FPT ALGORITHMS, PART IV:

TREE DECOMPOSITIONS

Contents:

- **Definitions, basic properties, finding tree decompositions**
- Algorithms: dynamic programming / monadic second order logic
- Tree width reduction algorithms and planar graphs

Definitions

- A *tree decomposition* of a graph G is a 2-tuple (T, X) where T is a tree and $X = \{X_v : v \in V(T)\}$ is a set of subsets of $V(G)$ such that:
(a) For every $xy \in E(G)$, there is a $v \in V(T)$ with $\{x, y\} \subseteq X_v$.
(b) For every $x \in V(G)$, the subgraph of T induced by $X^{-1}(x) = \{v \in V(T) : x \in X_v\}$ is non-empty and connected.

- Different formulation of (b): for every $u, v \in V(T)$ and every vertex w on the path between u and v, $X_u \cap X_v \subseteq X_w$.
(And every vertex of G appears in at least one X_v.)

- To distinguish between vertices of G and T, the vertices of T are called *nodes*.
- The sets X_v are called the *bags* of the tree decomposition.

- The *width* of a tree decomposition (T, X) is $\max_{v \in V(T)} |X_v| - 1$.
The *tree width* $tw(G)$ of a graph G is the minimum width over all tree decompositions of G.

Observation

$tw(G) = 0$ if and only if $E(G) = \emptyset$.

Proposition

If G is a forest with at least one edge, then $tw(G) = 1$.

Proof: $\mathrm{tw}(G) \geq 1$ by the above observation.
If G is a tree then let T be obtained from G by subdividing every edge $uv \in E(G)$ with a new vertex w_{uv}.

Set $X_u = \{u\}$ for all $u \in V(G)$, and $X_{w_{uv}} = \{u, v\}$ for every $uv \in E(G)$.

It is easily seen that this is a tree decomposition of G with width 1. A tree decomposition of a forest can be obtained by joining the tree decompositions for the components together by adding arbitrary edges. $\square$

Small tree decompositions

- A tree decomposition (T, X) is *small* if for distinct $u, v \in V(T)$, $X_u \not\subseteq X_v$ and $X_v \not\subseteq X_u$.

Proposition

When given a tree decomposition of G, in polynomial time we can construct a small tree decomposition of G with the same width.

Proof: Let (T, X) be a tree decomposition of G with $X_u \subseteq X_v$ for $u, v \in V(T)$.

By considering a path from u to v we can find adjacent nodes with this property, so w.l.o.g. $uv \in E(T)$.

Then contracting uv into a new node w with $X_w = X_v$ gives a smaller tree decomposition of G, continue until a small tree decomposition is obtained. $\square$

Proposition

Let (T, X) be a small tree decomposition of G. Then $|V(T)| \leq |V(G)|$.

Proof: By induction over $n = |V(G)|$. If $n = 1$ then $|V(T)| = 1$.

If $n \geq 2$ then consider a leaf u of T with neighbor v. Deleting u from T yields a small tree decomposition (T', X') of $G' = G - (X_u \setminus X_v)$.

Since $X_u \setminus X_v \neq \emptyset$, by induction $|V(T)| = |V(T')| + 1 \leq |V(G')| + 1 \leq |V(G)|$. $\square$

Minors

Observation

Let H be obtained from G by contracting an edge xy into z. Then $tw(H) \leq tw(G)$.

Proof: In a tree decomposition (T, X) of G, insert z in every bag containing x or y, and then remove x and y from every bag, to obtain (T, X').

This does not increase the bag size, and is a tree decomposition of H:
(a) Edges za are contained in the bag that previously contained xa or ya.
(b) Since $xy \in E(G)$, $X^{-1}(x) \cap X^{-1}(y) \neq \emptyset$. Since $X^{-1}(x)$ and $X^{-1}(y)$ induce connected subgraphs of T and are not disjoint, $X'^{-1}(z) = X^{-1}(x) \cup X^{-1}(y)$ induces a connected subgraph. $\square$

Observation

Let H be a subgraph of G. Then $tw(H) \leq tw(G)$.

• H is a *minor* of a graph G if it can be obtained from G by iteratively deleting and contracting edges.

Corollary

If H is a minor of G, then $tw(H) \leq tw(G)$.

Edges of tree decompositions correspond to cuts

- If (T, X) is a tree decomposition and $U \subseteq V(T)$, then *$X(U)$*$= \cup_{v \in U} X_v$.

- For a connected graph $G = (V, E)$, $C \subset V$ is a *cut* if $G - C$ is disconnected. It is a *k-cut* if $|C| \leq k$.
For disjoint non-empty $S, T \subset V(G)$, it is an *(S, T)-cut* or cut *separating S and T* if $G - C$ contains no paths with end vertices in both S and T.

