Introduction to graph minors and treewidth

(Lecture notes)

Jiří Fiala

fiala@kam.mff.cuni.cz

Contents

1 Introduction

In this short series of lectures we would like to explore some relationship between the notions of *graph minors* and *treewidth*. After this introductory course, the student should be familiar with these notions, and should be capable to apply these notions in the design of graph algorithms.

Some hard theorems in this course will be provided without proofs (the complete proof of Robertson-Seymour theorem has been given in more than twenty papers). In simpler cases proofs will be given, to distinguish hard and easy situations in this theory.

Students are welcome to solve exercises at the end of each lecture. Many exercised are taken from [14]. Some very easy "single thought" exercises are inserted into the text for better understanding of the subject.

The text is given it the first version. The author will be glad for all comments and suggestions.

I would like to express my thanks to Manuela Montangero and Martin Mareš, who carefully read the manuscript and suggested several improvements and corrections.

1.1 Basics definitions and properties of graph minors

If not specified otherwise, all graphs we consider are finite, undirected, simple and loopless. It means that a graph G is given as a pair (V, E), where V is a finite set of vertices and E is a set of unordered pairs which we call edges. Formally: $G = (V, E)$, $E \subseteq \binom{V}{2}$. We use the notation (u, v) to denote the undirected edges, while directed edges will be written as $[u, v]$.

A subgraph $H = (V_H, E_H)$ of a graph $G = (V_G, E_G)$, is a graph[1] that satisfies $V_H \subseteq V_G$ and $E_H \subseteq \binom{V_H}{2} \cap E_G$. If $E_H = \binom{V_H}{2} \cap E_G$, we say that H is a subgraph of G induced by the set V_H, or simply an induced subgraph. In such a case we write $H = G|_{V(H)}$.

Let G be a graph G, $w \in V_G$ be a vertex of G and $e = (u, v) \in E_G$ be an edge. We define the following operations:

- The *deletion* of the vertex w results into the graph $G - w$ which is the subgraph of G induced by the set $V_G \setminus \{w\}$.

[1] More precisely we would say that H is isomorphic to such graph. For simplicity we will skip this formalism. Similarly, we will not deal in details with the empty graph $(\emptyset, \emptyset)$ as the universal subgraph.

- The *deletion* of the edge e gives the graph $G - e = (V_G, E_G \setminus \{e\})$.
- The *contraction* of e provides the graph $G \circ e$ with the vertex set $V(G \circ e) = V_G \setminus \{v\}$ and edges $E(G \circ e) = E_G \setminus \{e : v \in e\} \cup \{(u, v') : (v, v') \in E_G, v' \neq u\}$.
- The *subdivision* of e yields the graph $G \cdot e$, where $V(G \cdot e) = V_G \cup \{v' : v' \notin V_G\}$ and $E(G \cdot e) = E_G \setminus \{e\} \cup \{(u, v'), (v, v')\}$.

Now we have enough power to give the first nontrivial definition:

Definition: *A graph H is an* minor *of a graph G if H can be obtained from a subgraph of G by a sequence of edge contractions. If H was derived from an induced subgraph of G we call H an* induced minor.

Definition: *A class of graphs $\mathcal{G}$ is called (induced) minor closed, if it contains with every $G \in \mathcal{G}$ all its (induced) minors.*

We illustrate these notions on standard graph examples. Let $V = \{u_1, \ldots, u_n\}$ be the vertex set. We define:

- the empty graph $E_n = (V, \emptyset)$, $n \geq 0$
- the complete graph (clique) $K_n = (V, \binom{V}{2})$, $n \geq 1$,
- the path $P_n = (V, \{(u_i, u_{i+1}), i = 1, \ldots, n-1\})$, $n \geq 2$,
- the cycle $C_n = (V, E(P_n) \cup \{(u_1, u_n)\})$, $n \geq 3$,

Exercise 1: Decide whether the class of all empty graphs, cliques, paths or cycles is (induced) minor-closed.

Solution: Empty graphs are minor closed and induced minor closed. Cliques are induced minor closed only. Paths and cycles are not closed under taking (induced) minors.

Exercise 2: Find the minimum (inclusion-wise) minor closed class of graphs that contains all paths, and all cycles, resp.

Solution: Disjoint union of paths, and graphs of maximum degree 2, resp.

Two vertices $u, v \in V_G$ are connected in G if either $u = v$ or G has a path as a subgraph that contains both u and v. A graph G is called connected if every two vertices u, v are connected. Every inclusion-maximal subset of vertices that are mutually connected is called a component.

A forest is a graph which has no cycle. A connected forest is called a tree.

Exercise 3: Are connected graphs, or trees or forest (induced)-minor closed? Which subclass of connected graph is (induced)-minor closed?

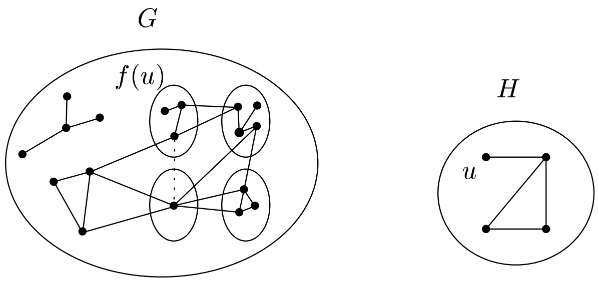


Figure 1: An illustration for Lemma 1.1. The dotted line should not be present when H is an induced minor of G.

Solution: Only forests are minor closed and induced minor closed. The class of cliques is the only infinite class of graphs which is closed under taking induced minors and contains only connected graphs. Each minor closed class of graphs which contains a graph on at least two vertices, contains also a graph on two isolated vertices.

Let's have a look at what a contraction operation makes in G:

Lemma 1.1 *If H is an (induced) minor of G then there exist a mapping f assigning a subsets of vertices of G to vertices of H, such that:*

- *For every $u \in V_H$, the graph $G|_{f(u)}$ is connected.*
- *For every $u, v \in V_H$, sets $f(u)$ and $f(v)$ are disjoint.*
- *If (and only if) there is an edge between $u, v \in V_H$, then there exists an edge $(u', v') : u \in f(u), v \in f(v)$.*

Exercise 4: Prove Lemma 1.1.

Solution: Proof: Assume that a vertex w of H is the result of contracting edges $e_1, \ldots, e_k$ of G. Then these edges form a connected subgraph of G, and these subgraphs are vertex-disjoint for different vertices w. Then set $f(w)$ is defined as the set of vertices of $e_1, \ldots, e_k$.

The third property follows from the following: If $(u, v) \in V_H$, then the same edge in G connects some vertex u' such that some path connecting u and u' was contracted onto u. A similar argument holds for v as well. $\square$

We further link the notion of minor with the subdivision operation:

Definition: *Call a graph F a* subdivision *(or also Topological minor) of H if F can be obtained by a series of subdivision operation on edges of the graph H.*

Lemma 1.2 *If a subdivision of H is an (induced) subgraph of G, then the graph H is an (induced) minor of G.*

Proof: Let H' be a subdivision of H that is an (induced) subgraph of G. It is enough to show that a series of contractions on H' gives H. If an edge e of H was subdivided into new edges e_1 and e_2, then the contraction either of e_1 or of e_2 in H' yields the edge e. $\square$

Lemma 1.3 ([11]) *If H is of maximum degree at most three then H is a minor of G if and only if G contains a subdivision of H as a subgraph.*

Exercise 5: Prove Lemma 1.3 by use of Lemma 1.1.

Solution: Proof: Only $\Rightarrow$ implication is sufficient to prove, the rest has been already proved in a more general case in Lemma 1.2. Let G' be the subgraph of G whose contraction is isomorphic to H. Find a mapping f from Lemma 1.1. From each $f(u)$ at most three edges links $f(u)$ to the rest of the graph G'. Denote by u_1, u_2 and u_3 the end vertices of these edges (if exist). Find the minimal connected subgraph of $G'|_{f(u)}$ containing all of them. This graph is a tree of maximum degree three and selection of such trees for every $u \in V_H$, together with edges in G' corresponding to E_H by Lemma 1.1, creates a subdivision of H. $\square$

Exercise 6: Show that the degree condition is necessary.

Solution: A star of four rays $K_{1,4}$ can be obtained by a contraction from the unique tree on six vertices with score $(1, 1, 1, 1, 3, 3)$.

Exercise 7: Show that Lemma 1.3 does not hold for induced minors.

Solution: E.g. a complete graph has no induced C_4 minor but contains many C_4 as a subgraph. If we require the subdivision to be an induced subgraph, then the statement is valid by the same argument.

Exercise 8: Which degree-bounded graphs are minor closed?

Solution: Only for $\Delta \leq 2$.

Planar graphs are graphs that can be drawn in the Euclidean plane such that vertices are represented by points and edges are drawn as curves connecting adjacent vertices which intersects only at their endpoints.

Exercise 9: Are planar graphs (induced) minor closed?

Solution: Planar graphs are minor closed, therefore are also induced minor closed.

The class of planar is well characterized by the celebrated Kuratowski theorem [9]

Theorem 1.4 *A graph is planar if and only if it contains no subdivision of K_5 nor of $K_{3,3}$ as a subgraph.*

We postpone the proof because of its complexity.

Exercise 10: Find a similar characterization theorem for the class of forests.

Solution: A graph is a forest iff it contains no C_3 minor.

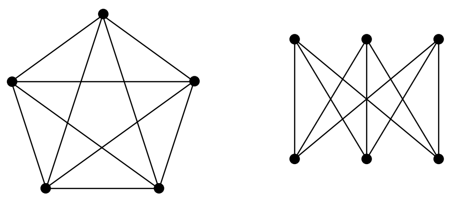


Figure 2: Forbidden minors for planar graphs.

We see that some particular minor-closed classes allow a good characterization by *forbidden subdivisions*. Even this characterization could be expressed in terms of minors, i.e. G is planar if none of K_5 and $K_{3,3}$ is its minor. The case of $K_{3,3}$ follows from Lemma 1.3.

Exercise 11: Show that if G has a K_5 minor, then it contains a subdivision of K_5 or $K_{3,3}$.

Solution: If G has K_5 minor but it contains no K_5 subdivision, then before the last contraction we can find a $K_{3,3}$ minor and hence also a $K_{3,3}$ subdivision.

To see that the existence of such characterization is a general principle, we let for the next lecture.

Exercises

Exercise 12: Find forbidden minors for outerplanar graphs.

Solution: The two graphs K_4 and $K_{2,3}$ are forbidden for outerplanar graphs.

They are minor-minimal with this property. The dual graph induced by the finite faces is a forest iff the original graph is outerplanar. If it is not a forest, then it either contains a cycle, which yield a K_4 minor or it contains a multiple edge, which yield a $K_{2,3}$ minor.

Almost-k-trees are subgraphs of connected graphs on n vertices with at most $n + k - 1$ edges.

Exercise 13: Show that forest are almost-0-trees.

Solution: Forest are subgraphs of trees. Trees have $|V| - 1$ edges and are connected.

Exercise 14: Show that unicyclic graphs (graph with at most one cycle) are almost-1-trees.

Solution: A connected graph with $|E| = |V|$ contains exactly one cycle. Its subgraphs have also at most one cycle.

Exercise 15: Are unicyclic graphs and almost-k-trees (induced) minor-closed?

Solution: Yes, they are closed under taking subgraphs. Each contraction reduces one vertex and at least one edge, hence the difference between the number of edges and vertices is maintained.

