

Tree Decompositions, Algorithms and Logic

Summer Term 2014

Isolde Adler
Institut for Computer Science
Goethe University Frankfurt

Contents

1. Introduction	3
1.1. Historical remarks and structure of the lecture	3
1.2. Algorithms on trees	4
2. Tree Decompositions	9
2.1. Definition, Basic Properties and Examples	9
2.2. Tree Width and Separators	17
2.3. Characterising Tree Width	19
3. Cops and Robbers	27
3.1. Cops and Robbers	27
3.2. Brambles	29
3.3. Monotonicity of the Cops-and-Robber Game	31
4. Computing Tree Decompositions	37
4.1. Tree Decompositions from Winning Strategies	38
4.2. Tree decompositions from separators	39
5. Algorithms on Tree Decompositions	45
5.1. Dynamic programming on graphs of bounded tree width	45
6. Monadic Second Order Logic	51
6.1. Monadic second order logic	51
6.2. MSO on trees	55
7. Courcelle's Theorem	61
7.1. Coding Tree Decompositions in Labelled Trees	61
7.2. Courcelle's Theorem	67
7.3. Seese's Theorem	69
A. Graphs: Basic notions and terminology	i
Appendix	iii

Foreword

This lecture is the latest revised version of a number of lectures I gave on similar topics. It includes recent new developements, and I experiment with the presentation, taking into account my previous experience in teaching this subject.

The lecture incorporates material from a joint lecture with Stephan Kreutzer, held in 2007 at Humboldt University Berlin, and ideas from related – very well elaborated – lectures by Martin Grohe.

Comments, typos etc. are welcome! Thank you to Philipp Krause and Lukas Larisch for comments and pointing out typos.

1. Introduction

1.1. Historical remarks and structure of the lecture

Many important algorithmic problems are NP-hard in general, but they nevertheless need to be solved in practice. In many cases this is possible because real-life instances have a ‘tree-like’ structure. The notion of ‘tree decomposition’ can make this precise and explain the phenomenon.

The tree width of a graph measures how close the graph is to being a tree. The smaller the tree width the closer the graph is to being a tree (trees have tree width 1).

The notion of tree width was introduced 1986 by Robertson and Seymour in the course of their famous proof of Wagner’s Conjecture. Indeed, the notion was independently introduced by at least two other groups of researchers (under different names).

- N. Robertson and P. D. Seymour: *tree width*.
(... the smallest non-negative integer k , such that G has a tree decomposition into bags of size at most k .)
- D. Rose, S. Arnborg, A. Proskurowski et al.: *Partial k -trees*.
(... the smallest non-negative integer k , such that G is a subgraph of a k -tree.)
- R. Halin: *Simplicial Decompositions*.
(... the smallest non-negative integer k , such that G has a triangulation without cliques of size $(k + 2)$.)

We will study these definitions in the first part of this lecture. In addition, we will give further characterizations of tree width, e.g. by a cops and robber game on graphs, to provide a solid understanding of tree width.

Meanwhile, tree width has numerous applications in different areas of computer science, such as Databases, Algorithm Design, AI, Bioinformatics, and Compiler Construction. We will discuss some of these applications in this lecture.

Is there an easy way to tell, whether a computational problem can be solved efficiently on tree-like graphs? Indeed, Courcelle’s prototypical metatheorem shows how monadic second order logic can help us here. In the second part of this lecture we will introduce monadic second order logic and prove Courcelle’s Theorem. Hereafter we will discuss some of its numerous applications and consequences.

Parts of these lecture notes are taken from a lecture by Martin Grohe (RWTH Aachen) and from a joint lecture with Stephan Kreutzer (TU Berlin).

1.2. Algorithms on trees

In this section we show that many NP-hard problems on graphs can be solved efficiently when we restrict them to instances that are trees.

The basic notions we use are defined in the appendix A.0.1 (and on the slides of the first lecture).

Definition 1.1 Let G be a graph. A subset $X \subseteq V(G)$ is a *clique* (in G), if for every two vertices $x, y \in X$ with $x \neq y$ we have $\{x, y\} \in E(G)$. If $V(G)$ is a clique we also (more simply) say that G is a clique.

CLIQUE

Input: Graph G , $k \in \mathbb{N}$.

Problem: Does G contain a clique of size $\geq k$?

