Graph Minor Theory and its algorithmic consequences
MPRI Graph Algorithms

Pierre Aboulker - pierreaboulker@gmail.com

1 - FPT algorithm via Graph Minor Theorem

Graph modification problem

Problem (Graph modification problem for $\mathcal{C}$)

Given: (G, k)
Question: Is there a set S of at most k vertices such that $G \setminus S \in \mathcal{C}$?

- Vertex Cover: $\mathcal{C}$ is the class of edgeless graphs.
- Feedback vertex set: $\mathcal{C}$ is the set of forest (forbidden (induce) subgraphs is the set of all cycles).
- You can take any class of graphs for $\mathcal{C}$: planar graphs, bipartite graphs, chordal graphs etc etc.

Graph modification problem

Problem (Graph modification problem for $\mathcal{C}$)

Given: (G, k)
Question: Is there a set S of at most k vertices such that $G \setminus S \in \mathcal{C}$?

- Vertex Cover: $\mathcal{C}$ is the class of edgeless graphs.
- Feedback vertex set: $\mathcal{C}$ is the set of forest (forbidden (induce) subgraphs is the set of all cycles).
- You can take any class of graphs for $\mathcal{C}$: planar graphs, bipartite graphs, chordal graphs etc etc.

Theorem

If $\mathcal{C}$ is closed under taking induced subgraph and can be characterized by a finite set $\mathcal{F}$ of forbidden induced subgraphs, then the graph modification problem corresponding to $\mathcal{C}$ is FPT.

Graph modification problem

By the Graph Minor Theorem:

Theorem

If $\mathcal{C}$ is closed under taking minor, then the graph modification problem for $\mathcal{C}$ is FPT.

Graph modification problem

By the Graph Minor Theorem:

Theorem

If $\mathcal{C}$ is closed under taking minor, then the graph modification problem for $\mathcal{C}$ is FPT.

Indeed, if $\mathcal{C}$ is closed udner taking minor, then:

- The set of YES-instance for a fixed k is also closed under taking minor.

Graph modification problem

By the Graph Minor Theorem:

Theorem

If $\mathcal{C}$ is closed under taking minor, then the graph modification problem for $\mathcal{C}$ is FPT.

Indeed, if $\mathcal{C}$ is closed udner taking minor, then:

- The set of YES-instance for a fixed k is also closed under taking minor.
- So, for each k, there exists a finit set of graphs $\mathcal{F}$ such that the question is: does G in $Forb_{minor}(\mathcal{F})$?

Graph modification problem

By the Graph Minor Theorem:

Theorem

If $\mathcal{C}$ is closed under taking minor, then the graph modification problem for $\mathcal{C}$ is FPT.

Indeed, if $\mathcal{C}$ is closed udner taking minor, then:

- The set of YES-instance for a fixed k is also closed under taking minor.
- So, for each k, there exists a finit set of graphs $\mathcal{F}$ such that the question is: does G in $Forb_{minor}(\mathcal{F})$?
- There is a $f(k)O(n^3)$ algorithm.

Graph modification problem

By the Graph Minor Theorem:

Theorem

If $\mathcal{C}$ is closed under taking minor, then the graph modification problem for $\mathcal{C}$ is FPT.

Indeed, if $\mathcal{C}$ is closed udner taking minor, then:

- The set of YES-instance for a fixed k is also closed under taking minor.
- So, for each k, there exists a finit set of graphs $\mathcal{F}$ such that the question is: does G in $Forb_{minor}(\mathcal{F})$?
- There is a $f(k)O(n^3)$ algorithm.

Graph modification problem

By the Graph Minor Theorem:

Theorem

If $\mathcal{C}$ is closed under taking minor, then the graph modification problem for $\mathcal{C}$ is FPT.

Indeed, if $\mathcal{C}$ is closed udner taking minor, then:

- The set of YES-instance for a fixed k is also closed under taking minor.
- So, for each k, there exists a finit set of graphs $\mathcal{F}$ such that the question is: does G in $Forb_{minor}(\mathcal{F})$?
- There is a $f(k)O(n^3)$ algorithm.

Problems:

- This is just an existential proof, we do not know how to get the set $\mathcal{F}$ of forbidden minor (the Grid Minor Theorem is not constructive)
- It is not uniform: not the same algorithm for different values of k.

2 - FPT Algorithms parametrized by treewidth

Max Weighted Independent Set

Problem (Maximum Weighted Independent Set - MWIS)

Input : A graph G with weight function $\omega : V(G) \to \mathbb{R}$
Output : an Independent set of maximum weight.

- NP-complete for general graphs.
- Polynomial for trees by Dynamical programming

MWIS for Trees

Fix a root r arbitrarily.

Denote by $ch(v)$ the set of children of v, by $T(v)$ the subtree rooted at v (hence $T(r) = T$) and set:

MWIS for Trees

Fix a root r arbitrarily.

Denote by $ch(v)$ the set of children of v, by $T(v)$ the subtree rooted at v (hence $T(r) = T$) and set:

- $f(v)$ denotes the maximum weight of an independent set of $T(v)$,
- $f^+(v)$ denotes the maximum weight of an independent set of $T(v)$ containing v
- $f^-(v)$ denotes the maximum weight of an independent set of $T(v)$ not containing v

MWIS for Trees

Fix a root r arbitrarily.

Denote by $ch(v)$ the set of children of v, by $T(v)$ the subtree rooted at v (hence $T(r) = T$) and set:

- $f(v)$ denotes the maximum weight of an independent set of $T(v)$,
- $f^+(v)$ denotes the maximum weight of an independent set of $T(v)$ containing v
- $f^-(v)$ denotes the maximum weight of an independent set of $T(v)$ not containing v

The value of a maximum weight independent set of T is precisely $f(r)$.

MWIS for Trees

Let v be a vertex of T, and let $ch(v)$ be the set of children of v. We have:

$$f^+(v) = \Sigma_{x \in ch(v)} f^-(x) + \omega(v)$$
$$f^-(v) = \Sigma_{x \in ch(v)} f(x)$$
$$f(v) = \max(f^+(v), f^-(v))$$

It only remains to compute these three functions in a bottom-up fashion (that is starting from the leaves and computing layer after layer until we rich the root), which take $O(|V(T)|)$ time.

MWIS for Trees

Let v be a vertex of T, and let $ch(v)$ be the set of children of v. We have:

$$f^+(v) = \Sigma_{x \in ch(v)} f^-(x) + \omega(v)$$
$$f^-(v) = \Sigma_{x \in ch(v)} f(x)$$
$$f(v) = \max(f^+(v), f^-(v))$$

It only remains to compute these three functions in a bottom-up fashion (that is starting from the leaves and computing layer after layer until we rich the root), which take $O(|V(T)|)$ time.

Many NP-hard problems are solvable in polytime on trees, using dynamic programming.
We are going to see that the same strategy stands when applied on tree decomposition.

MWIS parametrized by treewidth

Theorem

Given a tree decomposition of width k, Maximum Weighted Independant Set can be computed in time $O(2^k \cdot k^{O(1)} \cdot n)$.

MWIS parametrized by treewidth

Theorem

Given a tree decomposition of width k, Maximum Weighted Independant Set can be computed in time $O(2^k \cdot k^{O(1)} \cdot n)$.

For each vertex $t \in V(T)$, set:
$W_t \subseteq V(G)$: vertices appearing in node t
$V_t \subseteq V(G)$: vertices appearing in the subtree rooted at t.

MWIS parametrized by treewidth

Theorem

Given a tree decomposition of width k, Maximum Weighted Independant Set can be computed in time $O(2^k \cdot k^{O(1)} \cdot n)$.

