap X| \geqslant k$, then we can choose $Y \subseteq V_{t_i} \cap X$, $|Y| = k$ and $Z \subseteq (V \setminus V_{t_i}) \cap X$, $|Z| = k$.

But $S = X_{t_i} \cap X_t$ separates Y and Z and $|S| \leq k - 1$.

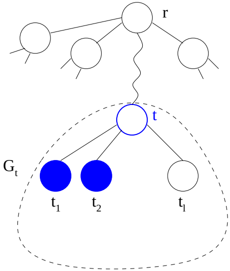
Highly linked graphs have large treewidth

Lemma

If G contains a $(k+1)$-well-linked set X with $|X| \geqslant 3k$, then $\mathrm{tw}(G) \geq k$.

skip

Contradiction: Consider a tree decomposition of G of width $< k$.



Let t be a "lowest" node with $|V_t \cap X| > 2k$.

Otherwise, let $W = V_{t_1} \cup \cdots \cup V_{t_i}$ with $|W \cap X| > k$ and $|(W \setminus V_{t_j}) \cap X| \leq k$ for $1 \leq j \leq i$.

$Y \subseteq W \cap X$, $|Y| = k+1$ and
$Z \subseteq (V \setminus W) \cap X$, $|Z| = k+1$ (why?).

But $S = X_t$ separates Y from Z and $|S| \leqslant k$.

Deciding linkedness is FPT

Lemma

Given a vertex set X of a graph G and $k \leq |X| \leq ck$ for some constant c, it is possible to decide whether X is k-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$.

Deciding linkedness is FPT

Lemma

Given a vertex set X of a graph G and $k \le |X| \le ck$ for some constant c, it is possible to decide whether X is k-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$.

- For every pair of subsets $Y, Z \subseteq X$ with $|Y| = |Z| \le k$, we can test whether Y and Z are separable in polynomial time (flow algorithm).

Deciding linkedness is FPT

Lemma

Given a vertex set X of a graph G and $k \leq |X| \leq ck$ for some constant c, it is possible to decide whether X is k-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$.

- For every pair of subsets $Y, Z \subseteq X$ with $|Y| = |Z| \leq k$, we can test whether Y and Z are separable in polynomial time (flow algorithm).
- Complexity: $4^{ck} \cdot n^{\mathcal{O}(1)}$. (why?)

Deciding linkedness is FPT

Lemma

Given a vertex set X of a graph G and $k \leq |X| \leq ck$ for some constant c, it is possible to decide whether X is k-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$.

- For every pair of subsets $Y, Z \subseteq X$ with $|Y| = |Z| \leq k$, we can test whether Y and Z are separable in polynomial time (flow algorithm).
- Complexity: $4^{ck} \cdot n^{\mathcal{O}(1)}$. (why?)

Remark: If X is not k-well-linked we can find, within the same running time, two separable subsets $Y, Z \subseteq X$.

Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $\mathcal{O}(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]
- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $\mathcal{O}(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]
- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $\mathcal{O}(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $\mathcal{O}(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $\mathcal{O}(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $5k + 4$ in time $2^{\mathcal{O}(k)} \cdot n$. [Bodlaender et al. 2016]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $\mathcal{O}(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $\mathcal{O}(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $2k+1$ in time $2^{\mathcal{O}(k)} \cdot n$. [Korhonen. 2021]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

- Deciding whether $\text{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]
- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $\mathcal{O}(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]
- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $\mathcal{O}(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]
- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $2k+1$ in time $2^{\mathcal{O}(k)} \cdot n$. [Korhonen. 2021]
- Either concludes that $\text{tw}(G) \geq k$ or finds a tree decomposition of width at most $\mathcal{O}(k \cdot \sqrt{\log k})$ in time $n^{\mathcal{O}(1)}$. [Feige, Hajiaghayi, Lee. 2008]

Complexity of computing the treewidth of a graph

Given a graph G on n vertices and a positive integer k:

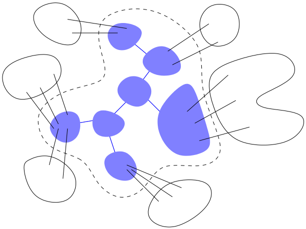
- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-complete. [Arnborg, Corneil, Proskurowski. 1987]
- Can be solved in time $k^{\mathcal{O}(k^3)} \cdot n$. [Bodlaender. 1996]

★ Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $4k$ in time $\mathcal{O}(3^{3k} \cdot k \cdot n^2)$. [Robertson and Seymour. 1995]

- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $9k/2$ in time $\mathcal{O}(2^{3k} \cdot k^{3/2} \cdot n^2)$. [Amir. 2010]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $2k+1$ in time $2^{\mathcal{O}(k)} \cdot n$. [Korhonen. 2021]
- Either concludes that $\mathrm{tw}(G) \geq k$ or finds a tree decomposition of width at most $\mathcal{O}(k \cdot \sqrt{\log k})$ in time $n^{\mathcal{O}(1)}$. [Feige, Hajiaghayi, Lee. 2008]

4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]

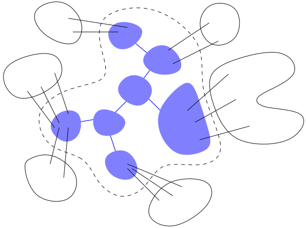


Idea

- We add vertices to a set U in a greedy way, until $U = V(G)$.

4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]

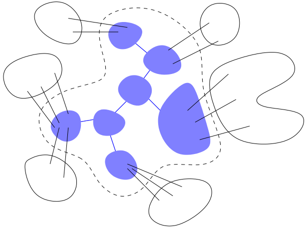


Idea

- We add vertices to a set U in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. $\text{width}(\mathcal{T}_U) < 4k$,

4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]

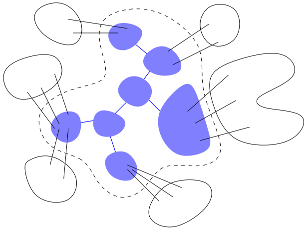


Idea

- We add vertices to a set U in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. $\text{width}(\mathcal{T}_U) < 4k$, unless we stop the algorithm and conclude that $\text{tw}(G) \geq k$.

4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]



Idea

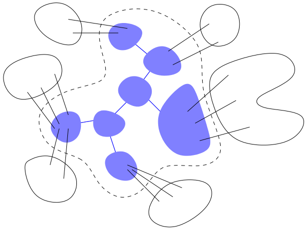
- We add vertices to a set U in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. width$(\mathcal{T}_U) < 4k$, unless we stop the algorithm and conclude that tw$(G) \geq k$.

Invariant

- Every connected component of $G - U$ has at most $3k$ neighbors in U.

4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]



Idea

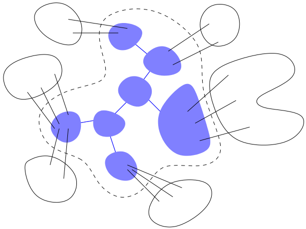
- We add vertices to a set U in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. width$(\mathcal{T}_U) < 4k$, unless we stop the algorithm and conclude that tw$(G) \geq k$.

Invariant

- Every connected component of $G - U$ has at most $3k$ neighbors in U.
- There exists a bag X_t of $\mathcal{T}_U$ containing all these neighbors.

4-approximation of Robertson and Seymour

[slides borrowed from Christophe Paul]



Idea

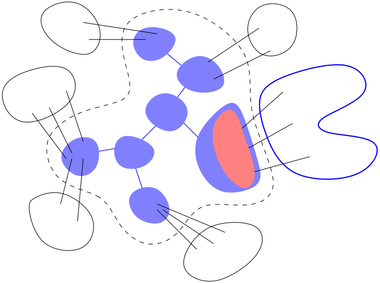
- We add vertices to a set U in a greedy way, until $U = V(G)$.
- We maintain a tree decomposition $\mathcal{T}_U$ of $G[U]$ s.t. $\text{width}(\mathcal{T}_U) < 4k$, unless we stop the algorithm and conclude that $\text{tw}(G) \geq k$.

Invariant

- Every connected component of $G - U$ has at most $3k$ neighbors in U.
- There exists a bag X_t of $\mathcal{T}_U$ containing all these neighbors.

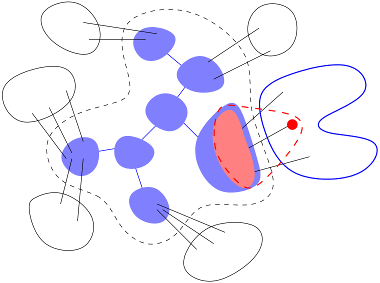
Initially, we start with U being any set of $3k$ vertices.

4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

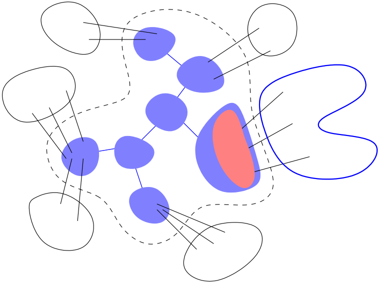
4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| < 3k$: we add a node t' neighbor of t such that $X_{t'} = \{x\} \cup X$, with $x \in C$ being a neighbor of X_t. The invariant is respected (why?).

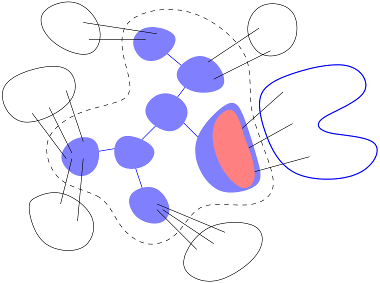
4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: test if X is $(k+1)$-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$:

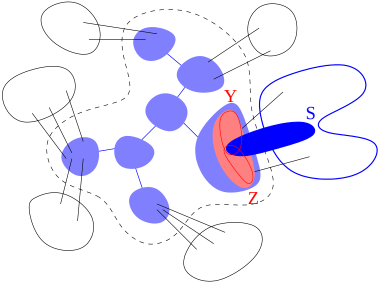
4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: test if X is $(k+1)$-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$:
  1. If X is $(k+1)$-well-linked, then $tw(G) \geq k$, and we stop.

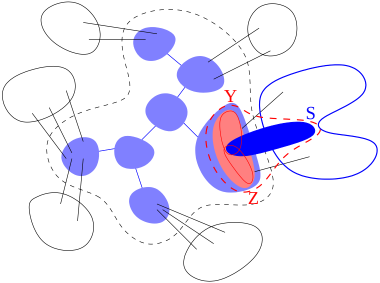
4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: test if X is $(k+1)$-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$:
  1. If X is $(k+1)$-well-linked, then $tw(G) \geq k$, and we stop.
  2. Otherwise, we find sets Y, Z, S with $|S| < |Y| = |Z| \leq k+1$ and such that S separates Y and Z.

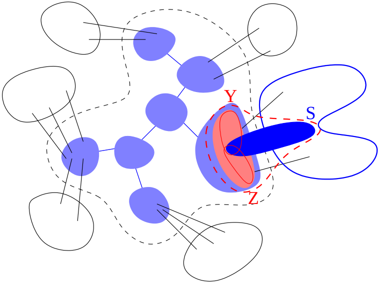
4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: test if X is $(k+1)$-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$:
  1. If X is $(k+1)$-well-linked, then $tw(G) \geq k$, and we stop.
  2. Otherwise, we find sets Y, Z, S with $|S| < |Y| = |Z| \leq k+1$ and such that S separates Y and Z.
     We create a node t' neighbor of t s.t. $X_{t'} = (S \cap C) \cup X$.

4-approximation of Robertson and Seymour (2)

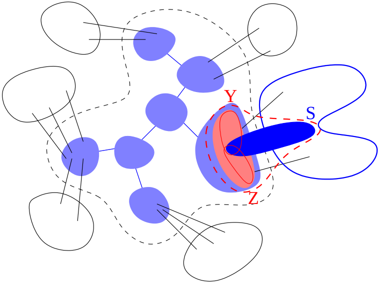


Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: test if X is $(k+1)$-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$:
  1. If X is $(k+1)$-well-linked, then $tw(G) \geq k$, and we stop.
  2. Otherwise, we find sets Y, Z, S with $|S| < |Y| = |Z| \leq k+1$ and such that S separates Y and Z.
     We create a node t' neighbor of t s.t. $X_{t'} = (S \cap C) \cup X$.

     Obs: the neighbors of every new component $C' \subseteq C$ are in $(X \setminus Z) \cup (S \cap C)$ or in $(X \setminus Y) \cup (S \cap C)$

4-approximation of Robertson and Seymour (2)



Let X be the neighbors of a component C and t be the node s.t. $X \subseteq X_t$.

- If $|X| = 3k$: test if X is $(k+1)$-well-linked in time $f(k) \cdot n^{\mathcal{O}(1)}$:
  1. If X is $(k+1)$-well-linked, then $tw(G) \geq k$, and we stop.
  2. Otherwise, we find sets Y, Z, S with $|S| < |Y| = |Z| \leq k+1$ and such that S separates Y and Z.
     We create a node t' neighbor of t s.t. $X_{t'} = (S \cap C) \cup X$.

     Obs: the neighbors of every new component $C' \subseteq C$ are in $(X \setminus Z) \cup (S \cap C)$ or in $(X \setminus Y) \cup (S \cap C)$ $\Rightarrow$ $\leq 3k$ neighbors.

Next subsection is...

1 Introduction to graph minors
2 Introduction to parameterized complexity
3 Treewidth
- Definition and simple properties
- Brambles and duality
- Computing treewidth
- Dynamic programming on tree decompositions
- Exploiting topology in dynamic programming

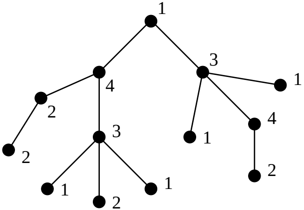
4 Bidimensionality
- Some ingredients and an illustrative example
- Meta-algorithms

5 Irrelevant vertex technique
6 Application to hitting minors
- Parameterized by treewidth
- Parameterized by solution size
- More general modification operations

7 Kernelization (?)

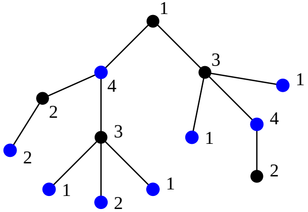
Weighted Independent Set on trees

[slides borrowed from Christophe Paul]



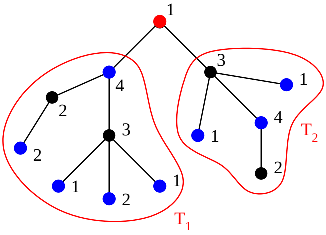
Weighted Independent Set on trees

[slides borrowed from Christophe Paul]



Weighted Independent Set on trees

[slides borrowed from Christophe Paul]

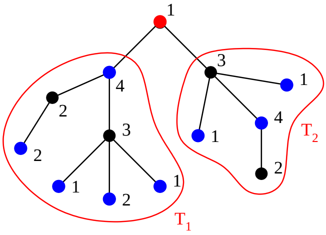


Observations:

1. Every vertex of a tree is a separator.
2. The union of independent sets of distinct connected components is an independent set.

Weighted Independent Set on trees

[slides borrowed from Christophe Paul]

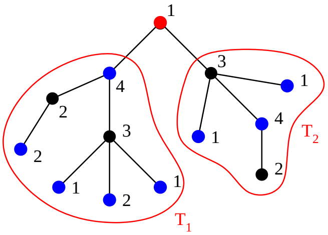


Let x be the root of T, $x_1 \dots x_\ell$ its children, $T_1, \dots T_\ell$ subtrees of $T - x$:

- $wIS(T, x)$: maximum weighted independent set containing x.
- $wIS(T, \overline{x})$: maximum weighted independent set not containing x.

Weighted Independent Set on trees

[slides borrowed from Christophe Paul]



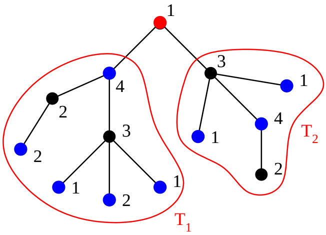
Let x be the root of T, $x_1 \dots x_\ell$ its children, $T_1, \dots T_\ell$ subtrees of $T - x$:

- $wIS(T, x)$: maximum weighted independent set containing x.
- $wIS(T, \overline{x})$: maximum weighted independent set not containing x.

$$\begin{cases} wIS(T,x) = \omega(x) + \sum_{i \in [\ell]} wIS(T_i, \overline{x_i}) \\ \end{cases}$$

Weighted Independent Set on trees

[slides borrowed from Christophe Paul]



Let x be the root of T, $x_1 \dots x_\ell$ its children, $T_1, \dots T_\ell$ subtrees of $T - x$:

- $wIS(T, x)$: maximum weighted independent set containing x.
- $wIS(T, \overline{x})$: maximum weighted independent set not containing x.

$$\begin{cases} wIS(T,x) & = & \omega(x) + \sum_{i \in [\ell]} wIS(T_i, \overline{x_i}) \\ wIS(T, \overline{x}) & = & \sum_{i \in [\ell]} \max\{wIS(T_i, x_i), wIS(T_i, \overline{x_i})\} \end{cases}$$

Dynamic programming on tree decompositions

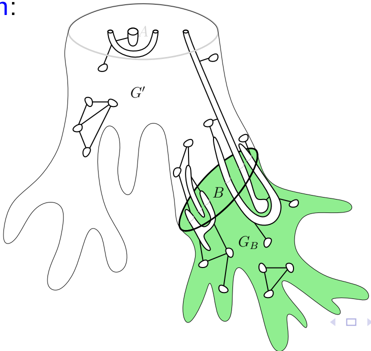
- Typically, FPT algorithms parameterized by treewidth are based on dynamic programming (DP) over a tree decomposition.

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]

Back to tree decompositions

Let $(T, \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.

Back to tree decompositions

Let $(T, \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.

Notation: If we root $(T, \{X_t \mid t \in V(T)\})$, then:

Back to tree decompositions

Let $(T, \{X_t \mid t \in V(T)\})$ be a tree decomposition of a graph G.

- For every $t \in V(T)$, X_t is a separator in G.
- For every edge $\{t_1, t_2\} \in E(T)$, $X_{t_1} \cap X_{t_2}$ is a separator in G.

Notation: If we root $(T, \{X_t \mid t \in V(T)\})$, then:

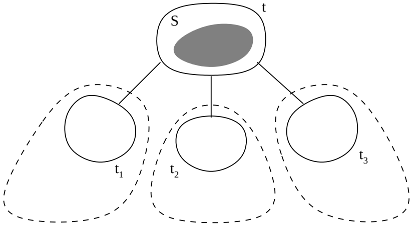
- V_t: all vertices of G appearing in bags that are descendants of t.
- $G_t = G[V_t]$.

INDEPENDENT SET on tree decompositions

$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S}$

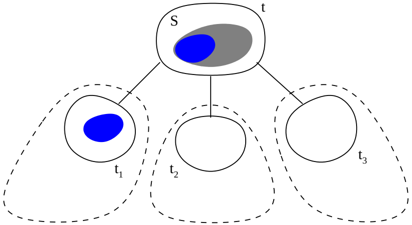
INDEPENDENT SET on tree decompositions

$\forall S \subseteq X_t$, $IS(S,t)$ = maximum independent set I of G_t s.t. $I \cap X_t = S$

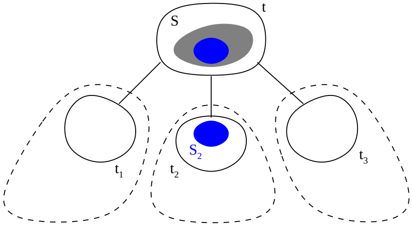


Independent Set on tree decompositions

$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S}$



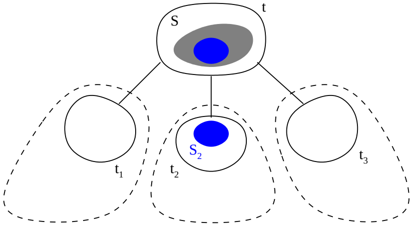
$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S}$



$\boxed{\text{Lemma}}$ If $S \subseteq X_t$ and $S_j = S \cap X_{t_j}$, then $|IS(S,t) \cap V_{t_j}| = |IS(S_j,t_j)|$.

Independent Set on tree decompositions

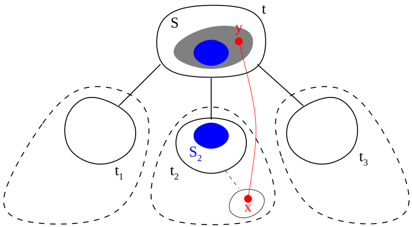
$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S}$



$\boxed{\text{Lemma}}$ If $S \subseteq X_t$ and $S_j = S \cap X_{t_j}$, then $|IS(S,t) \cap V_{t_j}| = |IS(S_j, t_j)|$.

For contradiction: suppose $IS(S,t) \cap V_{t_j}$ is not maximum in G_{t_j}.

$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S}$



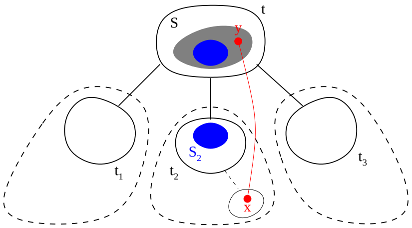
$\boxed{\text{Lemma}}$ If $S \subseteq X_t$ and $S_j = S \cap X_{t_j}$, then $|IS(S,t) \cap V_{t_j}| = |IS(S_j,t_j)|$.

For contradiction: suppose $IS(S,t) \cap V_{t_j}$ is not maximum in G_{t_j}.

$\Rightarrow$ $\exists y \in (S \setminus S_j) \subseteq X_t$ and $\exists x \in IS(S_j,t_j) \setminus X_{t_j}$ such that $\{x,y\} \in E(G)$.

Independent Set on tree decompositions

$$\boxed{\forall S \subseteq X_t,\ IS(S,t)\ = \text{maximum independent set } I \text{ of } G_t \text{ s.t. } I \cap X_t = S}$$



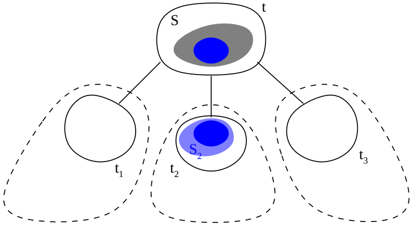
$\boxed{\text{Lemma}}$ If $S \subseteq X_t$ and $S_j = S \cap X_{t_j}$, then $|IS(S,t) \cap V_{t_j}| = |IS(S_j, t_j)|$.

For contradiction: suppose $IS(S,t) \cap V_{t_j}$ is not maximum in G_{t_j}.

$\Rightarrow$ $\exists y \in (S \setminus S_j) \subseteq X_t$ and $\exists x \in IS(S_j, t_j) \setminus X_{t_j}$ such that $\{x, y\} \in E(G)$.

Contradiction! X_{t_j} is not a separator.

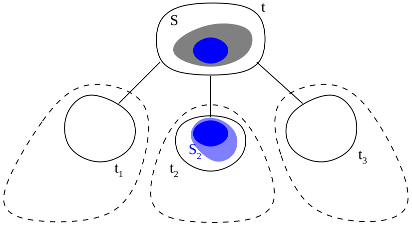
Idea of the dynamic programming algorithm:



How to compute $|IS(S,t)|$ from $|IS(S^i_j,t_j)|$, $\forall j \in [\ell]$, $\forall S^i_j \subseteq X_{t_j}$:

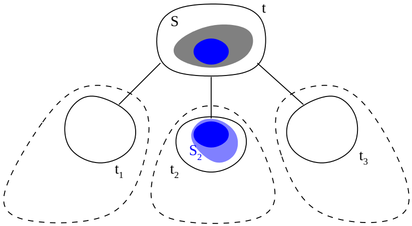
Independent Set on tree decompositions

Idea of the dynamic programming algorithm:



How to compute $|IS(S,t)|$ from $|IS(S_j^i, t_j)|$, $\forall j \in [\ell]$, $\forall S_j^i \subseteq X_{t_j}$:

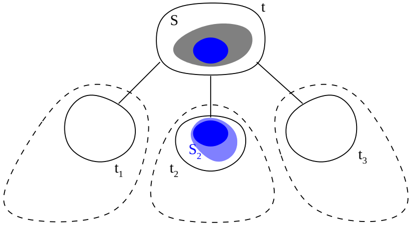
Idea of the dynamic programming algorithm:



How to compute $|IS(S,t)|$ from $|IS(S_j^i, t_j)|$, $\forall j \in [\ell]$, $\forall S_j^i \subseteq X_{t_j}$:

- verify that $S_j^i \cap X_t = S \cap X_{t_j} = S_j$ and $S_j \subseteq S_j^i$.

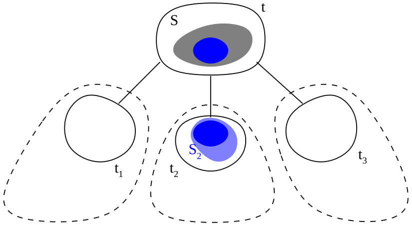
Idea of the dynamic programming algorithm:



How to compute $|IS(S,t)|$ from $|IS(S_j^i, t_j)|$, $\forall j \in [\ell]$, $\forall S_j^i \subseteq X_{t_j}$:

- verify that $S_j^i \cap X_t = S \cap X_{t_j} = S_j$ and $S_j \subseteq S_j^i$.
- verify that S_j^i is an independent set.

Idea of the dynamic programming algorithm:



How to compute $|IS(S,t)|$ from $|IS(S_j^i, t_j)|$, $\forall j \in [\ell]$, $\forall S_j^i \subseteq X_{t_j}$:

- verify that $S_j^i \cap X_t = S \cap X_{t_j} = S_j$ and $S_j \subseteq S_j^i$.
- verify that S_j^i is an independent set.

$$|IS(S,t)| = \left\{ \sum_{i\in[\ell]} \max \begin{array}{l} |S| + \\ \{|IS(S_j^i, t_j)| - |S_j| : \\ S_j^i \cap X_t = S_j \ \wedge \ S_j \subseteq S_j^i \text{ independent}\} \end{array} \right.$$

Independent Set on tree decompositions

$$|IS(S,t)| = \left\{ \sum_{i \in [\ell]} \max \begin{array}{l} |S| + \\ \{|IS(S_j^i, t_j)| - |S_j| : \\ S_j^i \cap X_t = S_j \ \wedge \ S_j \subseteq S_j^i \text{ independent}\} \end{array} \right.$$

Analysis of the running time, with bags of size k:

Independent Set on tree decompositions

$$|IS(S,t)| = \left\{ \begin{array}{ll} & |S| + \\ \sum_{i\in[\ell]} \max & \{|IS(S_j^i, t_j)| - |S_j| : \\ & S_j^i \cap X_t = S_j \ \wedge \ S_j \subseteq S_j^i \text{ independent}\} \end{array} \right.$$

Analysis of the running time, with bags of size k:

- Computing $IS(S,t)$: $\mathcal{O}(2^k \cdot k^2 \cdot \ell)$.

$$|IS(S,t)| = \left\{ \sum_{i\in[\ell]} \max \begin{array}{l} |S| + \\ \{|IS(S_j^i, t_j)| - |S_j| : \\ S_j^i \cap X_t = S_j \ \wedge \ S_j \subseteq S_j^i \text{ independent}\} \end{array} \right.$$

Analysis of the running time, with bags of size k:

- Computing $IS(S,t)$: $\mathcal{O}(2^k \cdot k^2 \cdot \ell)$.
- Computing $IS(S,t)$ for every $S \subseteq X_t$: $\mathcal{O}(2^k \cdot 2^k \cdot k^2 \cdot \ell)$.