Lemma

Let (T, X) be a tree decomposition of $G = (V, E)$ with $uv \in E(T)$, and let T_u and T_v be the vertex sets of the two components of $T - uv$. ($u \in T_u$.)
Let $C = X_u \cap X_v$, and let $S_u = X(T_u) \setminus X(T_v)$ and $S_v = X(T_v) \setminus X(T_u)$ be non-empty.
Then C is an (S_u, S_v)-cut in G.

Lemma

Let (T, X) be a tree decomposition of $G = (V, E)$ with $uv \in E(T)$, and let T_u and T_v be the vertex sets of the two components of $T - uv$. ($u \in T_u$.)
Let $C = X_u \cap X_v$, and let $S_u = X(T_u) \setminus X(T_v)$ and $S_v = X(T_v) \setminus X(T_u)$ be non-empty.
Then C is an (S_u, S_v)-cut in G.

Proof: $C = X_u \cap X_v = X(T_u) \cap X(T_v)$ (2nd property).

So $\{S_u, C, S_v\}$ is a partition of $V(G)$, therefore it suffices to show that $G - C$ contains no edge xy with $x \in S_u$ and $y \in S_v$.

For every edge $xy \in E(G - C)$, there is a $w \in V(T)$ with $\{x, y\} \subseteq X_w$ (1st property).

If $w \in T_u$ then $x, y \in X(T_u)$ so $x, y \notin S_v$.
If $w \in T_v$ then $x, y \in X(T_v)$ so $x, y \notin S_u$. $\square$

Lemma

Let G be a connected graph with $tw(G) = k$. Then $|V(G)| = k + 1$, or G has a k-cut.

Proof: Consider a *small* tree decomposition (T, X) of G of width k.

If $|V(G)| > k + 1$, then $|V(T)| \geq 2$, so we may consider any two adjacent nodes $u, v \in V(T)$.
Since (T, X) is small, $X_u \setminus X_v \neq \emptyset$ and $X_v \setminus X_u \neq \emptyset$, and $|X_u \cap X_v| \leq k$.
Then by the previous lemma, $C = X_u \cap X_v$ is a k-cut in G. $\square$

Corollary

If $tw(G) = 1$, then G is a forest.

Proof: A cycle C contains no 1-cut, so has $\text{tw}(C) \geq 2$. Therefore G contains no cycle subgraphs. $\square$

Corollary

For K_n, the complete graph on n vertices, $tw(K_n) = n - 1$.

- Let $[k] = \{1, \ldots, k\}$.

- The *$k \times l$-grid* $G_{k\times l}$ is the graph (V, E) with

- $V = \{(i,j) : i \in [k], j \in [l]\}$
- $E = \{(i,j)(i,j+1) : i \in [k], j \in [l-1]\} \cup$
  $\{(i,j)(i+1,j) : i \in [k-1], j \in [l]\}$.

Proposition

$tw(G_{k\times l}) \leq \min\{k, l\}$.

Proof: Exercise. $\square$

Proposition

$tw(G_{k\times l}) \geq \min\{k, l\}$.

Computing tree width

- Deciding whether $\mathrm{tw}(G) \leq k$ is NP-hard.

k-Tree Width
INSTANCE: A graph G on n vertices and integer k.
PARAMETER: k
QUESTION: Is $\mathrm{tw}(G) \leq k$?

Theorem (Bodlaender)

An FPT algorithm exists for k-Tree Width, with complexity $f(k) \cdot O(n)$.

FPT ALGORITHMS, PART IV:

TREE DECOMPOSITIONS

Contents:

- Definitions, basic properties, finding tree decompositions
- **Algorithms: dynamic programming / monadic second order logic**
- Tree width reduction algorithms and planar graphs

Dynamic programming over tree decompositions

Let (T, X) be a tree decomposition of G, of width k.
• Make T into a *rooted* (directed) tree by choosing a root $r \in V(T)$, and replacing edges by arcs in such a way that every vertex is reachable from r.

• For $v \in V(T)$, $R_T(v)$ denotes the vertices that are reachable from v, including v itself.

• For $v \in V(T)$, $V(v) = X(R_T(v))$, and $G(v) = G[V(v)]$.

Vertex k-colorability

- Recall: A proper k-vertex coloring or *k-coloring* of a graph $G = (V, E)$ is a function $\alpha : V \rightarrow \{1, \ldots, k\}$ such that for all $uv \in E$, $\alpha(u) \neq \alpha(v)$.

- Let H_1 and H_2 be two subgraphs of G, with k-colorings α_1 and α_2 respectively. α_2 is *α_1-compatible* if for all $v \in V(H_1) \cap V(H_2)$, $\alpha_1(v) = \alpha_2(v)$.

Let (T, X) be a rooted tree decomposition of G.