For each vertex $t \in V(T)$, set:
$W_t \subseteq V(G)$: vertices appearing in node t
$V_t \subseteq V(G)$: vertices appearing in the subtree rooted at t.

Generalizing the strategy used for tree:
Instead of computing two values $f^+(t)$ and $f^-(t)$, we compute $2^{|W_t|} \leq 2^k$ values for each bag W_t.

MWIS parametrized by treewidth

Theorem

Given a tree decomposition of width k, Maximum Weighted Independant Set can be computed in time $O(2^k \cdot k^{O(1)} \cdot n)$.

For each vertex $t \in V(T)$, set:
$W_t \subseteq V(G)$: vertices appearing in node t
$V_t \subseteq V(G)$: vertices appearing in the subtree rooted at t.

Generalizing the strategy used for tree:
Instead of computing two values $f^+(t)$ and $f^-(t)$, we compute $2^{|W_t|} \leq 2^k$ values for each bag W_t.

For each node t and each subset S of W_t:
$M[t, S]$ = max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.

MWIS parametrized by treewidth

Theorem

Given a tree decomposition of width k, Maximum Weighted Independant Set can be computed in time $O(2^k \cdot k^{O(1)} \cdot n)$.

For each vertex $t \in V(T)$, set:
$W_t \subseteq V(G)$: vertices appearing in node t
$V_t \subseteq V(G)$: vertices appearing in the subtree rooted at t.

Generalizing the strategy used for tree:
Instead of computing two values $f^+(t)$ and $f^-(t)$, we compute $2^{|W_t|} \leq 2^k$ values for each bag W_t.

For each node t and each subset S of W_t:
$M[t, S]$ = max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.

It is easy to compute $M[t, S]$ if the values are known for the children of t. But we are going to define a tree decomposition with a particular structure to ease it even more.

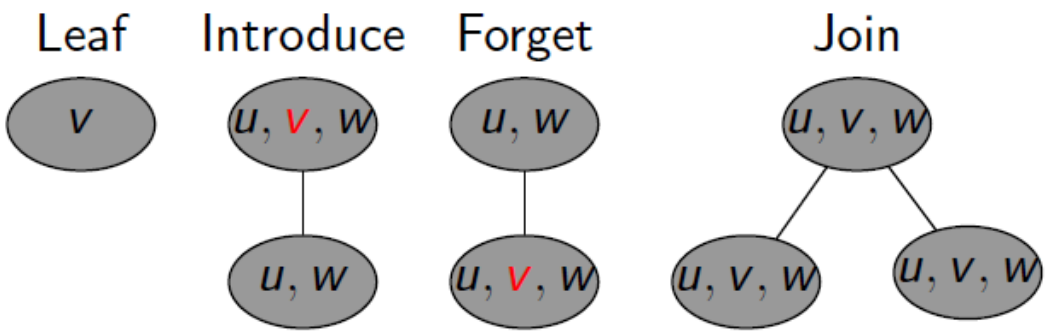
Nice Tree Decompositions

To design algorithms parametrized by treewidth, it is convenient to use the following particular tree decompositions.

Definition

A nice tree decomposition of G is a tree decomposition where T is a rooted binary tree with bags $(W_t)_{t \in V(T)}$ and each inner node t is of three possible kind :

- Leaf: t has no child and $|W_t| = 1$.
- Introduce: t has one child t' and $W_t = W_{t'} \cup \{v\}$ for some $v \notin W_{t'}$
- Forget: t has one child t' and $W_t = W_t \setminus \{v\}$ for some $v \notin W_t$.
- Join: t has two children t_1 and t_2 and $W_t = W_{t_1} = W_{t_2}$.



From tree decomposition to nice tree decomposition

Theorem

A tree decomposition of width k and n nodes can be turned into a nice tree decomposition of width k and $O(k \cdot n)$ nodes in time $O(k^2 \cdot n)$.

Proof Sketch:

- Root the decomposition arbitrarily.

From tree decomposition to nice tree decomposition

Theorem

A tree decomposition of width k and n nodes can be turned into a nice tree decomposition of width k and $O(k \cdot n)$ nodes in time $O(k^2 \cdot n)$.

Proof Sketch:

- Root the decomposition arbitrarily.
- For each internal node with p children, it is possible to add $2p$ new join nodes to make it binary.

From tree decomposition to nice tree decomposition

Theorem

A tree decomposition of width k and n nodes can be turned into a nice tree decomposition of width k and $O(k \cdot n)$ nodes in time $O(k^2 \cdot n)$.

Proof Sketch:

- Root the decomposition arbitrarily.
- For each internal node with p children, it is possible to add $2p$ new join nodes to make it binary.
- For each edge $t_1 t_2$ replace $t_1 t_2$ by a path with at most k forget nodes and at most k introduce nodes.

From tree decomposition to nice tree decomposition

Theorem

A tree decomposition of width k and n nodes can be turned into a nice tree decomposition of width k and $O(k \cdot n)$ nodes in time $O(k^2 \cdot n)$.

Proof Sketch:

- Root the decomposition arbitrarily.
- For each internal node with p children, it is possible to add $2p$ new join nodes to make it binary.
- For each edge $t_1 t_2$ replace $t_1 t_2$ by a path with at most k forget nodes and at most k introduce nodes.

From tree decomposition to nice tree decomposition

Theorem

A tree decomposition of width k and n nodes can be turned into a nice tree decomposition of width k and $O(k \cdot n)$ nodes in time $O(k^2 \cdot n)$.

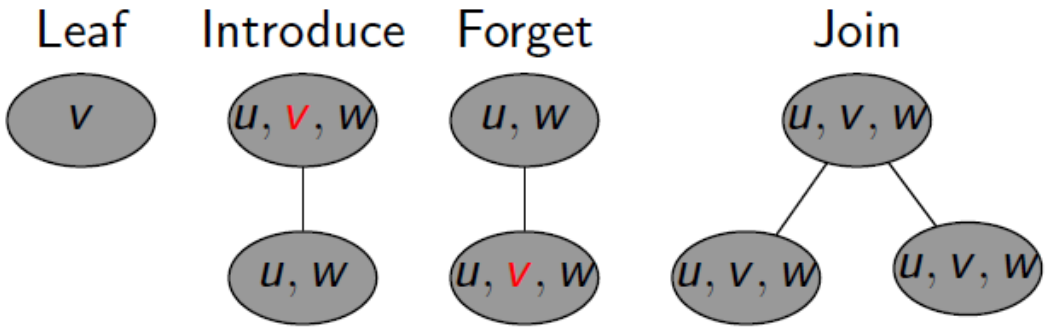
Proof Sketch:

- Root the decomposition arbitrarily.
- For each internal node with p children, it is possible to add $2p$ new join nodes to make it binary.
- For each edge $t_1 t_2$ replace $t_1 t_2$ by a path with at most k forget nodes and at most k introduce nodes.

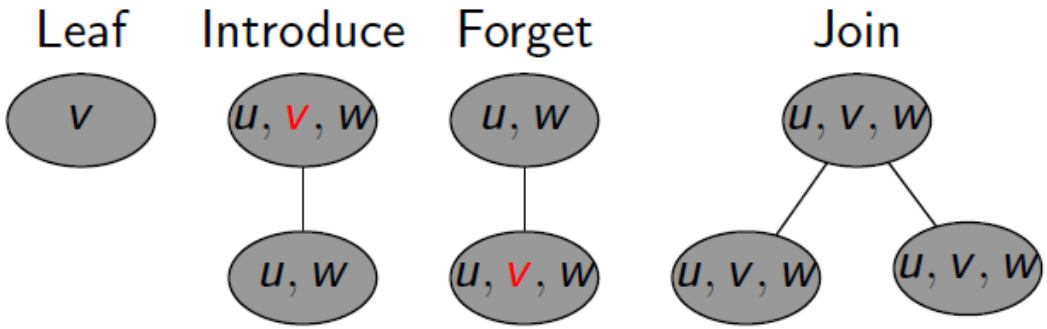
Using nice decomposition, it becomes super easy to compute $M[t, S]$ in a bottom-up fashion.