For later exercises we introduce also a notion of intersection graph. Let X be a (finite) family of sets, then the intersection graph I_X of X is defined as

$V(I_X) = X$ and edge connects sets with nonempty intersection, i.e. $E(I_X) = \{(x, y) : x, y \in X, x \cap y \neq \emptyset\}$.

A spaghetti graph is the intersection graph of a set of curves in the plane. Interval graphs are intersection graphs of intervals of real numbers.

Exercise 16: Are spaghetti/interval graphs (induced) minor closed?

Solution: They are only induced minor closed. Arbitrary large clique is a spaghetti/interval graph, but no every graph is a spaghetti/interval graph.

Line graphs are intersection graphs of edges of a graph.

Exercise 17: Are line graphs (induced) minor closed?

Solution: They are not induced minor closed (hence nor minor closed): The star with three rays is not a line graph. It can be obtained as a contraction of a line graph depicted in Fig. 3.

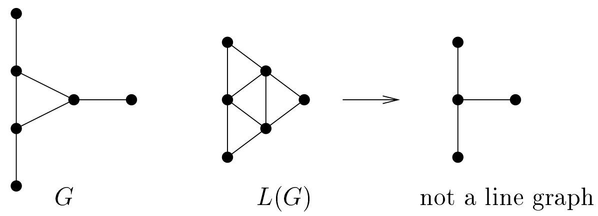


Figure 3: Line graphs are not induced minor-closed.

Line graphs are also well characterized by minimal forbidden induced subgraphs.

2 Well quasi ordering and the minor theorem

The material of this section is more-or-less reproduced from the monograph [7], chapter 12.

Observe that the relation "H is a minor of G" is reflexive and transitive, in other words it defines a quasi-order on the class of finite graphs $\mathcal{G}$. For such situation we write $H \preceq G$.

Observe also that $\preceq$ is also antisymmetric when we restrict ourselves on non-isomorphic graphs.

Exercise 18: Find all non-isomorphic quasi-orders on a three-element set X.

Solution: In total 9 possibilities.

Exercise 19: Find the $\preceq$ ordering on all non-isomorphic graphs on four vertices.

Solution: See Fig. 4.

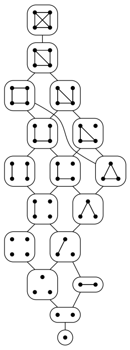


Figure 4: The $\preceq$ order on graphs with at most four vertices.

Definition: *A quasi-ordered set $(X, \preceq)$ is called well quasi-ordered, if for every infinite sequence $(x_i)_{i=1}^{\infty}$ of elements from X there exist elements x_i and x_j, such that $x_i \leq x_j$ and $i < j$.*

Exercise 20: Are vectors of $\mathbb{N}^d$ or of $\mathbb{Q}_+^d$ well (quasi) ordered either with the order on all coordinates $\leq^d$ or with the lexicographic order $\leq_{\text{Lex}}^d$, respectively?

Solution: An infinitely long decreasing sequence of rational numbers can be found, hence $\mathbb{Q}$ is not well ordered.

$(\mathbb{N}^d, \leq^d)$ is well ordered. Let a sequence $(A_i)_{i=1}^{\infty}$ be given. If $A_1 \not\leq A_i$ for all $i \geq 2$ we find a coordinate such that an infinite subsequence of $A_1, A_2, \ldots$ agrees on that coordinate. By excluding this coordinate we derive a sequence of vectors of less dimension and finish the proof by induction on the dimension.

The order $\leq^d$ is a suborder of $\leq_{\text{Lex}}^d$, hence the statement is valid also for the lexicographic order.

The following theorem is one of the jewels of graph theory:

Theorem 2.1 (Robertson and Seymour) *The class of all finite graphs ordered by the minor relation is well quasi-ordered.*

The complete proof of this theorem, formerly known as Wagner conjecture, has been published in a series of papers in Journal of Combinatorial Theory, Ser. B in late 80s.

In this course we will not reproduce the proof, we rather show an easier version of this theorem for trees and present a general scheme which was used to prove the theorem.

Let us first study several corollaries of the theorem:

Definition: *A set $\mathcal{O}$ is called the set of forbidden minors (or the obstruction set) for a minor closed class of graphs $\mathcal{C}$, iff:*

(1) $\mathcal{O} \cap \mathcal{C} = \emptyset$

(2) $\forall G : G \notin \mathcal{C} \iff \exists H \in \mathcal{O} : H \preceq G$

Corollary 2.2 *For every minor-closed class $\mathcal{C}$ the number of minor-minimal elements of $\mathcal{O}$ is* finite, *i.e. the class $\mathcal{C}$ can be characterized by* finitely many *forbidden minors.*

Proof: Consider a $H \notin \mathcal{C}$. Then every $G : H \preceq G$ does not belong to $\mathcal{C}$, otherwise $\mathcal{C}$ is not minor-closed. Take all graphs H that do not belong to $\mathcal{C}$, but all their minors do. Order this set arbitrarily. Due to Theorem 2.1, this set is finite. □

Exercise 21: Show that a class of graphs that is closed on taking subgraphs may have infinitely many forbidden subgraphs.

Solution: Bipartite graphs are closed under taking subgraphs but all odd cycles are the forbidden subgraphs.

Exercise 22: Find forbidden minors for the class of all forests and for the class of forests of maximum degree two.

Solution: Triangle C_3 is the only forbidden minor for forests, C_3 and $K_{1,3}$ are forbidden for forests of $\Delta \leq 2$

We note that Corollary 2.2 gives us a positive answer to the question whether Kuratowski theorem can be generalized for surfaces of higher genus, since such graphs are minor-closed. However, the number of forbidden minors grows fast: the graphs embedable into the projective plane have 35 forbidden minors.

The test "Is H a minor of G?" can be done in time $O(|V_G|^3)$ for a fixed graph H. Hence the computational result follows:

Corollary 2.3 *The membership test for a minor closed class of graph can be performed in $O(n^3)$ time. (The constant hidden in the O-notation depends on the graph class.)*

Let's focus our attention back to (well) quasi-ordered sets. A pair a_i, a_j of a sequence $(a_i)_{i=1}^{\infty}$ in a quasi ordered set $(X, \preceq)$ is called good, if $a_i \preceq a_j, i < j$. We call a sequence $a_1, a_2, \ldots$ bad, if it contains no good pair. Obviously well quasi-ordered sets are exactly those without bad sequences.

Observation 2.4 *In any well quasi-ordered set, every sequence $(a_i)_{i=1}^{\infty}$ contains an increasing subsequence.*

Proof: The fact that a sequence is good implies directly that it contains infinitely many increasingly ordered pairs. However, by use of the infinite Ramsey theorem, there must be also an increasing subsequence: We construct a infinite graph on $\{a_i\}$, with edges connecting $(a_i, a_j) : a_i \preceq a_j, i < j$. Since the independence number of this graph is finite, it must contain an infinite clique, which corresponds to the infinite increasing subsequence. $\square$

Assume that a set X has a quasi-order $\preceq$. We extend this quasiorder onto the class of finite subsets of X denoted by $X^{<\infty}$ as follows: We say that finite sets $A, B \subset X$ satisfy $A \preceq B$, if there exists an injective mapping $f : A \to B$ such that $a \preceq f(a)$ for all $a \in A$.

Lemma 2.5 *If a set X is well quasi-ordered by $\preceq$, then $(X^{<\infty}, \preceq)$ is also a well quasi-ordered set.*

Proof: We prove the Lemma by contradiction. Construct the sequence $(A_i)_{i=1}^{\infty}$ inductively as follows: We select A_i such that $|A_i|$ is minimal among all possible A_i such that an infinite bad sequence starts with $A_1, \ldots, A_i$. Such A_i exists for all $i \in \mathbb{N}$, and hence the sequence $(A_i)_{i=1}^{\infty}$ is well defined.

For each i we select some $a_i \in A_i$, and set $B_i = A_i \setminus \{a_i\}$. Since the set X itself is well quasi-ordered, there exists an infinite increasing subsequence $a_{i_1}, a_{i_2}, \ldots$. The sequence $A_1, A_2, \ldots, A_{i_1-1}, B_{i_1}, B_{i_2}, \ldots$ contains a good pair, due to the minimality of selection of $A_{i_1} : |A_{i_1}| > |B_{i_1}|$.

The good pair of this sequence should not be of the form $(A_j, A_{j'})$, since $(A_i)_{i=1}^{\infty}$ is bad. Similarly the good pair is not of the form (A_j, B_{i_k}). In this case the relation $A_j \preceq B_{i_k} \preceq A_{i_k}$ yields the same contradiction.

However, the existence of a bad pair of form $(B_{i_k}, B_{i'_k})$, causes a contradiction as well: Due to $a_{i_k} \preceq a_{i'_k}$ we get that $A_{i_k} \preceq A_{i'_k}$. $\square$

Due to Lemma 2.5 it suffices to show well quasi-ordering on connected graphs (which usually do not form a minor-closed set), and the well quasi-order will be also transferred to finite disjoint unions of such graphs. (E.g., from trees to forests.)

Now we are ready to prove the minor theorem for the class of forests. We prove a stronger version of the theorem (with an easier proof): Consider only rooted trees and only those minor relations that obey the parent-child relation (also in the sense of Lemma 1.1). More precisely we define:

Every rooted tree T has a unique orientation $\vec{T}$ such that each edge is oriented towards the parent vertex.

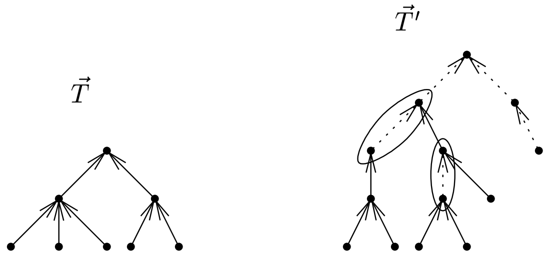


Figure 5: The topological minor relation.

A rooted tree T is a topological minor of a rooted tree T', if $\vec{T}$ is isomorphic to a minor of $\vec{T'}$ (see Fig. 5).

Theorem 2.6 (Kruskal) *The class of finite rooted trees is well quasi-ordered by the topological minor relation.*

Proof: As in the previous proof, we will proceed by contradiction. First, we construct a bad sequence of rooted trees $(T_i)_{i=1}^{\infty}$, such that T_i is selected as the minimal rooted tree such that $T_1, \ldots, T_i$ has an infinite bad extension.

Denote by r_i the root of tree T_i, and let F_i be the forest of rooted trees $T_i - r_i$, where the children of r_i are selected as roots of the new trees.

It is enough to prove that the set $\mathcal{F} = \bigcup_{i=1}^{\infty} F_i$ is well quasi-ordered by the topological minor relation. Consider any sequence $(B_j)_{j=1}^{\infty}$ of trees from $\mathcal{F}$, and for each j select arbitrarily $i(j)$, such that $B_j \in F_{i(j)}$. Select a subsequence $B_{j_1}, B_{j_2}, \ldots$ such that $i(j_1), i(j_2), \ldots$ is non-decreasing.

The sequence $T_1, T_2, \ldots, T_{i(j_1)-1}, B_{j_1}, B_{j_2}, \ldots$ is not bad due to the minimal selection of $T_{i(j_1)}$, hence it contains a good pair, either of form (T_i, B_j) or $(B_j, B_{j'})$. Any good pair of the first form satisfies $T_i \preceq B_j \preceq T_{i(j)}$ and $i < i(j_1) \leq i(j)$. The existence of such pair contradicts the assumption that the sequence $(T_i)_{i=1}^{\infty}$ is bad. A good pair of the second form is also a good pair for the sequence $(B_j)_{j=1}^{\infty}$. Hence, the set $\mathcal{F}$ is well quasi-ordered.