INDEPENDENT SET on tree decompositions

$$|IS(S,t)| = \left\{ \sum_{i\in[\ell]} \max \begin{array}{l} |S| + \\ \{|IS(S_j^i, t_j)| - |S_j| : \\ S_j^i \cap X_t = S_j \ \wedge \ S_j \subseteq S_j^i \text{ independent}\} \end{array} \right.$$

Analysis of the running time, with bags of size k:

- Computing $IS(S,t)$: $\mathcal{O}(2^k \cdot k^2 \cdot \ell)$.
- Computing $IS(S,t)$ for every $S \subseteq X_t$: $\mathcal{O}(2^k \cdot 2^k \cdot k^2 \cdot \ell)$.
- Computing an optimal solution: $\mathcal{O}(4^k \cdot k^2 \cdot n)$.

Independent Set on tree decompositions

$$|IS(S,t)| = \left\{ \sum_{i\in[\ell]} \max \begin{array}{l} |S| + \\ \{|IS(S_j^i, t_j)| - |S_j| : \\ S_j^i \cap X_t = S_j \ \wedge \ S_j \subseteq S_j^i \text{ independent}\} \end{array} \right.$$

Analysis of the running time, with bags of size k:

- Computing $IS(S,t)$: $\mathcal{O}(2^k \cdot k^2 \cdot \ell)$.
- Computing $IS(S,t)$ for every $S \subseteq X_t$: $\mathcal{O}(2^k \cdot 2^k \cdot k^2 \cdot \ell)$.
- Computing an optimal solution: $\mathcal{O}(4^k \cdot k^2 \cdot n)$.

★ We have to add the time in order to compute a "good" tree decomposition of the input graph (as we have seen before).

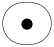
Helpful tool: nice tree decompositions

Helpful tool: nice tree decompositions

A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus \text{root}$ is of one of the following four types:

Helpful tool: nice tree decompositions

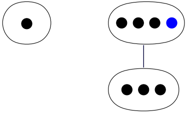
A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus \text{root}$ is of one of the following four types:



- Leaf: no children and $|X_t| = 1$.

Helpful tool: nice tree decompositions

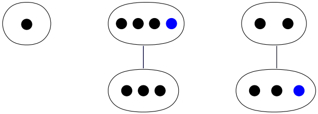
A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus \text{root}$ is of one of the following four types:



- Leaf: no children and $|X_t| = 1$.
- Introduce: a unique child t' and $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$.

Helpful tool: nice tree decompositions

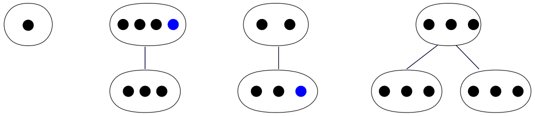
A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus$ root is of one of the following four types:



- Leaf: no children and $|X_t| = 1$.
- Introduce: a unique child t' and $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$.
- Forget: a unique child t' and $X_t = X_{t'} \setminus \{v\}$ with $v \in X_{t'}$.

Helpful tool: nice tree decompositions

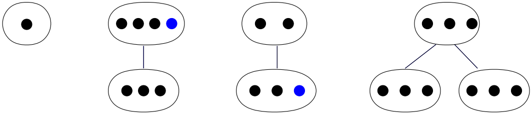
A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus$ root is of one of the following four types:



- Leaf: no children and $|X_t| = 1$.
- Introduce: a unique child t' and $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$.
- Forget: a unique child t' and $X_t = X_{t'} \setminus \{v\}$ with $v \in X_{t'}$.
- Join: two children t_1 and t_2 with $X_t = X_{t_1} = X_{t_2}$.

Helpful tool: nice tree decompositions

A rooted tree decomposition $(T, \{X_t : t \in T\})$ of a graph G is nice if every node $t \in V(T) \setminus \text{root}$ is of one of the following four types:



- Leaf: no children and $|X_t| = 1$.
- Introduce: a unique child t' and $X_t = X_{t'} \cup \{v\}$ with $v \notin X_{t'}$.
- Forget: a unique child t' and $X_t = X_{t'} \setminus \{v\}$ with $v \in X_{t'}$.
- Join: two children t_1 and t_2 with $X_t = X_{t_1} = X_{t_2}$.

Lemma

A tree decomposition $(T, \{X_t : t \in T\})$ of width k and x nodes of an n-vertex graph G can be transformed in time $\mathcal{O}(k^2 \cdot n)$ into a nice tree decomposition of G of width k and $\mathcal{O}(k \cdot x)$ nodes. (why?)

Simpler algorithm for INDEPENDENT SET

How to compute $IS(S, t)$ for every $S \subseteq X_t$:

Simpler algorithm for Independent Set

How to compute $IS(S,t)$ for every $S \subseteq X_t$:

- If t is a leaf: trivial.

Simpler algorithm for INDEPENDENT SET

How to compute $IS(S,t)$ for every $S \subseteq X_t$:

- If t is a leaf: trivial.
- t is an introduce node: $X_t = X_{t'} \cup \{v\}$

$$|IS(S,t)| = \begin{cases} |IS(S,t')| & \text{if } v \notin S \\ |IS(S \setminus \{v\}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\ -\infty & \text{otherwise} \end{cases}$$

Simpler algorithm for INDEPENDENT SET

How to compute $IS(S,t)$ for every $S \subseteq X_t$:

- If t is a leaf: trivial.
- t is an introduce node: $X_t = X_{t'} \cup \{v\}$

$$|IS(S,t)| = \begin{cases} |IS(S,t')| & \text{if } v \notin S \\ |IS(S \setminus \{v\}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\ -\infty & \text{otherwise} \end{cases}$$

- If t is a forget node: $X_t = X_{t'} \setminus \{v\}$

$$|IS(S,t)| = \max\{|IS(S,t')|, |IS(S \cup \{v\}, t')|\}$$

Simpler algorithm for INDEPENDENT SET

How to compute $IS(S,t)$ for every $S \subseteq X_t$:

- If t is a leaf: trivial.
- t is an introduce node: $X_t = X_{t'} \cup \{v\}$

$$|IS(S,t)| = \begin{cases} |IS(S,t')| & \text{if } v \notin S \\ |IS(S \setminus \{v\}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\ -\infty & \text{otherwise} \end{cases}$$

- If t is a forget node: $X_t = X_{t'} \setminus \{v\}$

$$|IS(S,t)| = \max\{|IS(S,t')|, |IS(S \cup \{v\}, t')|\}$$

- If t is a join node: $X_t = X_{t_1} = X_{t_2}$

$$|IS(S,t)| = |IS(S,t_1)| + |IS(S,t_2)| - |S|$$

Simpler algorithm for INDEPENDENT SET

How to compute $IS(S,t)$ for every $S \subseteq X_t$:

- If t is a leaf: trivial.
- t is an introduce node: $X_t = X_{t'} \cup \{v\}$

$$|IS(S,t)| = \begin{cases} |IS(S,t')| & \text{if } v \notin S \\ |IS(S \setminus \{v\}, t')| + 1 & \text{if } v \in S \text{ and } S \text{ independent} \\ -\infty & \text{otherwise} \end{cases}$$

- If t is a forget node: $X_t = X_{t'} \setminus \{v\}$

$$|IS(S,t)| = \max\{|IS(S,t')|, |IS(S \cup \{v\}, t')|\}$$

- If t is a join node: $X_t = X_{t_1} = X_{t_2}$

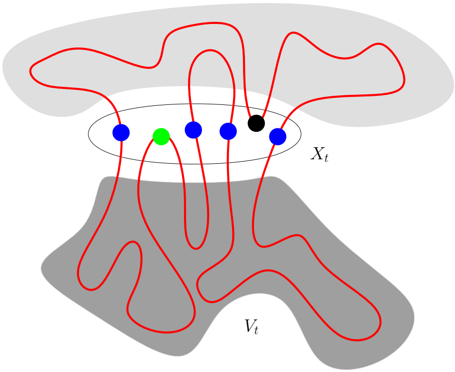
$$|IS(S,t)| = |IS(S,t_1)| + |IS(S,t_2)| - |S|$$

$$\boxed{\text{Complexity}: \mathcal{O}(2^k \cdot k^2 \cdot n)}$$

[slides borrowed from Christophe Paul]

Let $\mathcal{C}$ be a Hamiltonian cycle.

- Note that $\mathcal{C} \cap G[V_t]$ is a collection of paths.

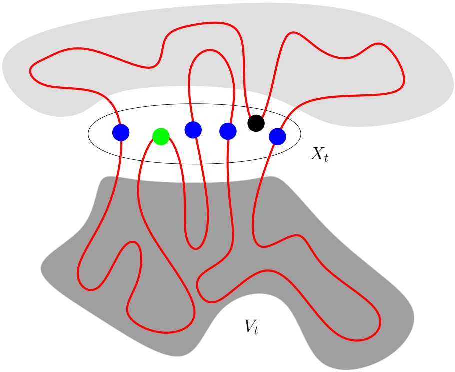


Hamiltonian Cycle on tree decompositions

[slides borrowed from Christophe Paul]

Let $\mathcal{C}$ be a Hamiltonian cycle.

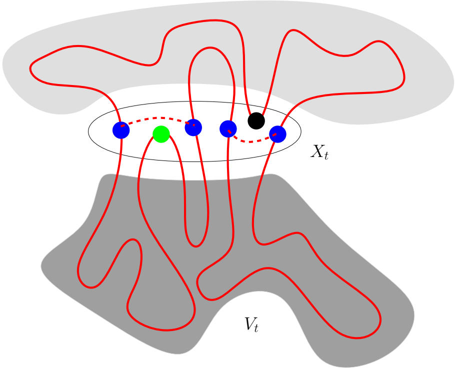
- Note that $\mathcal{C} \cap G[V_t]$ is a collection of paths.
- Partition of the bag X_t:
  - X_t^0: isolated in $G[V_t]$.
  - X_t^1: extremities of paths.
  - X_t^2: internal vertices.



[slides borrowed from Christophe Paul]

Let $\mathcal{C}$ be a Hamiltonian cycle.

- Note that $\mathcal{C} \cap G[V_t]$ is a collection of paths.
- Partition of the bag X_t:
  - X_t^0: isolated in $G[V_t]$.
  - X_t^1: extremities of paths.
  - X_t^2: internal vertices.



For every node t of the tree decomposition, we need to know if

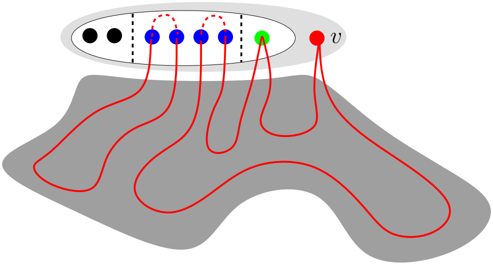
$$(X_t^0, X_t^1, X_t^2, M)$$

where M is a **matching** on X_t^1, corresponds to a partial solution.

skip

Forget node

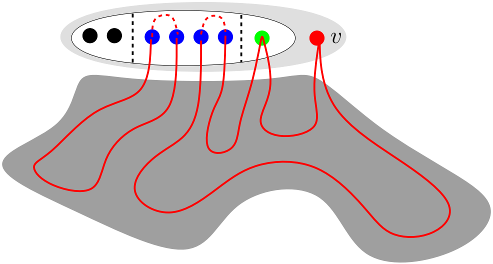
Let t be a forget node and t' its child such that $X_t = X_{t'} \setminus \{v\}$.



Claim X_t is a separator $\Rightarrow$
$\forall v \in V_t \setminus X_t$, v is internal in every partial solution.

Forget node

Let t be a forget node and t' its child such that $X_t = X_{t'} \setminus \{v\}$.



$\boxed{\text{Claim}}$ X_t is a separator $\Rightarrow$
$\forall v \in V_t \setminus X_t$, v is internal in every partial solution.

$$(X^0_{t'}, X^1_{t'}, X^2_{t'} \setminus \{v\}, M) \text{ is a partial solution for } t$$
$$\Leftrightarrow$$
$$(X^0_{t'}, X^1_{t'}, X^2_{t'}, M) \text{ is a partial solution for } t' \text{ with } v \in X^2_{t'}$$

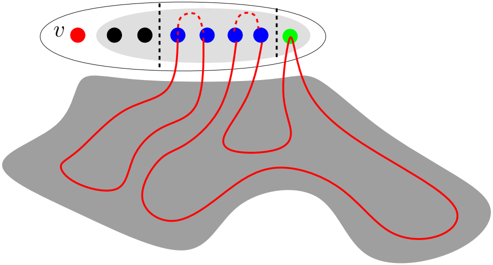
Introduce node

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

Introduce node

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

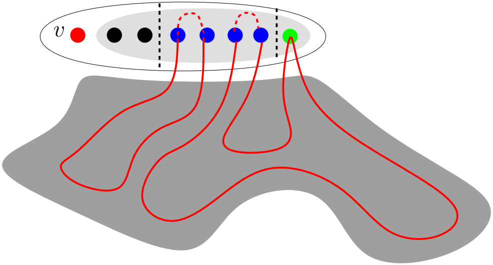
- Suppose: $v \in X_t^0$.



Introduce node

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^0$.

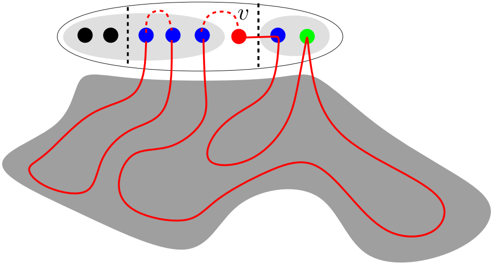


$$(X_{t'}^0 \cup \{v\}, X_{t'}^1, X_{t'}^2, M) \text{ is a partial solution for } t$$
$$\Leftrightarrow$$
$$(X_{t'}^0, X_{t'}^1, X_{t'}^2, M) \text{ is a partial solution for } t'$$

Introduce node (2)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

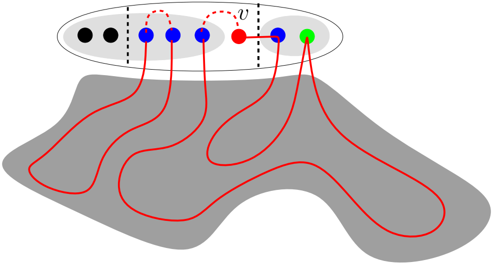
- Suppose: $v \in X_t^1$.



Introduce node (2)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.

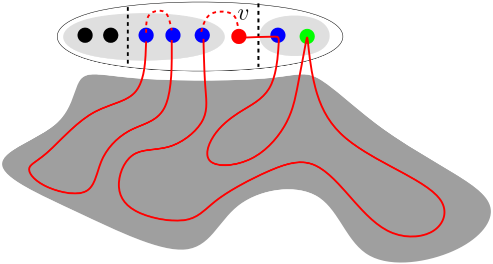


$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

Introduce node (2)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.



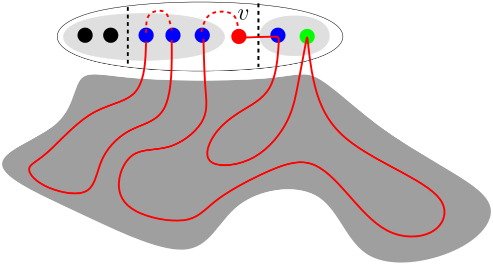
$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

- a vertex $u \in X_{t'}^1$ becomes internal $\Rightarrow$ $u \in X_t^2$.

Introduce node (2)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.



$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

- a vertex $u \in X_{t'}^1$ becomes internal $\Rightarrow$ $u \in X_t^2$.

$(X_{t'}^0, X_{t'}^1 \cup \{v\} \setminus \{u\}, X_{t'}^2 \cup \{u\}, M')$ is a partial solution for t

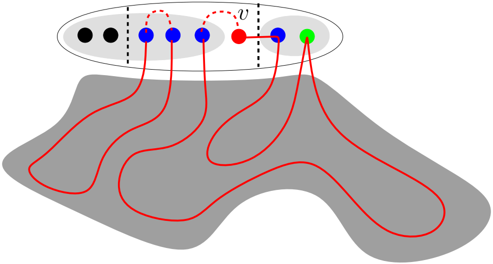
$\Leftrightarrow$

$(X_{t'}^0, X_{t'}^1, X_{t'}^2, M)$ is a partial solution for t'

Introduce node (2)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose: $v \in X_t^1$.



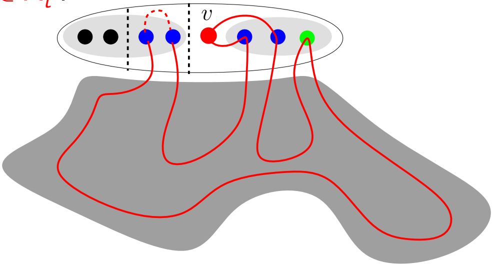
$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

- a vertex $u \in X_{t'}^1$ becomes internal $\Rightarrow$ $u \in X_t^2$.
- or a vertex $w \in X_{t'}^0$ becomes extremity of a path $\Rightarrow$ $w \in X_t^1$ (similar).

Introduce node (3)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.

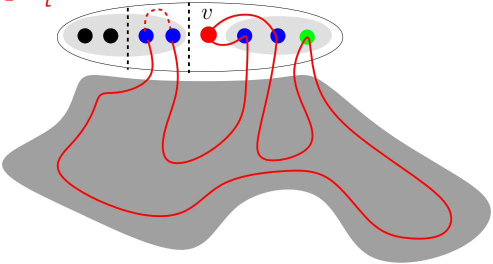


$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

Introduce node (3)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.



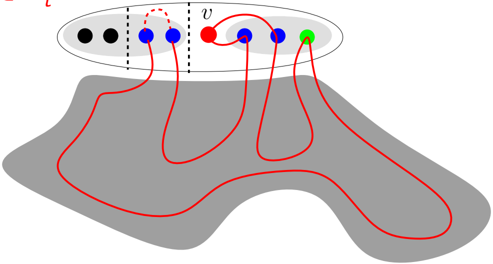
$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

1. two vertices $u, u' \in X_{t'}^1$ become internal $\Rightarrow u, u' \in X_t^2$.

Introduce node (3)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.



$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

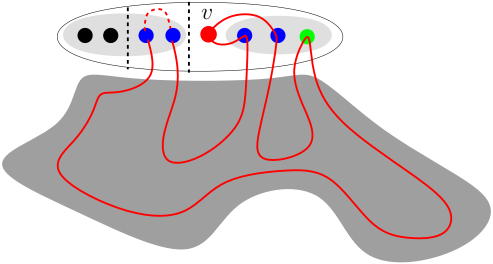
1. two vertices $u, u' \in X_{t'}^1$ become internal $\Rightarrow$ $u, u' \in X_t^2$.

$$(X_{t'}^0, X_{t'}^1 \setminus \{u, u'\}, X_{t'}^2 \cup \{v, u, u'\}, M') \text{ is a partial solution for } t$$
$$\Leftrightarrow$$
$$(X_{t'}^0, X_{t'}^1, X_{t'}^2, M) \text{ is a partial solution for } t'$$

Introduce node (3)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.



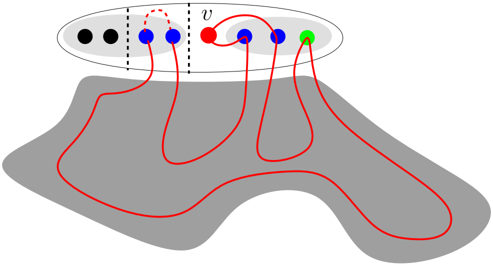
$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

1. two vertices $u, u' \in X_{t'}^1$ become internal $\Rightarrow u, u' \in X_t^2$.
2. two vertices $w, w' \in X_{t'}^0$ become extremities $\Rightarrow w, w' \in X_t^1$.

Introduce node (3)

Let t be an introduce node and t' its child such that $X_t = X_{t'} \cup \{v\}$.

- Suppose. $v \in X_t^2$.

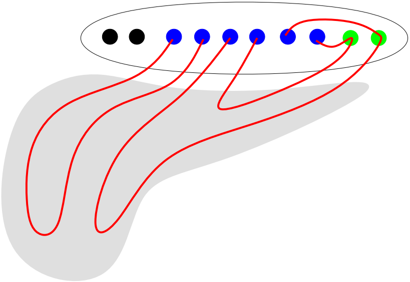


$\boxed{\text{Fact}}$ $X_{t'}$ is a separator $\Rightarrow$ $N(v) \cap V_t \subseteq X_t$.

1. two vertices $u, u' \in X_{t'}^1$ become internal $\Rightarrow u, u' \in X_t^2$.
2. two vertices $w, w' \in X_{t'}^0$ become extremities $\Rightarrow w, w' \in X_t^1$.
3. $w \in X_{t'}^0$ becomes extremity and $v \in X_{t'}^1$ internal $\Rightarrow w \in X_t^1, v \in X_t^2$.

Join node

Let t be a join node and t_1, t_2 its children such that $X_t = X_{t_1} = X_{t_2}$

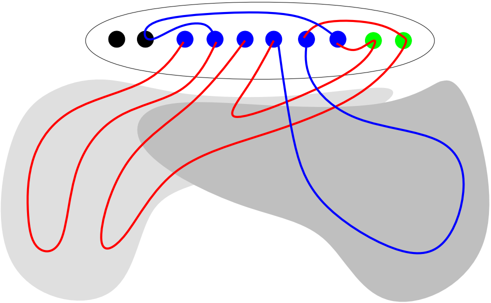


Fact For being compatible, partial solutions should verify:

- $X_{t_1}^2 \subseteq X_{t_2}^0$ and $X_{t_1}^1 \subseteq X_{t_2}^1 \cup X_{t_2}^0$.
- $X_{t_2}^2 \subseteq X_{t_1}^0$ and $X_{t_2}^1 \subseteq X_{t_1}^1 \cup X_{t_1}^0$.
- The union of matchings M_1 et M_2 does not create any cycle.

Join node

Let t be a join node and t_1, t_2 its children such that $X_t = X_{t_1} = X_{t_2}$



$\boxed{\text{Fact}}$ For being compatible, partial solutions should verify:

- $X^2_{t_1} \subseteq X^0_{t_2}$ and $X^1_{t_1} \subseteq X^1_{t_2} \cup X^0_{t_2}$.
- $X^2_{t_2} \subseteq X^0_{t_1}$ and $X^1_{t_2} \subseteq X^1_{t_1} \cup X^0_{t_1}$.
- The union of matchings M_1 et M_2 does not create any cycle.

HAMILTONIAN CYCLE on tree decompositions

Analysis of the running time, given a tree decomposition of width k:

Analysis of the running time, given a tree decomposition of width k:

- Number of subproblems at each node: : $3^k \cdot k!$.

HAMILTONIAN CYCLE on tree decompositions

Analysis of the running time, given a tree decomposition of width k:

- Number of subproblems at each node: : $3^k \cdot k!$.
- Number of nodes in a nice tree decomposition: $k \cdot n$.

HAMILTONIAN CYCLE on tree decompositions

Analysis of the running time, given a tree decomposition of width k:

- Number of subproblems at each node: : $3^k \cdot k!$.
- Number of nodes in a nice tree decomposition: $k \cdot n$.

Total running time of the algorithm: $k^{\mathcal{O}(k)} \cdot n$.

Hamiltonian Cycle on tree decompositions

Analysis of the running time, given a tree decomposition of width k:

- Number of subproblems at each node: : $3^k \cdot k!$.
- Number of nodes in a nice tree decomposition: $k \cdot n$.

Total running time of the algorithm: $k^{\mathcal{O}(k)} \cdot n$.

Can this approach be generalized to more problems?

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \mathrm{vertex}, \mathrm{edge}, I)$, where

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\vee$, $\wedge$, $\Rightarrow$, $\neg$, $=$, $\neq$.

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\vee$, $\wedge$, $\Rightarrow$, $\neg$, $=$, $\neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\vee$, $\wedge$, $\Rightarrow$, $\neg$, $=$, $\neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.
- Relations $\in$, $\subseteq$ on vertex/edge sets.

Monadic second order logic of graphs

We represent a graph $G = (V, E)$ with a structure $\mathcal{G} = (U, \text{vertex}, \text{edge}, I)$, where

- $U = V \cup E$ is the universe.
- "vertex" and "edge" are unary relations that allow to distinguish vertices and edges.
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ is the incidence relation.

An MSO formula is built using the following:

- Logical connectors $\vee$, $\wedge$, $\Rightarrow$, $\neg$, $=$, $\neq$.
- Predicates $\text{adj}(u, v)$ and $\text{inc}(e, v)$.
- Relations $\in$, $\subseteq$ on vertex/edge sets.
- Quantifiers $\exists$, $\forall$ on vertex/edge variables or vertex/edge **sets**.

(MSO_1/MSO_2)

Monadic second order logic of graphs: examples

$\boxed{\text{Example 1}}$ Expressing that $\{u, v\} \in E(G)$: $\quad \exists e \in E, \mathrm{inc}(u, e) \wedge \mathrm{inc}(v, e)$.

Monadic second order logic of graphs: examples

$\boxed{\text{Example 1}}$ Expressing that $\{u, v\} \in E(G)$: $\exists e \in E, \mathrm{inc}(u, e) \wedge \mathrm{inc}(v, e)$.

$\boxed{\text{Example 2}}$ Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

Monadic second order logic of graphs: examples

$\boxed{\text{Example 1}}$ Expressing that $\{u, v\} \in E(G)$: $\quad \exists e \in E, \mathrm{inc}(u, e) \wedge \mathrm{inc}(v, e)$.

$\boxed{\text{Example 2}}$ Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

$\boxed{\text{Example 3}}$ Expressing that a graph $G = (V, E)$ is connected.

Monadic second order logic of graphs: examples

$\boxed{\text{Example 1}}$ Expressing that $\{u, v\} \in E(G)$: $\exists e \in E, \mathrm{inc}(u, e) \wedge \mathrm{inc}(v, e)$.

$\boxed{\text{Example 2}}$ Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

$\boxed{\text{Example 3}}$ Expressing that a graph $G = (V, E)$ is connected.

- For every bipartition de V, there is a transversal edge:

Monadic second order logic of graphs: examples

Example 1 Expressing that $\{u, v\} \in E(G)$: $\exists e \in E, \text{inc}(u, e) \wedge \text{inc}(v, e)$.

Example 2 Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

Example 3 Expressing that a graph $G = (V, E)$ is connected.