For each node $t \in V(T)$ and each independant subset S of W_t:
$M[t, S] =$ max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.

For each node $t \in V(T)$ and each independant subset S of W_t:
$M[t, S] =$ max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.

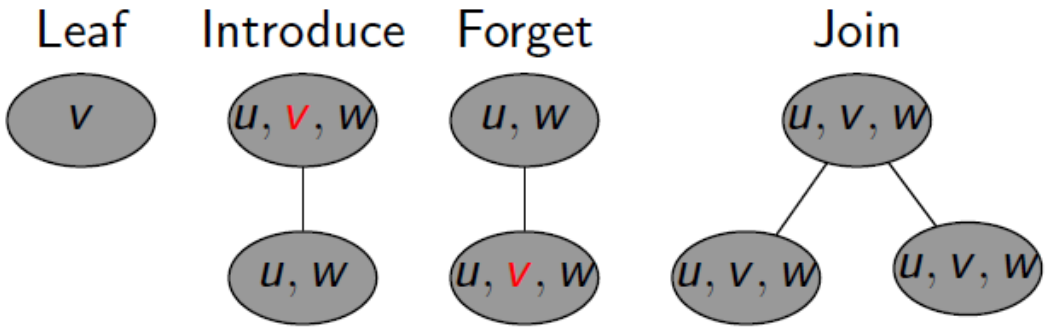


For each node $t \in V(T)$ and each independant subset S of W_t:
$M[t, S] =$ max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.



- **Leaf**: $|W_t| = 1$, trivial

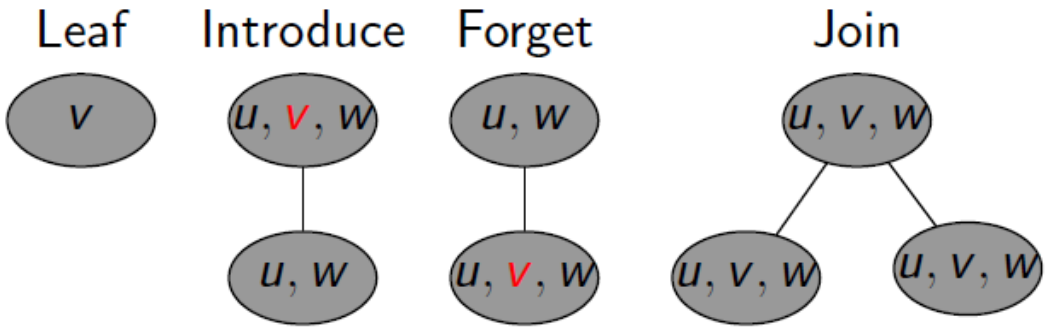
For each node $t \in V(T)$ and each independant subset S of W_t:
$M[t, S] =$ max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.



- **Leaf**: $|W_t| = 1$, trivial
- **Introduce**: one child t' with $W_t = W_{t'} \cup v$:

$$\begin{aligned} M[t, S] &= M[t', S] && \text{if } v \notin S \\ &= M[t', S \setminus \{v\}] + \omega(v) && \text{if } v \in S \end{aligned}$$

For each node $t \in V(T)$ and each independant subset S of W_t:
$M[t, S] =$ max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.



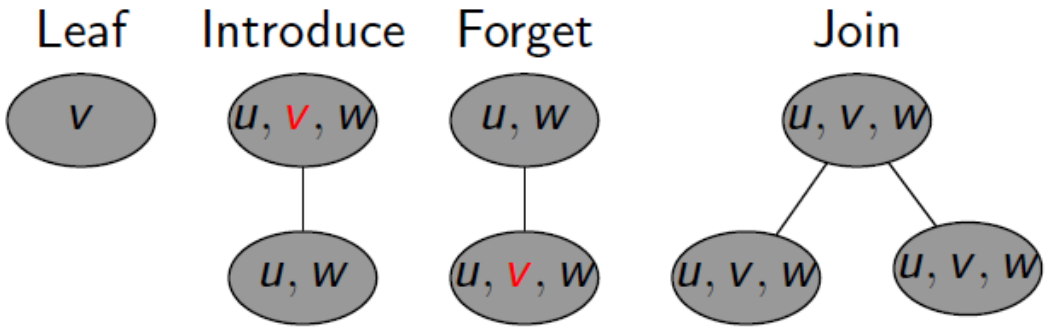
- **Leaf**: $|W_t| = 1$, trivial
- **Introduce**: one child t' with $W_t = W_{t'} \cup v$:

$$\begin{aligned} M[t, S] &= M[t', S] && \text{if } v \notin S \\ &= M[t', S \setminus \{v\}] + \omega(v) && \text{if } v \in S \end{aligned}$$

- **Forget**: one child t' with $W_t = W_{t'} \setminus v$:

$$M[t, S] = \max(M[t', S], M[t', S \cup \{v\}])$$

For each node $t \in V(T)$ and each independant subset S of W_t:
$M[t, S] =$ max weighted independant I such that $I \subseteq V_t$ and $I \cap W_t = S$.



- **Leaf**: $|W_t| = 1$, trivial
- **Introduce**: one child t' with $W_t = W_{t'} \cup v$:

$$\begin{aligned} M[t, S] &= M[t', S] && \text{if } v \notin S \\ &= M[t', S \setminus \{v\}] + \omega(v) && \text{if } v \in S \end{aligned}$$

- **Forget**: one child t' with $W_t = W_{t'} \setminus v$:

$$M[t, S] = \max(M[t', S], M[t', S \cup \{v\}])$$

- **Join**: t has two children t_1 and t_2 such that $W_t = W_{t_1} = W_{t_2}$:

$$M[t, S] = M[t_1, S] + M[t_2, S] - \omega(S)$$

Other Problems

Here is a list of results one can prove similarly using a tree decomposition of treewidth k.

Theorem

Let G be given with a tree decomposition of width at most k.

1. *Computing $vc(G)$ can be done in time $O\left(2^k \cdot k^{O(1)} \cdot n\right)$.*
2. *Computing $\chi(G)$ can be done in time $O(f(k) \cdot n)$.*
3. *Computing $\omega(G)$ can be done in time $O\left(2^k \cdot k^k \cdot n\right)$.*
4. *Computing $\gamma(G) := \min\{|X| : X \cup N(X) = V(G)\}$, can be done in time $O\left(4^k \cdot k^{O(1)} n\right)$. (dominating set).*
5. *Deciding if G has a hamiltonian cycle can be done in time $O\left(k^{O(k)} \cdot n\right)$.*

3 - Courcelle's Theorem

Monadic Second Order Logic

A celebrated algorithmic meta-theorem of Courcelle generalises all the previous results to monadic second order formulas.

Logical formulas on graphs are constructed inductively using

- atomic formulas : $x = y$, $v \in X$, $e \in F$ for subsets of vertices or edges.
- the binary relation $Inc(x, e)$ which is satisfied if $x \in V$ and x is incident with $e \in E$.
- logical operators $\vee$ and $\wedge$ and $\neg$
- quantifiers $\forall$ and $\exists$

Monadic Second Order Logic

A celebrated algorithmic meta-theorem of Courcelle generalises all the previous results to monadic second order formulas.