Finally, we apply Lemma 2.5 and get that $\mathcal{F}^{<\infty}$ is well quasi-ordered too. Therefore, any extension of the topological minor relation $F_i \preceq F_{i'}$ can be extended to a minor relation $T_i \preceq T_{i'}$. Observe that to get a minor of T_i in $T_{i'}$, it is enough to contract paths in $T_{i'}$ to the roots of minors in $F_{i'}$ into a single edge. □

The proof of Robertson-Seymour theorem follows in some situations the above proof of Kruskal theorem. The central notion on the general proof takes the notion of treewidth. Graphs with bounded treewidth have a tree-like structure, hence similar arguments might be used. We introduce this notion in the next section.

Exercises

Exercise 23: Show that the finite trees are not well quasi-ordered by the subgraph relation.

Solution: An infinite set of pairwise incomparable trees can be defines as follows: for any $n \in \mathbb{N}$ take a two 3-stars ($K_{1,3}$) whose centers are joined by a path of length n.

3 Treewidth

3.1 Definition and basic properties

We start with the formal definition of tree-decomposition and treewidth.

Definition: *The* tree decomposition *of a graph G is a tree T satisfying the following properties:*

- $V(T) \subseteq \mathcal{P}(V_G)$ *i.e. the nodes of T are subsets of V_G.*
- *For each edge $(u, v) \in E_G$ there is a node X_i of T, such that both $u, v \in X_i$.*
- *For each vertex $u \in V_G$, the subgraph of T induced by sets containing u is connected (i.e. it is a subtree).*

The width of a decomposition T is $\max_{X_i \in V_T} |X_i| - 1$*. The* treewidth *of a graph G, denoted by* $\mathrm{tw}(G)$*, is the minimum width among all its decompositions.*

(To deal well with graphs of treewidth 0, we may also require that every vertex appears in at least one node.)

In the following text we call the vertices of the tree T *nodes*, to distinguish them from the vertices of the graph G.

Exercise 24: Prove, that the last property in the definition of the tree decomposition is the equivalent with the condition: "If a node X_j of T lies on the unique path between nodes X_i and X_k then $X_i \cap X_k \subseteq X_j$".

Solution: The nodes containing any vertex of $X_i \cap X_k$ form a connected subgraph of T. This subgraph contains also X_j and the claim follows.

Exercise 25: The treewidth of a disjoint union of two graphs G_1 and G_2 is $\max\{\mathrm{tw}(G_1), \mathrm{tw}(G_2)\}$.

Solution: Joint the two trees by an arbitrary edge.

Exercise 26: Determine the treewidth of a tree.

Solution: Treewidth of a tree on at least two vertices is 1. The decomposition contains a node for each vertex and also for each edge.

Lemma 3.1 *For every graph G it holds that* $\mathrm{tw}(G) \geq \omega(G) - 1$.

Exercise 27: Prove Lemma 3.1

Solution: Proof: Assume that a set $K \subseteq V_G$ induces a clique in G, i.e. a complete subgraph, on k vertices and that all nodes of a tree-decomposition of G have size at most $k-1$.

Now take the node X_0 that contains the maximum number of vertices of K. Since $|X_0| < |K|$ there exists at least one vertex $u \in K \setminus X$. Let X_1 be the node in T containing u that is closest to the node X_0 and let (X_1, X_2) be the first edge of the unique path from X_1 to X_0 in T. Since u is adjacent to all vertices of $X_0 \cap K$, the intersection $X_1 \cap X_2$ must contain all vertices of $X_0 \cap K$. Otherwise there exists a vertex $v \in X_0 \cap K \setminus X_1$, such that no node X of T contains both u and v, but (u, v) is an edge of G.

We have got that $X_0 \cap K \subset X_1 \cap K$, a contradiction with the maximality of X_0. □

Corollary 3.2 *Vertices of each complete subgraph of G form a subset of some node in any tree decomposition of G.*

Lemma 3.3 *If H is a minor of G then* $\mathrm{tw}(H) \leq \mathrm{tw}(G)$.

Exercise 28: Prove Lemma 3.3.

Solution: Proof: If H is a subgraph of G, then $\mathrm{tw}(H) \leq \mathrm{tw}(G)$, because from a minimal tree-decomposition of G we delete all vertices outside H and get a tree decomposition of H of width at most $\mathrm{tw}(G)$. (On edge removal no modification of the tree decomposition is necessary.)

So the only non-trivial part corresponds to the edge contraction. Assume inductively that H is a graph that came out of G after contracting an edge (u, u') into the new vertex v. Then in the minimal tree-decomposition replace each occurrence of u or u' by v and get a tree-decomposition of H. □

Exercise 29: What is the treewidth of a cycle, or of an outerplanar graph, respectively?

Solution: Two, the lower bound follows from K_3 minor. A tree decomposition of width two can be constructed easily.

Exercise 30: Find other examples of graphs are of treewidth 1, or 2 respectively.

Solution: Graphs of treewidth 1 are forests, graphs of treewidth 2 can be defined e.g. as subgraphs of series-parallel graphs.

Exercise 31: Find a graph of treewidth 2, that is not outerplanar.

Solution: The smallest example is the complete bipartite graph $K_{2,3}$.

Definition: *Series-parallel graphs are constructed from $G_{u,v} = K_2$ by the recursive application of the following rules:*

- *If $G'_{u',v'}$ and $G''_{u'',v''}$ are series-parallel graphs, then the* series connection *of $G'_{u',v'}$ and $G''_{u'',v''}$ is a series-parallel graph $G_{u',v''}$, which appears from the disjoint union of $G'_{u',v'}$ and $G''_{u'',v''}$, by merging v' with u'' into a single vertex.*

- *Similarly, the* parallel connection *of two series-parallel graphs $G'_{u',v'}$ and $G''_{u'',v''}$ gives a series parallel graph $G_{u',v'}$. Here we take the disjoint union of $G'_{u',v'}$ and $G''_{u'',v''}$ and merge vertices u' with u'' and v' with v''.*

The binary tree describing the recursive application of these two rules is called construction tree*.*

Exercise 32: Prove that subgraphs of series-parallel graphs are exactly those of treewidth at most two.

Solution: The treewidth of a series-parallel graph is bounded by 2, since both parallel and series connections does not increase treewidth: W.l.o.g. we can require that the tree decomposition of a series-parallel graph contains both special vertices u, v in a common node. Then a tree decomposition of a graph composed as a connection of two fragments can be derived from the tree decompositions of its two fragments.

Now consider a graph G of treewidth at most two. Without loss of generality we may assume that it is connected. Any tree decomposition of G with nodes of size three can be modified by insertion of new nodes such that in the final decomposition T every two adjacent nodes intersect on exactly two vertices of G.

Let G' be the graph which arose by adding edges of (u, v) into G whenever u and v belong to the same node. We show that G' is series-parallel.

Any leaf node of T contains a vertex u such that u does not appear elsewhere in T. The graph $G' \setminus u$ is of treewidth 2 and is series-parallel by the induction hypothesis. Let v, w be the two vertices which appear in the same node as u. Since (v, w) is an edge of $G' \setminus u$, it had to be inserted as an edge K_2 in the its construction tree. But, at this moment we can extend K_2 onto the triangle K_3 by a parallel connection with the path vuw and get the construction tree for G'.

Observe that the graph G depicted in Fig. 3 is not series-parallel but has treewidth 2.

For a better understanding of the structure of graphs of bounded treewidth, we give another characterization in terms of k-trees.

Definition: *A graph is called* chordal*, if it contains no induced cycle of length at least four as a subgraph.*

We note here, that chordal graphs have nice properties from the algorithmic point of view. For example they are perfect and allow a polynomial time coloring algorithm. Later we show a similar coloring algorithm for graphs of bounded treewidth.

Observe that chordal graphs can be obtained from a complete graph by repeating the following operation: Find a set of mutually adjacent vertices and add a new vertex adjacent to all of them.

Lemma 3.4 *Every chordal graph G has a vertex whose neighborhood induces a complete subgraph in G.*

Proof: For a contradiction assume that this is not the case and take a graph G with this property. For an arbitrary vertex u exist two of its neighbors v and v' s.t. $(v, v') \notin E_G$. Then the set S_u of all vertices that are adjacent to both v and v' induces a clique, that is also a cut in G (otherwise we find an induced cycle of length at least 4).

Select u such that $G \setminus S_u$ contains a component $A \ni v$ of the smallest size. Let w, w' be the two non-adjacent neighbors of v separated by S_v. Clearly both, $w, w' \in A \cup S_u$ since other vertices are separated by S_u from v.

Denote by $B \ni w$, $B' \ni w'$ the two components separated by S_v. At most one of B, B' can intersect S_u, hence at least one of them is a subset of $A \setminus \{v\}$, a contradiction. □

Lemma 3.5 *For every chordal graph there exists a tree decomposition such that every node induces a clique.*

Proof: The insertion of a vertex u, whose neighborhood induces a clique, corresponds with an insertion of a leaf node on $\{u\} \cup N(u)$ which also forms a clique. □

Exercise 33: Show that chordal graphs are intersection graphs of subtrees of a tree.

Solution: Let the tree be the tree-decomposition from the previous lemma. The subtree corresponding to a vertex u is induced by nodes containing u.

Definition: *A chordal graph is called k-tree, if it can be constructed from the complete graph K_{k+1} by a sequence of insertions of vertices, which neighborhood at the insertion moment forms a clique of size k.*

Lemma 3.6 *Every graph of treewidth at most k is a subgraph of an k-tree and vice versa.*

Due to this fact, the graphs of treewidth at most k are also sometimes called *partial k-trees.*

Exercise 34: Prove Lemma 3.6

Solution: Proof: Observe that the insertion of edges that have both ends in the same node does not increase the treewidth of the graph, but yields to a k-tree. (The construction of the k-tree can be derived from the tree decomposition.)

For the opposite implication it is easy to show that every k-tree has treewidth at most k. The tree decomposition of width k can be derived from the construction of the graph. □

Exercise 35: Show that $\mathrm{tw}(G) = \min\{\omega(G') : G \subseteq G', G' \text{ is chordal}\} - 1$

Solution: Follows from Lemmas 3.1, 3.3 and 3.6.

Exercise 36: Show that k-decomposable graphs are not necessarily subgraphs of k-trees. (Any decomposable graph allows an ordering of its vertices, such that each vertex is adjacent with at most k successors.)

Solution: For example planar graphs are 5-decomposable but they have unbounded treewidth.

Exercise 37: Show that graphs G of treewidth at most k have at most $k|V_G|$ edges.

Solution: W.l.o.g we may assume that G is a k-tree. Then in each step of the construction of the k-tree we add with each vertex exactly k edges, but at the beginning we have started with the graph K_{k+1} which has $\binom{k+1}{2} \leq k(k+1)$ edges.

To relax from the formal presentation we would like at the end of this lecture to present a simple LaPaugh's "cops-and-robber" game played on a graph. Imagine that the graph represents a town with vertices as crossings and edges as streets. The game is played between one robber on a motorbike and several police helicopters. As the robber uses motorbike, he can quickly move from a vertex to any of its neighbors. The police helicopters do not need to follow edges of the graph, however their movement needs much more time. The robber may move through several edges during a single movement of a police helicopter, however he cannot pass through a crossing occupied by a helicopter. Even if a helicopter decides to land at the robber's standpoint, the robber has enough time to run away through any free adjacent vertex. The robber's aim is to escape forever, whereas police wants to encircle him, forbid any his movement and finally capture him by landing a helicopter on the vertex of his position.