- For every bipartition de V, there is a transversal edge:

Expressing that two sets V_1, V_2 define a bipartition of V:

$\forall v \in V, (v \in V_1 \vee v \in V_2) \wedge (v \in V_1 \Rightarrow v \notin V_2) \wedge (v \in V_2 \Rightarrow v \notin V_1)$.

Monadic second order logic of graphs: examples

$\boxed{\text{Example 1}}$ Expressing that $\{u, v\} \in E(G)$: $\quad \exists e \in E, \text{inc}(u, e) \wedge \text{inc}(v, e)$.

$\boxed{\text{Example 2}}$ Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

$\boxed{\text{Example 3}}$ Expressing that a graph $G = (V, E)$ is connected.

- For every bipartition de V, there is a transversal edge:

Expressing that two sets V_1, V_2 define a bipartition of V:

$\forall v \in V, (v \in V_1 \vee v \in V_2) \wedge (v \in V_1 \Rightarrow v \notin V_2) \wedge (v \in V_2 \Rightarrow v \notin V_1)$.

Connected: $\forall$ bipartition V_1, V_2, $\exists v_1 \in V_1, \exists v_2 \in V_2, \{v_1, v_2\} \in E(G)$.

Monadic second order logic of graphs: examples

Example 1 Expressing that $\{u, v\} \in E(G)$: $\exists e \in E, \text{inc}(u, e) \wedge \text{inc}(v, e)$.

Example 2 Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

Example 3 Expressing that a graph $G = (V, E)$ is connected.

- For every bipartition de V, there is a transversal edge:

Expressing that two sets V_1, V_2 define a bipartition of V:

$\forall v \in V, (v \in V_1 \vee v \in V_2) \wedge (v \in V_1 \Rightarrow v \notin V_2) \wedge (v \in V_2 \Rightarrow v \notin V_1)$.

Connected: $\forall$ bipartition V_1, V_2, $\exists v_1 \in V_1, \exists v_2 \in V_2, \{v_1, v_2\} \in E(G)$.

Other properties that can be expressed in MSO_2:

- a set being a vertex cover, independent set. (why?)

Monadic second order logic of graphs: examples

$\boxed{\text{Example 1}}$ Expressing that $\{u, v\} \in E(G)$: $\exists e \in E, \text{inc}(u, e) \wedge \text{inc}(v, e)$.

$\boxed{\text{Example 2}}$ Expressing that a set $S \subseteq V(G)$ is a dominating set.

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

$\boxed{\text{Example 3}}$ Expressing that a graph $G = (V, E)$ is connected.

- For every bipartition de V, there is a transversal edge:

Expressing that two sets V_1, V_2 define a bipartition of V:

$\forall v \in V, (v \in V_1 \vee v \in V_2) \wedge (v \in V_1 \Rightarrow v \notin V_2) \wedge (v \in V_2 \Rightarrow v \notin V_1)$.

Connected: $\forall$ bipartition V_1, V_2, $\exists v_1 \in V_1, \exists v_2 \in V_2, \{v_1, v_2\} \in E(G)$.

Other properties that can be expressed in MSO_2:

- a set being a vertex cover, independent set. (why?)
- a graph being k-colorable (for fixed k), having a Hamiltonian cycle.

Theorem (Courcelle. 1990)

Every problem expressible in MSO$_2$ *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and* *treewidth* *at most* tw.

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw.

The function $f(\mathrm{tw})$ depends on the structure of the MSO_2 formula.

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw*.*

The function $f(\mathrm{tw})$ depends on the structure of the MSO_2 formula.

Withing the same running time, one can also optimize the size of a vertex/edge set satisfying an MSO_2 formula.

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw*.*

The function $f(\mathrm{tw})$ depends on the structure of the MSO_2 formula.

Withing the same running time, one can also optimize the size of a vertex/edge set satisfying an MSO_2 formula.

Examples: Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, k-Coloring for fixed k, ...

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw*.*

The function $f(\mathrm{tw})$ depends on the structure of the MSO_2 formula.

Withing the same running time, one can also optimize the size of a vertex/edge set satisfying an MSO_2 formula.

Examples: Vertex Cover, Dominating Set, Hamiltonian Cycle, Clique, Independent Set, k-Coloring for fixed k, ...

In parameterized complexity: FPT parameterized by treewidth.

Are there only good news for treewidth?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *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.

Are there only good news for treewidth?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *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.

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

Are there only good news for treewidth?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 can be solved in time $f(\mathsf{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

In parameterized complexity: FPT parameterized by treewidth.

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]

Are there only good news for treewidth?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 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.

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 are even NP-hard on graphs of constant treewidth: Steiner Forest ($\mathrm{tw} = 3$), Bandwidth ($\mathrm{tw} = 1$).

Are there only good news for treewidth?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 can be solved in time $f(\mathsf{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

In parameterized complexity: FPT parameterized by treewidth.

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 are even NP-hard on graphs of constant treewidth: Steiner Forest ($\mathsf{tw} = 3$), Bandwidth ($\mathsf{tw} = 1$).

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

Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Is it enough to prove that a problem is FPT?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\text{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw.

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

$$f(\text{tw}) \cdot n$$

Is it enough to prove that a problem is FPT?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw*.*

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

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

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

Is it enough to prove that a problem is FPT?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 can be solved in time $f(\mathsf{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

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

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

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

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

This is a very active area in parameterized complexity.

Is it enough to prove that a problem is FPT?

Theorem (Courcelle. 1990)

Every problem expressible in MSO_2 *can be solved in time* $f(\mathrm{tw}) \cdot n$ *on graphs on* n *vertices and treewidth at most* tw*.*

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

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

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

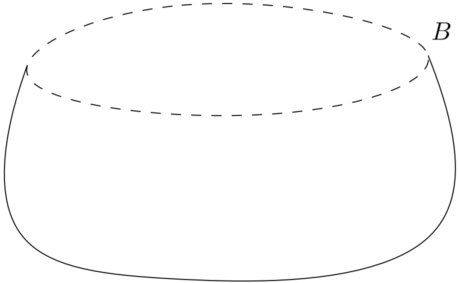
$\boxed{\text{Major goal}}$ find the smallest possible function $f(\mathrm{tw})$.

This is a very active area in parameterized complexity.

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

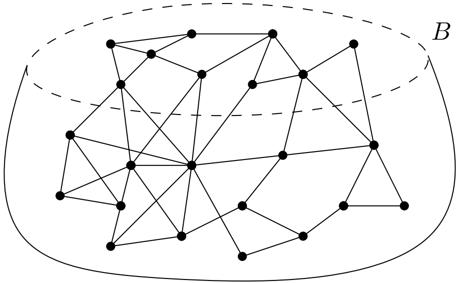
Two behaviors for problems parameterized by treewidth

$\boxed{\text{Local problems}}$ Vertex Cover, Dominating Set, Clique, Independent Set, q-Coloring for fixed q.



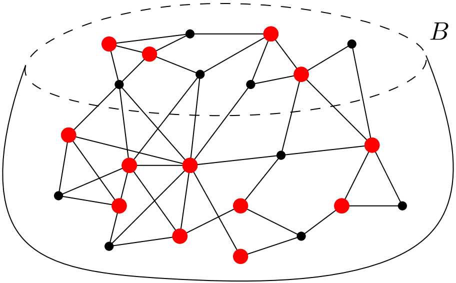
Two behaviors for problems parameterized by treewidth

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



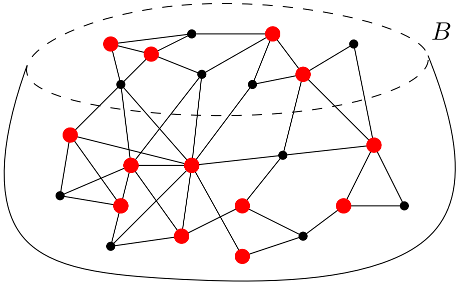
Two behaviors for problems parameterized by treewidth

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



Two behaviors for problems parameterized by treewidth

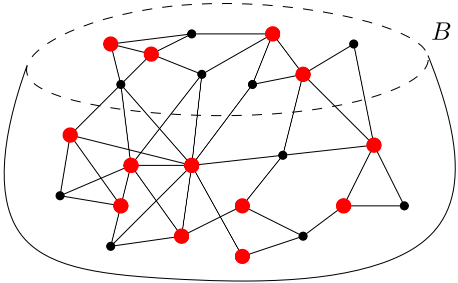
Local problems: VERTEX COVER, DOMINATING SET, CLIQUE, INDEPENDENT SET, q-COLORING for fixed q.



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

Two behaviors for problems parameterized by treewidth

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

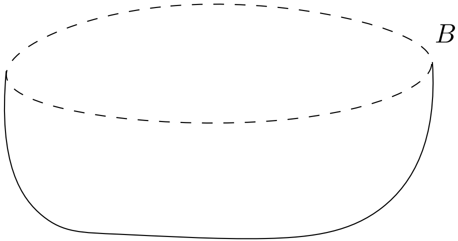


- It is sufficient to store, for each bag B, the subset of vertices of B that belong to a partial solution: 2^{tw} choices
- The "natural" DP algorithms lead to (optimal) single-exponential algorithms:

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

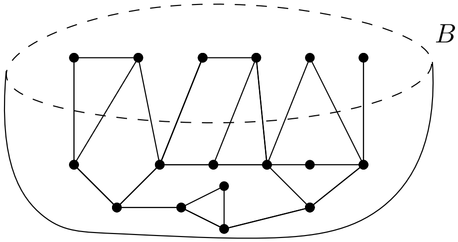
Connectivity problems seem to be more complicated...

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



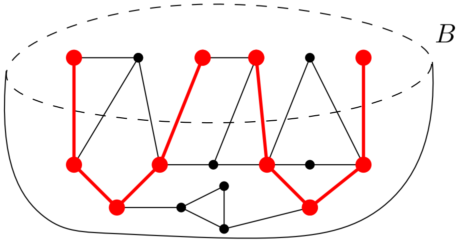
Connectivity problems seem to be more complicated...

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



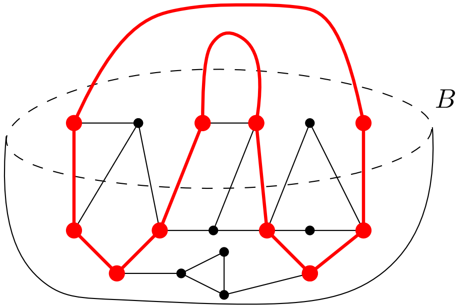
Connectivity problems seem to be more complicated...

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



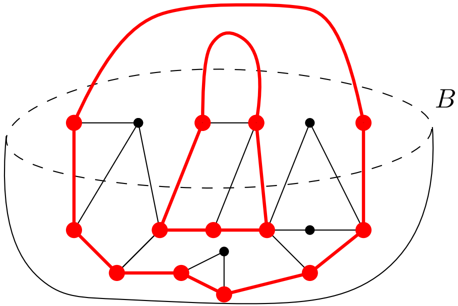
Connectivity problems seem to be more complicated...

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



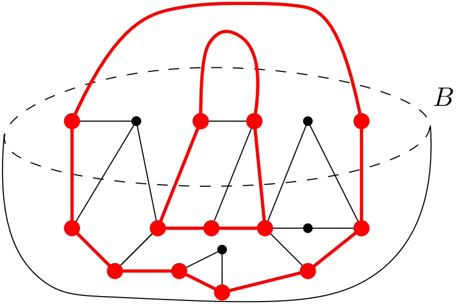
Connectivity problems seem to be more complicated...

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



Connectivity problems seem to be more complicated...

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

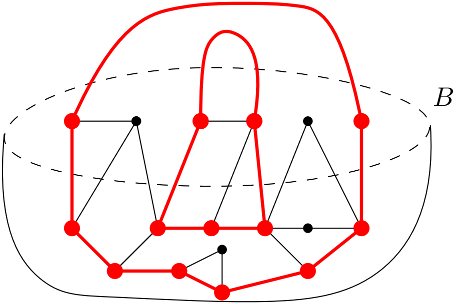


- Now it is not sufficient to store the subset of vertices of B that belong to a partial solution, but also how they are matched:

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

Connectivity problems seem to be more complicated...

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



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

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

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

Two types of behavior

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

- Local problems:

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

Vertex Cover, Dominating Set, ...

- Connectivity problems:

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

Longest Path, Steiner Tree, ...

How topology helps for dynamic programming?

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

How topology helps for dynamic programming?

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

- We consider a special tree-decomposition of a sparse graph, and exploit the structure of the subgraph induced by the bags.

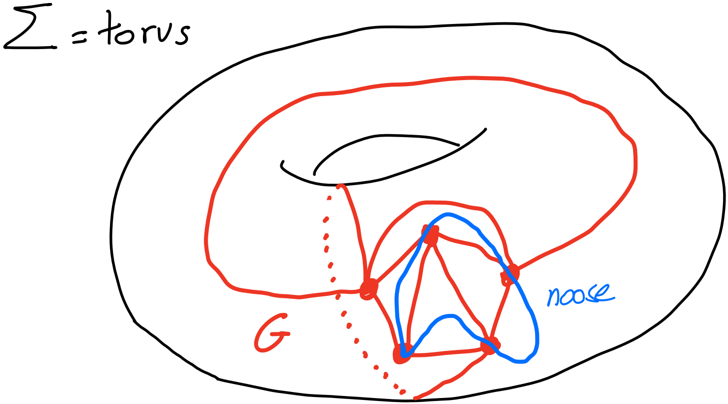
How topology helps for dynamic programming?

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

- We consider a special tree-decomposition of a sparse graph, and exploit the structure of the subgraph induced by the bags.
- More precisely, we use the existence of tree decompositions of small width and with nice topological properties.
- These nice properties do $\boxed{\text{not}}$ change the DP algorithms, but the analysis of their running time.

Nooses

Let G be a graph embedded in a surface Σ. A noose is a subset of Σ homeomorphic to $\mathbb{S}^1$ that meets G only at vertices.



- Let G be a planar graph. A sphere cut decomposition of G is a tree decomposition $(T, \{X_t : t \in V(T)\})$ of G such that the vertices in each bag X_t are situated around a noose in the plane.

[NB: several details are missing in this definition]

- Let G be a planar graph. A sphere cut decomposition of G is a tree decomposition $(T, \{X_t : t \in V(T)\})$ of G such that the vertices in each bag X_t are situated around a noose in the plane.

Theorem (Seymour and Thomas. 1994)

Every planar graph G has a sphere cut decomposition whose width is at most $\frac{3}{2} \cdot \mathrm{tw}(G)$, and that can be computed in polynomial time.

- Let G be a planar graph. A sphere cut decomposition of G is a tree decomposition $(T, \{X_t : t \in V(T)\})$ of G such that the vertices in each bag X_t are situated around a noose in the plane.

Theorem (Seymour and Thomas. 1994)

Every planar graph G has a sphere cut decomposition whose width is at most $\frac{3}{2} \cdot \mathrm{tw}(G)$, and that can be computed in polynomial time.

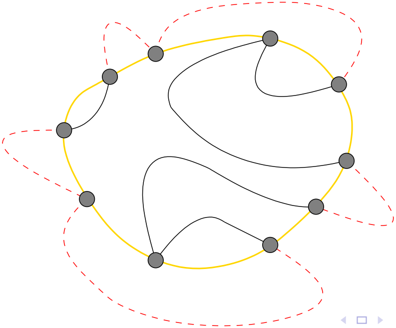
- The size of the tables of a DP algorithm depends on how many ways a partial solution can intersect the vertices in a bag X_t.

- Let G be a planar graph. A sphere cut decomposition of G is a tree decomposition $(T, \{X_t : t \in V(T)\})$ of G such that the vertices in each bag X_t are situated around a noose in the plane.

Theorem (Seymour and Thomas. 1994)

Every planar graph G has a sphere cut decomposition whose width is at most $\frac{3}{2} \cdot \mathrm{tw}(G)$, and that can be computed in polynomial time.

- The size of the tables of a DP algorithm depends on how many ways a partial solution can intersect the vertices in a bag X_t.

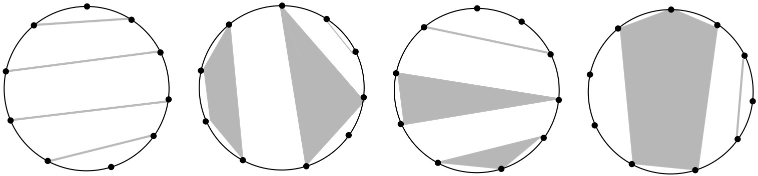


Using sphere cut decompositions

- Suppose we do DP on a sphere cut decomposition of width $\leq k$.

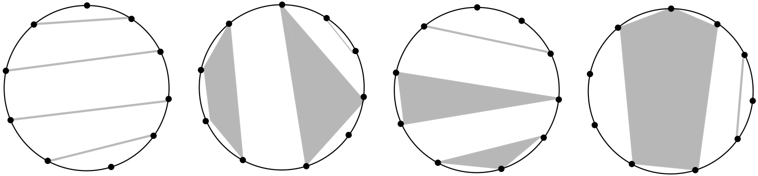
Using sphere cut decompositions

- Suppose we do DP on a sphere cut decomposition of width $\leq k$.
- In how many ways can we draw polygons inside a circle such that they touch the circle only on its k vertices and they do not intersect?



Using sphere cut decompositions

- Suppose we do DP on a sphere cut decomposition of width $\leq k$.
- In how many ways can we draw polygons inside a circle such that they touch the circle only on its k vertices and they do not intersect?



- Exactly the number of *non-crossing partitions* over k elements, which is given by the k-th Catalan number:

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

How to use this framework?

1. Let **P** be a "packing-encodable" problem on a planar graph G.

How to use this framework?

1. Let **P** be a "packing-encodable" problem on a planar graph G.

2. As a preprocessing step, build a surface cut decomposition of G, using the theorem of Seymour and Thomas.

How to use this framework?

1. Let **P** be a "packing-encodable" problem on a planar graph G.
2. As a preprocessing step, build a surface cut decomposition of G, using the theorem of Seymour and Thomas.
3. Run a "natural" DP algorithm to solve **P** over the obtained surface cut decomposition.

How to use this framework?

1. Let **P** be a "packing-encodable" problem on a planar graph G.
2. As a preprocessing step, build a surface cut decomposition of G, using the theorem of Seymour and Thomas.
3. Run a "natural" DP algorithm to solve **P** over the obtained surface cut decomposition.
4. The single-exponential running time is just a consequence of the topological properties of surface cut decomposition.

How to use this framework?

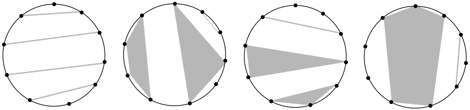
1. Let **P** be a "packing-encodable" problem on a planar graph G.
2. As a preprocessing step, build a surface cut decomposition of G, using the theorem of Seymour and Thomas.
3. Run a "natural" DP algorithm to solve **P** over the obtained surface cut decomposition.
4. The single-exponential running time is just a consequence of the topological properties of surface cut decomposition.

This idea was first used in [Dorn, Penninkx, Bodlaender, Fomin. 2005]

Generalizations to other sparse graph classes

Main idea special type of decomposition with nice topological properties:

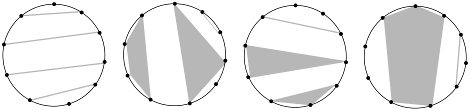
partial solutions $\Longleftrightarrow$ non-crossing partitions



Generalizations to other sparse graph classes

Main idea special type of decomposition with nice topological properties:

partial solutions $\Longleftrightarrow$ non-crossing partitions



This idea has been generalized to other graph classes and problems:

- Graphs on surfaces: [Dorn, Fomin, Thilikos '06] [Rué, S., Thilikos '10]
- H-minor-free graphs: [Dorn, Fomin, Thilikos '08] [Rué, S., Thilikos '12]

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

No!

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

End of the story?

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

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

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

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

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

End of the story?

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

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

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

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

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

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

There are other examples of such problems (as we may see later)...

Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

A few representative problems

VERTEX COVER
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $C \subseteq V$ of size at most k such that $G[V \setminus C]$ is an independent set?

A few representative problems

VERTEX COVER
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $C \subseteq V$ of size at most k such that $G[V \setminus C]$ is an independent set?

LONG PATH
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a path P in G of length at least k?

A few representative problems (II)

Feedback Vertex Set
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $F \subseteq V$ of size at most k such that for $G[V \setminus F]$ is a forest?

A few representative problems (II)

FEEDBACK VERTEX SET
Input: A graph $G = (V, E)$ and a positive integer k.
Parameter: k.
Question: Does there exist a subset $F \subseteq V$ of size at most k such that for $G[V \setminus F]$ is a forest?

DOMINATING SET
Input: A graph $G = (V, E)$ and a positive integers k.
Parameter: k.
Question: Does there exist a subset $D \subseteq V$ of size at most k such that for all $v \in V$, $N[v] \cap D \neq \emptyset$?

Minor-closed parameters

- A graph class $\mathcal{G}$ is *minor (contraction)-closed* if any minor (contraction) of a graph in $\mathcal{G}$ is also in $\mathcal{G}$.

Minor-closed parameters

- A graph class $\mathcal{G}$ is *minor (contraction)-closed* if any minor (contraction) of a graph in $\mathcal{G}$ is also in $\mathcal{G}$.
- A parameter P is any function mapping graphs to nonnegative integers.

Minor-closed parameters

- A graph class $\mathcal{G}$ is *minor (contraction)-closed* if any minor (contraction) of a graph in $\mathcal{G}$ is also in $\mathcal{G}$.

- A parameter P is any function mapping graphs to nonnegative integers.

- The *parameterized problem associated with* P asks, for some fixed k, whether for a given graph G, $P(G) \leq k$ (for minimization) or $P(G) \geq k$ (for maximization problem).

Minor-closed parameters

- A graph class $\mathcal{G}$ is *minor (contraction)-closed* if any minor (contraction) of a graph in $\mathcal{G}$ is also in $\mathcal{G}$.
- A parameter P is any function mapping graphs to nonnegative integers.
- The *parameterized problem associated with* P asks, for some fixed k, whether for a given graph G, $P(G) \leq k$ (for minimization) or $P(G) \geq k$ (for maximization problem).
- We say that a parameter P is *closed under taking of minors/contractions* (or, briefly, *minor/contraction-closed*) if for every graph H, $H \preceq_m G$ / $H \preceq_{cm} G$ implies that $P(H) \leq P(G)$.

Examples of minor/contraction closed parameters

- Minor-closed parameters:

  VERTEX COVER, FEEDBACK VERTEX SET, LONG PATH, TREEWIDTH, . . . (why?)

Examples of minor/contraction closed parameters

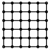
- Minor-closed parameters:

  Vertex Cover, Feedback Vertex Set, Long Path, Treewidth, ... (why?)

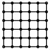
- Contraction-closed parameters:

  Dominating Set, Connected Vertex Cover, r-Dominating Set, ... (why?)

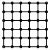
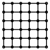
Grid Exclusion Theorem

- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  ℓ

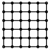
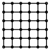
Grid Exclusion Theorem

- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_{\ell}$ We have $\text{tw}\,(H_{\ell,\ell}) = \ell$.

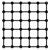
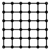
Grid Exclusion Theorem

- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_{\ell}$ We have $\text{tw}\,(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if  $_{\ell} \preceq_m G$, then $\text{tw}(G) \geq \text{tw}(H_{\ell,\ell}) = \ell$.

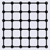
Grid Exclusion Theorem

- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_\ell$ We have $\mathrm{tw}\,(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if  $_\ell \preceq_m G$, then $\mathrm{tw}(G) \geq \mathrm{tw}(H_{\ell,\ell}) = \ell$. Does the reverse implication hold?

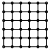
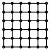
Grid Exclusion Theorem

- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid: $_\ell$ We have $\mathbf{tw}(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if $_\ell \preceq_m G$, then $\mathbf{tw}(G) \geq \mathbf{tw}(H_{\ell,\ell}) = \ell$. $\boxed{\text{Does the reverse implication hold?}}$

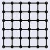
Theorem (Robertson and Seymour. 1986)

For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains $_\ell$ *as a minor.*

Grid Exclusion Theorem

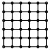
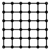
- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_\ell$ We have $\mathrm{tw}\,(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if  $_\ell \preceq_m G$, then $\mathrm{tw}(G) \geq \mathrm{tw}(H_{\ell,\ell}) = \ell$. Does the reverse implication hold?

Theorem (Robertson and Seymour. 1986)

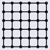
For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains  $_\ell$ *as a minor.*

- Smallest possible function $c(\ell)$?

Grid Exclusion Theorem

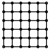
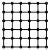
- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_\ell$ We have $\text{tw}(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if  $_\ell \preceq_m G$, then $\text{tw}(G) \geq \text{tw}(H_{\ell,\ell}) = \ell$. Does the reverse implication hold?

Theorem (Robertson and Seymour. 1986)

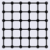
For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains  $_\ell$ *as a minor.*

- Smallest possible function $c(\ell)$? $\Omega(\ell^2 \log \ell) \leq c(\ell) \leq 20^{2^{\ell^5}}$

Grid Exclusion Theorem

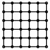
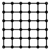
- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_\ell$ We have $\text{tw}(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if  $_\ell \preceq_m G$, then $\text{tw}(G) \geq \text{tw}(H_{\ell,\ell}) = \ell$. Does the reverse implication hold?

Theorem (Robertson and Seymour. 1986)

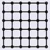
For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains  $_\ell$ *as a minor.*

- Smallest possible function $c(\ell)$? $\Omega(\ell^2 \log \ell) \leq c(\ell) \leq 20^{2\ell^5}$
- Some improvement: $c(\ell) = 2^{O(\ell \log \ell)}$. [Leaf and Seymour. 2012]

Grid Exclusion Theorem

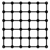
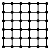
- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid: $_\ell$ We have $\mathrm{tw}\,(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if $_\ell \preceq_m G$, then $\mathrm{tw}(G) \geq \mathrm{tw}(H_{\ell,\ell}) = \ell$. $\boxed{\text{Does the reverse implication hold?}}$

Theorem (Robertson and Seymour. 1986)

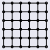
For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains $_\ell$ *as a minor.*

- Smallest possible function $c(\ell)$? $\Omega(\ell^2 \log \ell) \leq c(\ell) \leq 20^{2\ell^5}$
- Some improvement: $c(\ell) = 2^{O(\ell \log \ell)}$. [Leaf and Seymour. 2012]
- Recent breakthrough: $c(\ell) = \mathrm{poly}(\ell)$. [Chekuri and Chuzhoy. 2013]

Grid Exclusion Theorem

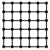
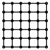
- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid: $_\ell$ We have $\text{tw}\,(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if $_\ell \preceq_m G$, then $\text{tw}(G) \geq \text{tw}(H_{\ell,\ell}) = \ell$. Does the reverse implication hold?