Logical formulas on graphs are constructed inductively using

- atomic formulas : $x = y$, $v \in X$, $e \in F$ for subsets of vertices or edges.
- the binary relation $Inc(x, e)$ which is satisfied if $x \in V$ and x is incident with $e \in E$.
- logical operators $\vee$ and $\wedge$ and $\neg$
- quantifiers $\forall$ and $\exists$
- First Order formulas : quantifiers over vertices and edges ($\forall v \in V(G)$; $\exists e \in E(G)$)
- MSO_1 = FO + quantify over sets of vertices,
- MSO_2 = MSO_1 + quantify over sets of edges.

Formula for 3-colorability

This is a second order formula for 3 colourability :

$$\begin{aligned}
& & \exists X_1 \subset V\ \exists X_2 \subset V\ \exists X_3 \subset V & \\
& (\forall x \in V & (x \in X_1 \vee x \in X_2 \vee x \in X_3) & \\
& & \wedge \neg(x \in X_1 \wedge x \in X_2) \wedge \neg(x \in X_1 \wedge x \in X_3) \wedge \neg(x \in X_2 \wedge x \in X_3)) & \\
\wedge & (\forall xy \in E & \neg(x \in X_1 \wedge y \in X_1) \wedge \neg(x \in X_2 \wedge y \in X_2) \wedge \neg(x \in X_3 \wedge y \in X_3)) &
\end{aligned}$$

Courcelle Theorem

The theorem of Courcelle asserts that every such property is easy to decide for bounded treewidth graphs.

Theorem (Courcelle, 1990)

Let G be a graph and ϕ a formula of MSO_2. Assume that we are given a tree decomposition of G of width at most k. Then there is an algorithm that verify if φ is satisfied in G in time $f(|\varphi|, k) \cdot n$ for some computable function f.

Courcelle Theorem

The theorem of Courcelle asserts that every such property is easy to decide for bounded treewidth graphs.

Theorem (Courcelle, 1990)

Let G be a graph and ϕ a formula of MSO_2. Assume that we are given a tree decomposition of G of width at most k. Then there is an algorithm that verify if φ is satisfied in G in time $f(|\varphi|, k) \cdot n$ for some computable function f.

Note: The dependance on k can be very large (double, triple exponential etc.), therefore a direct dynamic programming algorithm can be more efficient.

If we can express a property in MSO_2, then we immediately get that testing this property is FPT parameterized by the treewidth k of the input graph.

4 - Computing Tree Decomposition

Computing tree width

Problem

Input : A graph G and an integer k
Output : TRUE if and only if $\mathrm{tw}(G) \leq k$

Computing tree width

Problem

Input : A graph G and an integer k
Output : TRUE if and only if $\mathrm{tw}(G) \leq k$

- **NP-Hard**: Arnborg, Corneil, Proskurowski '87 (note that polytime open for planar)

Computing tree width

Problem

Input : A graph G and an integer k
Output : TRUE if and only if $\mathrm{tw}(G) \leq k$

- **NP-Hard**: Arnborg, Corneil, Proskurowski '87 (note that polytime open for planar)

Anyway, we want to use tree decomposition to design *FPT* algorithm with parameter $\mathrm{tw}(G)$, so we would be happy with time $O\big(f(\mathrm{tw}(G)) \cdot n^{0(1)}\big)$.

Computing tree width

Problem

Input : A graph G and an integer k
Output : TRUE if and only if $\mathrm{tw}(G) \leq k$

- **NP-Hard**: Arnborg, Corneil, Proskurowski '87 (note that polytime open for planar)

Anyway, we want to use tree decomposition to design *FPT* algorithm with parameter $\mathrm{tw}(G)$, so we would be happy with time $O\left(f(\mathrm{tw}(G)) \cdot n^{0(1)}\right)$.

- **FPT** : $O\left(\mathrm{tw}(G)^{\mathrm{tw}(G)^3} n\right)$ algorithm (Bodlaender, 96)

Approximate the treewidth

We don't really need to compute an *optimal tree decomposition, the following is enough.*

Theorem (Robertson and Seymour)

Given a graph G and an integer k, there is an algorithm running in time $O(f(k).n^2)$ that output:

- *either a small certificate showing that $\mathrm{tw}(G) \geq k$*
- *or a tree decomposition of width at most $4k+1$.*

This is enough for our FPT algorithms seen before : simply run this for $k=1$, $k=2$, $k=3$,... one is guaranteed to find a tree decomposition of G of width at most $4\mathrm{tw}(G)$ in time $O(f(\mathrm{tw}(G)).n^2))$.

Approximate the treewidth

We don't really need to compute an *optimal tree decomposition, the following is enough.*

Theorem (Robertson and Seymour)

Given a graph G and an integer k, there is an algorithm running in time $O(f(k).n^2)$ that output:

- *either a small certificate showing that $\mathrm{tw}(G) \geq k$*
- *or a tree decomposition of width at most $4k+1$.*

This is enough for our FPT algorithms seen before : simply run this for $k=1$, $k=2$, $k=3$,... one is guaranteed to find a tree decomposition of G of width at most $4\mathrm{tw}(G)$ in time $O(f(\mathrm{tw}(G)).n^2))$.

The rest of this section is dedicated to design this algorithm.

Computing the treewidth: state of the art

Approximation	*f(k)*	*g(n)*	reference
exact	$O(1)$	$O(n^{k+2})$	Arnborg, Corneil & Proskurowski (1987)
$4k+3$	$O(3^{3k})$	$O(n^2)$	Robertson & Seymour (1995)
$8k+7$	$2^{O(k \log k)}$	$n \log^2 n$	Lagergren (1996)
$5k+4$	$2^{O(k \log k)}$	$n \log n$	Reed (1992)
exact	$k^{O(k^3)}$	$O(n)$	Bodlaender (1996)
$O(k \cdot \sqrt{\log k})$	$O(1)$	$n^{O(1)}$	Feige, Hajiaghayi & Lee (2008)
$4.5k+4$	2^{3k}	n^2	Amir (2010)
$\frac{11}{3}k+4$	$2^{3.6982k}$	n^2	Amir (2010)
exact	$O(1)$	$O(1.7347^n)$	Fomin, Todinca & Villanger (2015)
$3k+2$	$2^{O(k)}$	$O(n \log n)$	Bodlaender et al. (2016)
$5k+4$	$2^{O(k)}$	$O(n)$	Bodlaender et al. (2016)
k^2	$O(k^7)$	$O(n \log n)$	Fomin et al. (2018)
$5k+4$	$2^{8.765k}$	$O(n \log n)$	Belbasi & Fürer (2020)
$2k+1$	$2^{O(k)}$	$O(n)$	Korhonen (2021)

picture from wikipedia

Approximate the treewidth

Theorem

There exists an algorithm with input a graph G and an integer k and that outputs in time $O\left(f(k).n^2\right)$:

- *either a small certificate that $\mathrm{tw}(G) \geq k$*
- *or a tree decomposition of width at most $4k + 3$.*

The rest of this section is dedicated to design this algorithm.

Good separator

Let G be a graph. A set of vertices S is a separator (or vertex cutset) if S disconnects G, that is $G \setminus S$ has at least two connected components.

Let S, X be two sets of vertices, S is a good separator with respect to X if:

- S disconnects G into two parts V_1 and V_2 both intersecting X.
- For $i = 1, 2$, V_i contains at most $2|X|/3$ vertices of X.

(Such a separator is sometime called a 2/3-separator w.r.t. X).

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.
- For each node $t \in T$, the bag W_t separates G into 2 or 3 connected components.