Exercise 38: On which graphs has police a winning strategy with one, or with two helicopters, respectively?

Solution: With single helicopter the robber must be placed on an isolated vertex. With two helicopters it is possible to isolate robber in a forest.

Exercise 39: On which graphs the robber has a winning strategy, if police has only 2 helicopters?

Solution: When the graph contains a cycle, the robber can always move to some free vertex of the cycle, if the helicopter is flying.

Lemma 3.7 *The police had a winning strategy with $k+1$ helicopters if and only if the graph has treewidth at most k.*

For the $\Rightarrow$ implication we need the notion of separators, that will be discussed later. Now we prove only the opposite implication.

Exercise 40: Prove the $\Leftarrow$ implication in Lemma 3.7.

Solution: **Proof:** Assume that G has treewidth k and is isomorphic to a k tree. Police first concentrate all its helicopters on vertices of a node X in T. Then robber is in some component G' of $G - X$. In the next moves police should move its helicopters to the vertices of the node adjacent to X towards the location of robber. In finitely many steps robber will be captured. $\square$

EXERCISES

Exercise 41: Have planar graphs bounded treewidth?

Solution: No, see the next exercise.

Exercise 42: What is the treewidth of the $n \times n$ grid graph?

Solution: **Proof:** We show that the grid has treewidth at least n, by use of Lemma 3.7. The police strategy with $n+1$ helicopters is easy: sweep across the square. For the lower bound consider n helicopters in the mesh. Then either all are placed on a diagonal, or there exists a "large" region which connects the right and left side of the mesh or top and bottom. When any helicopter takes off, such region exists also in the "diagonal" case. The robber strategy is to stay at any point of this large region. $\square$

Exercise 43: Design a polynomial-time coloring algorithm for chordal graphs.

Solution: Use Lemma 3.4 and remove the vertex whose neighborhood induces a clique. Chordal graphs are closed on induced subgraphs, so this graph can be colored optimally. For the removed vertex use the first available color.

3.2 Treewidth and the minor theorem

In this section we will overview main ideas that were used in the proof of Robertson-Seymour minor theorem. We refer to [7] as a source for the contents of this section, where an interested reader can find more details. Due to the complexity all proofs are omitted here.

Lemma 3.3 implies, that every class of graphs of treewidth bounded by a constant is minor-closed. The following generalization of Kruskal Theorem was one of the first basic steps in the proof of Robertson-Seymour theorem.

Theorem 3.8 *For every k, the class of graphs of treewidth bounded by k is well quasi-ordered by the minor relation.*

We showed in the section 2 the case $k = 1$. The case of general k is solved by k-iterations of arguments used in the proof of Kruskal theorem.

Conversely, we can characterize those classes of graphs that have bounded treewidth and exclude a certain minor.

Theorem 3.9 *Any class of graphs, which contains no minor of some fixed graph H, has bounded treewidth if an only if H is planar.*

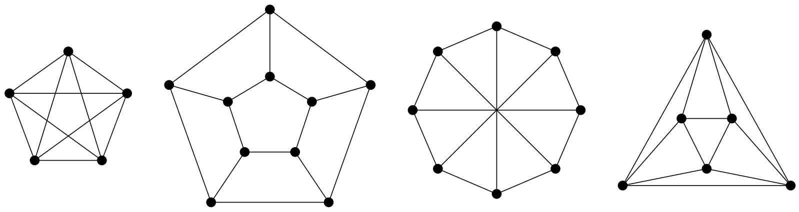


Figure 6: Forbidden minors for graphs of treewidth at most three.

Since every planar graph is a minor of a grid of sufficient size, it is sufficient to prove:

Theorem 3.10 *For every integer n exists k_n such that every graph of treewidth at least k_n contains the $n \times n$ grid as a minor.*

Now consider a sequence of finite graphs $G_0, G_1, \ldots$. We want show that this sequence had a good pair $G_i \preceq G_j$, $i < j$. If G_0 is planar then, graphs in the rest of the sequence either had bounded treewidth (and they are well quasi-ordered), or at least one graph contains G_0 as a minor. If G_0 is not planar and moreover it is not a minor of the rest of the sequence, then we may assume none of $G_1, G_2, \ldots$ contains $K_{|V(G_0)|}$ as a minor. Such graphs might be characterized in terms or tree decomposition and embedding into surfaces of bounded genus, and it has been proven that these graphs are also well quasi-ordered by the minor relation.

Finally we would like to show (without proof), that some classes of graphs of bounded treewidth, have known a good characterization in terms of forbidden minors [2].

Theorem 3.11 *A graph has tree-width at most*

- *1, if it does not contain $K_3 = C_3$ as a minor.*
- *2, if it contains no K_4 minor.*
- *3, if no graph depicted in Fig. 6 is its minor.*

3.3 Separators and brambles

We call a set of vertices $S \subseteq V_G$ a *separator*, if the subgraph of G induced by $V_G \backslash S$ (abbreviated by $G \backslash S$) has more components than G. Separators or also

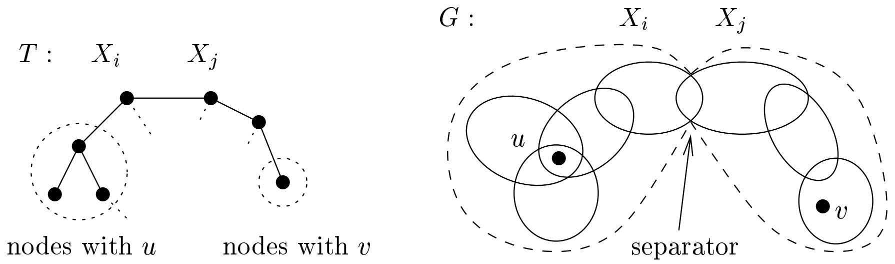


Figure 7: An illustration for Lemma 3.12.

called cutsets, play important role in the finding good tree-decompositions. First observe, that a the existence of a tree decomposition of width t implies, that if $V_G \geq t$ then G has a separator with at most t vertices.

Lemma 3.12 *For every edge $(X_i, X_j) \in E_T$ it holds, that*

- *either the intersection $X_i \cap X_j$ is either a separator in G*
- *or all nodes of the component of $T - (X_i, X_j)$ containing the node X_i are subsets of X_i.*
- *or the vice-versa for X_j*

Exercise 44: Prove Lemma 3.12.

Solution: Proof: Without loss of generality we may assume that G is connected.

Observe first that if $X_i \subsetneq X_j$ but some other node of the component of $T - (X_i, X_j)$ containing the node X_i is not a subset of X_i, then $X_i = X_i \cap X_j$ is a separator.

Assume for a contradiction, that T is a tree-decomposition of G and X_i and X_j are two nodes of T violating all three statements of the lemma. Then there exists a vertex $u \in X_i \setminus X_j$ as well as $v \in X_j \setminus X_i$ and u and v are connected in $G \setminus (X_i \cap X_j)$. See Fig. 7.

Since both subgraphs of T induced by nodes containing u and v, respectively are connected and are separated by the edge (X_i, X_j), there cannot exist a node containing both vertices u and v, which is in a contradiction with the existence of the edge (u, v) in G. $\square$

We say that S is a separator for a set $W \subseteq V$, if at least two vertices of W belong to different components of $G \setminus S$. Lemma 3.12 could be extended as follows:

Corollary 3.13 *For each $W \subseteq V_G$, and any tree decomposition T of G either there exist a node $X \supseteq W$ or W has a separator of form $X_i \cap X_j$ for some edge $(X_i, X_j) \in E_T$.*

Exercise 45: Prove Corollary 3.13.

Solution: Proof: If $W \not\subseteq X_i$ for all nodes X_i, then for each node X_i the set W has a vertex $u_i \in W \setminus X_i$. Consider a path in T from X_i to any node containing u_i and orient the first edge of the path in this direction (such edge is unique since all nodes containing u_i form a subtree of T and X_i does not belong to this subtree).

One edge (X_i, X_j) of T must be oriented in both directions, and therefore $X_i \cap X_j$ is a separator for vertices u_i and u_j as follows from Lemma 3.12. □

Call a separator S for a set W good, if no component of $G \setminus S$ contains more than $\frac{|W|}{2}$ vertices of W.

Corollary 3.14 *If G has treewidth bounded by k, then any $W \subseteq V$ has a good separator S of size at most $k+1$.*

Exercise 46: Prove Corollary 3.14.

Solution: Proof: Check every node X_i of T if X_i is a good separator for W. If not, orient one of the incident edges towards the component of $G \setminus X_i$ with more than $\frac{|W|}{2}$ vertices of W.

Then there exist an edge (X_i, X_j) of T directed in both directions, i.e. $X_i \cap X_j$ separates two subsets of W, each of size more than $\frac{|W|}{2}$, a contradiction. □

Some classes of graphs allows separators of sufficiently small size. Observe that planar graphs on at least five vertices have separators of size at most five, since they always have a vertex of degree at most five. However, more specific separators (similar to good separators) can be found:

Theorem 3.15 (Lipton-Tarjan [10]) *Every planar graph G has a separator S of size at most $2\sqrt{2|V_G|}$ such that each component of $G \setminus S$ has at most $\frac{2}{3}|V_G|$ vertices.*

Proof: (Sketch) We show a weaker result where the separator has size at most $4\sqrt{n}$, where $n = |V_G|$ Assume without loss of generality that G is a planar triangulation. If a spanning tree of G of diameter at most $2\sqrt{n}$ exists, then we check all fundamental cycles incident with the non-tree edges. These edges correspond to a spanning tree of the dual graph. We weight these edges by the number of vertices separated by the corresponding fundamental cycle. It is not hard to deduce that an edge of weight in the interval $\langle \frac{1}{3}, \frac{2}{3})$ exists, which corresponds to the desired separator. (The length of the cycle is bounded by the diameter of the spanning tree, i.e. $2\sqrt{n}$.)

To find a short spanning tree we proceed vertices by breadth-first search. It the $\sqrt{n}$-th level L is of size at most $\sqrt{n}$ we get the desired separator. If not, observe that at most $\sqrt{n}$ levels around L can have size $> \sqrt{n}$. Then we use the above method with the BFS spanning tree for these large levels, and extend this partial separator by the two thin surrounding levels (of size at most $\sqrt{n}$). The final separator for G is then of size at most $4\sqrt{n}$. $\square$

Note that the above method can be implemented as a linear time algorithm.

We have already showed that planar $n \times n$ grid G has treewidth n, i.e. $\text{tw}(G) = \sqrt{|V_G|}$. In fact this is an example of a planar graph with asymptotically maximal treewidth.

Exercise 47: Show that $tw(G) = O(\sqrt{|V_G|})$ for all planar graphs G.

Solution: Use theorem 3.15 to find a separator S of size at most $2\sqrt{2|V_G|}$ and recursively find the tree decomposition T' of $G \setminus S$ of width at most $c\sqrt{\frac{2}{3}|V_G|}$. Then insert S into each node of T' which yield the tree decomposition of width $2\sqrt{2|V_G|} + c\sqrt{\frac{2}{3}|V_G|} \leq c\sqrt{|V_G|}$ for $c = 16$.

We now focus our attention on a method to prove a lower bound on treewidth. For this purpose we define a notion of bramble:

Definition: *A family of subsets of vertices $\mathcal{B}$ is called* bramble*, if*

- *Each $B \in \mathcal{B}$ induces a connected subgraph of G.*
- *For each pair of sets B, B' in $\mathcal{B}$ the union $B \cup B'$ induces a connected subgraph in G. In other words either B and B' have at least one vertex in common, or there exist an edge between B and B'.*

We say that a set W covers a bramble $\mathcal{B}$ if the set W intersects every set B of the bramble $\mathcal{B}$. The order *of a bramble $\mathcal{B}$ is the minimum number of vertices of a set that covers $\mathcal{B}$.*

Exercise 48: Find a bramble of order n in the $n \times n$ grid graph.