Theorem (Robertson and Seymour. 1986)

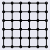
For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains $_\ell$ *as a minor.*

- Smallest possible function $c(\ell)$? $\Omega(\ell^2 \log \ell) \leq c(\ell) \leq 20^{2\ell^5}$
- Some improvement: $c(\ell) = 2^{O(\ell \log \ell)}$. [Leaf and Seymour. 2012]
- Recent breakthrough: $c(\ell) = \text{poly}(\ell)$. [Chekuri and Chuzhoy. 2013]
  $c(\ell) = O(\ell^9 \text{polylog} \ell)$. [Chuzhoy and Tan. 2021]

Grid Exclusion Theorem

- Let $H_{\ell,\ell}$ be the $(\ell \times \ell)$-grid:  $_\ell$ We have $\text{tw}\,(H_{\ell,\ell}) = \ell$.
- As TREEWIDTH is minor-closed, if  $_\ell \preceq_m G$, then $\text{tw}(G) \geq \text{tw}(H_{\ell,\ell}) = \ell$. Does the reverse implication hold?

Theorem (Robertson and Seymour. 1986)

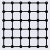
For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains  $_\ell$ *as a minor.*

- Smallest possible function $c(\ell)$? $\Omega(\ell^2 \log \ell) \leq c(\ell) \leq 20^{2\ell^5}$
- Some improvement: $c(\ell) = 2^{O(\ell \log \ell)}$. [Leaf and Seymour. 2012]
- Recent breakthrough: $c(\ell) = \text{poly}(\ell)$. [Chekuri and Chuzhoy. 2013]
  $c(\ell) = O(\ell^9 \text{polylog} \ell)$. [Chuzhoy and Tan. 2021]

Important message grid-minors are the certificate of large treewidth.

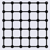
Grid Exclusion Theorems on sparse graphs

Theorem (Robertson, Seymour, Thomas. 1994)

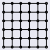
Every planar graph of treewidth $\geq 6 \cdot \ell$ *contains* $_{\ell}$ *as a minor.*

Grid Exclusion Theorems on sparse graphs

Theorem (Robertson, Seymour, Thomas. 1994)

Every planar graph of treewidth $\geq 6 \cdot \ell$ contains $_\ell$ *as a minor.*

Theorem (Demaine, Fomin, Hajiaghayi, Thilikos. 2005)

For every fixed g, there is a constant c_g such that every graph of genus g and of treewidth $\geq c_g \cdot \ell$ contains $_\ell$ *as a minor.*

Grid Exclusion Theorems on sparse graphs

Theorem (Robertson, Seymour, Thomas. 1994)

Every planar graph of treewidth $\geq 6 \cdot \ell$ contains $\boxplus_\ell$ as a minor.

Theorem (Demaine, Fomin, Hajiaghayi, Thilikos. 2005)

For every fixed g, there is a constant c_g such that every graph of genus g and of treewidth $\geq c_g \cdot \ell$ contains $\boxplus_\ell$ as a minor.

Theorem (Demaine and Hajiaghayi. 2008)

For every fixed graph H, there is a constant c_H such that every H-minor-free graph of treewidth $\geq c_H \cdot \ell$ contains $\boxplus_\ell$ as a minor.

Best constant in the above theorem is by [Kawarabayashi and Kobayashi. 2012]

Grid Exclusion Theorems on sparse graphs

Theorem (Robertson, Seymour, Thomas. 1994)

Every planar graph of treewidth $\geq 6 \cdot \ell$ contains $\boxplus_\ell$ as a minor.

Theorem (Demaine, Fomin, Hajiaghayi, Thilikos. 2005)

For every fixed g, there is a constant c_g such that every graph of genus g and of treewidth $\geq c_g \cdot \ell$ contains $\boxplus_\ell$ as a minor.

Theorem (Demaine and Hajiaghayi. 2008)

For every fixed graph H, there is a constant c_H such that every H-minor-free graph of treewidth $\geq c_H \cdot \ell$ contains $\boxplus_\ell$ as a minor.

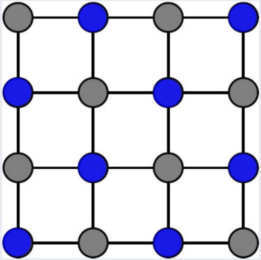
Best constant in the above theorem is by [Kawarabayashi and Kobayashi. 2012]

In sparse graphs: linear dependency between treewidth and grid-minors

How to use Grid Theorems algorithmically?

Example: FPT algorithm for Planar Vertex Cover

A vertex cover of a graph G is a set of vertices C such that every edge of G has at least one endpoint in C. Min size: $\mathbf{vc}(G)$.



Example: FPT algorithm for Planar Vertex Cover

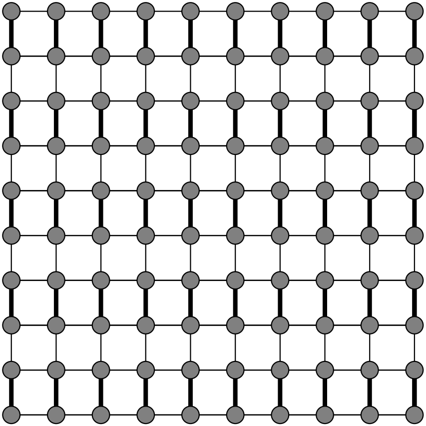
INPUT: Planar graph G on n vertices, and an integer k.

OUTPUT: Either a vertex cover of G of size $\leq k$, or a proof that G has no such a vertex cover.

RUNNING TIME: $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

Objective subexponential FPT algorithm for PLANAR VERTEX COVER.

Example: FPT algorithm for Planar Vertex Cover



$\mathbf{vc}(H_{\ell,\ell}) \geq \frac{\ell^2}{2}$

Example: FPT algorithm for Planar Vertex Cover

Let G be a planar graph of treewidth $\geq 6 \cdot \ell$

Example: FPT algorithm for Planar Vertex Cover

Let G be a planar graph of treewidth $\geq 6 \cdot \ell$ $\Longrightarrow$ G contains the $(\ell \times \ell)$-grid $H_{\ell,\ell}$ as a minor

Example: FPT algorithm for Planar Vertex Cover

Let G be a planar graph of treewidth $\geq 6 \cdot \ell$ $\Longrightarrow$ G contains the $(\ell \times \ell)$-grid $H_{\ell,\ell}$ as a minor

- The size of any vertex cover of $H_{\ell,\ell}$ is at least $\ell^2/2$.
- Recall that Vertex Cover is a minor-closed parameter.
- Since $H_{\ell,\ell} \preceq_m G$, it holds that $\mathbf{vc}(G) \geq \mathbf{vc}(H_{\ell,\ell}) \geq \ell^2/2$.

We are already very close to an algorithm...

Recall:

- k is the parameter of the problem.
- We have that $\text{tw}(G) = 6 \cdot \ell$ and ℓ is the size of a grid-minor of G.
- Therefore, $\mathbf{vc}(G) \geq \ell^2/2$.

We are already very close to an algorithm...

Recall:

- k is the parameter of the problem.
- We have that $\mathrm{tw}(G) = 6 \cdot \ell$ and ℓ is the size of a grid-minor of G.
- Therefore, $\mathbf{vc}(G) \geq \ell^2/2$.

WIN/WIN approach:

- If $k < \ell^2/2$, we can safely answer "NO".

We are already very close to an algorithm...

Recall:

- k is the parameter of the problem.
- We have that $\mathrm{tw}(G) = 6 \cdot \ell$ and ℓ is the size of a grid-minor of G.
- Therefore, $\mathbf{vc}(G) \geq \ell^2/2$.

WIN/WIN approach:

- If $k < \ell^2/2$, we can safely answer "NO".
- If $k \geq \ell^2/2$, then $\mathrm{tw}(G) = O(\ell) = O(\sqrt{k})$,

We are already very close to an algorithm...

Recall:

- k is the parameter of the problem.
- We have that $\mathrm{tw}(G) = 6 \cdot \ell$ and ℓ is the size of a grid-minor of G.
- Therefore, $\mathbf{vc}(G) \geq \ell^2/2$.

WIN/WIN approach:

- If $k < \ell^2/2$, we can safely answer "NO".
- If $k \geq \ell^2/2$, then $\mathrm{tw}(G) = O(\ell) = O(\sqrt{k})$, and we can solve the problem by standard DP in time $2^{O(\mathrm{tw}(G))} \cdot n^{O(1)}$

We are already very close to an algorithm...

Recall:

- k is the parameter of the problem.
- We have that $\mathrm{tw}(G) = 6 \cdot \ell$ and ℓ is the size of a grid-minor of G.
- Therefore, $\mathbf{vc}(G) \geq \ell^2/2$.

WIN/WIN approach:

- If $k < \ell^2/2$, we can safely answer "NO".
- If $k \geq \ell^2/2$, then $\mathrm{tw}(G) = O(\ell) = O(\sqrt{k})$, and we can solve the problem by standard DP in time $2^{O(\mathrm{tw}(G))} \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.

We are already very close to an algorithm...

Recall:

- k is the parameter of the problem.
- We have that $\mathrm{tw}(G) = 6 \cdot \ell$ and ℓ is the size of a grid-minor of G.
- Therefore, $\mathbf{vc}(G) \geq \ell^2/2$.

WIN/WIN approach:

- If $k < \ell^2/2$, we can safely answer "NO".
- If $k \geq \ell^2/2$, then $\mathrm{tw}(G) = O(\ell) = O(\sqrt{k})$, and we can solve the problem by standard DP in time $2^{O(\mathrm{tw}(G))} \cdot n^{O(1)} = 2^{O(\sqrt{k})} \cdot n^{O(1)}$.

This gives a subexponential FPT algorithm!

Was VERTEX COVER really just an example...?

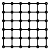
$\boxed{\text{What is so special in VERTEX COVER?}}$

$\boxed{\text{Where did we use planarity?}}$

Was VERTEX COVER really just an example...?

What is so special in VERTEX COVER?

★ Nothing special! It is just a minor bidimensional parameter:

minor-closed $\quad + \quad$ **vc**($_k$) $= \Omega(k^2)$.

Where did we use planarity?

Was VERTEX COVER really just an example...?

What is so special in VERTEX COVER?

★ Nothing special! It is just a minor bidimensional parameter:

$$\text{minor-closed} \quad + \quad \mathbf{vc}(\boxplus_k) = \Omega(k^2).$$

Where did we use planarity?

★ Only the linear Grid Exclusion Theorem!

Arguments go through up to H-minor-free graphs.

Next subsection is...

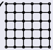
1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Minor Bidimensionality:

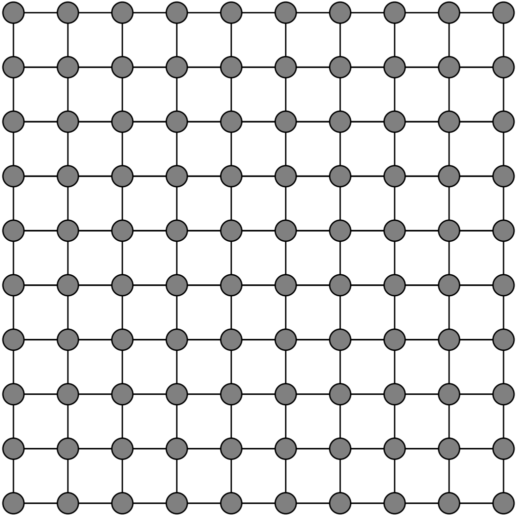
[Demaine, Fomin, Hajiaghayi, Thilikos. 2005]

Definition

A parameter **p** is *minor bidimensional* if

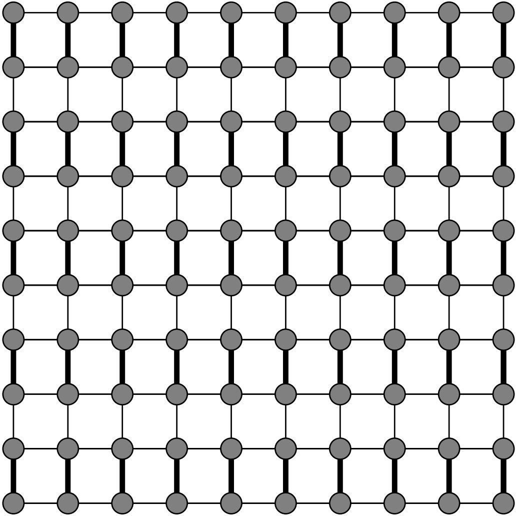
1. **p** is closed under taking of minors (minor-closed), and
2. **p** $\Bigg($  $_{k}\Bigg) = \Omega(k^2)$.

Vertex Cover of a Grid



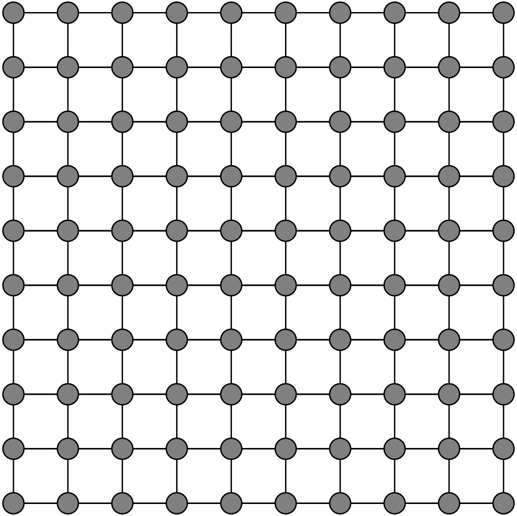
$H_{\ell,\ell}$ for $\ell = 10$

Vertex Cover of a Grid

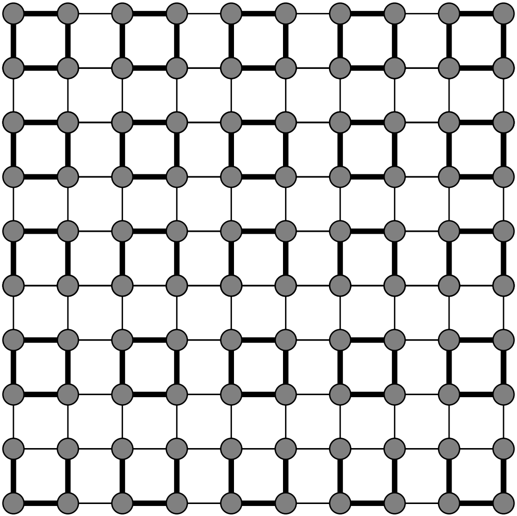


$\mathbf{vc}(H_{\ell,\ell}) \geq \ell^2/2$

Feedback Vertex Set of a Grid



Feedback Vertex Set of a Grid



$\mathbf{fvs}(H_{\ell,\ell}) \geq \ell^2/4$

How to obtain subexponential algorithms for BP?

- First we must restrict ourselves to special graph classes, like planar or H-minor-free graphs.

How to obtain subexponential algorithms for BP?

- First we must restrict ourselves to special graph classes, like planar or H-minor-free graphs.
  - Show that if the graph has large treewidth $(> c\sqrt{k})$ then it has a $(\sqrt{k} \times \sqrt{k})$-grid as a minor, and hence the answer to the problem is YES (or NO) immediately.

How to obtain subexponential algorithms for BP?

- First we must restrict ourselves to special graph classes, like planar or H-minor-free graphs.
  - Show that if the graph has large treewidth $(> c\sqrt{k})$ then it has a $(\sqrt{k} \times \sqrt{k})$-grid as a minor, and hence the answer to the problem is YES (or NO) immediately.
  - Otherwise, the treewidth is bounded by $c\sqrt{k}$, and hence we can use a dynamic programming (DP) algorithm on graphs of bounded treewidth.

How to obtain subexponential algorithms for BP?

- First we must restrict ourselves to special graph classes, like planar or H-minor-free graphs.
  - Show that if the graph has large treewidth $(> c\sqrt{k})$ then it has a $(\sqrt{k} \times \sqrt{k})$-grid as a minor, and hence the answer to the problem is YES (or NO) immediately.
  - Otherwise, the treewidth is bounded by $c\sqrt{k}$, and hence we can use a dynamic programming (DP) algorithm on graphs of bounded treewidth.
- If we have a DP algorithm for bounded treewidth running in time c^t or t^t, then it implies $2^{O(\sqrt{k})}$ or $2^{O(\sqrt{k}\log k)}$ algorithm.

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

1. Compute (or approximate) $\mathbf{tw}(G)$.
2. If $\mathbf{tw}(G) = \Omega(\sqrt{k})$, then answer NO.
3. Otherwise $\mathbf{tw}(G) = O(\sqrt{k})$, and we solve the problem by DP.

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

$\boxed{1}$ Compute (or approximate) $\mathbf{tw}(G)$.
We can use a fast FPT algorithm or a constant-factor approx.

$\boxed{2}$ If $\mathbf{tw}(G) = \Omega(\sqrt{k})$, then answer NO.

$\boxed{3}$ Otherwise $\mathbf{tw}(G) = O(\sqrt{k})$, and we solve the problem by DP.

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

1. Compute (or approximate) $\mathbf{tw}(G)$.
   We can use a fast FPT algorithm or a constant-factor approx.
2. If $\mathbf{tw}(G) = \Omega(\sqrt{k})$, then answer NO.
   This follows because of the linear Grid Exclusion Theorems.
3. Otherwise $\mathbf{tw}(G) = O(\sqrt{k})$, and we solve the problem by DP.

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

$\boxed{1}$ Compute (or approximate) $\mathbf{tw}(G)$.
We can use a fast FPT algorithm or a constant-factor approx.

$\boxed{2}$ If $\mathbf{tw}(G) = \Omega(\sqrt{k})$, then answer NO.
This follows because of the linear Grid Exclusion Theorems.

$\boxed{3}$ Otherwise $\mathbf{tw}(G) = O(\sqrt{k})$, and we solve the problem by DP.
Doing DP in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$ is a whole area of research:

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

1. Compute (or approximate) $\mathbf{tw}(G)$.
   We can use a fast FPT algorithm or a constant-factor approx.
2. If $\mathbf{tw}(G) = \Omega(\sqrt{k})$, then answer NO.
   This follows because of the linear Grid Exclusion Theorems.
3. Otherwise $\mathbf{tw}(G) = O(\sqrt{k})$, and we solve the problem by DP.
   Doing DP in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$ is a whole area of research:
   - Exploiting Catalan structures on sparse graphs. [Dorn et al. 2005-2008] [Rué, S., Thilikos. 2010]

Piecing everything together

Theorem

Let G be an H-minor-free graph, and let $\mathbf{p}$ be a minor bidimensional graph parameter computable in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$.
Then deciding "$\mathbf{p}(G) = k$" can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

1. Compute (or approximate) $\mathbf{tw}(G)$.
   We can use a fast FPT algorithm or a constant-factor approx.
2. If $\mathbf{tw}(G) = \Omega(\sqrt{k})$, then answer NO.
   This follows because of the linear Grid Exclusion Theorems.
3. Otherwise $\mathbf{tw}(G) = O(\sqrt{k})$, and we solve the problem by DP.
   Doing DP in time $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$ is a whole area of research:
   - Exploiting Catalan structures on sparse graphs. [Dorn et al. 2005-2008] [Rué, S., Thilikos. 2010]
   - Randomized algorithms using Cut&Count. [Cygan et al. 2011]
   - Deterministic algorithms based on matrix rank. [Boadlaender et al. 2012]
   - Deterministic algorithms based on matroids. [Fomin et al. 2013]

Minor Bidimensionality provides a meta-algorithm

- This result applies to all $\boxed{\text{minor-closed}}$ parameters:

  VERTEX COVER, FEEDBACK VERTEX SET, LONG PATH, CYCLE COVER, ...

Minor Bidimensionality provides a meta-algorithm

- This result applies to all $\boxed{\text{minor-closed}}$ parameters:

  Vertex Cover, Feedback Vertex Set, Long Path, Cycle Cover, ...

- What about $\boxed{\text{contraction-closed}}$ parameters??

  Dominating Set, Connected Vertex Cover, r-Dominating Set, ...

skip

Extensions: contraction bidimensionality

- Dominating Set is NOT minor-closed,
  so we cannot use Grid Exclusion Theorems!!

Extensions: contraction bidimensionality

- Dominating Set is NOT minor-closed,
  so we cannot use Grid Exclusion Theorems!!
- But it is contraction-closed...

- DOMINATING SET is NOT minor-closed, so we cannot use Grid Exclusion Theorems!!
- But it is contraction-closed...

Contraction Bidimensionality:

[Demaine, Fomin, Hajiaghayi, Thilikos. 2005]

Definition

A parameter **p** is *contraction bidimensional* if

1. **p** is closed under taking of contractions (contraction-closed), and
2. for a "$(k \times k)$-grid-like graph" Γ, $\mathbf{p}(\Gamma) = \Omega(k^2)$.

- Dominating Set is NOT minor-closed,
  so we cannot use Grid Exclusion Theorems!!
- But it is contraction-closed...

Contraction Bidimensionality:

[Demaine, Fomin, Hajiaghayi, Thilikos. 2005]

Definition

A parameter **p** is *contraction bidimensional* if

1. **p** is closed under taking of contractions (contraction-closed), and
2. for a "$(k \times k)$-grid-like graph" Γ, $\mathbf{p}(\Gamma) = \Omega(k^2)$.

What is a $(k \times k)$-grid-like graph...?

Contraction bidimensionality: old setting

A "$(k \times k)$-grid-like graph" was different for each graph class:

Contraction bidimensionality: old setting

A "$(k \times k)$-grid-like graph" was different for each graph class:

★ For planar graphs this is a partially triangulated $(k \times k)$-grid.

[Demaine, Fomin, Hajiaghayi, Thilikos. 2006]

Contraction bidimensionality: old setting

A "$(k \times k)$-grid-like graph" was different for each graph class:

★ For planar graphs this is a partially triangulated $(k \times k)$-grid.

[Demaine, Fomin, Hajiaghayi, Thilikos. 2006]

★ For graphs of Euler genus γ, this is a partially triangulated $(k \times k)$-grid with up to γ additional handles.

[Demaine, Hajiaghayi, Thilikos. 2006]

Contraction bidimensionality: old setting

A "$(k \times k)$-grid-like graph" was different for each graph class:

★ For planar graphs this is a partially triangulated $(k \times k)$-grid.

[Demaine, Fomin, Hajiaghayi, Thilikos. 2006]

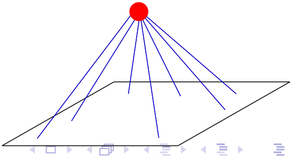
★ For graphs of Euler genus γ, this is a partially triangulated $(k \times k)$-grid with up to γ additional handles.

[Demaine, Hajiaghayi, Thilikos. 2006]

★ For apex-minor-free graphs, this is a $(k \times k)$-augmented grid, i.e., partially triangulated grid augmented with additional edges such that each vertex is incident to $O(1)$ edges to non-boundary vertices of the grid.

[Demaine, Fomin, Hajiaghayi, Thilikos. 2005]

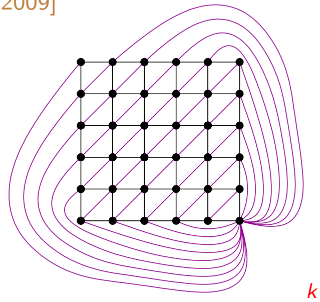
H is an *apex graph* if $\exists v \in V(H)$: $H - v$ is planar



Contraction bidimensionality: new definition

Finally, the right "($k \times k$)-grid-like graph" was found:

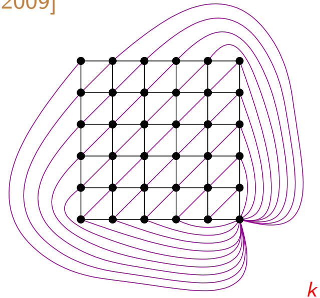
[Fomin, Golovach, Thilikos. 2009]



Contraction bidimensionality: new definition

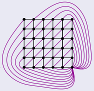
Finally, the right "$(k \times k)$-grid-like graph" was found:

[Fomin, Golovach, Thilikos. 2009]



Definition

A parameter **p** is *contraction bidimensional* if the following hold:

1. **p** is contraction-closed, and
2. **p**( $_k$) $= \Omega(k^2)$.

Meta-algorithms for contraction bidimensional parameters

Theorem

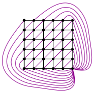
Let H be a fixed $\boxed{\text{apex}}$ graph, let G be an H-minor free graph, and let $\mathbf{p}$ be a contraction bidimensional parameter computable in $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$. Then deciding $\mathbf{p}(G) = k$ can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

Meta-algorithms for contraction bidimensional parameters

Theorem

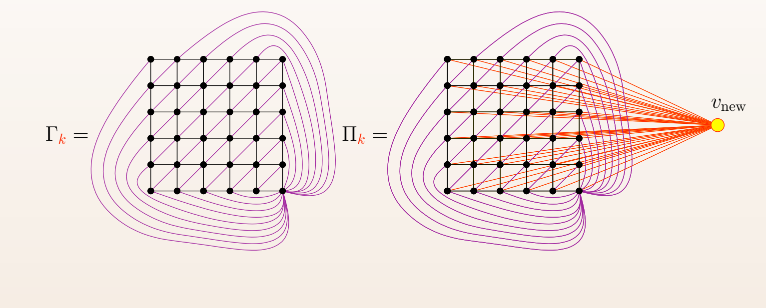
Let H be a fixed $\boxed{\text{apex}}$ graph, let G be an H-minor free graph, and let $\mathbf{p}$ be a contraction bidimensional parameter computable in $2^{O(\mathbf{tw}(G))} \cdot n^{O(1)}$. Then deciding $\mathbf{p}(G) = k$ can be done in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

As for minor bidimensionality, we need to prove that

- If $\mathbf{tw}(G) = \Omega(k)$ then G contains  $_k$ as a contraction.

Two important grid-like graphs

Two pattern graphs Γ_k and Π_k:



$\Pi_k = \Gamma_k +$ a new universal vertex v_{new}.

The "contraction-certificates" for large treewidth

Theorem (Fomin, Golovach, Thilikos. 2009)

For any integer $\ell > 0$, there is c_ℓ such that every connected graph of treewidth at least c_ℓ contains K_ℓ, Γ_ℓ, or Π_ℓ as a contraction.

The "contraction-certificates" for large treewidth

Theorem (Fomin, Golovach, Thilikos. 2009)

For any integer $\ell > 0$, there is c_ℓ such that every connected graph of treewidth at least c_ℓ contains K_ℓ, Γ_ℓ, or Π_ℓ as a contraction.

Theorem (Fomin, Golovach, Thilikos. 2009)

For every graph H, there is $c_H > 0$ such that every connected H-minor-free graph of treewidth at least $c_H \cdot \ell^2$ contains Γ_ℓ or Π_ℓ as a contraction.

The "contraction-certificates" for large treewidth

Theorem (Fomin, Golovach, Thilikos. 2009)

For any integer $\ell > 0$, there is c_ℓ such that every connected graph of treewidth at least c_ℓ contains K_ℓ, Γ_ℓ, or Π_ℓ as a contraction.