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.
- For each node $t \in T$, the bag W_t separates G into 2 or 3 connected components.
- If one contains more than half of the vertices of X, then orient the corresponding edge out from t.

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.
- For each node $t \in T$, the bag W_t separates G into 2 or 3 connected components.
- If one contains more than half of the vertices of X, then orient the corresponding edge out from t.
- Prove that there exists an internal node t with no outgoing edge (observe that any edge incident with a leaf is oriented out from the leaf).

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.
- For each node $t \in T$, the bag W_t separates G into 2 or 3 connected components.
- If one contains more than half of the vertices of X, then orient the corresponding edge out from t.
- Prove that there exists an internal node t with no outgoing edge (observe that any edge incident with a leaf is oriented out from the leaf).
- Show that W_t is a good separator with respect to X.

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.
- For each node $t \in T$, the bag W_t separates G into 2 or 3 connected components.
- If one contains more than half of the vertices of X, then orient the corresponding edge out from t.
- Prove that there exists an internal node t with no outgoing edge (observe that any edge incident with a leaf is oriented out from the leaf).
- Show that W_t is a good separator with respect to X.

Certificate that $\mathrm{tw}(G) \geq k$

Proposition

If $\mathrm{tw}(G) < k$, then every $X \subseteq V(G)$ of size at least $2k+1$ admits a good separator of size at most k

Proof Ideas:

- Take a tree decomposition (T, W) of width $k-1$ where T has maximum degree 3.
- For each node $t \in T$, the bag W_t separates G into 2 or 3 connected components.
- If one contains more than half of the vertices of X, then orient the corresponding edge out from t.
- Prove that there exists an internal node t with no outgoing edge (observe that any edge incident with a leaf is oriented out from the leaf).
- Show that W_t is a good separator with respect to X.

Certificate that $\mathrm{tw}(G) \geq k$:
If G contains a set X of at least $2k+1$ vertices that do not admit a good separator of size a most k, then $\mathrm{tw}(G) \geq k$.

Proof of the previous theorems

We prove by induction on the number of vertices of G the following algorithm (apply it with $X = \emptyset$ to get the desired algorithm).

Problem

Input : A graph G, an integer k and a set $X \subseteq V(G)$ such that $|X| \leq 3k$
Output : A certificate that $tw(G) \geq k$ or a rooted tree decomposition T of G of width at most $4k + 1$ where $X \subseteq root(G)$

Proof of the previous theorems

We prove by induction on the number of vertices of G the following algorithm (apply it with $X = \emptyset$ to get the desired algorithm).

Problem

Input : A graph G, an integer k and a set $X \subseteq V(G)$ such that $|X| \leq 3k$
Output : A certificate that $tw(G) \geq k$ or a rooted tree decomposition T of G of width at most $4k + 1$ where $X \subseteq root(G)$

- If G has at most $4k$ vertices then put all vertices in a single bag.

Proof of the previous theorems

We prove by induction on the number of vertices of G the following algorithm (apply it with $X = \emptyset$ to get the desired algorithm).

Problem

Input : A graph G, an integer k and a set $X \subseteq V(G)$ such that $|X| \leq 3k$
Output : A certificate that $tw(G) \geq k$ or a rooted tree decomposition T of G of width at most $4k + 1$ where $X \subseteq root(G)$

- If G has at most $4k$ vertices then put all vertices in a single bag.
- If X has less than $2k + 1$ vertices, then augment X arbitrarily by adding vertices until its size is at least $2k + 1$.

Proof of the previous theorems

We prove by induction on the number of vertices of G the following algorithm (apply it with $X = \emptyset$ to get the desired algorithm).

Problem

Input : A graph G, an integer k and a set $X \subseteq V(G)$ such that $|X| \leq 3k$
Output : A certificate that $tw(G) \geq k$ or a rooted tree decomposition T of G of width at most $4k + 1$ where $X \subseteq root(G)$

- If G has at most $4k$ vertices then put all vertices in a single bag.
- If X has less than $2k + 1$ vertices, then augment X arbitrarily by adding vertices until its size is at least $2k + 1$.
- Assume for the moment that we know how to compute a good separator of size at most k wrt X.

Proof of the previous theorems

We prove by induction on the number of vertices of G the following algorithm (apply it with $X = \emptyset$ to get the desired algorithm).

Problem

Input : A graph G, an integer k and a set $X \subseteq V(G)$ such that $|X| \leq 3k$
Output : A certificate that $tw(G) \geq k$ or a rooted tree decomposition T of G of width at most $4k + 1$ where $X \subseteq root(G)$

- If G has at most $4k$ vertices then put all vertices in a single bag.
- If X has less than $2k + 1$ vertices, then augment X arbitrarily by adding vertices until its size is at least $2k + 1$.
- Assume for the moment that we know how to compute a good separator of size at most k wrt X.
- If X admits no good separator of size at most k, then by what precedes it is a certificate that $tw(G) \geq k$.

Proof of the previous theorems

We prove by induction on the number of vertices of G the following algorithm (apply it with $X = \emptyset$ to get the desired algorithm).

Problem

Input : A graph G, an integer k and a set $X \subseteq V(G)$ such that $|X| \leq 3k$
Output : A certificate that $tw(G) \geq k$ or a rooted tree decomposition T of G of width at most $4k + 1$ where $X \subseteq root(G)$

- If G has at most $4k$ vertices then put all vertices in a single bag.
- If X has less than $2k + 1$ vertices, then augment X arbitrarily by adding vertices until its size is at least $2k + 1$.
- Assume for the moment that we know how to compute a good separator of size at most k wrt X.
- If X admits no good separator of size at most k, then by what precedes it is a certificate that $tw(G) \geq k$.
- So we may assume that X has a good separator S of size at most k.

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$
- Observe that $|V(G_i)| < |V(G)|$.

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$
- Observe that $|V(G_i)| < |V(G)|$.
- Apply the algorithm on $(G_i,\ k,\ X_i)$.

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$
- Observe that $|V(G_i)| < |V(G)|$.
- Apply the algorithm on $(G_i,\ k,\ X_i)$.
- Either certificate that $tw(G_i) \geq k$ for some i and therefore $tw(G) \geq k$

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$
- Observe that $|V(G_i)| < |V(G)|$.
- Apply the algorithm on $(G_i,\ k,\ X_i)$.
- Either certificate that $tw(G_i) \geq k$ for some i and therefore $tw(G) \geq k$
- Or get two rooted decompositions T_1, T_2 of G_1 and G_2 with $X_i \subseteq root(T_i)$

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$
- Observe that $|V(G_i)| < |V(G)|$.
- Apply the algorithm on $(G_i,\ k,\ X_i)$.
- Either certificate that $tw(G_i) \geq k$ for some i and therefore $tw(G) \geq k$
- Or get two rooted decompositions T_1, T_2 of G_1 and G_2 with $X_i \subseteq root(T_i)$
- Add a root bag containing all vertices in $X \cup S$ (note that $|X \cup S| \leq 4k$) attached to the roots of T_1 and T_2.

- S is a good separator for X (Recall $2k+1 \leq |X| \leq 3k$ and $|S| \leq k$)
- $G \setminus S$ disconnects G into two non-empty parts V_1 and V_2 such that $X \cap V_i \leq 2|X|/3$ for $i = 1, 2$
- Define $X_i = S \cup (X \cap V_i)$. Then $|X_i| \leq k + \frac{2}{3}3k = 3k$
- Set $G_i = G[V_i \cup S]$, and apply induction on (G_i, X_i) for $i = 1, 2$
- Observe that $|V(G_i)| < |V(G)|$.
- Apply the algorithm on $(G_i,\ k,\ X_i)$.
- Either certificate that $tw(G_i) \geq k$ for some i and therefore $tw(G) \geq k$
- Or get two rooted decompositions T_1, T_2 of G_1 and G_2 with $X_i \subseteq root(T_i)$
- Add a root bag containing all vertices in $X \cup S$ (note that $|X \cup S| \leq 4k$) attached to the roots of T_1 and T_2.
- Check that it is indeed a tree decomposition of the desired width.