Solution: $\mathcal{B} = \{B_i : B_i$ is the union of the i-th row and the i-th column of the grid$\}$.

Lemma 3.16 *If G has a bramble $\mathcal{B}$ of order k, then* $\text{tw}(G) \geq k - 1$.

Proof: Let T be a tree-decomposition of G. We claim that there exists a node of T that intersects all sets in the bramble $\mathcal{B}$.

If not, then for every node X_i at least one $B_i \in \mathcal{B}$ exists such that X_i does not intersect B We mimic the proof of Corollary 3.13 and we orient one of the edges incident with X_i towards nodes containing vertices of B_i which form an connected subtree of T.

Then the edge (X_i, X_j) oriented in both directions determines a cutset $X_i \cap X_j$ that separates sets B_i and B_j, a contradiction. $\square$

In fact, the opposite implication in Lemma 3.16 holds, but it is not trivial to prove. The theorem is called the *treewidth minimax theorem* and was proved by Seymour and Thomas in 1993 [13].

Theorem 3.17 *For any $k \geq 1$, G has treewidth at least $k-1$ if and only if it contains a bramble of order at least k.*

We in fact prove a stronger theorem from which Theorem 3.17 follows by setting $\mathcal{B} = \emptyset$:

Theorem 3.18 *Let G has no bramble of order at least k. Then for each bramble $\mathcal{B}$ exists a tree decomposition T, such that for any node $X \in V_T$ of size at least k holds that*

- *X is a leaf in T*
- *$X \cap B = \emptyset$ for some $B \in \mathcal{B}$.*

Proof: We use backward induction on $\mathcal{B}$. i.e. for any bramble $\mathcal{B}$ we assume that the statement is valid for all brambles that contain $\mathcal{B}$ as a subset.

Without loss of generality we may assume that $|V_G| \geq k$ since otherwise we can take T as the one-vertex tree with $X = V_G$.

For a given bramble $\mathcal{B}$ select the set W of the minimal size that covers $\mathcal{B}$. Since no bramble has order at least k, we have $|W| < k$. Denote by $A_1, \dots A_r$ the components of the graph $G \setminus W$. Since $|V_G| \geq k$ we have that at least one such component exists.

We now prove an auxiliary statement:

For any $i = 1, \dots, r$ there exists a tree decomposition T_i of $G_i = G|_{A_i \cup W}$ such that:

- *$X_i = W$ for some node $X_i \in T_i$,*
- *any node X of size at least k in T_i is a leaf, and $X \cap B = \emptyset$ for some $B \in \mathcal{B}$.*

If $\mathcal{B} \cup \{A_i\}$ is not a bramble, we may set T_i to be a two-vertex tree, with nodes W and $A_i \cup N(A_i)$. Even if A_i is of size at least k then it is separated from some $B \in \mathcal{B}$ which imply that $B \cap N(A_i) = \emptyset$.

Now assume that $\mathcal{B} \cup \{A_i\}$ is a bramble. It has size larger than $\mathcal{B}$, so by the induction hypothesis there is a tree decomposition T' of G according

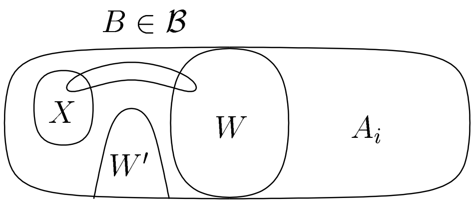


Figure 8: No small separation exists between X and W.

to the statement of the theorem. Without loss of generality we may assume that T' is not valid also for $\mathcal{B}$, i.e. T' contains a node X of size at least k, which is a leaf, but $X \cap B \neq \emptyset$ for all $B \in \mathcal{B}$, but $X \cap A_i = \emptyset$.

Observe that no set W' of size less than $|W|$ separates W and X, since it would had to cover $\mathcal{B}$ (see Fig. 8). By Menger's theorem there exists a set of $s = |W|$ vertex-disjoint paths $P_1, \ldots, P_s$ which connect W and X.

We now modify the tree decomposition T' as follows:

1. Restrict nodes of T' only to vertices of $A_i \cup W \cup V_{P_1} \cup \cdots \cup V_{P_s}$.

2. Contract each path P_i into a single vertex $u \in W$.

At the end of the procedure we obtain the desired auxiliary tree decomposition T_i with the special node $X_i = X = W$. Observe that any node of T_i of size at least k is a leaf and that it does not intersect some $B \in \mathcal{B}$, because every such node contains at least one vertex of A_i.

The final step in the construction is the composition of all auxiliary tree decompositions $T_1, \ldots, T_r$ into T. This is possible, since each T_i contains the node $X_i = W$ and we can take disjoint union $T_1 \cup \cdots \cup T_r$ and unify all X_i into a single node. $\square$

Exercise 49: Prove the $\Rightarrow$ implication of Lemma 3.7 with help of Theorem 3.17.

Solution: If $tw(G) > k$ then G contains a bramble $\mathcal{B}$ of order at least $k+1$. With k helicopters the police can not cover all $B_i \in \mathcal{B}$, hence the robber then can stay in (and move between) the uncovered sets of the bramble.

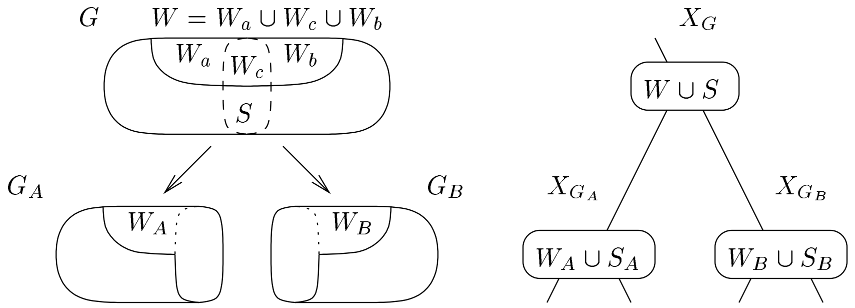


Figure 9: Recursive finding good separators and building tree decomposition.

4 Algorithms for bounded treewidth graphs

4.1 Computing treewidth

The problem to determine whether a given graph G has at most treewidth k is NP-complete [1].

On the other hand for fixed k the problem $\mathrm{tw}(G) \leq k$ could be decided in linear time [4]: We here present a simpler quadratic algorithm that either answers that the $\mathrm{tw}(G) > k$ or outputs a tree-decomposition of width at most $3k + 3$ [12]. Then the exact treewidth of G could be computed by the technique described in Section 4.2.

The basic idea is that we try to find recursively a good separator S of size at most $k+1$ for the set W of size at most $2k+3$. If no such separator exists then, by Corollary 3.14, the treewidth of G is more than k. If we succeed in all recursive steps, we construct a tree decomposition of width at most $3k+2$ as depicted in Fig. 9. We note here that we may start with an arbitrary set W of size $2k+3$, and also observe, that we put a leaf to the tree, if the size of the component reaches $k+1$.

The recursion rules assure that the size of the set W_A is bounded (the same for W_B), since it follows from the definition of good separator that

$$|W_A| \leq |S| + \frac{|W|}{2} \leq k + 1 + \frac{2k+3}{2} \leq 2k + 3.$$

The constructed tree decomposition T is of width at most $|W|+|S|-1 \leq 3k+3$. Observe also, that we have defined a proper tree decomposition, since every edge of G will be represented either in the node $W \cup S$ or in one of recursive trees for G_A or G_B. Moreover all nodes containing a fixed vertex form a connected subtree, since if it occurs in X_G and not in X_{G_A}, it means that it does not belong to G_A and therefore cannot occur later in T below G_A.

> Tree decomposition
> **Input:** *A graph G of width at most k, a set $W \subseteq V_G$, $|W| \leq 2k+3$.*
> **Output:** *A rooted tree decomposition T of G of width at most $3k+3$, where $W \subseteq root(T)$.*
> **begin**
> find a good separator S for W in G
> construct recursively a tree decomposition T_A for G_A, W_A
> construct recursively a tree decomposition T_B for G_B, W_B
> take $T = T_A \cup T_B \cup \{W \cup S\}$, where the roots of T_A and T_B are the only children of $root(T) = \{W \cup S\}$,
> **end.**

Now the only missing step is to find good separators quickly:

Lemma 4.1 *For any G on n vertices and $W \subseteq V$ of size at most $2k+3$, it is possible to find a good separator of size at most $k+1$ or answer that no such separator exists in time $O(n \cdot 9^k)$.*

Proof: Such good separator exists, if and only if W can be partitioned into three parts W_a, W_b and W_c, each of size at most $k+1$, and W_c is a part of a separator of size at most $k+1$ (consult Fig. 9). In other words, in such case the graph $G \setminus W_c$ has no $k+1-|W_c|$ vertex disjoint paths between sets W_a and W_b and this could be tested in $O(|E(G \setminus W_c)|) = O(n)$ time (e.g. by the maximum flow technique).

We try all possible partitions of W into three parts (there are at most 3^{2k+3} of them) and for each we solve the vertex-disjoint path problem. Observe that we find a good separator if and only if at least one exist, so we may answer "NO", if all tests fail. $\square$

The time complexity of the Tree decomposition algorithm composes of at most n iterations of the Separator search algorithm, because the final tree-decomposition has at most n nodes. In total we get $O(n^2 9^k)$ required operations.

```
SEPARATOR SEARCH
Input: A graph G a set W of size at most 2k + 3.
Output: Either a good separator S for W of size at most k + 1
        or a conclusion that it does not exist.
begin
  extend W to 2k + 3 vertices
  for all partitions of W into sets W_a, W_b and W_c of size at most k + 1
    do begin
      if there is no (k − 1 − |W_c|)-flow from W_a to W_b
        then find a separator S and exit;
    end;
  answer "no good separator for W of size at most k + 1 exists".
end.
```

4.2 More applications and nice decompositions

In this section we show how some hard problems allow a tractable solution when restricted to graphs of bounded treewidth.

The general scheme is based on a "brute force" enumeration of all feasible solutions of the given problem when restricted onto graph G_i, i.e. the induced subgraph of G by the node X_i. We assume that the tree T is rooted, and for each node X_i we build a table $\texttt{Tab}_i$ of constant or polynomial size, whose entries describe necessary properties of partial solutions on the graph G_i, and sometimes also on the graph G'_i, that is the subgraph of G induced by the union of X_i and all its descendants. The data entries for X_i should be computed in polynomial time from the entries in $\texttt{Tab}_j$, corresponding the children X_j of X_i.

The situation when the table $\texttt{Tab}_m$ is nonempty for the root node X_m usually indicates that a proper solution of the entire graph G exists.

We illustrate this approach on the problem of computing the size of the maximal independent set $\alpha(G)$.

Each entry in $\texttt{Tab}_i$ is the pair (S, a), where S is an independent set in G_i (possibly empty) and a is the size of the maximum independent set of G'_i with S as its intersection with X_i.

In the following algorithm we call an independent set S_j of G_j feasible for S_i if $X_j \cap S_i = X_i \cap S_j$.