Definition 1.2 Let G be a graph. A subset $X \subseteq V(G)$ is *independent* (in G), if for every two vertices $x, y \in X$ we have $\{x, y\} \notin E(G)$.

INDEPENDENTSET

Input: Graph G , $k \in \mathbb{N}$.

Problem: Does G contain an independent set of size $\geq k$?

Definition 1.3 Let $G = (V, E)$ be a graph. The graph $\overline{G} := (V, [V]^2 \setminus E)$ is called *complement* of G .

Remark 1.4 Let G be a graph. A subset $X \subseteq V(G)$ is a *clique* on G if and only if X is an *independent* set in \overline{G} .

Definition 1.5 Let $G = (V, E)$ be a graph and let $k \in \mathbb{N}$. A k -coloring of G is a mapping $f: V \rightarrow \{1, \dots, k\}$, such that all edges $\{x, y\} \in E(G)$ satisfy $f(x) \neq f(y)$. A coloring of G is a k -coloring of G for some $k \in \mathbb{N}$. If G has a k -coloring, we say that G is k -colorable.

COL

Input: Graph G , $k \in \mathbb{N}$.

Problem: Is G k -colorable?

Definition 1.6 Let G be a graph. A cycle in G that visits each vertex of G exactly once is called a *Hamilton cycle* in G .

HAM (HAMILTON CYCLE)
Input: Graph G .
Problem: Does G contain a Hamilton cycle?

Definition 1.7 Let G be a graph. A subset $X \subseteq V(G)$ is a *vertex cover* (of G), if every edge $\{x, y\} \in E(G)$ satisfies $\{x, y\} \cap X \neq \emptyset$.

VERTEXCOVER
Input: Graph G , $k \in \mathbb{N}$.
Problem: Does G have a vertex cover of size $\leq k$?

Remark 1.8 Let G be a graph and $X \subseteq V(G)$ a vertex cover. Then $V(G) \setminus X$ is an independent set.

Definition 1.9 Let G be a graph. A subset $X \subseteq V(G)$ is a *feedback vertex set* (of G), if $G \setminus X$ does not contain a cycle.

FVS (FEEDBACK VERTEX SET)
Input: Graph G , $k \in \mathbb{N}$.
Problem: Does G contain a feedback vertex set of size $\leq k$?

Theorem 1.10 (Karp 1972) The problems CLIQUE, INDEPENDENTSET, COL, HAM, VERTEXCOVER and FVS are NP-complete.

Theorem 1.11 If we restrict the inputs to trees, the problems CLIQUE, INDEPENDENTSET, COL, HAM, VERTEXCOVER and FVS are solvable in time polynomial (even linear) in the size of the vertex set of the input graph.

Proof.

- CLIQUE, COL, HAM und FVS sind trivial auf Bäumen.
- INDEPENDENTSET: cf. Corollary 1.13
- VERTEXCOVER: cf. Exercise 2, sheet 1.

□

The algorithmic problems that we saw so far are *decision problems* (their answer is either ‘yes’ or ‘no’). Many of these problems come with an optimization variant. Here the task is to compute subsets with certain properties of optimal size (maximum or minimum – depending on the problem).

MAXIS (MAXIMUM INDEPENDENT SET)
Input: Graph G .
Problem: Compute an independent set in G of maximum size.

Theorem 1.12 *On trees, MAXIS is solvable in linear time (linear in the number of vertices/edges).*

Proof. We root the input tree T arbitrarily. We describe an algorithm \mathcal{A} that works bottom-up along T . Starting from the leaves of T , for all subtrees, algorithm \mathcal{A} computes the partial independent sets and remembers the ones of minimum cardinality. At every node $u \in V(T)$ we distinguish two cases.

Case 1: u is contained in the partial minimum independent set of T .

Case 2: u is not contained in the partial minimum independent set of T .

Algorithm \mathcal{A}

Step 1: Choose a node $w \in V(T)$ as root.