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,
  - X_0 is a subset of a separator of size at most k separating V_1 and V_2 where $X_i \subseteq V_i$

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,
  - X_0 is a subset of a separator of size at most k separating V_1 and V_2 where $X_i \subseteq V_i$
- Equivalently if and only if in $G \setminus X_0$, there are at most $k - |X_0|$ disjoint paths from X_1 to X_2.

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,
  - X_0 is a subset of a separator of size at most k separating V_1 and V_2 where $X_i \subseteq V_i$
- Equivalently if and only if in $G \setminus X_0$, there are at most $k - |X_0|$ disjoint paths from X_1 to X_2.
- Ford Fulkerson : $O\left(k^2 n\right)$ [1]

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,
  - X_0 is a subset of a separator of size at most k separating V_1 and V_2 where $X_i \subseteq V_i$
- Equivalently if and only if in $G \setminus X_0$, there are at most $k - |X_0|$ disjoint paths from X_1 to X_2.
- Ford Fulkerson : $O\left(k^2 n\right)$ [1]
- 3^{3k} ways of defining the partition X_0, X_1, X_2 so $O\left(27^k . k^2 n\right)$ for this step

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,
  - X_0 is a subset of a separator of size at most k separating V_1 and V_2 where $X_i \subseteq V_i$
- Equivalently if and only if in $G \setminus X_0$, there are at most $k - |X_0|$ disjoint paths from X_1 to X_2.
- Ford Fulkerson : $O\left(k^2 n\right)$ [1]
- 3^{3k} ways of defining the partition X_0, X_1, X_2 so $O\left(27^k . k^2 n\right)$ for this step

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

Computing the good separator

Here is how to compute a good separator S of size at most k wrt X:

- S exists if and only if one can partition X into three subsets X_1, X_2, X_0 such that
  - X_1 and X_2 have size at most $2|X|/3$,
  - X_0 is a subset of a separator of size at most k separating V_1 and V_2 where $X_i \subseteq V_i$
- Equivalently if and only if in $G \setminus X_0$, there are at most $k - |X_0|$ disjoint paths from X_1 to X_2.
- Ford Fulkerson : $O\left(k^2 n\right)$ [1]
- 3^{3k} ways of defining the partition X_0, X_1, X_2 so $O\left(27^k . k^2 n\right)$ for this step

So the **total complexity** for the algorithm is $O\left(27^k . k^2 n^2\right)$ since the tree decomposition has at most n nodes.

[1] FF runs a number of iterations; each iteration takes $O(n+m)$ time and either concludes that the currently found flow is maximum, or augments it by 1. Since we are interested only in situations when the maximum flow is of size at most $k+1$, we may terminate the computation after $k+2$ iterations. Moreover $m \leq kn$, otherwise $\mathrm{tw}(G) > k$

5 - Win/Win approach and planar graph problems

Vertex Cover via treewidth

Observation

If $vc(G) \leq k$, then $tw(G) \leq k$

Vertex Cover via treewidth

Observation

If $vc(G) \leq k$, then $tw(G) \leq k$

FPT algorithm for Vertex Cover:

- Run our algorithm to compute tree decomposition on (G, k).

Vertex Cover via treewidth

Observation

If $vc(G) \leq k$, then $tw(G) \leq k$

FPT algorithm for Vertex Cover:

- Run our algorithm to compute tree decomposition on (G, k).
- If it ouputs that $tw(G) \geq k$, then (G, k) is a NO-instance.

Vertex Cover via treewidth

Observation

If $vc(G) \leq k$, then $tw(G) \leq k$

FPT algorithm for Vertex Cover:

- Run our algorithm to compute tree decomposition on (G, k).
- If it ouputs that $tw(G) \geq k$, then (G, k) is a NO-instance.
- Otherwise we have a tree decomposition of width at most $4k + 3$ at hand.

Vertex Cover via treewidth

Observation

If $vc(G) \leq k$, then $tw(G) \leq k$

FPT algorithm for Vertex Cover:

- Run our algorithm to compute tree decomposition on (G, k).
- If it ouputs that $tw(G) \geq k$, then (G, k) is a NO-instance.
- Otherwise we have a tree decomposition of width at most $4k + 3$ at hand.
- Use Dynamic Programming to compute a minimum vertex cover.

Subexponential FPT algorithm for planar graphs

Grid Minor for planar graphs

We denote by $\boxplus_t$ the $t \times t$ grid.

Planar grid minor Theorem

Every planar graph G with $tw(G) \geq 9t/2$ contains $\boxplus_t$ as a minor.

Grid Minor for planar graphs

We denote by $\boxplus_t$ the $t \times t$ grid.

Planar grid minor Theorem

Every planar graph G with $tw(G) \geq 9t/2$ contains $\boxplus_t$ as a minor.
Moreover, there is a $O(n^2)$-time algorithm that, given a planar graph, either output a tree-decomposition of width $9t/2$, or constructs a $\boxplus_t$-model.

Grid Minor for planar graphs

We denote by $\boxplus_t$ the $t \times t$ grid.

Planar grid minor Theorem

Every planar graph G with $tw(G) \geq 9t/2$ contains $\boxplus_t$ as a minor.
Moreover, there is a $O(n^2)$-time algorithm that, given a planar graph, either output a tree-decomposition of width $9t/2$, or constructs a $\boxplus_t$-model.

Corollary

Let G a planar graph on n vertices. Then:

- $tw(G) \leq \frac{9}{2}\sqrt{n+1}$ *and*
- *a tree decomposition of width $\frac{9}{2}\sqrt{n+1}$ can be constructed in $O(n^2)$ time.*

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.
- So, by the Planar Grid Minor Theorem, if $vc(G) \leq k$, then $tw(G) \leq \frac{9}{2}\sqrt{2k+2}$.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.
- So, by the Planar Grid Minor Theorem, if $vc(G) \leq k$, then $tw(G) \leq \frac{9}{2}\sqrt{2k+2}$.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.
- So, by the Planar Grid Minor Theorem, if $vc(G) \leq k$, then $tw(G) \leq \frac{9}{2}\sqrt{2k+2}$.

We now have the following algorithm:

- In $O(n^2)$, we get either a $\boxplus_{\sqrt{2k+2}}$-model, and in this cas we output NO.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.
- So, by the Planar Grid Minor Theorem, if $vc(G) \leq k$, then $tw(G) \leq \frac{9}{2}\sqrt{2k+2}$.

We now have the following algorithm:

- In $O(n^2)$, we get either a $\boxplus_{\sqrt{2k+2}}$-model, and in this cas we output NO.
- Or we get a tree decomposition of width at most $\frac{9}{2}\sqrt{2k+2}$.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.
- So, by the Planar Grid Minor Theorem, if $vc(G) \leq k$, then $tw(G) \leq \frac{9}{2}\sqrt{2k+2}$.

We now have the following algorithm:

- In $O(n^2)$, we get either a $\boxplus_{\sqrt{2k+2}}$-model, and in this cas we output NO.
- Or we get a tree decomposition of width at most $\frac{9}{2}\sqrt{2k+2}$.
- In the later case, we use dynamic programming to computer the minimum vertex cover in time $2^{\sqrt{2k+2}} \cdot k^{0(1)} \cdot n$.