The algorithm works as follows:

```
MAXIMUM INDEPENDENT SET
Input: A tree decomposition of a graph G of fixed width c.
Output: Size of the maximal independent set α(G).
begin
  mark all nodes as unfinished
  while exists an unfinished node X_i with all children finished do
    begin
      for all independent sets S_i in G_i do
        if for all children X_j of X_i there exists
          a set S_j feasible for S_i, s.t. (S_j, a_j) ∈ Tab_j
          then
            begin
              for each X_j select a S_j feasible for S_i
                s.t. (S_j, a_j) ∈ Tab_j and a_j is maximized
              a_i := |S_i| + ∑_j (a_j − |S_j ∩ S_i|)
              store (S_i, a_i) into Tab_i
            end;
      mark X_i as finished
    end;
  α(G) = max{a_m : (S_m, a_m) ∈ Tab_m}.
end.
```

Observe that for any X_i the computation of the table $\texttt{Tab}_i$ requires at most $O(2^{2|X_i|})$ operations. By assuming that the treewidth of G i bounded by is a constant, the algorithm runs in time $O(|T|)$, that means, it is linear in n.

Exercise 50: Modify the algorithm such that it outputs also an independent set of size $\alpha(G)$.

Solution: First compute the independence number from the botom-up manner and then traverse the tree in the opposite direction and decompose each partial solution of each parent into partial solutions of its children.

In some cases a better structured tree-decompositions might be helpful in the design of an algorithm.

The *nice tree decomposition* of G ([8]) is a tree decomposition such that: T is a rooted binary tree and for each $i \in V(T)$ exactly one of the following cases applies:

- X_i is a leaf and $|X_i| = 1$, then X_i is a *leaf node.*

- X_i has one child j and $X_i = X_j \cup \{v\}$ for some $v \in V(G) \setminus X_j$, then we call X_i an *introduce node*.

- X_i has one child j and $X_i = X_j \setminus \{v\}$ where $v \in X_j$, then X_i is a *forget node*.

- X_i has two children j, j', and $X_i = X_j = X_{j'}$ then we call X_i a *join node*.

Exercise 51: Show that any tree decomposition can be transformed in polynomial time into a nice tree decomposition of the same width [8].

Solution: This can be done by means of local replacements. Each node will be substituted by at most $\log k$ union nodes, each edge will be substituted by at most k forget and at most k indroduce nodes of a nice decomposition.

We show an application of this method on the graph coloring problem, e.g. a decision problem, testing whether $\chi(G) \le k$. Observe that $\chi(G) \le \mathrm{tw}(G)$, so we can assume $k \le \mathrm{tw}(G)$.

Now the table $\texttt{Tab}_i$ stores all colorings (viewed as vertex partitions) of G_i with at most k colors, that allow an extension to G'_i.

```
k-COLORABILITY
Input: A tree decomposition of a graph G of fixed width c.
Output: Answer to the question: Is χ(G) ≤ k?
begin
  mark all nodes as unfinished
  for all leaves X_i put the unique 1-coloring φ
    of G_i: φ_i = {{v}} into Tab_i and mark X_i as finished
  while exists an unfinished node X_i with all children finished do
    begin
      case X_i of
        — introduce node with child X_j and v = X_i \ X_j
          then store in Tab_i all extensions φ_i of φ_j ∈ Tab_j onto G_i
            where no neighbor of v is in the same color class as v
            and φ_i has at most k color classes
        — forget node with child X_j,
          then store in Tab_i all φ_j ∈ Tab_j restricted to X_i
            and remove duplicate entries
        — join node with children X_j, X_j'
          then Tab_i = Tab_j ∩ Tab_j'
        esac
      mark X_i as finished
    end;
  answer "YES" if Tab_root ≠ ∅ and "NO" otherwise.
end.
```

Exercise 52: Modify the algorithm such that it finds $\chi(G)$ and outputs also an optimal coloring of G.

Solution: First compute the chromatic number from the botom-up manner and then traverse the tree in the opposite direction and decompose the partial solution of each parent into partial solutions of its children.

4.3 General scheme and MSOL

A general scheme for the design of an algorithm for graphs of bounded treewidth can be formalized as follows [5]: denote first a *k-terminal graph* a graph with distinguished k-tuple of vertices. For two terminal graphs H, H' we define $H \oplus H'$ as the disjoint union of H and H' where the corresponding elements of the distinguished k-tuple are merged together. Now for a given decision problem (and the associated predicate $P(G)$) define relations $sol(G, s)$, $psol(H, s)$, $ext(H, s, H', s')$ and a function $ch(H, s)$ which capture the following situations: $sol(G, s)$ means that a solution string s satisfies the given problem on G, i.e.

$$\forall G : P(G) \Leftrightarrow \exists s : sol(G, s)$$

The relation $psol(H, s)$ stands for a similar situation when s is a partial solution for a terminal graph H. We write $ext(G, s, H, s')$ when H is a terminal subgraph of G and s' is the restriction of the solution string s on H. It has to satisfy

$$\forall G, s, H, H' : (sol(G, s) \wedge G = H \oplus H') \Rightarrow \exists s' : psol(H, s') \wedge ext(G, s, H, s')$$

Finally, $ch(H, s)$ is a characteristic function that essentially captures the differences of possible solutions. It has to satisfy the following statement for all H, H', K and s, s':

$$ch(H, s) = ch(H', s') \Rightarrow [(\exists t : ext(H \oplus K, t, H, s)) \Leftrightarrow (\exists t' : ext(H' \oplus K, t', H', s'))]$$

Moreover we require $ch(H, s)$ to have finite range for a given problem.

In this setting we can define an equivalence relation on k-terminal graphs

$$H \sim_{P,k} H' \Leftrightarrow (\forall K : P(H \oplus K) \Leftrightarrow P(H' \oplus K)).$$

This equivalence relation groups those terminal graph graph which cannot be distinguished by possible solutions on their extensions, and essentially capture the same structure as the characteristic function: the graphs equivalent under $\sim_{P,k}$ should be of the same characteristic and vice versa. We mention here that it can be proven:

Proposition 4.2 *If the graph predicate P impose finitely many equivalence classes in $\sim_{P,k}$ for any k, then the corresponding decision problem can be solved in linear time on graphs of bounded treewidth.*

Observe that if $H \sim_{P,k} H'$ and $|V'_H| < |V_H|$, we can reduce the size of an instance $H \oplus K$ by replacing it with $H' \oplus K$ without changing the value of the predicate P. Such a replacement is is called a *reduction rule*. One can show that for graphs of bounded treewidth and predicates P that allow only finitely many equivalence classes (for any k) there exists a finite set of reduction rules that can be used in a design of a linear-time algorithm.

So, the natural question is to recognize those graph predicates P that allow only finitely many equivalence classes on k-terminal graphs. A positive answer for a variety of graph problems brings the notion of Monadic Second Order Logic (MSOL). Formulas belonging to MSOL may contain:

- variables for vertices, edges and vertex sets and edge sets that can be quantified over V_G, E_G, $\mathcal{P}(V_G)$ and $\mathcal{P}(E_G)$, resp.
- the inclusion $\in$ predicate
  — it can be extended to the adjacency predicate
  $(u,v) \in E_G \Leftrightarrow \exists e \in E_G : (u \in e) \wedge (v \in e)$
  — and also to the equality predicates, e.g.
  $u = v \Leftrightarrow \forall V \subseteq V_G : (u \in V) \Leftrightarrow (v \in V)$
- the usual logical operations $\neg, \wedge, \vee, \Leftrightarrow, \Rightarrow \ldots$.

Example: The property that a graph has an Hamiltonian circuit can be expressed as follows:

$$
\begin{aligned}
&\exists E \subseteq E_G : \\
&[\forall u \in V_G\, \exists v, w \in V_G : (v \neq w) \wedge ((u,v) \in E_G) \wedge ((u,w) \in E_G) \wedge \\
&\qquad \wedge [\forall x \in V_G : (u,x) \in E_G \Rightarrow ((x = v) \vee (x = w))]] \\
&\qquad\qquad\qquad \ldots \textit{E is a 2-factor} \\
&\wedge [\forall V \subseteq V_G : ((V \neq V_G) \wedge (\exists u \in V_G : u \in V)) \Rightarrow \\
&\qquad \Rightarrow (\exists v, w \in V_G : (v,w) \in E_G \wedge (v \in V) \wedge (w \notin V))] \\
&\qquad\qquad\qquad \ldots \textit{and it is connected}
\end{aligned}
$$

Finally note that for such a formula a finite tree automaton can be constructed, which accepts those tree decompositions of graphs that satisfy the given formula. The state set of the tree automaton more-or-less corresponds to the possible contents of $\mathtt{Tab}_i$ and the transition function describes the computation of the parent database from the children's databases.

Exercises

Exercise 53: Decide whether the problem $\chi(G^2) \leq k$ is expressible in MSOL and if so, construct a linear-time algorithm for graphs of bounded treewidth.

Solution: Yes, test the existence of k vertex disjoint color sets and then check that no two elements of the same class are at distance at most two.

Exercise 54: Design an algorithm for testing whether a graph of fixed treewidth c is Hamiltonian, provided that the tree-decomposition of width at most c is a part of the input.

Solution: The partial solutions of a node are stored as a possible intersection of a Hamiltonian cycle through the node. The extension and restriction of a partial solution in introduce and forget nodes is straightforward. The union node should correctly combine two such intersections into a comon solution.

5 Some different notions of graph width

5.1 Treewidth versus pathwidth

The path decomposition of a graph is a tree decomposition where the underlying tree forms a path. The notion of pathwidth is defined analogously. Clearly $\mathrm{tw}(G) \leq \mathrm{pw}(G)$ for any graph G, because a path decomposition can be viewed as a tree decomposition. In the opposite direction the graphs with bounded treewidth may attain arbitrarily high pathwidth.

Exercise 55: For every k find a tree with pathwidth at least k.

Solution: Consider a rooted ternary tree T_k with $k+1$ levels (i.e. each inner vertex has three children). It can be proven by induction that $\mathrm{pw}(T_k) \geq k$ as follows: Let P be a path decomposition of $T = T_k$ and let T_1, T_2 and T_3 be the three trees isomorphic to T_{k-1} rooted at the three chidren of the root of T. Without loss of generality we may assume that the leftmost node of P contains some vertices of T_1 and that the rightmost node contains some vertices of T_1 or of T_3. Then all nodes of P that contain some vertices of T_2 incude also some vertex of $T \setminus T_2$, hence $\mathrm{pw}(T) \geq \mathrm{pw}(T_2) + 1$.

Theorem 5.1 *For an arbitrary graph G on n vertices it holds that $\mathrm{pw}(G) = O(\mathrm{tw}(G)\log n)$ and this upper bound is asymptotically tight.*

Proof: Consider a rooted tree decomposition T of G of width $\mathrm{tw}(G)$ which have at most n nodes and where no node is a subset of another. Such a decomposition can be derived from an embedding of G into a k-tree.

We first define an auxiliary labeling of the tree T as follows: Put 1 as the label for every leaf of T. If an internal vertex has only one child, we label it by the child's label. If an internal vertex has more children we select as the label of the parent the second highest label among its children and increase it by one. Observe that the root has label at most $\log_2(n)$. This can be proved by induction, because below a vertex of label i are at least 2^i leaves.

We now define a path decomposition P according to the labeling of T recursively. For any subtree T_i rooted in a node X_i we define a path decomposition of G_i, i.e. of the subgraph of G induced by the vertices of $\bigcup_{X \in T_i}$.

The path decomposition has the property that X_i is the last node of the path decomposition.