Step 2: For all leaves u of T let:

$$\begin{aligned} i_0(u) &:= 0 & I_0(u) &:= \emptyset \\ i_1(u) &:= 1 & I_1(u) &:= \{u\} \end{aligned}$$

Step 3. For all nodes $u \in V(T)$ with children v_1, \dots, v_k let

- $i_0(u) := \sum_{j=1}^k \max\{i_0(v_j), i_1(v_j)\}$
// Size of a maximum independent set in T_u , *not* containing u
- $i_1(u) := 1 + \sum_{j=1}^k i_0(v_j)$
// Size of a maximum independent set in T_u , containing u
- $I_0(u) := \bigcup_{j=1}^k I(v_j)$, where

$$I(v_j) = \begin{cases} I_0(v_j) & \text{if } i_0(v_j) \geq i_1(v_j), \\ I_1(v_j) & \text{otherwise.} \end{cases}$$

// maximum independent set in T_u , *not* containing u

- $I_1(u) := \bigcup_{j=1}^k I_0(v_j) \cup \{u\}$
// maximum independent set in T_u , containing u

Step 4: If $u = w$: if $i_0(w) \geq i_1(w)$, return $I_0(w)$, otherwise return $I_1(w)$.

Correctness of \mathcal{A}

Claim: Every node $u \in V(T)$ satisfies

- (1) $I_0(u)$ is an independent set in T_u that does not contain u and $I_0(u)$ is maximum with these properties.
- (2) $I_1(u)$ is an independent set in T_u that contains u , and $I_1(u)$ is maximum with these properties.

It is easy to see that we are done, once the claim is proven. (Why is this true?) We prove it by induction from the leaves to the root.

If u is a leaf, the claim holds by definition (cf. Step 2). Let u be a node with children v_1, \dots, v_k and assume the claim holds for v_1, \dots, v_k .

Part (1) of the claim:

$I_0(u)$ is an independent set in T_u , because by the inductive hypothesis the sets $I_0(v_j)$, $I_1(v_j)$ are independent sets in T_{v_j} (for $j = 1 \dots, k$) and because any two nodes $x \in V(T_{v_i})$ and $y \in V(T_{v_j})$ (for $i \neq j$) are not connected by an edge. By Definition (cf. Step 3) we have $u \notin I_0(u)$. The set $I_0(u)$ is maximum with these properties, because otherwise there would be an independent set $X \subseteq V(T_u)$ with $u \notin X$ such that $|I_0(u)| < |X|$. But then for every $j = 1, \dots, k$ the set $V(T_{v_j}) \cap X$ is independent in T_{v_j} , and there would be a $j \in \{1, \dots, k\}$ satisfying $|V(T_{v_j}) \cap X| > \max\{i_0(v_j), i_1(v_j)\}$, a contradiction to the inductive hypothesis.

Part (2) of the claim:

$I_1(u)$ is an independent set in T_u , because by the inductive hypothesis the sets $I_0(v_j)$ are independent in T_{v_j} (for $j = 1 \dots, k$) and because any two nodes $x \in V(T_{v_i})$ and $y \in V(T_{v_j})$ (for $i \neq j$) are not connected by an edge, and because the distance from u to every node in $I_0(v_j)$ is at least 2. By Definition (cf. Step 3) we have $u \in I_0(u)$. The set $I_1(u)$ is maximum with these properties, because otherwise there would be an independent set $X \subseteq V(T_u)$ with $u \in X$ such that $|I_1(u)| < |X|$. But then for every $j = 1, \dots, k$ the set $V(T_{v_j}) \cap X$ is independent in T_{v_j} and $v_j \notin V(T_{v_j}) \cap X$ and there would be a $j \in \{1, \dots, k\}$ satisfying $|V(T_{v_j}) \cap X| > i_0(v_j)$, a contradiction to the inductive hypothesis.

Runtime analysis of \mathcal{A}

Let $n := |V(G)|$. Step 1 can be done in time $\mathcal{O}(n)$.

Steps 2 and 3:

The Algorithm \mathcal{A} visits every node $u \neq w$ twice: First for computing the sums $i_0(u)$ and $i_1(u)$ and the sets $I_0(u)$ and $I_1(u)$, and the second time when \mathcal{A} computes $i_0(v)$, $i_1(v)$, $I_0(v)$ and $I_1(v)$ for the predecessor v of u . Every visit uses constant time. Altogether, these steps can be computed in time $\mathcal{O}(n)$. Step 4 can also be done in constant time. Hence algorithm \mathcal{A} has a total running time of $\mathcal{O}(n)$. \square

Corollary 1.13 *On trees, INDEPENDENTSET is solvable in linear time.*

Proof. Given a tree G and a $k \in \mathbb{N}$, we solve MAXIS and check whether the maximum independent set in G that we computed has size at least k . \square

2. Tree Decompositions

In this chapter we introduce the concept of the tree width of a graph, the central notion of this book. Tree decompositions and the associated tree width have been introduced by Robertson and Seymour [5] as a measure of similarity of a graph to being a tree. It has subsequently been shown to be equivalent to previously studied concepts such as partial k -trees and we will present some alternative characterisations below.