We want to solve k-Vertex Cover for an instance (G, k) where G is planar.

- Observe that $vc(\boxplus_t) = \lceil \frac{t^2}{2} \rceil$ (because it has a matching of size $\lceil \frac{t^2}{2} \rceil$).
- So if G contains $\boxplus_t$ as a minor for some $t \geq \sqrt{2k+2}$, it has no vertex cover of size k.
- So, by the Planar Grid Minor Theorem, if $vc(G) \leq k$, then $tw(G) \leq \frac{9}{2}\sqrt{2k+2}$.

We now have the following algorithm:

- In $O(n^2)$, we get either a $\boxplus_{\sqrt{2k+2}}$-model, and in this cas we output NO.
- Or we get a tree decomposition of width at most $\frac{9}{2}\sqrt{2k+2}$.
- In the later case, we use dynamic programming to computer the minimum vertex cover in time $2^{\sqrt{2k+2}} \cdot k^{0(1)} \cdot n$.
- In total, we get an algorithm in $2^{0(\sqrt{k})} \cdot n \cdot O(n^2)$.

Subexponential parameterized algorithm

Any problem satisfying the following properties has a subexponential time FPT algorithm:

- The size of a solution in $\boxplus_k$ is of order $\Omega(k^2)$.
- Given a tree decomposition of width $O(k)$, the problem can be solved in time $O(2^k) \cdot n^{0(1)}$.
- If G has a solution of size at most k, then every minor of G too.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

- A dominating set of $\boxplus_k$ has size at least $\frac{k^2}{4}$.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

- A dominating set of $\boxplus_k$ has size at least $\frac{k^2}{4}$.
- Given a tree decomposition of width $O(k)$, we can compute a minimum dominating set in time $O(2^{O(k)}) \cdot n^{0(1)}$.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

- A dominating set of $\boxplus_k$ has size at least $\frac{k^2}{4}$.
- Given a tree decomposition of width $O(k)$, we can compute a minimum dominating set in time $O(2^{O(k)}) \cdot n^{0(1)}$.
- But it might be that G has a smaller dominating set then one of its minor.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

- A dominating set of $\boxplus_k$ has size at least $\frac{k^2}{4}$.
- Given a tree decomposition of width $O(k)$, we can compute a minimum dominating set in time $O(2^{O(k)}) \cdot n^{0(1)}$.
- But it might be that G has a smaller dominating set then one of its minor.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

- A dominating set of $\boxplus_k$ has size at least $\frac{k^2}{4}$.
- Given a tree decomposition of width $O(k)$, we can compute a minimum dominating set in time $O(2^{O(k)}) \cdot n^{0(1)}$.
- But it might be that G has a smaller dominating set then one of its minor.

Indeed, deleting a vertex (that sees every other vertices for example), might increase a lot the size of a smallest dominating set.

Dominating set

A vertex set S of a graph G is a dominating set if $S \cup N(S) = V(G)$.
In other words, every vertex has a neighbour in S or is in S.

Question: Does the above strategy works?

- A dominating set of $\boxplus_k$ has size at least $\frac{k^2}{4}$.
- Given a tree decomposition of width $O(k)$, we can compute a minimum dominating set in time $O(2^{O(k)}) \cdot n^{0(1)}$.
- But it might be that G has a smaller dominating set then one of its minor.

Indeed, deleting a vertex (that sees every other vertices for example), might increase a lot the size of a smallest dominating set.

Solution: Observe that contracting an edge can only decrease the size of a smallest dominating set!

Planar grid minor theorem for edge contraction

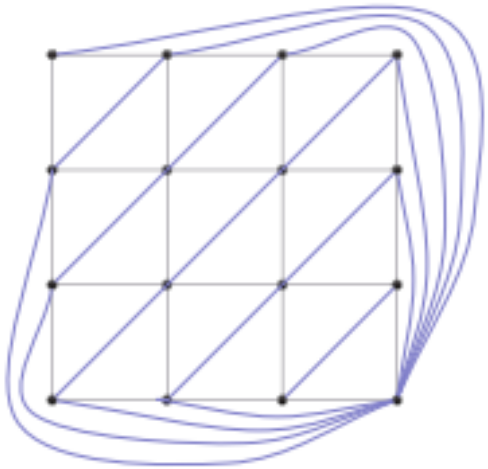
Given two graphs G and H, we say that G contains H as a contraction, if H can be obtained from G by contracting some edges.

Planar grid minor theorem for edge contraction

Given two graphs G and H, we say that G contains H as a contraction, if H can be obtained from G by contracting some edges.

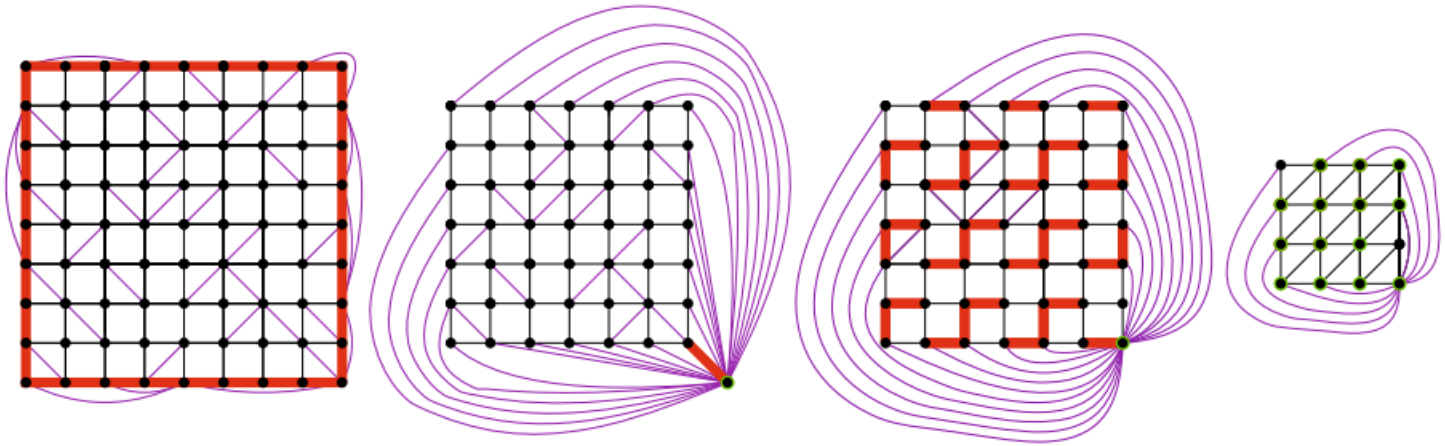
Planar grid minor theorem for edge contraction

Let G be a planar graph. If $tw(G) \geq 9t + 5$, then G contain Γ_t as a contraction. Moreover, there is an algorithm running in time $O(n^2)$ that either output a tree decomposition of width $9t + 5$, or output a set of edges whose contraction resuts in Γ_t.