Theorem (Fomin, Golovach, Thilikos. 2009)

For every graph H, there is $c_H > 0$ such that every connected H-minor-free graph of treewidth at least $c_H \cdot \ell^2$ contains Γ_ℓ or Π_ℓ as a contraction.

Theorem (Fomin, Golovach, Thilikos. 2009)

For every apex graph H, there is $c_H > 0$ such that every connected H-minor-free graph of treewidth at least $c_H \cdot \ell$ contains Γ_ℓ as a contraction.

Further applications of Bidimensionality

1. Bidimensionality + DP $\Rightarrow$ $\boxed{\text{Subexponential FPT algorithms}}$

   [Demaine, Fomin, Hajiaghayi, Thilikos. 2004-2005]
   [Fomin, Golovach, Thilikos. 2009]

Further applications of Bidimensionality

1. Bidimensionality + DP ⇒ $\boxed{\text{Subexponential FPT algorithms}}$

   [Demaine, Fomin, Hajiaghayi, Thilikos. 2004-2005]
   [Fomin, Golovach, Thilikos. 2009]

2. Bidimensionality + separation properties ⇒ $\boxed{\text{(E)PTAS}}$

   [Demaine and Hajiaghayi. 2005]
   [Fomin, Lokshtanov, Raman, Saurabh. 2011]

Further applications of Bidimensionality

1. Bidimensionality + DP $\Rightarrow$ $\boxed{\text{Subexponential FPT algorithms}}$

   [Demaine, Fomin, Hajiaghayi, Thilikos. 2004-2005]
   [Fomin, Golovach, Thilikos. 2009]

2. Bidimensionality + separation properties $\Rightarrow$ $\boxed{\text{(E)PTAS}}$

   [Demaine and Hajiaghayi. 2005]
   [Fomin, Lokshtanov, Raman, Saurabh. 2011]

3. Bidimensionality + separation properties $\Rightarrow$ $\boxed{\text{Kernelization}}$

   [Fomin, Lokshtanov, Saurabh, Thilikos. 2009-2010]

Further applications of Bidimensionality

1. Bidimensionality + DP $\Rightarrow$ $\boxed{\text{Subexponential FPT algorithms}}$

   [Demaine, Fomin, Hajiaghayi, Thilikos. 2004-2005]
   [Fomin, Golovach, Thilikos. 2009]

2. Bidimensionality + separation properties $\Rightarrow$ $\boxed{\text{(E)PTAS}}$

   [Demaine and Hajiaghayi. 2005]
   [Fomin, Lokshtanov, Raman, Saurabh. 2011]

3. Bidimensionality + separation properties $\Rightarrow$ $\boxed{\text{Kernelization}}$

   [Fomin, Lokshtanov, Saurabh, Thilikos. 2009-2010]

4. Bidimensionality + new Grid Theorems $\Rightarrow$ $\boxed{\text{Geometric graphs}}$

   [Fomin, Lokshtanov, Saurabh. 2012]
   [Grigoriev, Koutsonas, Thilikos. 2013]

Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Basic principle of the irrelevant vertex technique

This technique was invented in [Robertson and Seymour. 1995]

Basic principle of the irrelevant vertex technique

This technique was invented in [Robertson and Seymour. 1995]

Disjoint Paths

Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.

Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Basic principle of the irrelevant vertex technique

This technique was invented in [Robertson and Seymour. 1995]

DISJOINT PATHS
Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Strategy:

1. If $\text{tw}(G) > f(k)$, find an irrelevant vertex:

   A vertex $v \in V(G)$ such that (G, T, k) and $(G \setminus v, T, k)$ are equivalent instances.

Basic principle of the irrelevant vertex technique

This technique was invented in [Robertson and Seymour. 1995]

DISJOINT PATHS
Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

Strategy:

1. If $\mathrm{tw}(G) > f(k)$, find an irrelevant vertex:

   A vertex $v \in V(G)$ such that (G, T, k) and $(G \setminus v, T, k)$ are equivalent instances.

2. Otherwise, if $\mathrm{tw}(G) \leq f(k)$, solve the problem using dynamic programming (by Courcelle).

How to find an irrelevant vertex when the treewidth is large?

How to find an irrelevant vertex when the treewidth is large?

By using the Grid Exclusion Theorem!

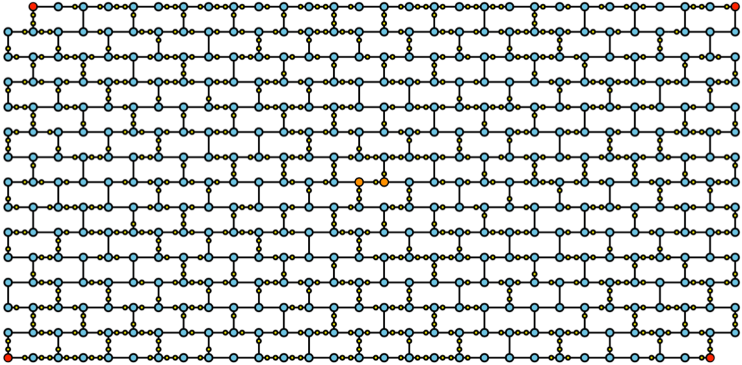
How to find an irrelevant vertex when the treewidth is large?

By using the $\boxed{\text{Wall}}$ Exclusion Theorem!

How to find an irrelevant vertex when the treewidth is large?

Theorem (Robertson and Seymour. 1986)

For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains an ℓ-wall as a minor.

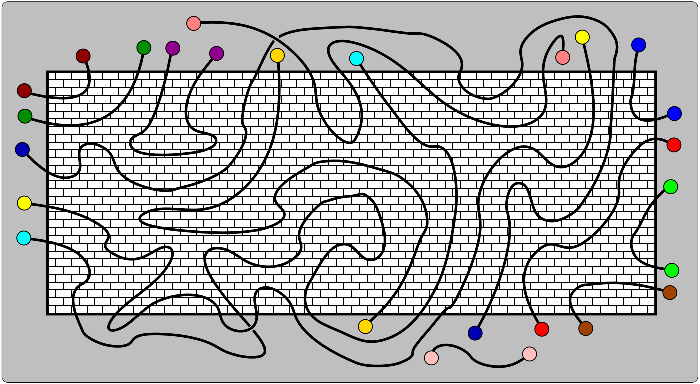


[Figure by Dimitrios M. Thilikos]

How to find an irrelevant vertex when the treewidth is large?

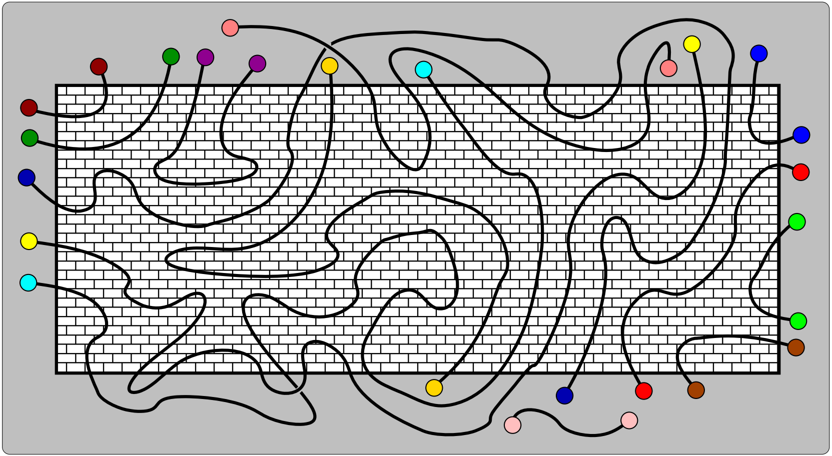
Theorem (Robertson and Seymour. 1986)

For every integer $\ell > 0$, there is an integer $c(\ell)$ such that every graph of treewidth $\geq c(\ell)$ contains an ℓ-wall as a minor.

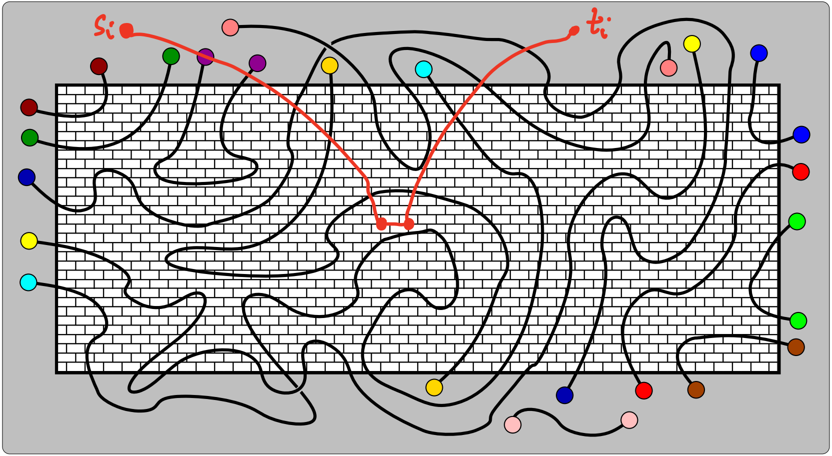


[Figure by Dimitrios M. Thilikos]

Goal: declare one of the central vertices of the wall irrelevant.



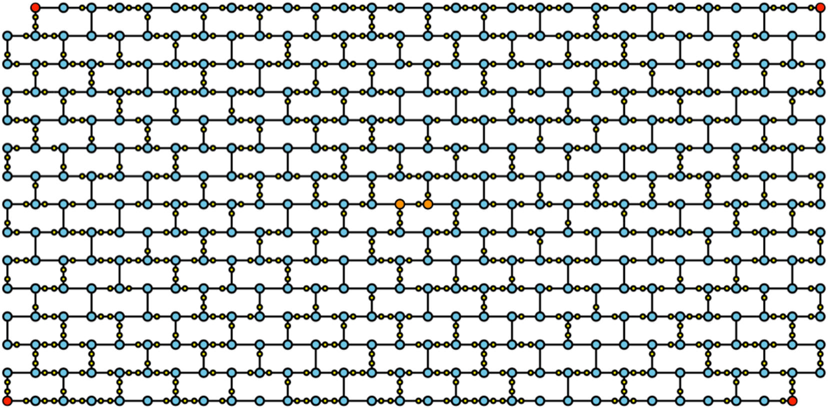
Goal: declare one of the central vertices of the wall irrelevant.



This is only possible if the wall is insulated from the exterior!

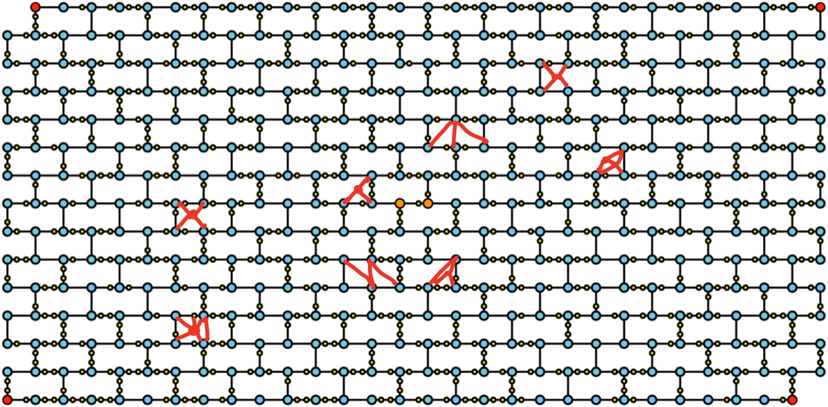
Flat walls

Goal: enrich the notion of wall so that we can insulate it from the exterior.



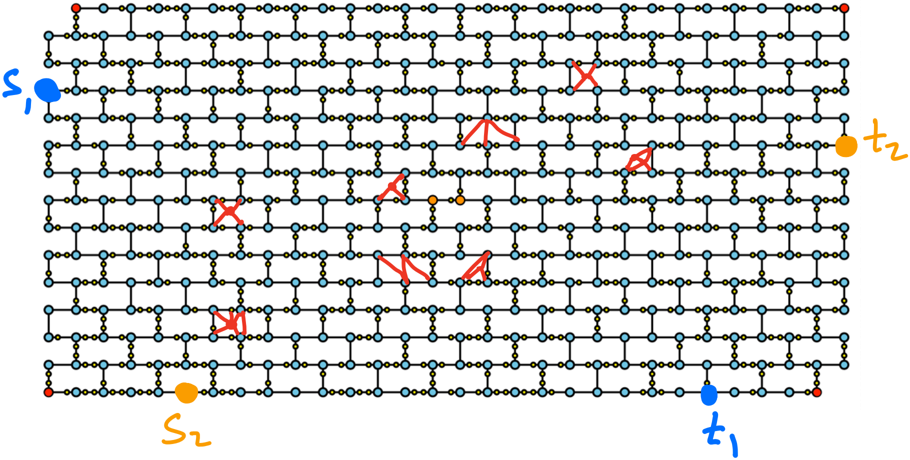
Flat walls

We need to allow some extra edges in the interior of the wall.



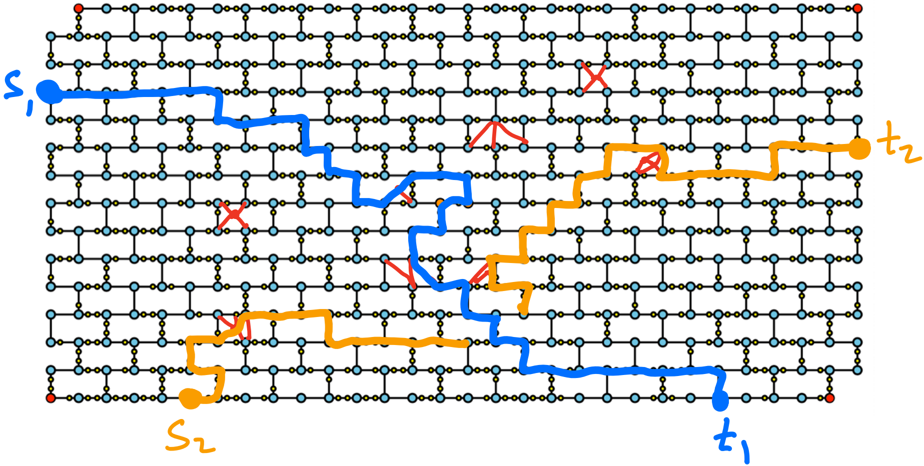
Flat walls

We impose a topological property that defines the "flatness" of the wall.

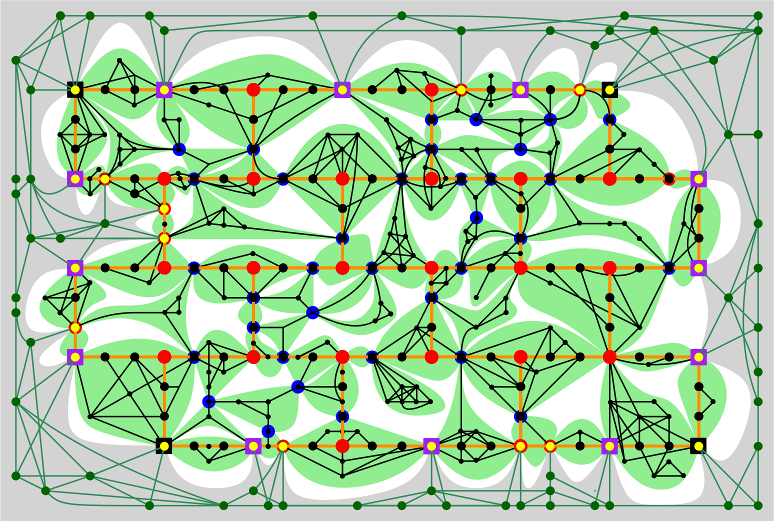


Flat walls

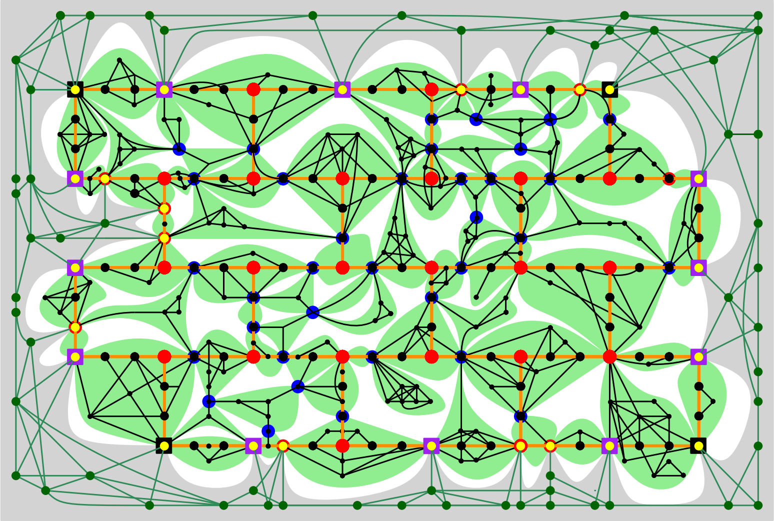
There are no crossing paths $s_1 - t_1$ and $s_2 - t_2$ from/to the perimeter.



A real flat wall can be quite wild... [Figure by Dimitrios M. Thilikos]

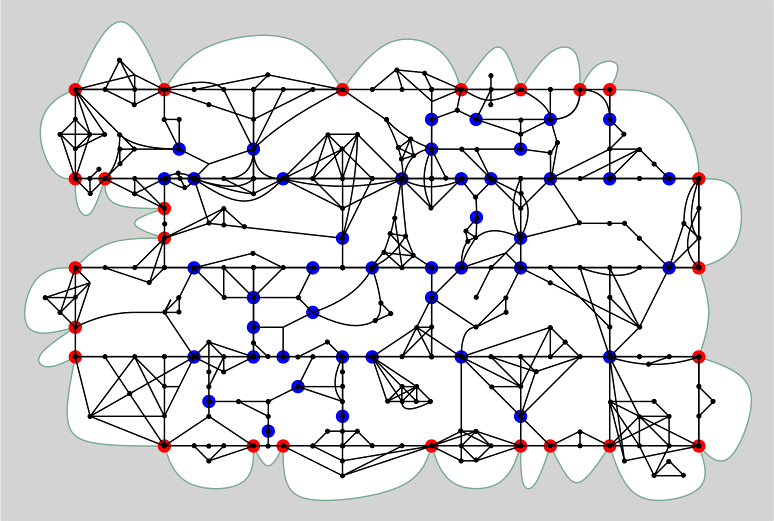


Flat walls: a bit more formal



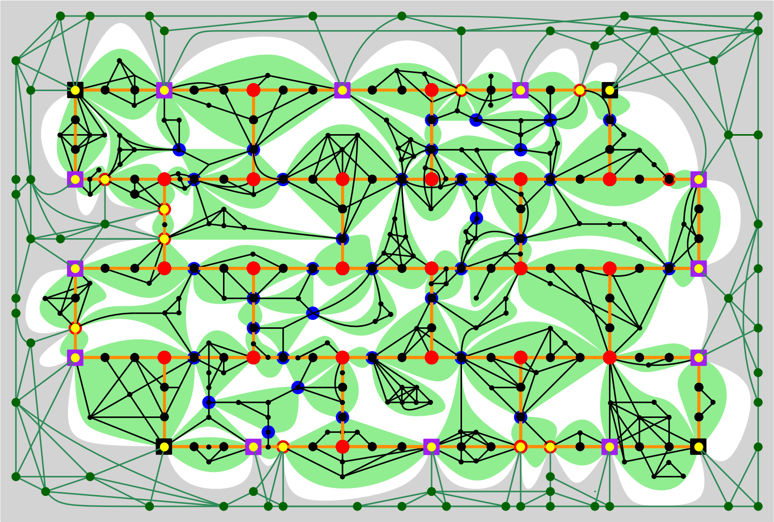
[Figures by Dimitrios M. Thilikos]

Flat walls: a bit more formal



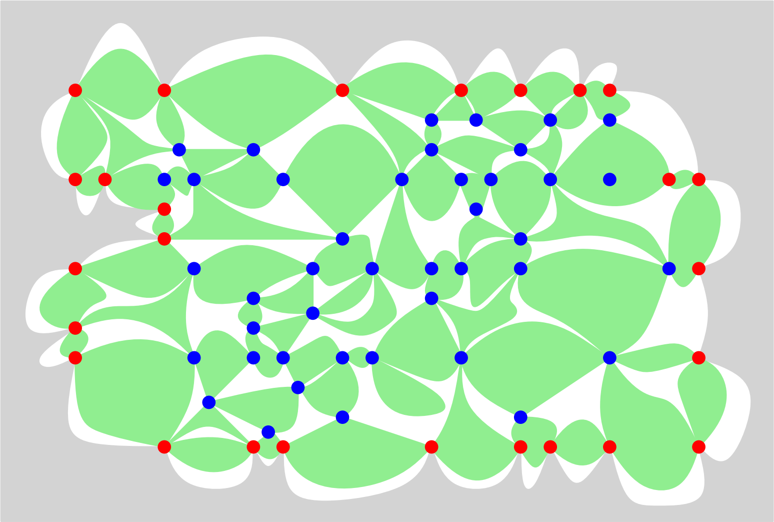
[Figures by Dimitrios M. Thilikos]

Flat walls: a bit more formal



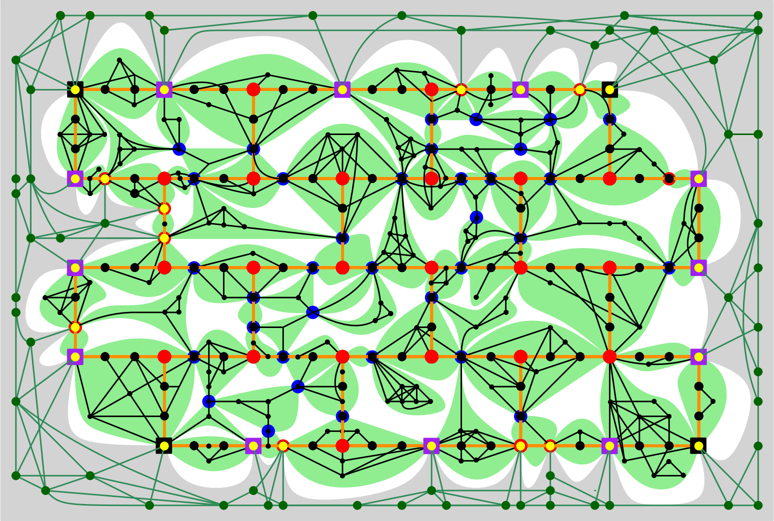
[Figures by Dimitrios M. Thilikos]

Flat walls: a bit more formal



[Figures by Dimitrios M. Thilikos]

Flat walls: a bit more formal



[Figures by Dimitrios M. Thilikos]

The Weak Structure Graph Minors Theorem

Theorem (Robertson and Seymour. 1995)

There exist recursive functions $f_1 : \mathbb{N}^2 \to \mathbb{N}$ *and* $f_2 : \mathbb{N} \to \mathbb{N}$, *such that for every graph* G *and every* $q, r \in \mathbb{N}$, *one of the following holds:*

The Weak Structure Graph Minors Theorem

Theorem (Robertson and Seymour. 1995)

There exist recursive functions $f_1 : \mathbb{N}^2 \to \mathbb{N}$ *and* $f_2 : \mathbb{N} \to \mathbb{N}$*, such that for every graph* G *and every* $q, r \in \mathbb{N}$*, one of the following holds:*

1. K_q *is a minor of* G.

The Weak Structure Graph Minors Theorem

Theorem (Robertson and Seymour. 1995)

There exist recursive functions $f_1 : \mathbb{N}^2 \to \mathbb{N}$ and $f_2 : \mathbb{N} \to \mathbb{N}$, such that for every graph G and every $q, r \in \mathbb{N}$, one of the following holds:

1. *K_q is a minor of G.*
2. *The treewidth of G is at most $f_1(q, r)$.*

The Weak Structure Graph Minors Theorem

Theorem (Robertson and Seymour. 1995)

There exist recursive functions $f_1 : \mathbb{N}^2 \to \mathbb{N}$ and $f_2 : \mathbb{N} \to \mathbb{N}$, such that for every graph G and every $q, r \in \mathbb{N}$, one of the following holds:

1. K_q *is a minor of* G.
2. *The treewidth of* G *is at most* $f_1(q, r)$.
3. *There exists* $A \subseteq V(G)$ *(apices) with* $|A| \leq f_2(q)$ *such that* $G \setminus A$ *contains as a subgraph a flat wall* W *of height* r.

The Weak Structure Graph Minors Theorem

Theorem (Robertson and Seymour. 1995)

There exist recursive functions $f_1 : \mathbb{N}^2 \to \mathbb{N}$ and $f_2 : \mathbb{N} \to \mathbb{N}$, such that for every graph G and every $q, r \in \mathbb{N}$, one of the following holds:

1. K_q *is a minor of* G.
2. *The treewidth of* G *is at most* $f_1(q, r)$.
3. *There exists* $A \subseteq V(G)$ *(apices) with* $|A| \leq f_2(q)$ *such that* $G \setminus A$ *contains as a subgraph a flat wall* W *of height* r.

There are many different variants and optimizations of this theorem...

[Chuzhoy. 2015]

[Kawarabayashi, Thomas, Wollan. 2018]

[S., Stamoulis, Thilikos. 2021]

The Weak Structure Graph Minors Theorem

Theorem (Robertson and Seymour. 1995)

There exist recursive functions $f_1 : \mathbb{N}^2 \to \mathbb{N}$ and $f_2 : \mathbb{N} \to \mathbb{N}$, such that for every graph G and every $q, r \in \mathbb{N}$, one of the following holds:

1. *K_q is a minor of G.*
2. *The treewidth of G is at most $f_1(q, r)$.*
3. *There exists $A \subseteq V(G)$ (apices) with $|A| \leq f_2(q)$ such that $G \setminus A$ contains as a subgraph a flat wall W of height r.*

There are many different variants and optimizations of this theorem...

[Chuzhoy. 2015]

[Kawarabayashi, Thomas, Wollan. 2018]

[S., Stamoulis, Thilikos. 2021]

Important: possible to find one of the outputs in time $f(q, r) \cdot |V(G)|$.

Back to the DISJOINT PATHS problem

DISJOINT PATHS
Input: a graph G and k pairs of vertices $T = \{s_1, \dots, s_k, t_1, \dots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \dots, P_k$ such that P_i connects s_i to t_i?

Back to the Disjoint Paths problem

Disjoint Paths
Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

By the Weak Structure Theorem:

- If $\mathrm{tw}(G) \leq f(k)$: solve using dynamic programming.

Back to the Disjoint Paths problem

Disjoint Paths
Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

By the Weak Structure Theorem:

- If $\mathrm{tw}(G) \leq f(k)$: solve using dynamic programming.
- If G contains a $K_{g(k)}$-minor: "easy" to find an irrelevant vertex.