2.1. Definition, Basic Properties and Examples

For convenience, we denote edges $\{x, y\}$ by xy (and we continue to regard edges as 2-element subsets of the vertex set).

Definition 2.1 *A tree decomposition of a graph G is a pair (T, β) consisting of a tree T and a function $\beta: V(T) \rightarrow 2^{V(G)}$ associating with each node $t \in V(T)$ a set of vertices $\beta(t) \subseteq V(G)$ such that* *tree decomposition*

(T1) *for every edge $e \in E(G)$ there is a node $t \in V(T)$ with $e \subseteq \beta(t)$, and*

(T2) *for all $v \in V(G)$ the set* $\beta^{-1}(v)$

$$\beta^{-1}(v) := \{t \in V(T) : v \in \beta(t)\}$$

is non-empty and connected in T .

The width of (T, β) is defined as $w(T, \beta) := \max\{|\beta(t)| - 1 : t \in V(T)\}$ and the tree width of G is defined as the minimum width over all tree decompositions of G , i. e. *width of (T, β)*
tree width

$$\text{tw}(G) := \min\{w(T, \beta) \mid (T, \beta) \text{ is a tree decomposition of } G\}.$$

We refer to the sets $\beta(t)$ of a tree decomposition as *bags*. We say that a node $t \in V(T)$ *covers* the edge $e \in E(G)$, if $e \subseteq \beta(t)$. Similarly, we say that t *covers* $v \in V(G)$, if $v \in \beta(t)$. *bags*
cover

Example 2.2 *Figure 2.1 shows a graph with a corresponding tree decomposition of width 3.*

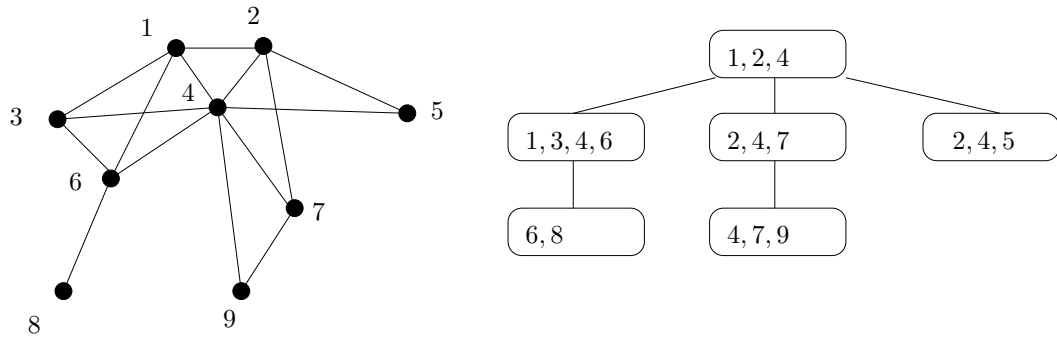


Figure 2.1.: A graph with a tree decomposition of width 3.

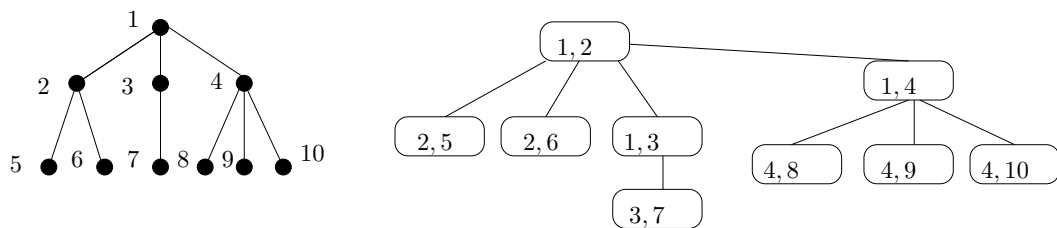


Figure 2.2.: A tree with a tree decomposition of width 1.