- For every $v \in V(T)$ and every proper k-coloring α of $G[X_v]$, we define $P_v(\alpha) = 1$ if $G(v)$ has an α-compatible k-coloring β, and $P_v(\alpha) = 0$ otherwise.

Proposition

$P_u(\alpha) = 1$ if and only if for all children w of u, there is an α-compatible coloring β of $G[X_v]$ with $P_v(\beta) = 1$.

Proof: $\Rightarrow$: Let γ be an α-compatible coloring of $G(u)$.
$G(v)$ is a subgraph of $G(u)$, so restricting γ to X_v gives the desired coloring β.

$\Leftarrow$: Consider two children v and w of u, and suppose they have α-compatible colorings β and γ respectively.
Since (T, X) is a tree decomposition, $V(v) \cap V(w) \subseteq X_u$, so β is γ-compatible.
Combining β and γ now gives $\delta : V(u) \to \{1, \ldots, k\}$.
Since (T, X) is a tree decomposition, there are no edges $xy \in E(G)$ with $x \in V(v) \setminus X_u$ and $y \in V(w) \setminus X_u$, so δ is a proper k-coloring of $G(u)$.
The same can be done for all children of u simultaneously. $\square$

Conclusion:

Theorem

Let (T, X) be a small tree decomposition of G of width w.
In time $k^{w+1} \cdot n^{O(1)}$ we can decide if G is k-colorable.

Proof: For every $v \in V(T)$ and every k-coloring α of $G[X_v]$, we compute $P_v(\alpha)$: start at the leaves of T, and use the previous proposition for the other nodes, in the right order.

$G = G(r)$ is k-colorable iff $P_r(\alpha) = 1$ for some α.

By storing some more data, testing whether α is a $G[X_v]$ coloring and computing $P_v(\alpha)$ can be done in polynomial time $n^{O(1)}$, so the total complexity is mainly determined by the number of candidates for α, which is $k^{|X_v|}$.
Complexity: $|V(T)| \cdot k^{w+1} \cdot n^{O(1)} = k^{w+1} \cdot n^{O(1)}$. $\square$

Parameterizing by tree width

tw-k-Colorability?
INSTANCE: A graph G and integer k.
PARAMETER: $\mathrm{tw}(G) + k$
QUESTION: Is G k-colorable?

PROBLEM: $\mathrm{tw}(G)$ cannot be computed in polynomial time (unless P=NP).

SOLUTION 1:
tw-k-Colorability (1)
INSTANCE: A graph G, a tree decomposition (T, X) of G of width w, and integer k.
PARAMETER: $w + k$.
QUESTION: Is G k-colorable?

SOLUTION 2:
tw-k-Colorability (2)
INSTANCE: A graph G and integers w and k.
PARAMETER: $w + k$
QUESTION: Is $\mathrm{tw}(G) \leq w$ and is G k-colorable?

- Since an FPT algorithm for deciding $\mathrm{tw}(G) \leq w$ exists, both problems above allow FPT algorithms.

In fact, the following problem allows an FPT algorithm as well (exercise):

tw-k-Colorability (3)
INSTANCE: A graph G and integers w and k.
PARAMETER: w
QUESTION: Is $\mathrm{tw}(G) \leq w$ and is G k-colorable?

- Nevertheless, from now on we will simply choose $\mathrm{tw}(G)$ as parameter: this should be interpreted as problem variant (1) or (2).

Nice tree decompositions

- For more complicated dynamic programming algorithms, *nice* tree decompositions make life much easier:

- A *rooted* tree decomposition $(T, X), r$ of G is *nice* if for every $u \in V(T)$:
- $|X_u| = 1$ *(leaf)*, or
- u has one child v with $X_u \subset X_v$ and $|X_u| = |X_v| - 1$ *(forget)*, or
- u has one child v with $X_v \subset X_u$ and $|X_u| = |X_v| + 1$ *(introduce)*, or
- u has two children v and w with $X_u = X_v = X_w$ *(join)*.

Lemma

When given a tree decomposition of width w of G, in polynomial time we can construct a nice tree decomposition (T', X') of G of width w, with $|V(T)| \in O(wn)$, where $n = |V(G)|$.

Proof: Exercise. $\square$

tw-Vertex Cover

tw-Vertex Cover
INSTANCE: A graph G.
PARAMETER: tw(G) (!)
TASK: Compute the size of a *minimum* vertex cover of G.

Let (T, X) be a nice tree decomposition of G.

- For all $v \in V(T)$ and all $C \subseteq X_v$, let $s_v(C)$ be the minimum size of a vertex cover C' of $G(v)$ with $C' \cap X_v = C$ if such a C' exists, and $s_v(C) = \infty$ otherwise.