Back to the DISJOINT PATHS problem

DISJOINT PATHS
Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

By the Weak Structure Theorem:

- If $\mathrm{tw}(G) \leq f(k)$: solve using dynamic programming.
- If G contains a $K_{g(k)}$-minor: "easy" to find an irrelevant vertex.
- If G contains a "small" apex set A and a flat wall W in $G \setminus A$ of size at least $h(k)$: declare the central vertex of the flat wall irrelevant.

Back to the DISJOINT PATHS problem

DISJOINT PATHS
Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

By the Weak Structure Theorem:

- If $\mathrm{tw}(G) \leq f(k)$: solve using dynamic programming.
- If G contains a $K_{g(k)}$-minor: "easy" to find an irrelevant vertex.
- If G contains a "small" apex set A and a flat wall W in $G \setminus A$ of size at least $h(k)$: declare the central vertex of the flat wall irrelevant.

The irrelevant vertex technique has been applied to many problems...

Back to the DISJOINT PATHS problem

DISJOINT PATHS

Input: a graph G and k pairs of vertices $T = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$.

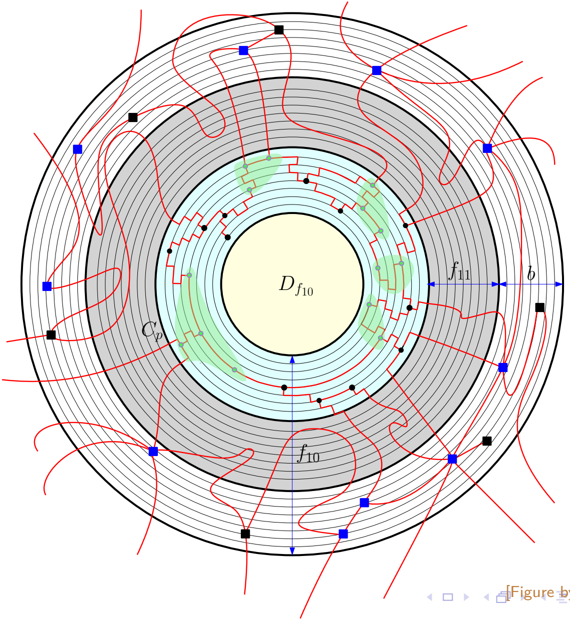
Question: does G contain k vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i connects s_i to t_i?

By the Weak Structure Theorem:

- If $\mathrm{tw}(G) \leq f(k)$: solve using dynamic programming.
- If G contains a $K_{g(k)}$-minor: "easy" to find an irrelevant vertex.
- If G contains a "small" apex set A and a flat wall W in $G \setminus A$ of size at least $h(k)$: declare the central vertex of the flat wall irrelevant.

The irrelevant vertex technique has been applied to many problems... usually with a lot of technical pain.

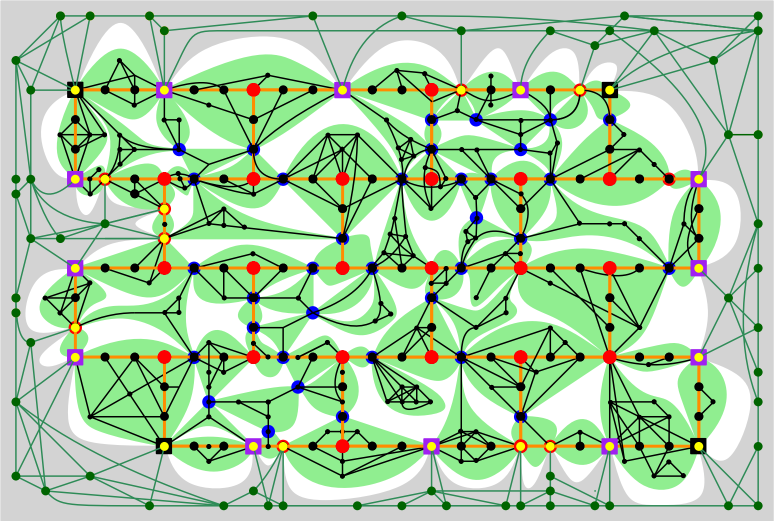
Rerouting inside a big flat wall...



[Figure by Dimitrios M. Thilikos]

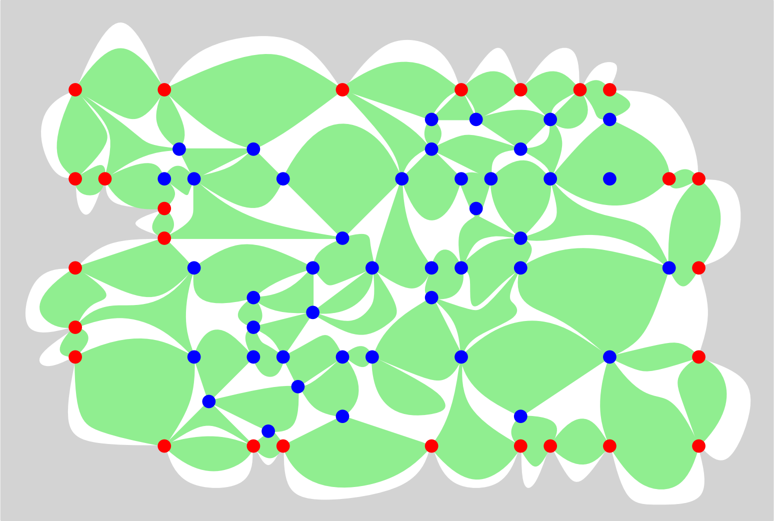
Crucial notion: homogeneity

In order to declare a vertex irrelevant for some problem, usually we need to consider a homogenous flat wall, which we proceed to define.



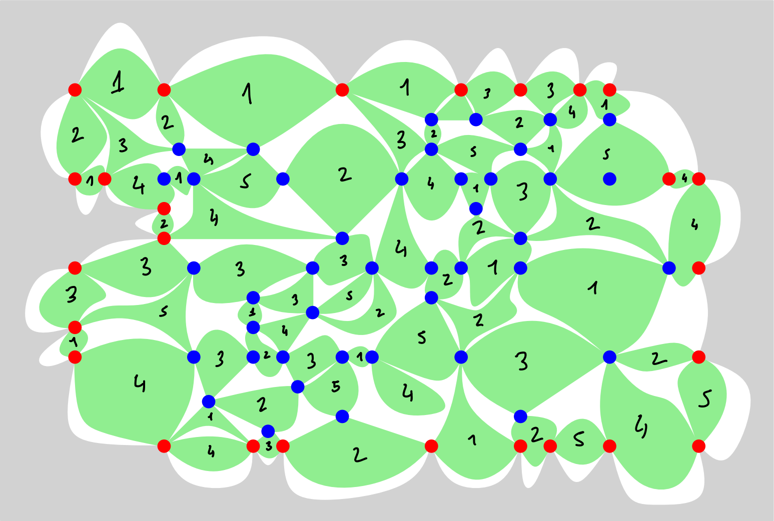
Crucial notion: homogeneity

We consider a flap-coloring encoding the relevant information of our favorite problem inside each flap (similar to tables of DP).



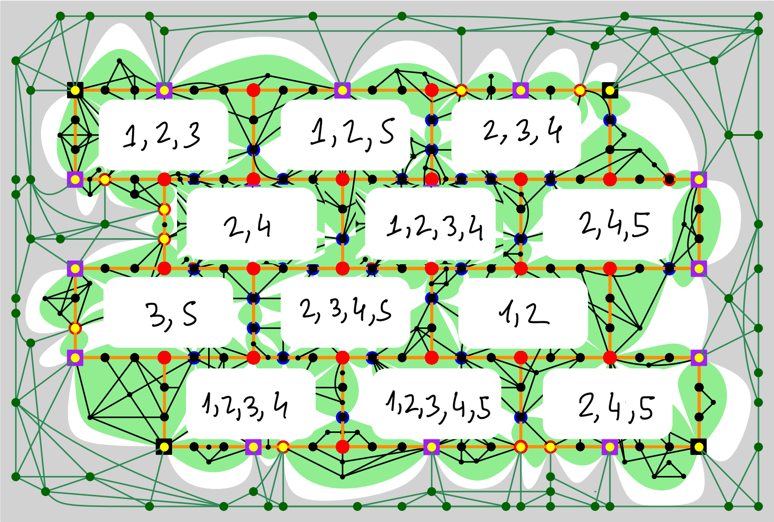
Crucial notion: homogeneity

We consider a flap-coloring encoding the relevant information of our favorite problem inside each flap (similar to tables of DP).



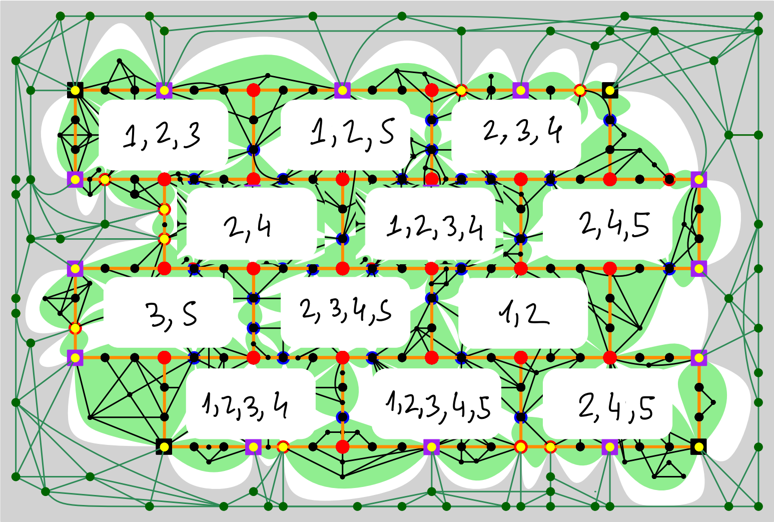
Crucial notion: homogeneity

For every brick of the wall, we define its palette as the colors appearing in the flaps it contains.



Crucial notion: homogeneity

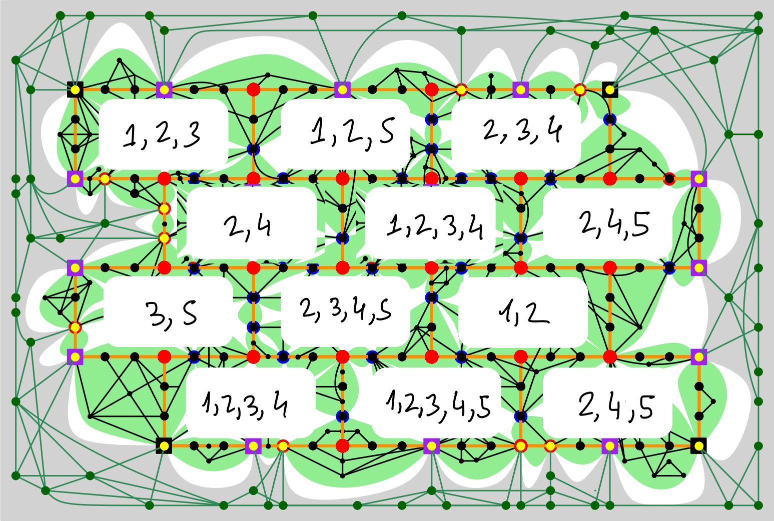
A flat wall is homogenous if every (internal) brick has the same palette.
Fact: every brick of a homogenous flat wall has the same "behavior".



Crucial notion: homogeneity

Price of homogeneity to obtain a homogenous flat r-wall (zooming):

If we have c colors, we need to start with a flat r^c-wall. (why?)



Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Hitting forbidden minors

- If $\mathcal{C} = \{\text{edgeless graphs}\}$, then $\mathcal{F} = \{K_2\}$.
- If $\mathcal{C} = \{\text{forests}\}$, then $\mathcal{F} = \{K_3\}$.
- If $\mathcal{C} = \{\text{outerplanar graphs}\}$, then $\mathcal{F} = \{K_4, K_{2,3}\}$.
- If $\mathcal{C} = \{\text{planar graphs}\}$, then $\mathcal{F} = \{K_5, K_{3,3}\}$.

Hitting forbidden minors

- If $\mathcal{C}$ = {edgeless graphs}, then $\mathcal{F} = \{K_2\}$.
- If $\mathcal{C}$ = {forests}, then $\mathcal{F} = \{K_3\}$.
- If $\mathcal{C}$ = {outerplanar graphs}, then $\mathcal{F} = \{K_4, K_{2,3}\}$.
- If $\mathcal{C}$ = {planar graphs}, then $\mathcal{F} = \{K_5, K_{3,3}\}$.

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

Hitting forbidden minors

- If $\mathcal{C}$ = {edgeless graphs}, then $\mathcal{F} = \{K_2\}$.
- If $\mathcal{C}$ = {forests}, then $\mathcal{F} = \{K_3\}$.
- If $\mathcal{C}$ = {outerplanar graphs}, then $\mathcal{F} = \{K_4, K_{2,3}\}$.
- If $\mathcal{C}$ = {planar graphs}, then $\mathcal{F} = \{K_5, K_{3,3}\}$.

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

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

Hitting forbidden minors

- If $\mathcal{C}$ = {edgeless graphs}, then $\mathcal{F} = \{K_2\}$.
- If $\mathcal{C}$ = {forests}, then $\mathcal{F} = \{K_3\}$.
- If $\mathcal{C}$ = {outerplanar graphs}, then $\mathcal{F} = \{K_4, K_{2,3}\}$.
- If $\mathcal{C}$ = {planar graphs}, then $\mathcal{F} = \{K_5, K_{3,3}\}$.

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

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

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.
- $\mathcal{F} = \{K_3\}$: FEEDBACK VERTEX SET.
- $\mathcal{F} = \{K_5, K_{3,3}\}$: VERTEX PLANARIZATION.
- $\mathcal{F}$ = {diamond}: CACTUS VERTEX DELETION.

Hitting forbidden minors

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

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

Hitting forbidden minors

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

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

NP-hard if $\mathcal{F}$ contains a graph with some edge. [Lewis, Yannakakis. 1980]

Hitting forbidden minors

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

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

NP-hard if $\mathcal{F}$ contains a graph with some edge. [Lewis, Yannakakis. 1980]

We consider the following two parameterizations of $\mathcal{F}$-M-Deletion:

1. Structural parameter: $\mathrm{tw}(G)$.
2. Solution size: k.

Joint work with Dimitrios M. Thilikos, Julien Baste, Giannos Stamoulis, and Laure Morelle.

Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Theorem (Courcelle. 1990)

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

Theorem (Courcelle. 1990)

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

It is not difficult to see that can $\mathcal{F}$-M-Deletion be expressed in MSOL:

$\mathcal{F}$-M-Deletion is FPT parameterized by tw...

Theorem (Courcelle. 1990)

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

It is not difficult to see that can $\mathcal{F}$-M-Deletion be expressed in MSOL:

$\mathcal{F}$-M-Deletion is FPT parameterized by tw...

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

Theorem (Courcelle. 1990)

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

It is not difficult to see that can $\mathcal{F}$-M-DELETION be expressed in MSOL:

$\mathcal{F}$-M-DELETION is FPT parameterized by tw...

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

Theorem (Courcelle. 1990)

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

It is not difficult to see that can $\mathcal{F}$-M-Deletion be expressed in MSOL:

$\mathcal{F}$-M-Deletion is FPT parameterized by tw...

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

$\boxed{\text{Goal}}$ For every $\mathcal{F}$, find the smallest possible function $f_{\mathcal{F}}(\mathrm{tw})$.

Theorem (Courcelle. 1990)

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

It is not difficult to see that can $\mathcal{F}$-M-Deletion be expressed in MSOL:

$\mathcal{F}$-M-Deletion is FPT parameterized by tw...

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

$\boxed{\text{Goal}}$ For every $\mathcal{F}$, find the $\boxed{\text{smallest possible}}$ function $f_{\mathcal{F}}(\mathrm{tw})$.

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

[Impagliazzo, Paturi. 1999]

What was known for particular collections $\mathcal{F}$

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 $\lvert S \rvert \leqslant k$ such that $G \setminus S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

What was known for particular collections $\mathcal{F}$

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

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

What was known for particular collections $\mathcal{F}$

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 $\lvert S \rvert \leqslant k$ such that $G \setminus S$ does not contain any of the graphs in $\mathcal{F}$ as a minor?

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

What was known for particular collections $\mathcal{F}$

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

- $\mathcal{F} = \{K_2\}$: VERTEX COVER.
  Easily solvable in time $2^{\Theta(\text{tw})} \cdot n^{\mathcal{O}(1)}$.
- $\mathcal{F} = \{K_3\}$: FEEDBACK VERTEX SET.

What was known for particular collections $\mathcal{F}$

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 $\lvert S \rvert \leqslant k$ such that $G \setminus S$ 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**: Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]

What was known for particular collections $\mathcal{F}$

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 $\|S\| \leqslant k$ such that $G \setminus S$ 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**: Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]
- $\mathcal{F} = \{K_5, K_{3,3}\}$: Vertex Planarization.

What was known for particular collections $\mathcal{F}$

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 $|S| \leqslant k$ such that $G \setminus S$ 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**: Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk. 2011]
- $\mathcal{F} = \{K_5, K_{3,3}\}$: Vertex Planarization.
  Solvable in time $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$.
  [Jansen, Lokshtanov, Saurabh. 2014 + Pilipczuk. 2015]

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 on n-vertex graphs can be solved in time

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

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 on n-vertex graphs can be solved in time

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

- 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.

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 on n-vertex graphs can be solved in time

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

- 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.

[Baste, S., Thilikos. **Hitting minors on bounded treewidth graphs. I. General upper bounds**. 2020]

[Baste, S., Thilikos. **Hitting minors on bounded treewidth graphs. II. Single-exponential algorithms**. 2020]

[Baste, S., Thilikos. **Hitting minors on bounded treewidth graphs. III. Lower bounds**. 2020]

[Baste, S., Thilikos. **Hitting minors on bounded treewidth graphs. IV. An optimal algorithm**. 2021]

Summary of our results

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

Summary of our results

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

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

Summary of our results

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

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

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

- For every $\mathcal{F}$: $\mathcal{F}$-M-Deletion in time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- For every ~~planar~~[1] $\mathcal{F}$: $\mathcal{F}$-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 every $\mathcal{F}$: $\mathcal{F}$-M-Deletion not solvable 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

- For every $\mathcal{F}$: $\mathcal{F}$-M-Deletion in time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- For every ~~planar~~[1] $\mathcal{F}$: $\mathcal{F}$-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 every $\mathcal{F}$: $\mathcal{F}$-M-Deletion not solvable 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

- For every $\mathcal{F}$: $\mathcal{F}$-M-Deletion in time $2^{2^{\mathcal{O}(\mathrm{tw}\cdot\log \mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$.
- For every ~~planar~~[1] $\mathcal{F}$: $\mathcal{F}$-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 every $\mathcal{F}$: $\mathcal{F}$-M-Deletion not solvable 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.

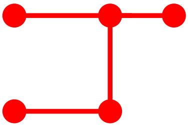
A dichotomy for hitting a connected minor

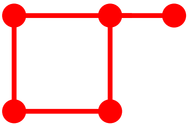
A dichotomy for hitting a connected minor

Theorem (Baste, S., Thilikos. 2016-2020)

Let H be a connected graph.

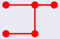
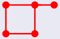
A dichotomy for hitting a connected minor



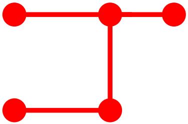


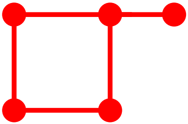
Theorem (Baste, S., Thilikos. 2016-2020)

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

- $2^{\mathcal{O}(\text{tw})} \cdot n^{\mathcal{O}(1)}$, *if* $H \leqslant_{\mathsf{c}}$  *or* $H \leqslant_{\mathsf{c}}$  .

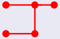
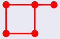
A dichotomy for hitting a connected minor



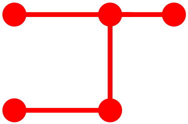


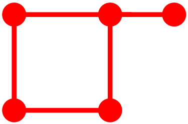
Theorem (Baste, S., Thilikos. 2016-2020)

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

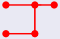
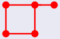
A dichotomy for hitting a connected minor





Theorem (Baste, S., Thilikos. 2016-2020)

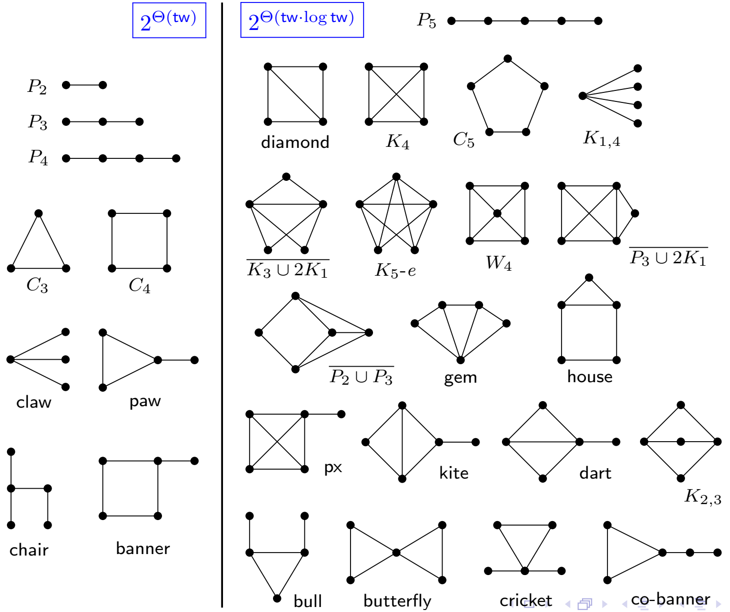
Let H be a connected graph.
The $\{H\}$-M-DELETION problem is solvable in time

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

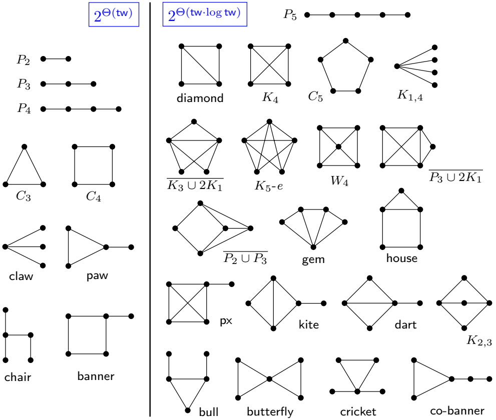
In both cases, the running time is asymptotically optimal under the ETH.

skip

Complexity of hitting a single connected minor H

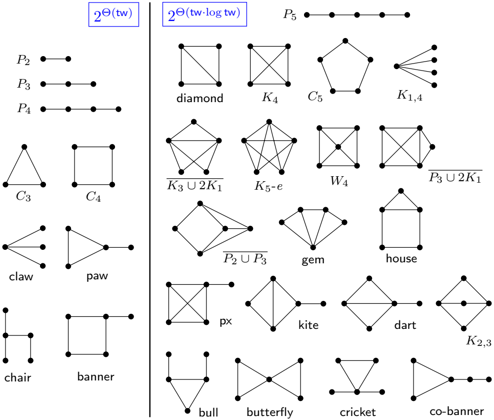


A compact statement for a single connected graph

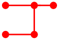
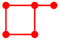


All these cases can be succinctly described as follows:

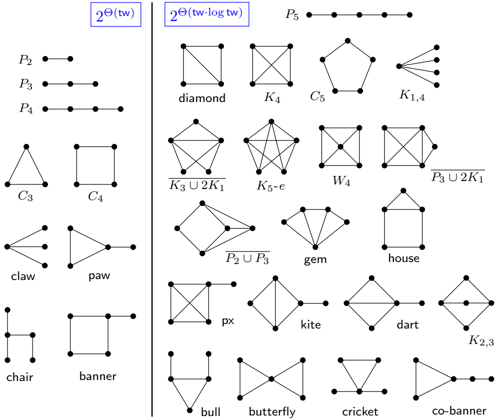
A compact statement for a single connected graph



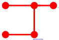
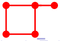
All these cases can be succinctly described as follows:

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

3. Lower bounds under the ETH
   - $2^{o(\text{tw})}$ is "easy".
   - $2^{o(\text{tw} \cdot \log \text{tw})}$ is much more involved and we get ideas from:

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

skip

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]

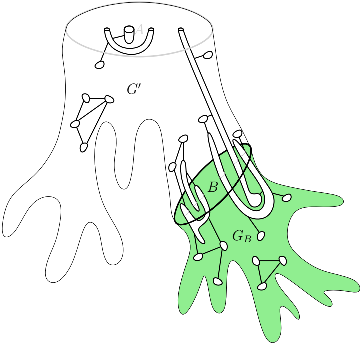
3. Lower bounds under the ETH
   - $2^{o(\mathrm{tw})}$ is "easy".
   - $2^{o(\mathrm{tw} \cdot \log \mathrm{tw})}$ is much more involved and we get ideas from:

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

skip

Algorithm in time $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw} \cdot \log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ for any collection $\mathcal{F}$

[Fig. by Valentin Garnero]

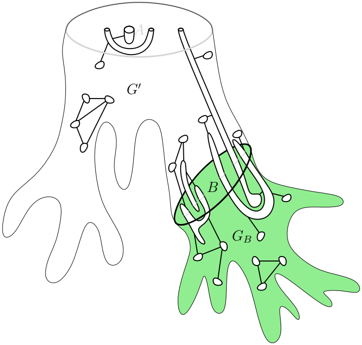


Algorithm in time $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ for any collection $\mathcal{F}$

[Fig. by Valentin Garnero]

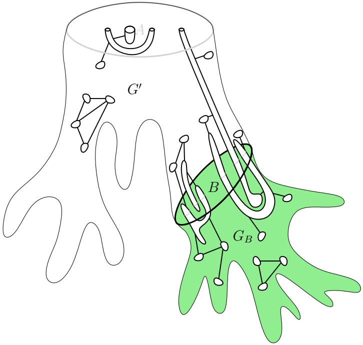
- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

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



Algorithm in time $2^{\mathcal{O}_{\mathcal{F}}(\text{tw} \cdot \log \text{tw})} \cdot n^{\mathcal{O}(1)}$ for any collection $\mathcal{F}$

[Fig. by Valentin Garnero]



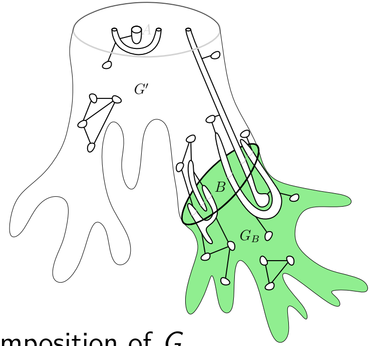
- For a fixed $\mathcal{F}$, we define an equivalence relation $\equiv^{(\mathcal{F},t)}$ on t-boundaried graphs:

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

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

Algorithm in time $2^{\mathcal{O}_{\mathcal{F}}(\mathrm{tw}\cdot\log \mathrm{tw})} \cdot n^{\mathcal{O}(1)}$ for any collection $\mathcal{F}$

[Fig. by Valentin Garnero]



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

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

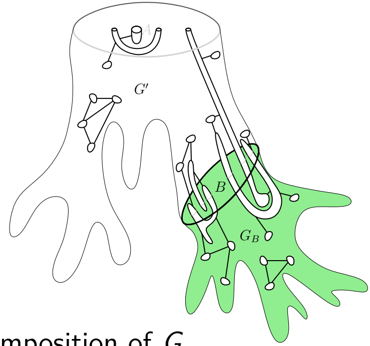
- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.