Proof:

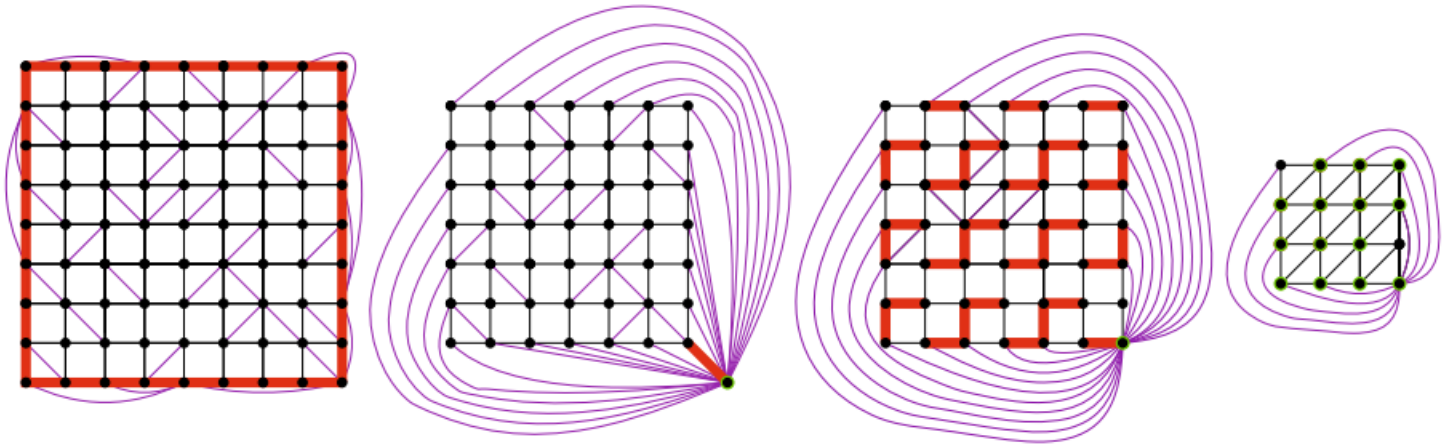
- If $tw(G) \geq 9t + 5$, then G contain a $\boxplus_{2k+1}$ model.



- Finally, the obtained Γ_k has no extra edge, since adding an edge to Γ_t spoils its planarity.

Proof:

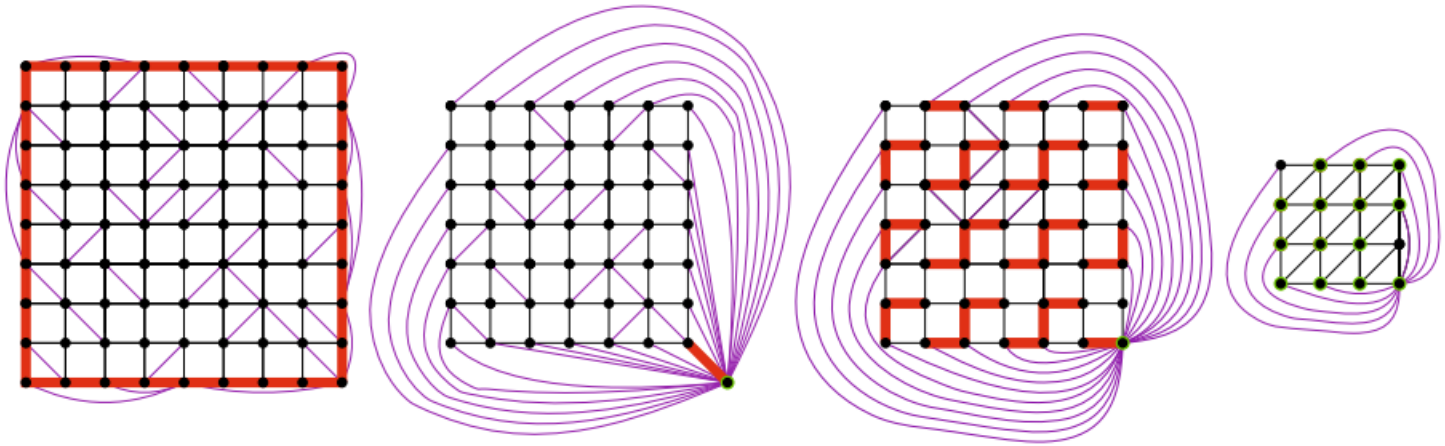
- If $tw(G) \geq 9t + 5$, then G contain a $\boxplus_{2k+1}$ model.
- i.e. after a sequence of vertex deletion, edge deletion and edge contraction.



- Finally, the obtained Γ_k has no extra edge, since adding an edge to Γ_t spoils its planarity.

Proof:

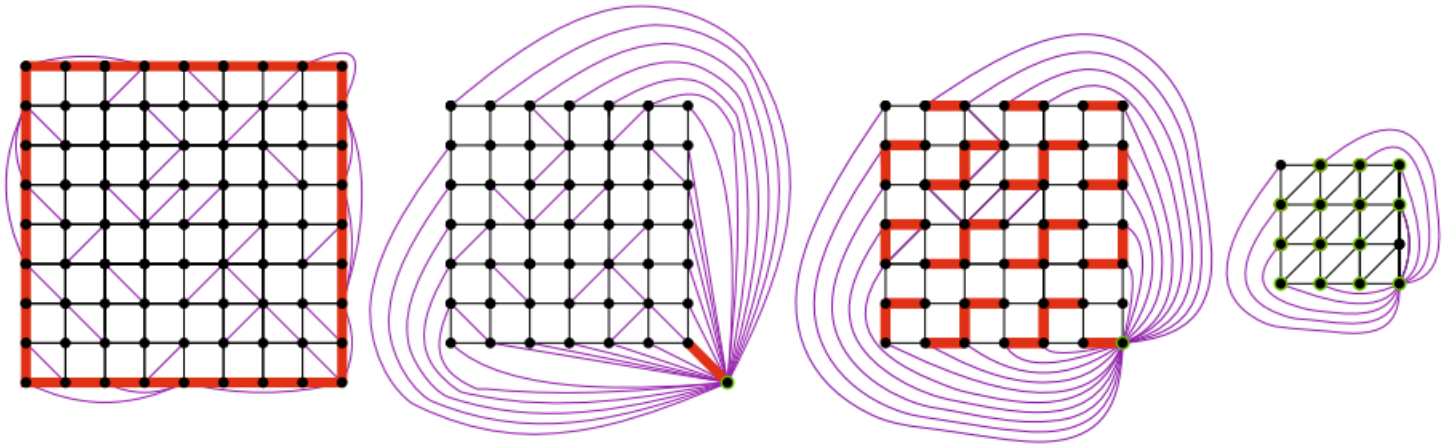
- If $tw(G) \geq 9t + 5$, then G contain a $\boxplus_{2k+1}$ model.
- i.e. after a sequence of vertex deletion, edge deletion and edge contraction.
- Instead of deleting the vertices, contract them with one of their neighbor and omit edge deletion.



- Finally, the obtained Γ_k has no extra edge, since adding an edge to Γ_t spoils its planarity.

Proof:

- If $tw(G) \geq 9t + 5$, then G contain a $\boxplus_{2k+1}$ model.
- i.e. after a sequence of vertex deletion, edge deletion and edge contraction.
- Instead of deleting the vertices, contract them with one of their neighbor and omit edge deletion.
- This way we get $\boxplus_{2k+1}$ plus some edges.



- Finally, the obtained Γ_k has no extra edge, since adding an edge to Γ_t spoils its planarity.

Subexponential parameterized algorithm for dominating set

Observation

A minimum dominating set of Γ_k has size $\Omega(k^2)$.

So the strategy works again, and we get a subexponential FPT time algorithm for dominating set in planar graphs.

Subexponential parameterized algorithm for dominating set

Observation

A minimum dominating set of Γ_k has size $\Omega(k^2)$.

So the strategy works again, and we get a subexponential FPT time algorithm for dominating set in planar graphs.

General strategy:

- The size of a solution in $\boxplus_k$ is of order $\Omega(k^2)$.
- Given a tree decomposition of width $O(k)$, the problem can be solved in time $O(2^k) \cdot n^{0(1)}$.
- Contracting edges can only decrease the size of the solution.