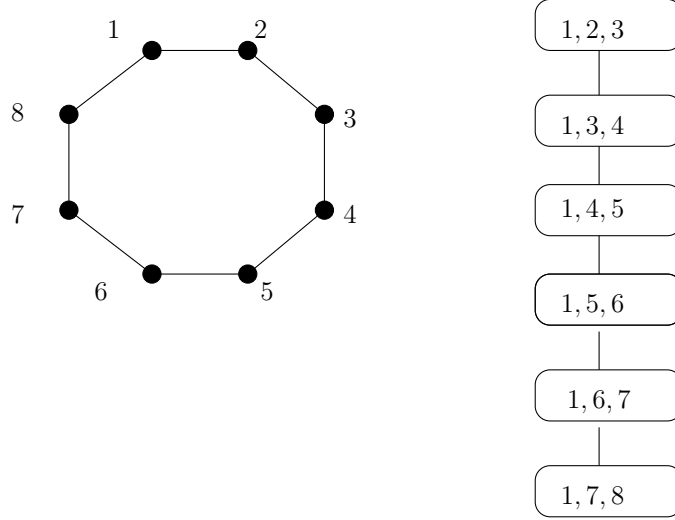


Figure 2.3.: A 6-cycle with a tree decomposition of width 2.

Example 2.3 *Trees have tree width at most 1. Given a tree T , the tree decomposition of T has a node t_e for each edge $e \in E(T)$ labelled by $\beta(t_e) := e$ and appropriate edges connecting the nodes following the tree shape of T (cf. Figure 2.2). \square*

Example 2.4 *Figure 2.3 shows a 6-cycle with a corresponding tree decomposition of width 2. Indeed, any cycle has tree width at most 2.*

Example 2.5 *Figure 2.4 shows a 6-cycle with an additional apex vertex (i. e. a vertex connected to all other vertices) with a tree decomposition of width 3. Indeed, any cycle with an additional apex vertex has tree width at most 3.*

Example 2.6 *Figure 2.5 shows a clique on four vertices with a corresponding tree decomposition of width 3. Indeed, any clique on n vertices has tree width at most $n - 1$.*

Remark 2.7 *Every graph G has the trivial tree decomposition (T, β) , where T consists of a single node t with bag $\beta(t) = V(G)$. Hence every graph G satisfies $\text{tw}(G) \leq |V(G)| - 1$.*

Let $m, n \in \mathbb{N}^+$. The $(m \times n)$ -grid is the graph $G_{m \times n} = (V, E)$ with $V := (m \times n)$ -grid $\{1, 2, \dots, m\} \times \{1, 2, \dots, n\}$ and

$$E := \{ \{(x, y), (z, w)\} \in [V]^2 \mid (x = z \text{ and } |y - w| = 1) \text{ or } (y = w \text{ and } |x - z| = 1) \}.$$

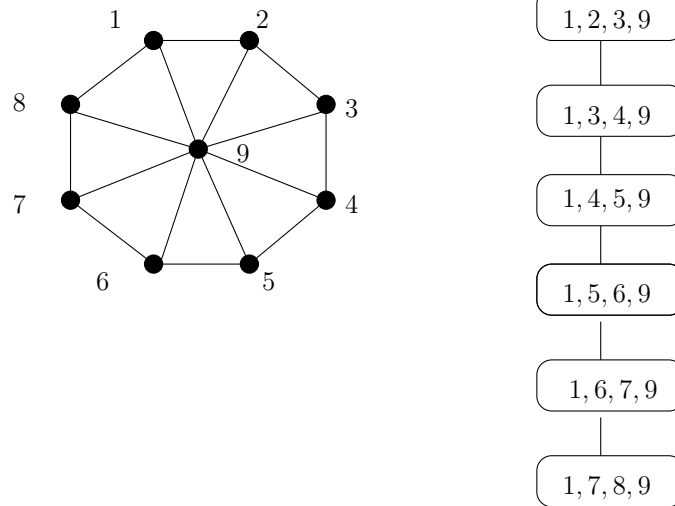


Figure 2.4.: A 6-cycle with an apex vertex with a tree decomposition of width 3.