Proposition (Leaf)

Let u be a leaf *of T with $X_u = \{x\}$. Then $s_u(\{x\}) = 1$ and $s_u(\emptyset) = 0$.*

Proposition (Forget)

Let u be a forget node *of T with child v and $X_v \setminus X_u = \{x\}$. Then for all $C \subseteq X_u$, $s_u(C) = \min\{s_v(C), s_v(C + x)\}$.*

Proof:

$\geq$: Let C' be a minimum vertex cover of $G(u) = G(v)$ with $C' \cap X_u = C$.

If $x \notin C'$ then $C' \cap X_v = C$ so $|C'| \geq s_v(C)$.

If $x \in C'$ then similarly $|C'| \geq s_v(C + x)$.

$\leq$: let C_1 and C_2 be the vertex covers that determine $s_v(C)$ and $s_v(C + x)$ respectively. Both are vertex covers of $G(u)$ compatible with C, so $s_v(C) \leq \min\{|C_1|, |C_2|\}$. $\square$

Proposition (Introduce)

Let u be an introduce *node of T with child v and $X_u \setminus X_v = \{x\}$.*
Then for all $C \subseteq X_u$:
(1) If C is not a vertex cover of $G[X_u]$ then $s_u(C) = \infty$.
(2) If C is a vertex cover of $G[X_u]$ and $x \in C$ then
$s_u(C) = s_v(C - x) + 1$.
(3) If C is a vertex cover of $G[X_u]$ and $x \notin C$ then $s_u(C) = s_v(C)$.

Proof sketch: (1) and (2) are straightforward.

(3) All neighbors of x in $G(u)$ are in X_v and therefore in C since C is a vertex cover of $G[X_u]$. $\square$

Proposition (Join)

Let u be a join *node of T with children v and w.*
Then for all $C \subseteq X_u$: $s_u(C) = s_v(C) + s_w(C) - |C|$.

Proof sketch: $\geq$: If C' is a vertex cover of $G(u)$ with $C' \cap X_u = C$, then $C' \cap V(v)$ is a vertex cover of $G(v)$ and $C' \cap V(w)$ is a vertex cover of $G(w)$, which share $|C|$ vertices.

$\leq$: Two C-compatible vertex covers of $G(v)$ and $G(w)$ of size $s_v(C)$ and $s_w(C)$ can be combined to a vertex cover of $G(u)$ of size $s_v(C) + s_w(C) - |C|$. $\square$

Theorem

Let (T, X) be a tree decomposition of width w of a graph G on n vertices. In time $2^{w+1} \cdot n^{O(1)}$ the size of a minimum vertex cover of G can be computed.

Proof: In polynomial time, make (T, X) into a nice tree decomposition (T', X') of width w, with $|V(T')| \in O(wn)$.
Using the above propositions, we can compute $s_v(C)$ for all $v \in V(T')$ and $C \subseteq X'_v$.
For every such v and C this takes time $n^{O(1)}$.
So the total complexity is $|V(T')| \cdot 2^{w+1} \cdot n^{O(1)} = 2^{w+1} \cdot n^{O(1)}$.

The size of a minimum vertex cover of G is $\min_{C \subseteq X_r} s_r(C)$. □

- We can *construct* a minimum vertex cover as well, by tracing back through the tree decomposition.

- By using the proper data structures, a *linear* time complexity $O^*(2^w) \cdot O(n)$ can also be proved.

A sketch of more advanced dynamic programming algorithms

Suppose we wish to find a minimum feedback vertex set (FVS) of G, when a tree decomposition (T, X) of width w is given.

• For every $v \in V(T)$, every $C \subseteq X_v$, and every partition P of $X_v \setminus C$, $F_v(C, P)$ denotes the size of a minimum FVS S of $G(v)$ with

• $S \cap X_v = C$ and

• for all $x, y \in X_v \setminus C$: x and y lie in the same component of $G(v) - S$ iff x and y are part of the same set in P,

if such a FVS exists, and $F_v(C, P) = \infty$ otherwise.

• One can show how to compute $F_v(C, P)$ for leaves, forget, introduce and join nodes.

• By considering all C and P for $F_r(C, P)$, the minimum FVS size can be found.

Dynamic programming over tree decompositions - Summary

- The majority of NP-hard problems can be solved in polynomial time on graphs of bounded treewidth!

- Using nice tree decompositions helps a lot, at almost no cost.

- The main challenge is finding out which information to store for tree nodes; when this is done, proving formulas for forget, introduce and join nodes is tedious but mostly straightforward.

- Another research subject is finding faster dynamic programming strategies: sometimes it may be possible to store and compute fewer combinations.

- If the goal is just to determine whether a problem can be solved efficiently for bounded tree width, giving a dynamic programming strategy is often not needed: use *monadic second order logic*.