- We compute, using DP over a tree decomposition of G, the following parameter for every representative $R \in \mathcal{R}^{(\mathcal{F},t)}$:

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

Algorithm in time $2^{\mathcal{O}_{\mathcal{F}}(\text{tw}\cdot\log \text{tw})} \cdot n^{\mathcal{O}(1)}$ for any collection $\mathcal{F}$

[Fig. by Valentin Garnero]



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

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

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

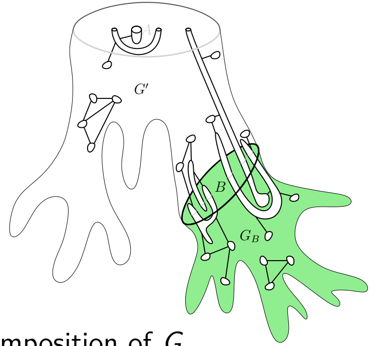
- We compute, using DP over a tree decomposition of G, the following parameter for every representative $R \in \mathcal{R}^{(\mathcal{F},t)}$:

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

- This gives an algorithm running in time $|\mathcal{R}^{(\mathcal{F},t)}|^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(1)}$.

Algorithm in time $2^{\mathcal{O}_{\mathcal{F}}(\text{tw}\cdot\log \text{tw})} \cdot n^{\mathcal{O}(1)}$ for any collection $\mathcal{F}$

[Fig. by Valentin Garnero]



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

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

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

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

- This gives an algorithm running in time $|\mathcal{R}^{(\mathcal{F},t)}|^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(1)}$.

- $\boxed{\text{Goal}}$ Bound the number of representatives: $|\mathcal{R}^{(\mathcal{F},t)}| = 2^{\mathcal{O}_{\mathcal{F}}(\text{tw}\cdot\log \text{tw})}$.

Bounding the set of representatives

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

Bounding the set of representatives

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

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

Bounding the set of representatives

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- Then, by the sparsity of the representatives,
$$|\mathcal{R}^{(\mathcal{F},t)}| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)},$$
and we are done!

Bounding the set of representatives

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- Then, by the sparsity of the representatives,
$$|\mathcal{R}^{(\mathcal{F},t)}| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)},$$
and we are done!
- Flat Wall Theorem [Robertson, Seymour. GMXIII. 1995]

Bounding the set of representatives

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- Then, by the sparsity of the representatives,
$$|\mathcal{R}^{(\mathcal{F},t)}| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)},$$
and we are done!
- $\boxed{\text{Flat Wall Theorem}}$ [Robertson, Seymour. GMXIII. 1995]
As a representative R is $\mathcal{F}$-minor-free, if $\mathrm{tw}(R \setminus B) > c_{\mathcal{F}}$,

Bounding the set of representatives

- $\mathcal{R}^{(\mathcal{F},t)}$: set of minimum-size representatives of $\equiv^{(\mathcal{F},t)}$.
- Suppose that we can prove that, for every $R \in \mathcal{R}^{(\mathcal{F},t)}$,
$$|V(R)| = \mathcal{O}_{\mathcal{F}}(t).$$
- Then, by the sparsity of the representatives,
$$|\mathcal{R}^{(\mathcal{F},t)}| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)},$$
and we are done!
- Flat Wall Theorem [Robertson, Seymour. GMXIII. 1995]

  As a representative R is $\mathcal{F}$-minor-free, if $\text{tw}(R \setminus B) > c_{\mathcal{F}}$, $R \setminus B$ contains a large flat wall,

Bounding the set of representatives

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

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

- Then, by the sparsity of the representatives,

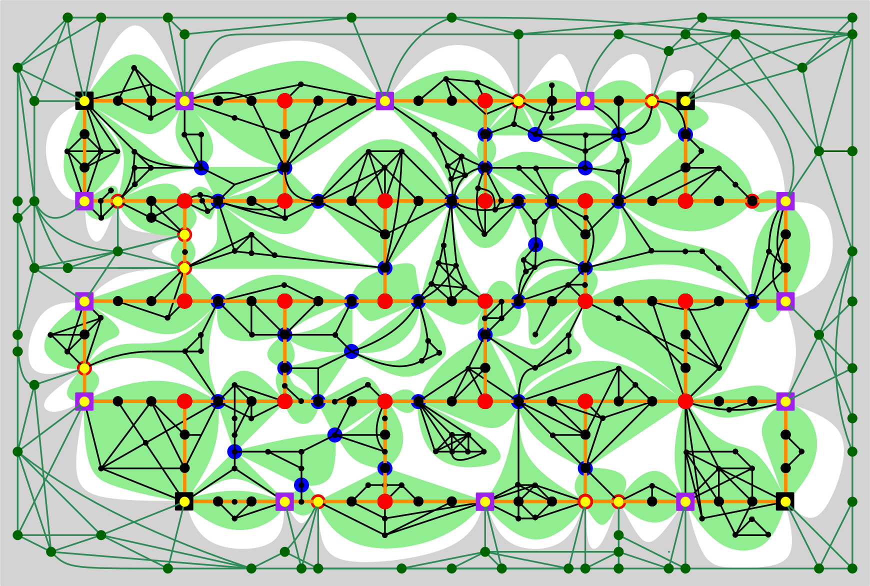
$$|\mathcal{R}^{(\mathcal{F},t)}| = \mathcal{O}_{\mathcal{F}}(1) \cdot \binom{t^2}{t} = 2^{\mathcal{O}_{\mathcal{F}}(t \cdot \log t)},$$

  and we are done!

- $\boxed{\text{Flat Wall Theorem}}$ [Robertson, Seymour. GMXIII. 1995]

  As a representative 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.

As we know, a flat wall can be quite wild...

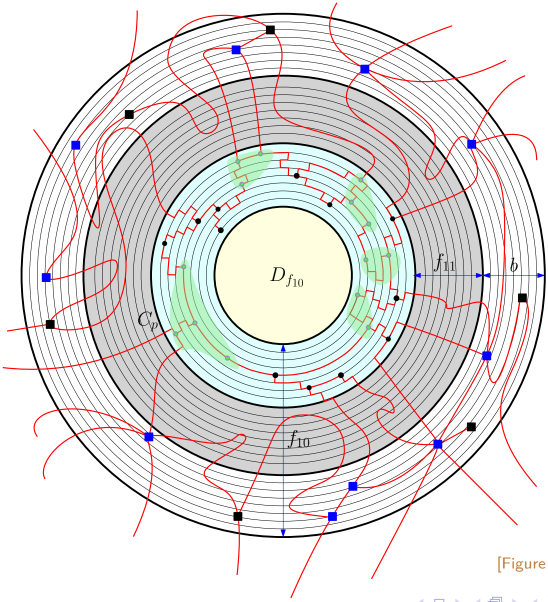


[Figure by Dimitrios M. Thilikos]

skip

Hard part: finding an irrelevant vertex inside a flat wall

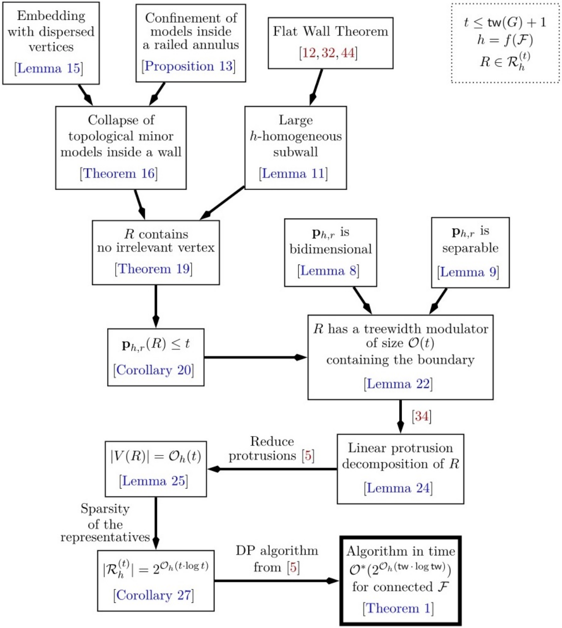
Hard part: finding an irrelevant vertex inside a flat wall



[Figure by Dimitrios M. Thilikos]

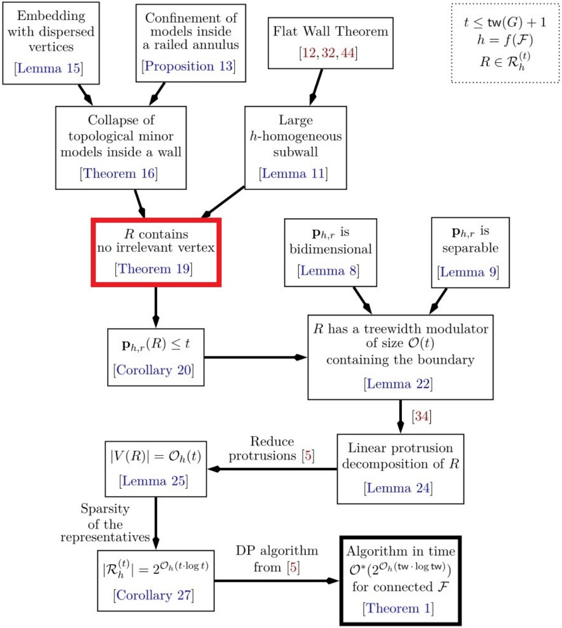
skip

Diagram of the algorithm for a general collection $\mathcal{F}$



skip

Diagram of the algorithm for a general collection $\mathcal{F}$



Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

We parameterize by the size of the desired solution

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

We parameterize by the size of the desired solution

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

It is easy to see that, for every $k \geqslant 1$, the class of graphs

$$\mathcal{C}_k = \{G \mid (G, k) \text{ is a positive instance of } \mathcal{F}\text{-M-Deletion}\}$$

is minor-closed.

We parameterize by the size of the desired solution

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

It is easy to see that, for every $k \geqslant 1$, the class of graphs

$$\mathcal{C}_k = \{G \mid (G,k) \text{ is a positive instance of } \mathcal{F}\text{-M-DELETION}\}$$

is minor-closed.

Theorem (Robertson and Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, deciding whether an n-vertex graph G belongs to $\mathcal{C}$ can be solved in time $f(\mathcal{C}) \cdot n^2$.

We parameterize by the size of the desired solution

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

It is easy to see that, for every $k \geqslant 1$, the class of graphs

$$\mathcal{C}_k = \{G \mid (G, k) \text{ is a positive instance of } \mathcal{F}\text{-M-Deletion}\}$$

is minor-closed.

Theorem (Robertson and Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, deciding whether an n-vertex graph G belongs to $\mathcal{C}$ can be solved in time $f(\mathcal{C}) \cdot n^2$.

For every $k \geq 1$, there exists an FPT algorithm for $\mathcal{F}$-M-Deletion.

We parameterize by the size of the desired solution

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

It is easy to see that, for every $k \geqslant 1$, the class of graphs

$$\mathcal{C}_k = \{G \mid (G, k) \text{ is a positive instance of } \mathcal{F}\text{-M-Deletion}\}$$

is minor-closed.

Theorem (Robertson and Seymour. 1983-2004)

For every minor-closed graph class $\mathcal{C}$, deciding whether an n-vertex graph G belongs to $\mathcal{C}$ can be solved in time $f(\mathcal{C}) \cdot n^2$.

For every $k \geq 1$, there exists an FPT algorithm for $\mathcal{F}$-M-Deletion.

But... only existential, non-uniform, $f(\mathcal{C}_k)$ astronomical.

Can we do better?

- The function $f(\mathcal{C}_k)$ is constructible. [Adler, Grohe, Kreutzer. 2008]

Can we do better?

- The function $f(\mathcal{C}_k)$ is constructible. [Adler, Grohe, Kreutzer. 2008]
- If $\mathcal{F}$ contains a planar graph: $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{\mathcal{O}(1)}$.

[Fomin, Lokshtanov, Misra, Saurabh. 2012]

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

Can we do better?

- The function $f(\mathcal{C}_k)$ is constructible. [Adler, Grohe, Kreutzer. 2008]
- If $\mathcal{F}$ contains a planar graph: $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{\mathcal{O}(1)}$.

  [Fomin, Lokshtanov, Misra, Saurabh. 2012]

  [Kim, Langer, Paul, Reidl, Rossmanith, S., Sikdar. 2013]
- For some non-planar collections $\mathcal{F}$:
  - $\mathcal{F} = \{K_5, K_{3,3}\}$: $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. [Jansen, Lokshtanov, Saurabh. 2014]

Can we do better?

- The function $f(\mathcal{C}_k)$ is constructible. [Adler, Grohe, Kreutzer. 2008]
- If $\mathcal{F}$ contains a planar graph: $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{\mathcal{O}(1)}$.

  [Fomin, Lokshtanov, Misra, Saurabh. 2012]

  [Kim, Langer, Paul, Reidl, Rossmanith, S., Sikdar. 2013]
- For some non-planar collections $\mathcal{F}$:
  - $\mathcal{F} = \{K_5, K_{3,3}\}$: $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. [Jansen, Lokshtanov, Saurabh. 2014]
  - Deletion to genus at most g: $2^{\mathcal{O}_g(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Ma. Pilipczuk. 2019]

Can we do better?

- The function $f(\mathcal{C}_k)$ is constructible. [Adler, Grohe, Kreutzer. 2008]
- If $\mathcal{F}$ contains a planar graph: $2^{\mathcal{O}_{\mathcal{F}}(k)} \cdot n^{\mathcal{O}(1)}$.
  [Fomin, Lokshtanov, Misra, Saurabh. 2012]
  [Kim, Langer, Paul, Reidl, Rossmanith, S., Sikdar. 2013]
- For some non-planar collections $\mathcal{F}$:
  - $\mathcal{F} = \{K_5, K_{3,3}\}$: $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. [Jansen, Lokshtanov, Saurabh. 2014]
  - Deletion to genus at most g: $2^{\mathcal{O}_g(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$. [Kociumaka, Ma. Pilipczuk. 2019]
- For every $\mathcal{F}$, some enormous explicit function $f_{\mathcal{F}}(k)$ can be derived from an FPT algorithm for hitting topological minors:

$$f_{\mathcal{F}}(k) \cdot n^{\mathcal{O}(1)}.$$

[Fomin, Lokshtanov, Panolan, Saurabh, Zehavi. 2020]

Our results

Theorem (S., Stamoulis, Thilikos. 2020)

For all $\mathcal{F}$, the $\mathcal{F}$-M-Deletion problem can be solved in time $2^{\mathrm{poly}(k)} \cdot n^3$.

Here, $\mathrm{poly}(k)$ is a polynomial whose degree depends on $\mathcal{F}$.

Our results

Theorem (S., Stamoulis, Thilikos. 2020)

For all $\mathcal{F}$, the $\mathcal{F}$-M-DELETION problem can be solved in time $2^{\mathsf{poly}(k)} \cdot n^3$.

Here, $\mathsf{poly}(k)$ is a polynomial whose degree depends on $\mathcal{F}$.

Theorem (S., Stamoulis, Thilikos. 2020)

If $\mathcal{F}$ contains an apex graph, the $\mathcal{F}$-M-DELETION problem can be solved in time $2^{\mathsf{poly}(k)} \cdot n^2$.

Again, $\mathsf{poly}(k)$ is a polynomial whose degree depends on $\mathcal{F}$.

Our results

Theorem (S., Stamoulis, Thilikos. 2020)

For all $\mathcal{F}$, the $\mathcal{F}$-M-DELETION problem can be solved in time $2^{\mathsf{poly}(k)} \cdot n^3$.

Here, $\mathsf{poly}(k)$ is a polynomial whose degree depends on $\mathcal{F}$.

Theorem (S., Stamoulis, Thilikos. 2020)

If $\mathcal{F}$ contains an apex graph, the $\mathcal{F}$-M-DELETION problem can be solved in time $2^{\mathsf{poly}(k)} \cdot n^2$.

Again, $\mathsf{poly}(k)$ is a polynomial whose degree depends on $\mathcal{F}$.

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

For all $\mathcal{F}$, the $\mathcal{F}$-M-DELETION problem can be solved in time $2^{\mathsf{poly}(k)} \cdot n^2$.

Sketch of the proofs

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

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

Sketch of the proofs

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

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

Theorem (S., Stamoulis, Thilikos. 2020)

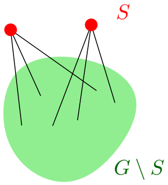
For all $\mathcal{F}$, the $\mathcal{F}$-M-Deletion problem can be solved in time $2^{\mathrm{poly}(k)} \cdot n^3$.

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]

General scheme of the algorithm:

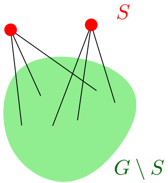
[whole slide shamelessly borrowed from Giannos Stamoulis]



Iterative compression: given solution S of size $k+1$, search solution of size k.

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]

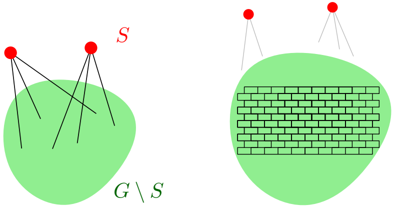


Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]



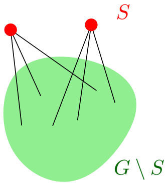
Iterative compression: given solution S of size $k + 1$, search solution of size k.

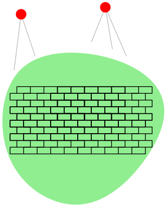
If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

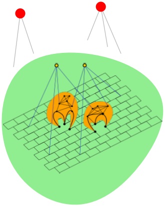
1. Find a "very very large" wall in $G \setminus S$.

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]





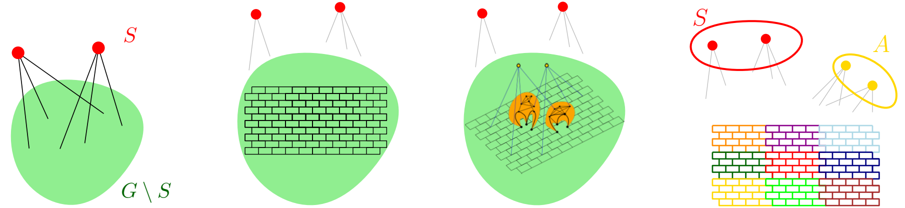


Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.

General scheme of the algorithm: [whole slide shamelessly borrowed from Giannos Stamoulis]



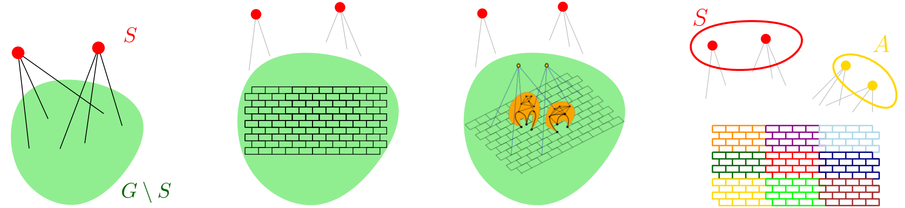
Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]



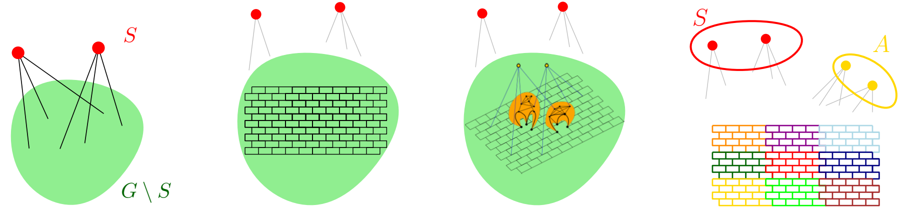
Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:
   - If every subwall has at least $|A|+1$ neighbors in $S \cup A$:

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]



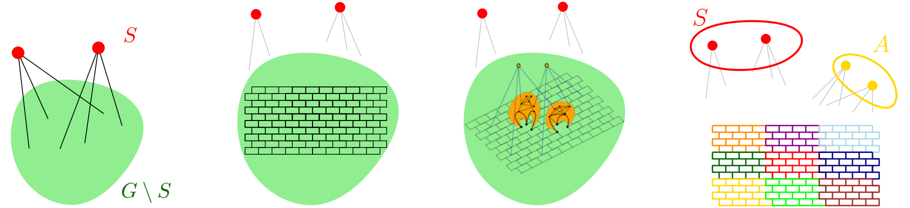
Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:
   - If every subwall has at least $|A|+1$ neighbors in $S \cup A$:
     Every solution intersects $S \cup A$ → we can branch!

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]



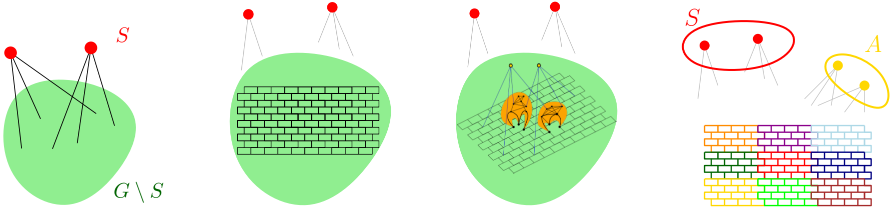
Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:
   - If every subwall has at least $|A|+1$ neighbors in $S \cup A$:
     Every solution intersects $S \cup A \rightarrow$ we can branch!
   - If one of these subwalls has at most $|A|$ neighbors in $S \cup A$:

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]

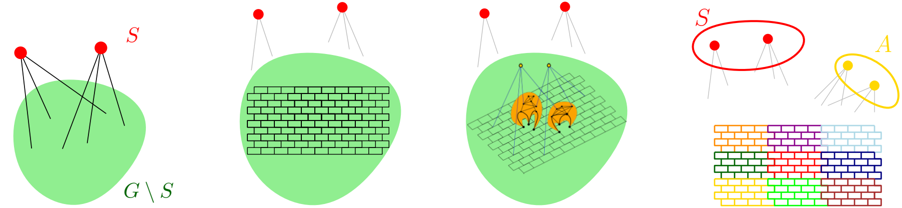


Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:
   - If every subwall has at least $|A|+1$ neighbors in $S \cup A$:
     Every solution intersects $S \cup A$ $\rightarrow$ we can branch!
   - If one of these subwalls has at most $|A|$ neighbors in $S \cup A$:
     Find an irrelevant vertex v inside this flat subwall.
     Update $G = G \setminus v$ and repeat.

General scheme of the algorithm: [whole slide shamelessly borrowed from Giannos Stamoulis]



Iterative compression: given solution S of size $k+1$, search solution of size k.

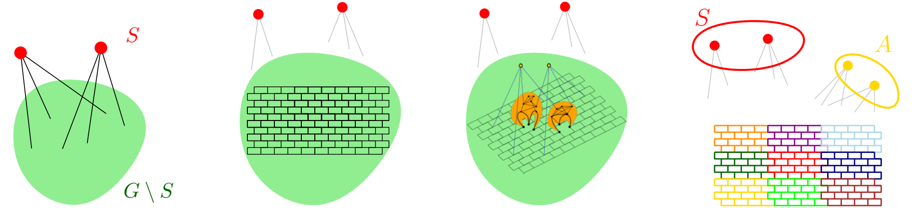
If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:
    - If every subwall has at least $|A|+1$ neighbors in $S \cup A$:
      Every solution intersects $S \cup A$ → we can branch!
    - If one of these subwalls has at most $|A|$ neighbors in $S \cup A$:
      Find an irrelevant vertex v inside this flat subwall.
      Update $G = G \setminus v$ and repeat.

Thus, $\mathrm{tw}(G \setminus S) = k^{\mathcal{O}_{\mathcal{F}}(1)}$.

General scheme of the algorithm:

[whole slide shamelessly borrowed from Giannos Stamoulis]



Iterative compression: given solution S of size $k+1$, search solution of size k.

If treewidth of $G \setminus S$ is "large enough" (as a polynomial function of k):

1. Find a "very very large" wall in $G \setminus S$.
2. Find a "very large" flat wall W of $G \setminus S$ with few apices A.
3. Find in W a packing of $\mathcal{O}_{\mathcal{F}}(k^4)$ disjoint "large" subwalls:
   - If every subwall has at least $|A|+1$ neighbors in $S \cup A$:
     Every solution intersects $S \cup A$ $\rightarrow$ we can branch!
   - If one of these subwalls has at most $|A|$ neighbors in $S \cup A$:
     Find an irrelevant vertex v inside this flat subwall.
     Update $G = G \setminus v$ and repeat.

Thus, $\mathrm{tw}(G \setminus S) = k^{\mathcal{O}_{\mathcal{F}}(1)}$: our previous FPT algo gives $2^{k^{\mathcal{O}_{\mathcal{F}}(1)}} \cdot n^2$.

Main idea of our improved algorithm

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

For all $\mathcal{F}$, the $\mathcal{F}$-M-Deletion problem can be solved in time $2^{\text{poly}(k)} \cdot n^2$.

Improvement from n^3 to n^2: avoiding iterative compression.

Main idea of our improved algorithm

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

For all $\mathcal{F}$, the $\mathcal{F}$-M-Deletion problem can be solved in time $2^{\mathsf{poly}(k)} \cdot n^2$.

Improvement from n^3 to n^2: avoiding iterative compression.

How to achieve it?

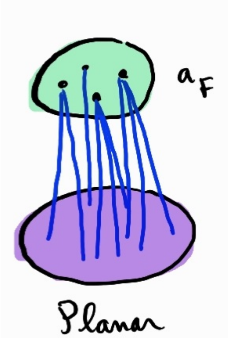
We are able to detect a vertex that must belong to every solution.

Approach inspired by [Marx, Schlotter. 2012] [S., Stamoulis, Thilikos. 2020]

skip

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

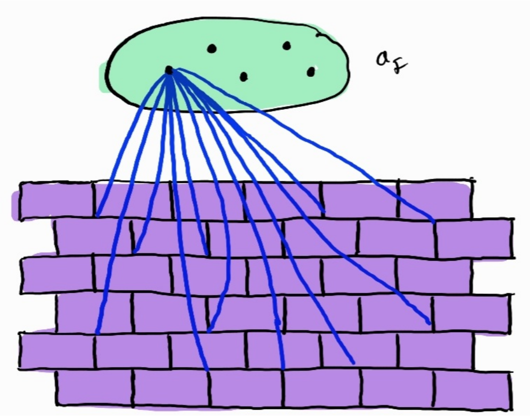
The **apex number** $a_{\mathcal{F}}$ is the smallest number of vertices that can be removed from a graph of $\mathcal{F}$ such that the remaining graph is planar.