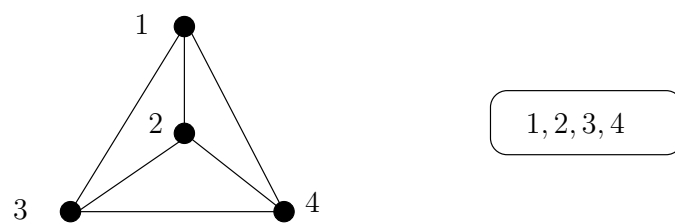
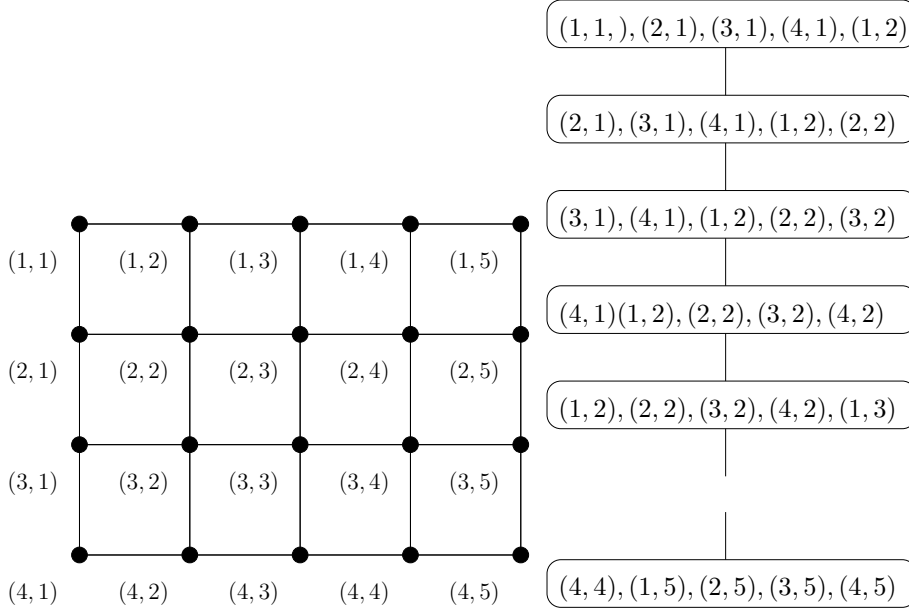


Figure 2.5.: A 4-clique with a tree decomposition of width 3.

Figure 2.6.: A (4×5) -grid with a tree decomposition of width 4.

Example 2.8 Figure 2.6 shows a (4×5) -grid $G_{4 \times 5}$ with a corresponding tree decomposition of width 4. Indeed, for any two positive integers m and n we have $\text{tw}(G_{m \times n}) \leq \min\{m, n\}$.

Let G be a graph and let (T, β) be a tree decomposition of G . For a subset $X \subseteq V(G)$ we let $\beta^{-1}(X) := \bigcup_{v \in X} \beta^{-1}(v) = \{t \in V(T) \mid X \cap \beta(t) \neq \emptyset\}$. Conversely, for a subset $S \subseteq V(T)$ we let $\beta(S) := \bigcup_{t \in S} \beta(t)$.

Lemma 2.9 Let (T, β) be a tree decomposition of a graph G . If $C \subseteq V(G)$ is connected in G , then the set $\beta^{-1}(C)$ is connected in T .

Proof. By (T2), for every vertex $v \in C$ the set $\beta^{-1}(v)$ is connected in T . By (T1) every edge $uv \in E(G[C])$ is covered by some node $t_{uv} \in V(T)$, i. e. $\{u, v\} \subseteq \beta(t_{uv})$. Hence $t_{uv} \in \beta^{-1}(u) \cap \beta^{-1}(v) \neq \emptyset$. Thus, since C is connected in G , the set $\beta^{-1}(C)$ is connected in T . \square

The class of *series-parallel graphs*, presented in the following example, has its origins in the famous Kirchhoff laws from 1847. The laws describe how to compute the resistance of electrical networks, where resistors can be combined either in series or in parallel (see [1]). The networks thus obtained are the series-parallel graphs.

Example 2.10 The class of series-parallel graphs (G, s, t) with source $s \in V(G)$ and sink $t \in V(G)$ is inductively defined as follows.

- (1) Every edge st is series-parallel.
- (2) If (G_1, s_1, t_1) and (G_2, s_2, t_2) are series parallel with $V(G_1) \cap V(G_2) = \emptyset$, then so are the following graphs:
 - a) the graph (G, s, t) obtained from $G_1 \cup G_2$ by identifying t_1 and s_2 and setting $s = s_1$ and $t = t_2$ (serial composition).
 - b) the graph (G, s, t) obtained from $G_1 \cup G_2$ by identifying s_1 and s_2 and also t_1 and t_2 and setting $s = s_1$ and $t = t_2$ (parallel composition).