A leaf X_i of T is viewed as a single-node path decomposition of G_i. If a node X_i has only one children X_j we take the path decomposition P_j and extend it by the node X_i. The remaining case corresponds to the situation when a node X_i has more children. Let X_j be the child of the highest label of X_j and let the other children be $X_{j'}, X_{j''}, \ldots$. We construct the path decomposition P_i as the concatenation of $P_j \circ P' \circ P'' \circ \cdots \circ X_i$, where $P^{(s)}$ is the path decomposition of $G_{j^{(s)'}}$, where each node is extended by the set X_i.

Observe that the above defined labeling of X_i indicates how many times were extended nodes in P_i. This concludes the argument for $\mathrm{pw}(G) = O(\mathrm{tw}(G) \log n)$.

The asymptotics follows straightforwardly from the previous exercise, since $\mathrm{pw}(T_k) \geq k = \lceil \log_3 |V_{T_k}| \rceil - 1$. □

Observe that graphs of pathwidth at most k are exactly those hraphs where $k+1$ police helicopters can capture an invisible robber in the LaPaugh's game. (I.e. police heicopters move without knowing the robber's position. The game ends when the robber cannot move.).

5.2 Treewidth versus branchwidth

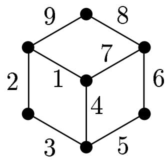
The notion of branchwidth in fact preceded the notion of treewidth. It is defined as follows:

Definition: *A* branch decomposition *of a graph G with at least two edges is a tree B where each internal node has degree three and leaves of B are in one-to-one correspondence with edges of G. Each edge $f \in E_B$ splits E_G into the two sets E_f and $E'_f = E_G \setminus E_f$ where the edges of E_f are indicated by the leaves of one of the two components of $B \setminus f$.*

The width of an edge f is the number of vertices that appear both in E_f and in E'_f as an endvertex of some edge, i.e. formally $w(f) = |V_f| = |\{(\bigcup E_f) \cap (\bigcup E'_f)\}|$. The width of a branch decomposition B is the maximum width among all its edges and the branchwidth *of G, denoted by* $\mathrm{bw}(G)$, *is the minimum width out of all possible branch decompositions of G.*

Tbe branchwidth of a graph with at most one edge is considered to be 0.

Exercise 56: Characterize graphs of branchwidth one.



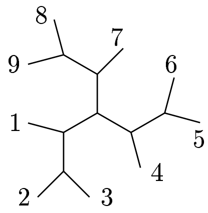


Figure 10: An example of a graph and its branch decomposition of width 3.

Solution: Disjoint union of stars. A vertex of degree at least two must appear, but each edge must have at least one of its two ends of degree one.

Exercise 57: For each tree, construct a branch decomposition of width at most two.

Solution: Take the original tree, and add to each edge a leaf as its representative. Finally replace each vertex of degree at least four by a ternary tree and eliminate vertices of degree two.

Exercise 58: Show that $\mathrm{bw}(H) \leq \mathrm{bw}(G)$ whenever H is a minor of G.

Solution: The vertex and edge removal cannot increase branchwidth, hence $\mathrm{bw}(H) \leq \mathrm{bw}(G)$ if H is a subgraph of G.

Now consider the case that an edge $e = (u, v)$ was contracted into the vertex w. In B we remove the leaf for e and eliminate the adjacent vertex of degree two to get the branch decomposition B' of $G' = G \circ e$.

The vertex w may newly appear as the intersection of previously disjoint edges, e.g. some incident with u and other incident with v. But if the vertex w appears in some V'_f for an edge $f \in E_{B'}$, we show that V_f contained before the elimination either u or v: Let A, B be the two components of $B \setminus f$ and without loss of generality assume that the leaf for e belongs to A. Then if $w \in V'_f$ then some edge e' incident either with u or v appears in B. But then also either u or v belong to V_f, so $|V'_f| \leq |V_f|$.

Exercise 59: Show that the branchwidth of the complete graph K_n on at least 3 vertices is at least $\lceil \frac{2}{3}n \rceil$.

Solution: Observe that for any $u \in V_G$ the set $\{f : u \in V_f\}$ induces a connected subtree. For $G = K_n$ each pair of such subtrees must share an edge of B (e.g. the edge incident with the leaf containing the edge on the representatives of the subtrees).

Then there is an internal node of B contained in all tress. At least one of the incident edges has width at least $\lceil \frac{2}{3}n \rceil$.

Theorem 5.2 *For any graph G of branchwidth at least two it holds that* $\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1 \leq \lfloor \frac{3}{2}\mathrm{bw}(G) \rfloor$.

Proof: Let T be a tree decomposition of G of width $\mathrm{tw}(G)$. For each edge $e \in E_G$ select a node X and join X with an exposed leaf representing e.

We now take the smallest subtree containing all the exposed leaves and contract all vertices of degree 2 and possibly replace internal vertices of higher degree by ternary trees. We claim that this branch decomposition B is of width at most $\mathrm{tw}(G) + 1$.

If $f \in E_B$ is incident with a node X of the original tree decomposition then $V_f \subseteq X$, since other vertices are separated by X (from E_f or E'_f), and cannot contribute to V_f. This establish the first inequality.

Let B an arbitrary branch decomposition of G of width $\mathrm{bw}(G)$. We construct a tree decomposition $T = B$ of G as follows: If a leaf X_e corresponds in B to the edge e of G we let $X_e = e = \{u, v\}$. For each pair of edges $e, e' \in E_G$ with a common vertex u we insert u into all internal nodes on the unique path between X_e and $X_{e'}$. This is a tree decomposition of G of width at most $\lfloor \frac{3}{2}\mathrm{bw}(G) \rfloor$, because each vertex is inserted twice in each internal node. $\square$

5.3 Treewidth versus cliquewidth

The major disadvantage of all already discussed decompositions is that they cannot capture well some classes of graphs, especially when they contain large cliques, e.g. complete graphs or cographs.

This is possible in the notion of cliquewidth. Similarly as in the previous cases a graph is described in a tree-like manner.

Definition: *Let $I = \{1, \ldots, k\}$ be a set of labels. A* expression tree *of a graph G is a rooted tree E of the following form:*

- *The nodes of E are of the four types i, $\oplus$, η and ρ:*
- *Introduce nodes $i(v)$ are leaves of E, corresponding to a graph, with only one vertex v of label $i \in I$.*
- *A union node $\oplus$ stands for a disjoint union of the two graphs associated with its two children.*
- *A join node $\eta_{i,j}$, where $i, j \in I, i \neq j$ are associated with the graph resulting by the insertion of all edges between nodes of label i and the nodes of label j into the graph corresponding to the only child of the node.*
- *A relabel node $\rho_{i \to j}$ is associated with the graph of its only child, where in addition each vertex label i is replaced by the label j.*
- *The graph G is isomorphic to the graph associated with the root of T with all vertex labels removed.*

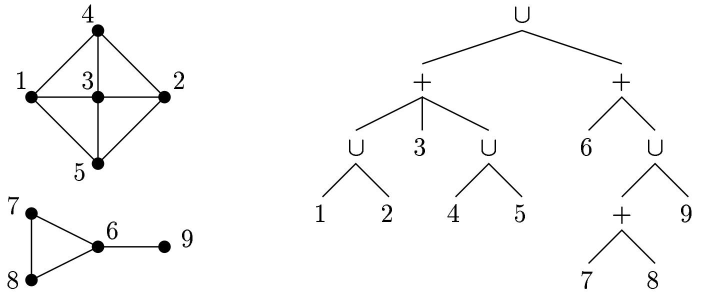


Figure 11: An example of a cograph and its cotree. Union nodes are indicated by $\cup$, while join nodes are marked by $+$.

The size of the minimum set of labels necessary to construct an expression tree for a graph G is denoted by $\mathrm{cw}(G)$ *and is called the* cliquewidth *of G.*

Exercise 60: Determine the cliquewidth of trees and of complete graphs.

Solution: Complete graphs can be constructed with two labels: whenever we add a new vertex we give it the label different from the rest of the graph, then we add all edges and finally relabel it such that all vertices of the complete graph have the same label.

Each rooted tree can be constructed inductively by use of three labels as follows: Assume that each smaller tree can be constructed such that the root has label 1 and the other vertices have label 3. We decompose a given tree T into two smaller trees - one is T', the tree rooted in a child of the root of T and the other tree T'' is the complement of T' in T. We construct the expression trees of T' and T'' by induction and relabel the root of T' with the label 2. Now take the disjoint union of E' and E'', add the missing edge ($\eta_{1,2}$) and finally relabel the auxiliary root of T' ($\rho_{2\to 3}$).

Exercise 61: Show that graph of cliquewidth at most 2 are exactly those graphs with no induced path of length four (so called complement-reducible graphs or cographs).

Solution: Each cograph can be associated with its cograph expression tree (shortly cotree) — where leaves represent vertices and internal nodes represent either disjoint union operation or join operation (see Fig. 11). The transformations between a cotree and an expression tree of width 2 is then straightforward.

We show that graphs of bounded treewidth have bounded cliquewidth.

Theorem 5.3 *For any graph G it holds that* $\mathrm{cw}(G) \leq 2^{\mathrm{tw}(G)+1} + \mathrm{tw}(G)$.

Proof: Let a graph G have a nice tree decomposition T of width k with root node X_r. We construct by induction an expression tree E that can be split into three parts:

- a subtree E_r for G_r, the subgraph of G induced by X_r,
- a subtree E' corresponding to $G' = G \setminus G_r$,

- a union node above E_r and E' followed by a chain of η nodes describing the edges between X_r and $V_G \setminus X_r$.

The labels on E are distributed as follows: vertices on G_r have $k+1$ distinct labels, different from the labels used on G'. Moreover two vertices in G' have the same label iff they have the same set of neighbors in G_r.

In total we need $k+1$ labels for G_r and at most $2^{k+1}-1$ labels for G'.

We now mimic the construction of E as was shown above for the case of trees. If X_r is

- the leaf node corresponding to a vertex v, we take $E = \{1(v)\}$,
- the introduce node, we modify only E_r (that can be done with at most $k+1$ labels),
- the forget node for a vertex v — we remove v from E_r, and extend E' such that v is:
  - introduced with an extra new label,
  - united with E' by use of the $\oplus$ operation,
  - connected to all its neighbors in G' by a proper η operation,
  - relabeled with respect its neighbors in the new X_r.

  We finally merge some labels in G' for the vertices which have the neighborhood in X_r but for v.
- the union node with respect to subtrees T' and T''. Observe that $V_{G'}$ must be disjoint from $V_{G''}$. Without loss of generality we may assume that vertices of G' and G'' have the same label iff they have the same set of neighbors in G_r. Then we place $E' \oplus E''$ below the union node with E_r (see Fig. 12).

In total we use $2^{k+1}-1+k+1$ labels and the claim follows. □

It is possible to construct graphs whose cliquewidth is exponential in treewidth. An example of a k-tree with cliquewidth at least $2^{\lfloor k/2 \rfloor -1}$ was shown in [6].