[Figure by Laure Morelle]

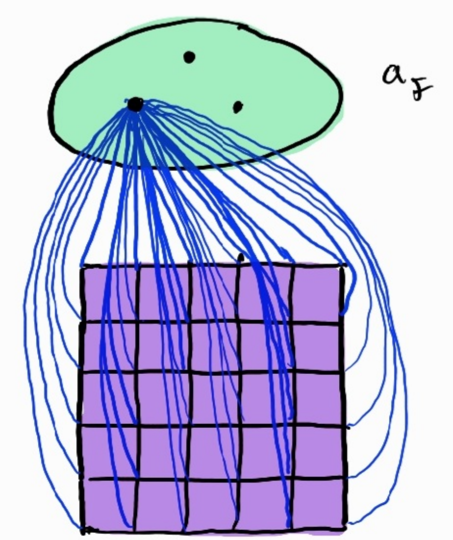
$a_{\mathcal{F}} = 1 \rightarrow$ apex graph

Finding a vertex belonging to every solution of size k



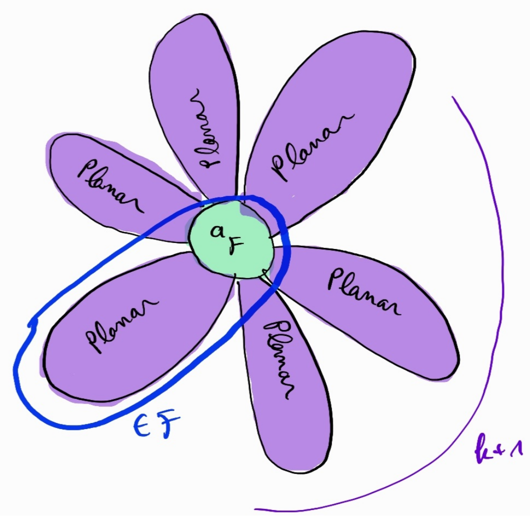
[Figure by Laure Morelle]

Finding a vertex belonging to every solution of size k



[Figure by Laure Morelle]

Finding a vertex belonging to every solution of size k



[Figure by Laure Morelle]

Strategy for solving $\mathcal{F}$-M-DELETION in time $2^{\text{poly}_{\mathcal{F}}(k)} \cdot n^2$:

Strategy for solving $\mathcal{F}$-M-DELETION in time $2^{\text{poly}_{\mathcal{F}}(k)} \cdot n^2$:

- If the treewidth of G is small (namely, $\text{tw} \leq \text{poly}_{\mathcal{F}}(k)$):

Strategy for solving $\mathcal{F}$-M-Deletion in time $2^{\text{poly}_{\mathcal{F}}(k)} \cdot n^2$:

- If the treewidth of G is small (namely, $\text{tw} \leq \text{poly}_{\mathcal{F}}(k)$):

  Dynamic programming using algorithm of [Baste, S., Thilikos. 2020]
  Solve in time $2^{\text{poly}_{\mathcal{F}}(\text{tw} \log \text{tw})} \cdot n$.

Strategy for solving $\mathcal{F}$-M-DELETION in time $2^{\text{poly}_{\mathcal{F}}(k)} \cdot n^2$:

- If the treewidth of G is small (namely, $\text{tw} \leq \text{poly}_{\mathcal{F}}(k)$):

  Dynamic programming using algorithm of [Baste, S., Thilikos. 2020]
  Solve in time $2^{\text{poly}_{\mathcal{F}}(\text{tw} \log \text{tw})} \cdot n$.

- If the treewidth of G is big, remove a vertex from G using one of the following approaches:

Strategy for solving $\mathcal{F}$-M-Deletion in time $2^{\mathrm{poly}_{\mathcal{F}}(k)} \cdot n^2$:

- If the treewidth of G is small (namely, $\mathrm{tw} \leq \mathrm{poly}_{\mathcal{F}}(k)$):

  Dynamic programming using algorithm of [Baste, S., Thilikos. 2020]
  Solve in time $2^{\mathrm{poly}_{\mathcal{F}}(\mathrm{tw} \log \mathrm{tw})} \cdot n$.

- If the treewidth of G is big, remove a vertex from G using one of the following approaches:

  - **Irrelevant vertex technique**: time $\mathcal{O}^*(n)$.
    Detect vertex v such that (G, k) and $(G \setminus \{v\}, k)$ are equivalent instances of $\mathcal{F}$-M-Deletion.

Strategy for solving $\mathcal{F}$-M-DELETION in time $2^{\text{poly}_{\mathcal{F}}(k)} \cdot n^2$:

- If the treewidth of G is small (namely, $\text{tw} \leq \text{poly}_{\mathcal{F}}(k)$):

  Dynamic programming using algorithm of [Baste, S., Thilikos. 2020]
  Solve in time $2^{\text{poly}_{\mathcal{F}}(\text{tw} \log \text{tw})} \cdot n$.

- If the treewidth of G is big, remove a vertex from G using one of the following approaches:

  - **Irrelevant vertex technique**: time $\mathcal{O}^*(n)$.
    Detect vertex v such that (G, k) and $(G \setminus \{v\}, k)$ are equivalent instances of $\mathcal{F}$-M-DELETION.

  - **Branching**: time $\mathcal{O}^*(n^2)$.
    Find set A of $a_{\mathcal{F}}$ vertices that intersects every k-apex set.
    "Guess" a vertex $v \in A$ in a k-apex set and solve $(G \setminus \{v\}, k-1)$.

Strategy for solving $\mathcal{F}$-M-DELETION in time $2^{\text{poly}_{\mathcal{F}}(k)} \cdot n^2$:

- If the treewidth of G is small (namely, $\text{tw} \leq \text{poly}_{\mathcal{F}}(k)$):

  Dynamic programming using algorithm of [Baste, S., Thilikos. 2020]
  Solve in time $2^{\text{poly}_{\mathcal{F}}(\text{tw} \log \text{tw})} \cdot n$.

- If the treewidth of G is big, remove a vertex from G using one of the following approaches:

  - **Irrelevant vertex technique**: time $\mathcal{O}^*(n)$.
    Detect vertex v such that (G, k) and $(G \setminus \{v\}, k)$ are equivalent instances of $\mathcal{F}$-M-DELETION.

  - **Branching**: time $\mathcal{O}^*(n^2)$.
    Find set A of $a_{\mathcal{F}}$ vertices that intersects every k-apex set.
    "Guess" a vertex $v \in A$ in a k-apex set and solve $(G \setminus \{v\}, k-1)$.

    (Branching tree is of size $a_{\mathcal{F}}^k$, so we do *not* get an extra factor n).

skip

Next subsection is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

Motivation: distance from triviality

Distance from triviality: [Guo, Hüffner, Niedermeier. 2004]

Concept to express the closeness of a graph G to a "trivial" graph class $\mathcal{H}$.

Motivation: distance from triviality

Distance from triviality: [Guo, Hüffner, Niedermeier. 2004]

Concept to express the closeness of a graph G to a "trivial" graph class $\mathcal{H}$.

Motivation: Solve problems parameterized by the "distance to $\mathcal{H}$".

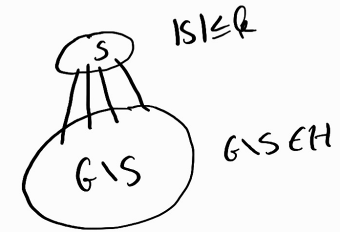
Motivation: distance from triviality

Distance from triviality: [Guo, Hüffner, Niedermeier. 2004]

Concept to express the closeness of a graph G to a "trivial" graph class $\mathcal{H}$.

Motivation: Solve problems parameterized by the "distance to $\mathcal{H}$".

$\rightarrow$ Vertex Deletion to $\mathcal{H}$



[Figure by Laure Morelle]

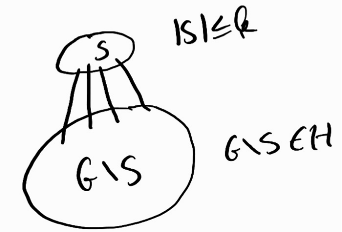
Motivation: distance from triviality

Distance from triviality: [Guo, Hüffner, Niedermeier. 2004]

Concept to express the closeness of a graph G to a "trivial" graph class $\mathcal{H}$.

Motivation: Solve problems parameterized by the "distance to $\mathcal{H}$".

$\rightarrow$ VERTEX DELETION TO $\mathcal{H}$



[Figure by Laure Morelle]

$\rightarrow$ ELIMINATION DISTANCE TO $\mathcal{H}$

Notion recently introduced by [Bulian, Dawar. 2016]

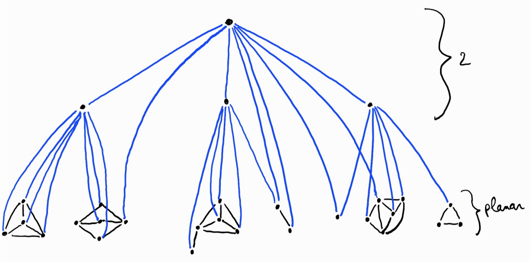
The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected}, \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$

Notion recently introduced by [Bulian, Dawar. 2016]

The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected}, \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$

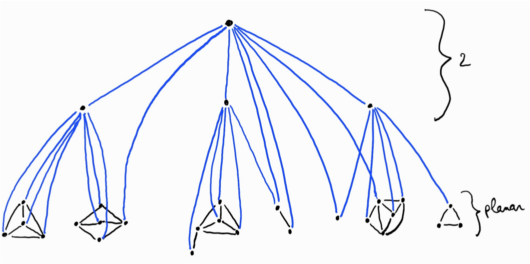


[Figure by Laure Morelle]

Notion recently introduced by [Bulian, Dawar. 2016]

The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected,} \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$



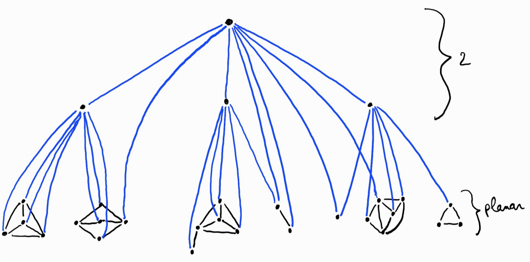
[Figure by Laure Morelle]

k-elimination set: set of removed vertices such that $\mathrm{ed}_{\mathcal{H}}(G) \leq k$.

Notion recently introduced by [Bulian, Dawar. 2016]

The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected}, \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$



[Figure by Laure Morelle]

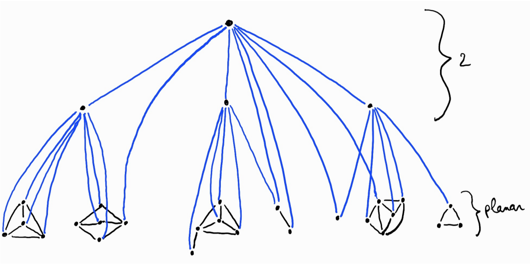
k-elimination set: set of removed vertices such that $\mathrm{ed}_{\mathcal{H}}(G) \leq k$.

Remark: the size of a k-elimination set is not necessarily a function of k!

Notion recently introduced by [Bulian, Dawar. 2016]

The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected}, \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$



[Figure by Laure Morelle]

k-elimination set: set of removed vertices such that $\mathrm{ed}_{\mathcal{H}}(G) \leq k$.

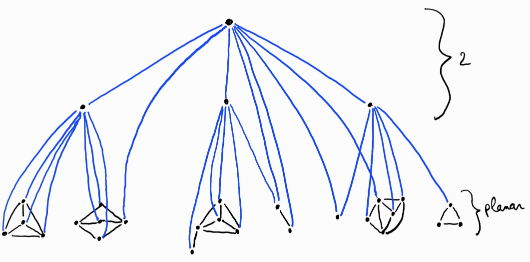
Remark: the size of a k-elimination set is not necessarily a function of k!

$\rightarrow \mathcal{H} = \{\emptyset\}$: treedepth

Notion recently introduced by [Bulian, Dawar. 2016]

The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected}, \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$



[Figure by Laure Morelle]

k-elimination set: set of removed vertices such that $\mathrm{ed}_{\mathcal{H}}(G) \leq k$.

Remark: the size of a k-elimination set is not necessarily a function of k!

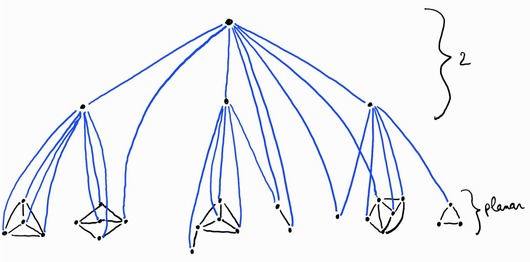
$\rightarrow \mathcal{H} = \{\emptyset\}$: treedepth

Stronger parameter than vertex deletion: $\mathrm{ed}_{\mathcal{H}}(G) \leq \mathrm{VertexDeletion}_{\mathcal{H}}(G)$

Notion recently introduced by [Bulian, Dawar. 2016]

The **elimination distance** of a graph G to a graph class $\mathcal{H}$ is:

$$\mathrm{ed}_{\mathcal{H}}(G) = \begin{cases} 0 & \text{if } G \in \mathcal{H}, \\ 1 + \min\{\mathrm{ed}_{\mathcal{H}}(G \setminus \{v\}) \mid v \in V(G)\} & \text{if } G \text{ is connected}, \\ \max\{\mathrm{ed}_{\mathcal{H}}(H) \mid H \text{ is a connected component of } G\} & \text{otherwise.} \end{cases}$$



[Figure by Laure Morelle]

ELIMINATION DISTANCE TO $\mathcal{H}$

Input: A graph G and a $k \in \mathbb{N}$.
Question: Is $\mathrm{ed}_{\mathcal{H}}(G) \leq k$?

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

What is known about $\textsc{Elimination Distance to } \mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of $\textsc{Elimination Distance to } \mathcal{H} \Leftrightarrow G \in \mathcal{E}_k(\mathcal{H})$.

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of ELIMINATION DISTANCE TO $\mathcal{H} \Leftrightarrow G \in \mathcal{E}_k(\mathcal{H})$.

$\mathcal{H}$ minor-closed

What is known about Elimination Distance to $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of Elimination Distance to $\mathcal{H}$ $\Leftrightarrow$ $G \in \mathcal{E}_k(\mathcal{H})$.

$\mathcal{H}$ minor-closed $\Rightarrow$ $\mathcal{E}_k(\mathcal{H})$ minor-closed

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of ELIMINATION DISTANCE TO $\mathcal{H}$ $\Leftrightarrow$ $G \in \mathcal{E}_k(\mathcal{H})$.

$\mathcal{H}$ minor-closed $\Rightarrow$ $\mathcal{E}_k(\mathcal{H})$ minor-closed $\Rightarrow$ non-constructive FPT-algo.

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of ELIMINATION DISTANCE TO $\mathcal{H} \Leftrightarrow G \in \mathcal{E}_k(\mathcal{H})$.

$\mathcal{H}$ minor-closed $\Rightarrow \mathcal{E}_k(\mathcal{H})$ minor-closed $\Rightarrow$ non-constructive FPT-algo.

If we are given $\mathcal{F} = \mathrm{Obs}(\mathcal{H})$, it is possible to construct $\mathrm{Obs}(\mathcal{E}_k(\mathcal{H}))$.

[Bulian, Dawar. 2017]

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of ELIMINATION DISTANCE TO $\mathcal{H}$ $\Leftrightarrow$ $G \in \mathcal{E}_k(\mathcal{H})$.

$\mathcal{H}$ minor-closed $\Rightarrow$ $\mathcal{E}_k(\mathcal{H})$ minor-closed $\Rightarrow$ non-constructive FPT-algo.

If we are given $\mathcal{F} = \mathrm{Obs}(\mathcal{H})$, it is possible to construct $\mathrm{Obs}(\mathcal{E}_k(\mathcal{H}))$.

[Bulian, Dawar. 2017]

$\Rightarrow$ constructive FPT-algorithm: $f(k) \cdot n^2$

What is known about ELIMINATION DISTANCE TO $\mathcal{H}$?

Let $\mathcal{E}_k(\mathcal{H}) = \{G \mid \mathrm{ed}_{\mathcal{H}}(G) \leq k\}$.

(G, k) yes-instance of ELIMINATION DISTANCE TO $\mathcal{H}$ $\Leftrightarrow$ $G \in \mathcal{E}_k(\mathcal{H})$.

$\mathcal{H}$ minor-closed $\Rightarrow$ $\mathcal{E}_k(\mathcal{H})$ minor-closed $\Rightarrow$ non-constructive FPT-algo.

If we are given $\mathcal{F} = \mathrm{Obs}(\mathcal{H})$, it is possible to construct $\mathrm{Obs}(\mathcal{E}_k(\mathcal{H}))$.

[Bulian, Dawar. 2017]

$\Rightarrow$ constructive FPT-algorithm: $f(k) \cdot n^2$

$\boxed{\text{Can we provide an explicit function } f(k)?}$

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathrm{tw})} \cdot n$.

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathrm{tw})} \cdot n$.

Since $\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G) \cdot \log n$

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathsf{tw})} \cdot n$.

Since $\mathsf{tw}(G) \leq \mathsf{td}(G) \leq \mathsf{tw}(G) \cdot \log n \rightarrow$ time $n^{\mathcal{O}(\mathsf{tw}^2)}$

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathrm{tw})} \cdot n$.

Since $\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G) \cdot \log n \rightarrow$ time $n^{\mathcal{O}(\mathrm{tw}^2)}$ and $2^{\mathcal{O}(k^2)} \cdot n$.

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathrm{tw})} \cdot n$.

Since $\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G) \cdot \log n \rightarrow$ time $n^{\mathcal{O}(\mathrm{tw}^2)}$ and $2^{\mathcal{O}(k^2)} \cdot n$.

(Open problem: computing td parameterized by tw is FPT?)

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathrm{tw})} \cdot n$.

Since $\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G) \cdot \log n \rightarrow$ time $n^{\mathcal{O}(\mathrm{tw}^2)}$ and $2^{\mathcal{O}(k^2)} \cdot n$.

(Open problem: computing td parameterized by tw is FPT?)

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and with treewidth at most tw, and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time $2^{\mathcal{O}_{\mathcal{H}}(k \cdot \mathrm{tw} + \mathrm{tw} \log \mathrm{tw})} \cdot n$.*

Taking the treewidth as the parameter

If $\mathcal{H} = \{\emptyset\}$ (treedepth): [Reidl, Rossmanith, Sanchez Villaamil, Sikdar. 2014]

Dynamic programming algorithm parameterized by treewidth in $2^{\mathcal{O}(k \cdot \mathrm{tw})} \cdot n$.

Since $\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G) \cdot \log n$ → time $n^{\mathcal{O}(\mathrm{tw}^2)}$ and $2^{\mathcal{O}(k^2)} \cdot n$.

(Open problem: computing td parameterized by tw is FPT?)

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and with treewidth at most tw, and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance* (G, k) *in time* $2^{\mathcal{O}_{\mathcal{H}}(k \cdot \mathrm{tw} + \mathrm{tw} \log \mathrm{tw})} \cdot n$.

→ algorithm in time $n^{\mathcal{O}_{\mathcal{H}}(\mathrm{tw}^2)}$ for Elimination Distance to $\mathcal{H}$.

Our results for Elimination Distance to $\mathcal{H}$

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time*

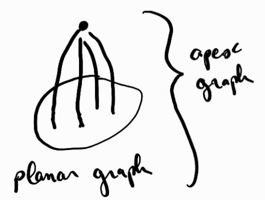
- $2^{2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}}} \cdot n^2$ *for a general minor-closed class $\mathcal{H}$,*

Our results for Elimination Distance to $\mathcal{H}$

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time*

- $2^{2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}}} \cdot n^2$ *for a general minor-closed class $\mathcal{H}$,*
- $2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}} \cdot n^2$ *if $\mathrm{Obs}(\mathcal{H})$ contains an apex graph.*



[Figure by Laure Morelle]

Our results for ELIMINATION DISTANCE TO $\mathcal{H}$

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves ELIMINATION DISTANCE TO $\mathcal{H}$ *for the instance (G, k) in time*

- $2^{2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}}} \cdot n^2$ *for a general minor-closed class $\mathcal{H}$,*
- $2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}} \cdot n^2$ *if $\mathrm{Obs}(\mathcal{H})$ contains an apex graph.*

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

If $\mathrm{Obs}(\mathcal{H})$ contains an apex graph, given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves ELIMINATION DISTANCE TO $\mathcal{H}$ *for the instance (G, k) in time $2^{\mathrm{poly}_{\mathcal{H}}(k)} \cdot n^3$.*

Our results for Elimination Distance to $\mathcal{H}$

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time*

- $2^{2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}}} \cdot n^2$ *for a general minor-closed class $\mathcal{H}$,*
- $2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}} \cdot n^2$ *if $\mathrm{Obs}(\mathcal{H})$ contains an apex graph.*

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

If $\mathrm{Obs}(\mathcal{H})$ contains an apex graph, given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time $2^{\mathrm{poly}_{\mathcal{H}}(k)} \cdot n^3$.*

Main challenge compared to Vertex Deletion to $\mathcal{H}$:

The size of a k-elimination set may be unbounded, so we cannot branch!

Our results for Elimination Distance to $\mathcal{H}$

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

Given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time*

- $2^{2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}}} \cdot n^2$ *for a general minor-closed class $\mathcal{H}$,*
- $2^{2^{\mathrm{poly}_{\mathcal{H}}(k)}} \cdot n^2$ *if $\mathrm{Obs}(\mathcal{H})$ contains an apex graph.*

Theorem (Morelle, S., Stamoulis, Thilikos. 2022)

If $\mathrm{Obs}(\mathcal{H})$ contains an apex graph, given a graph G on n vertices and $k \in \mathbb{N}$, there is an algorithm that solves Elimination Distance to $\mathcal{H}$ *for the instance (G, k) in time $2^{\mathrm{poly}_{\mathcal{H}}(k)} \cdot n^3$.*

Main challenge compared to Vertex Deletion to $\mathcal{H}$:

The size of a k-elimination set may be unbounded, so we cannot branch!
We always have to find an irrelevant vertex: larger treewidth bounds.

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

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- 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 $\mathcal{F}$-M-DELETION?

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-DELETION for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
- Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathsf{tw})}$ or $2^{\Theta(\mathsf{tw} \cdot \log \mathsf{tw})}$?

We can also consider the $\boxed{\text{topological minor}}$ version:

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
- Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathrm{tw})}$ or $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})}$?

We can also consider the $\boxed{\text{topological minor}}$ version:

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

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
- Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathrm{tw})}$ or $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})}$?

We can also consider 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.

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
- Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\text{tw})}$ or $2^{\Theta(\text{tw} \cdot \log \text{tw})}$?

We can also consider 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 $2^{o(\text{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.

$\boxed{\text{With parameter } k}$ We presented an algorithm in time $2^{k^{\mathcal{O}_{\mathcal{F}}(1)}} \cdot n^3$.

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
- Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\mathrm{tw})}$ or $2^{\Theta(\mathrm{tw} \cdot \log \mathrm{tw})}$?

We can also consider 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.

$\boxed{\text{With parameter } k}$ We presented an algorithm in time $2^{k^{\mathcal{O}_{\mathcal{F}}(1)}} \cdot n^3$.

Is $2^{\mathcal{O}_{\mathcal{F}}(k^c)} \cdot n^{\mathcal{O}(1)}$ possible for some constant c?

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

$\boxed{\text{With parameter tw}}$ Classify the asymptotic complexity of $\mathcal{F}$-M-Deletion for every family $\mathcal{F}$?

- We obtained a tight dichotomy when $|\mathcal{F}| = 1$ (connected).
- Missing: When $|\mathcal{F}| \geq 2$ (connected): $2^{\Theta(\text{tw})}$ or $2^{\Theta(\text{tw} \cdot \log \text{tw})}$?

We can also consider 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 $2^{o(\text{tw}^2)} \cdot n^{\mathcal{O}(1)}$ under the ETH.

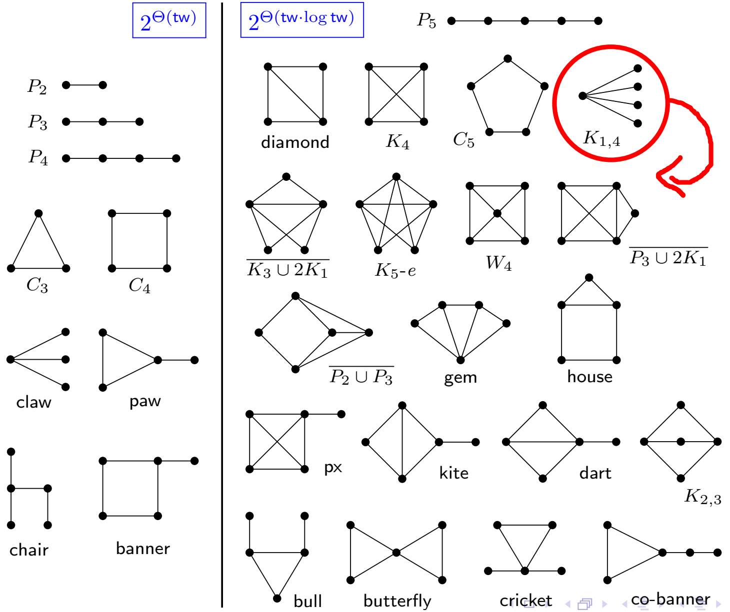
$\boxed{\text{With parameter } k}$ We presented an algorithm in time $2^{k^{\mathcal{O}_{\mathcal{F}}(1)}} \cdot n^3$.

Is $2^{\mathcal{O}_{\mathcal{F}}(k^c)} \cdot n^{\mathcal{O}(1)}$ possible for some constant c?

Is the price of homogeneity unavoidable?

skip

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



Next section is...

1. Introduction to graph minors
2. Introduction to parameterized complexity
3. Treewidth
   - Definition and simple properties
   - Brambles and duality
   - Computing treewidth
   - Dynamic programming on tree decompositions
   - Exploiting topology in dynamic programming
4. Bidimensionality
   - Some ingredients and an illustrative example
   - Meta-algorithms
5. Irrelevant vertex technique
6. Application to hitting minors
   - Parameterized by treewidth
   - Parameterized by solution size
   - More general modification operations
7. Kernelization (?)

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

Kernelization

Idea polynomial-time preprocessing.

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

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

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

Kernelization

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

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

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

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

The function g is called the size of the kernel.

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

Kernelization

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

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

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

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

The function g is called the size of the kernel.

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

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.

Now, on the board!

- Definitions.
- Some simple kernels.
- Crown decompositions.
- Kernels based on linear programming.
- Sunflower lemma.

References

1. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Parameterized Algorithms. Springer 2015. DOI: 10.1007/978-3-319-21275-3.
2. Meirav Zehavi, Saket Saurabh, Daniel Lokshtanov, and Fedor V. Fomin. Kernelization: Theory of Parameterized Preprocessing. Cambridge University Press 2019. DOI: 10.1017/9781107415157.

Gràcies!