To conclude the study on cliquewidth we present a k-coloring algorithm for graphs of bounded cliquewidth. We recall the notation used for treewidth-bounded graphs: X_l stands for a node of the expression tree and G_l is the

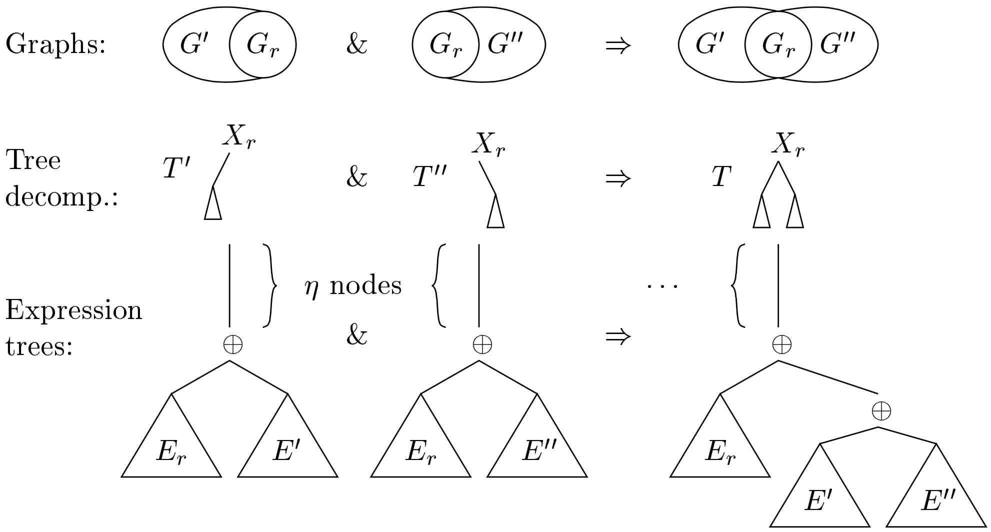


Figure 12: Transformation of a tree decomposition into an expression tree — union node.

induced subgraph of G induced by vertices of the introduce nodes which are descendants of X_l. Observe that the graph G_l has expression tree E_l composed by two parts: the subtree of E induced by all descendants of X_l together with the unique path from X_l to the root. Finally denote $E^{i,j}(G_l)$ the set of edges of G_l between vertices of labels i and j.

Consider a coloring c of a graph G_i, and let S be a color class of c, i.e. a set of vertices of G_i of the same color. We say that S is labeled by $J \subseteq I$, if J is exactly the set of labels used on S, i.e. if every label of J appears on some vertex of S and no vertex of S has a label outside J.

We again involve dynamic programming and store a table $\texttt{Tab}_l$ in each node X_l of the expression tree. Each element $\mathbf{x} \in \texttt{Tab}_l$ is an integer vector of length $2^{|I|}-1$ and describes a group of possible colorings of G_l. The entries of $\mathbf{x}$ are indexed by nonempty subsets $J \subseteq I$ and the value of $\mathbf{x}_J$ is the number of color classes which are labeled by J in a coloring corresponding to $\mathbf{x}$.

Observe that the table size is bounded by $(k+1)^{2^{|I|}-1}$. The algorithm for evaluating $\texttt{Tab}_i$ follows.

Its time complexity is asymptotically determined by solving the system of equations which is done for exactly $n-1$ union nodes. We have to test each of $t = (k+1)^{2^{|I|}-1}$ possible solutions $\mathbf{x}$. The feasibility of the system of equations can be tested e.g. by Gaussian elimination in time $O(t^6)$, since the number of variables $a_{J,J'}$ is bounded by t^2.

k-COLORABILITY
Input: *An expression tree E of a graph G of fixed width $|I|$.*
Output: *Answer to the question: Is $\chi(G) \leq k$?*

```
begin
  mark all nodes as unfinished
  for all leaves X_l = i(v) store in Tab_l the unique vector x:
    x_{i} = 1 and x_J = 0 for all J ≠ {i} and mark X_l as finished
  while exists an unfinished node X_i with all children finished do
    begin
      case X_l of
        — relabel node ρ_{i→j} with child X_m
          then for every y ∈ Tab_m store in Tab_l the vector x:
            x_J = 0 if i ∈ J
            x_J = y_J + y_{J∪{i}} if j ∈ J, i ∉ J
            x_J = y_J if i, j ∉ J
        — join node η_{i,j} with child X_m,
          then set Tab_l = Tab_m
        — union node with children X_m, X_m'
          then for each y ∈ Tab_m, z ∈ Tab_m' store in Tab_l all x
            for which the following set of equations has a solution:
              ∀J : Σ_{J'} a_{J,J'} = y_J
              ∀J' : Σ_J a_{J,J'} = z_{J'}
              ∀K : Σ_{J∪J'=K} a_{J,J'} = x_K
              ∀J, J' s.t. (∃i ∈ J, j ∈ J' ∧ E^{i,j}(G_l) ≠ ∅) : a_{J,J'} = 0
              ∀J, J' : a_{J,J'} ∈ {0, 1, ..., k}
        esac
      mark X_l as finished
    end;
  answer "YES" if Tab_root ≠ ∅ and "NO" otherwise.
end.
```

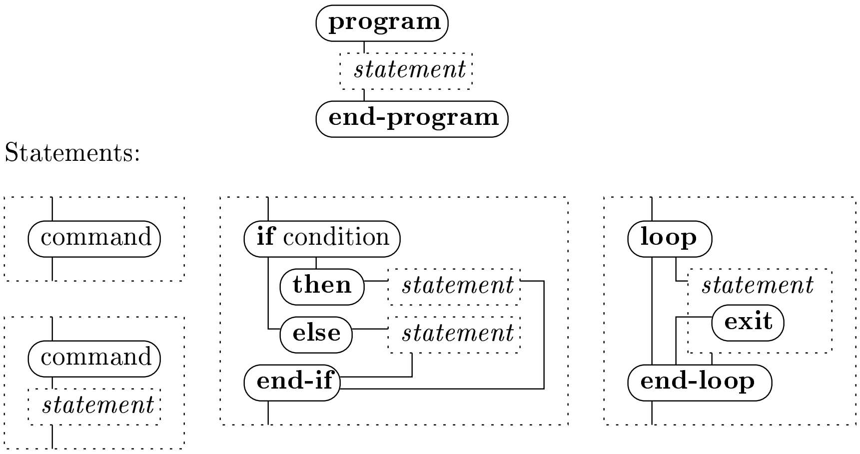


Figure 13: Program constructs and control flow diagrams.

6 Other applications of treewidth

6.1 Treewidth of flow control graphs and register allocation

As a practical application of the notion of treewidth we reproduce here a result on flow control graphs and register allocation [15].

Consider a simple programming language with **if** – **then** – **else** – **end-if** and **loop** – **exit** – **end-loop** constructs.

Roughly speaking, the control flow graph models the possible order of processing statements, i.e. those statements which might be processed contiguously are joined by a (directed) edge. For our purposes we assume that the program constructs and segments of the corresponding control flow graph are defined by the diagrams depicted in Fig. 13.

It is not hard to see that without the keyword **exit** the control flow graph of any program in our language is a series-parallel graph, hence of treewidth 2. The keyword **exit** deserves further explanation: It may appear within any statement inside the **loop** – **end-loop** cycle in a position of a command. In fact more **exit**s may appear there, but each passes the computation to the closest subsequent **end-loop**, as indicated in the figure.

Exercise 62: Prove that if on the program the keyword **exit** appears at most once, then the treewidth of the control flow graph is at most three.

In fact a stronger statement holds.

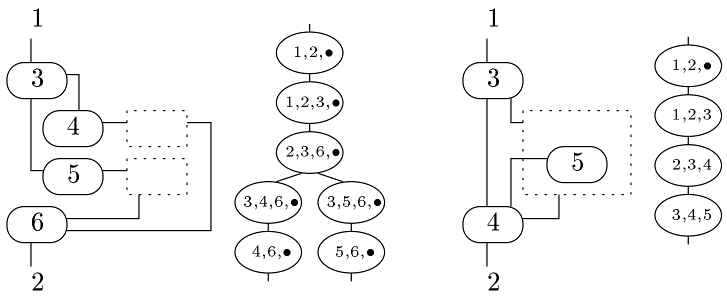


Figure 14: From control flow graph to tree decomposition.

Proposition 6.1 *The treewidth of the control flow graph for any program written in our simple language is bounded by 3.*

Exercise 63: Prove Proposition 6.1.

Solution: Proof: The tree decomposition T can be deduced directly from the control flow graph, see Fig. 14 for an example. For each statement we put in T a node containing the entry and exit point to the statement and also the **end-loop** keyword for possible **exit** jumps. These are in the figure marked by •. Observe that the validity of a certain **exit** jump ends with a new **loop** − **end-loop** cycle. The other three cases can be handled similarly, and are in fact even simpler. □

For widely used structured programming languages, the following result was obtained in [15]:

Theorem 6.2 *The treewidth of control flow graphs is bounded by 2 for* **goto**-*free Algol and Pascal, by 4 for Modula-2 and by 5 for* **goto**-*free C programs.*

The importance of these control flow graphs and their treewidth is closely related to the register allocation problem. Observe that every variable that might appear in our program will be used within connected part of the the control flow graph. Each variable has to be assigned a register, and no two variables may share the same register. Hence we may define the intersection graph with variables as vertices. Two vertices form an edge, if they are in use at the same time, i.e. if the two parts of the control flow graphs corresponding to use of these two variables intersect. Then, the minimum number of registers needed for execution of the program is equal to the chromatic number of the intersection graph. Straightforwardly, any proper vertex coloring corresponds to a feasible register assignment to variables.

Observe, that the fact that the control flow graph has bounded treewidth does *not* imply the same for the intersection graph. However, unlike general graph coloring problem that is hard to approximate (and decide for a fixed number of colors), under this assumption there were designed fast approximate algorithms for arbitrary number of registers [15] or linear-time exact (decision) algorithms for fixed number of registers [3].

References

[1] Arnborg, S., Corneil, D. G., and Proskurowski, A. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods 8* (1987), 277–284.

[2] Arnborg, S., Proskurowski, A., and Corneil, D. Forbidden minors characterization of partial 3-trees. *Discrete Math. 80* (1990), 1–19.

[3] Bodlaender, H., Gustedt, J., and Telle, J. A. Linear-time register allocation for a fixed number of registers. In *ACM-SIAM Symposium on Discrete Algorithms, 9th SODA'98* (1998), pp. 574–583.

[4] Bodlaender, H. L. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput. 25*, 6 (1996), 1305–1317.

[5] Bodlaender, H. L. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations in Computer Science, MFCS'97* (1997), no. 1295 in Lecture Notes in Computer Science, Springer Verlag, pp. 19–36.

[6] Corneil, D. C., and Rotics, U. On the relationship between clique-width and treewidth (extended abstract). In *Graph-Theoretical Concepts in Computer Science, 27th WG '01, Boltenhagen* (2001), no. 2204 in Lecture Notes in Computer Science, Springer Verlag, pp. 78–90.

[7] Diestel, R. *Graph Theory*, graduate texts in mathematics ed., vol. 173. Springer-Verlag, May 1997. chapt. 12.

[8] Kloks, T. *Treewidth. Computations and approximations.* No. 842 in Lecture Notes in Computer Science. Springer Verlag, 1994.

[9] Kuratowski, K. Sur le probleme des courbes gauches en topologie. *Fund. Math. 15* (1930), 271–283.

[10] Lipton, R. J., and Tarjan, R. E. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36* (1979), 177–189.

[11] Reed, B. Rooted routing. manuscript, 2000.

[12] Reed, B. A. Finding approximate separators and computing tree width quickly. In *ACM Symposium on Theory of Computing 24th STOC '92, Victoria, British Columbia, Canada* (1992), ACM, pp. 221–228.

[13] Seymour, P., and Thomas, R. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B 58*, 1 (1993), 22–33.

[14] Thomas, R. Tree decompositions of graphs. lecture notes, 1996.

[15] Thorup, M. Structured programs have small tree-width and good register allocation. In *Graph Theoretic Concepts in Computer Science, 23rd WG'97* (1997), no. 1335 in Lecture Notes in Computer Science, Springer Verlag, pp. 318–332.