Every series-parallel graphs has tree width ≤ 2 . Following the inductive definition of series-parallel graphs one can easily show that every such graph (G, s, t) has a tree decomposition of width at most 2 containing a bag $\{s, t\}$. This is trivial for edges. For parallel and serial composition the tree decompositions of the individual parts can be glued together at the node labelled by the respective source and sink vertices. \square

For proving that the tree width of a given graph G is equal to some integer k , we usually proceed in two steps. First we prove that $\text{tw}(G) \leq k$ and then we show that $\text{tw}(G) \geq k$. While for the former (the upper bound) it is sufficient to exhibit a tree decomposition of G of width k , the latter (the lower bound) is usually more difficult to prove. How do we show that none of the tree decompositions of G has width less than k ? We will present some tools for proving lower bounds in this and the next chapter. In particular, the following lemma will be useful.

Lemma 2.11 (Helly Property for Trees) *Let T be a tree and $k \in \mathbb{N}^+$. For $i \in \{1, \dots, k\}$ let S_i be a connected subset of $V(T)$, $S_i \subseteq V(T)$. If $S_i \cap S_j \neq \emptyset$ for all $i, j \in \{1, \dots, k\}$, then $\bigcap_{i=1}^k S_i \neq \emptyset$.*

Proof. We use induction on $|V(T)|$:

$|V(T)| = 1$ is trivial.

$|V(T)| > 1$: Choose a leaf $t \in V(T)$.

Case 1: There is an $i \in \{1, \dots, k\}$ with $S_i = \{t\}$. Then $t \in S_j$ for all $j \in \{1, \dots, k\}$ and hence $t \in \bigcap_{i=1}^k S_i \neq \emptyset$.

Case 2: $S_i \neq \{t\}$ for all $i \in \{1, \dots, k\}$. Then $T' := T \setminus \{t\}$ is a tree and $S'_i := S_i \setminus \{t\}$ is connected in T' for all $i \in \{1, \dots, k\}$. We claim that all $i, j \in \{1, \dots, k\}$ satisfy $S'_i \cap S'_j \neq \emptyset$: If $t \notin S_i \cap S_j$, we are done. Otherwise we have $t \in S_i \cap S_j$. Let t' be the neighbour of t in T . Since $S_i \neq \{t\}$ and $S_j \neq \{t\}$, and because S_i and S_j are connected, we have $t' \in S_i \cap S_j$. Hence also $t' \in S'_i \cap S'_j$. By the inductive hypothesis we thus have $\bigcap_{i=1}^k S'_i \neq \emptyset$. It follows that $\bigcap_{i=1}^k S_i \neq \emptyset$. \square

Example 2.12 *The triangle C_3 does not have the Helly Property, because the intersection of any two of the three edges is non-empty, whereas the common intersection of all three edges is empty.*

Lemma 2.13 *Let (T, β) be a tree decomposition of G , and let $W \subseteq V(G)$ be a clique in G . Then there exists a node $t \in V(T)$ with $W \subseteq \beta(t)$.*

Proof. Let $W = \{v_1, \dots, v_k\}$ and let $S_i = \beta^{-1}(v_i)$ for every $i \in [k]$. By Condition (T2), for all $i \in [k]$ the sets S_i are connected, and by Condition (T1), all $i, j \in [k]$ satisfy $S_i \cap S_j \neq \emptyset$. Hence by Lemma 2.11, $\bigcap_{i=1}^k S_i \neq \emptyset$, which proves the lemma. \square

From Lemma 2.13 and Remark 2.7 we obtain the following corollary.

Corollary 2.14 *For all $n \in \mathbb{N}$, the complete graph K_n satisfies $\text{tw}(K_n) = n - 1$.*

Proof. The statement follows from Lemma 2.13 and Remark 2.7. \square

Corollary 2.15 *Any tree T with at least one edge satisfies $\text{tw}(T) = 1$.*

Proof. The statement follows from Lemma 2.13 and Example 2.3. \square

Let G be a graph. A set $S \subseteq V(G)$ is a *separator* of G , or *separates* G , if $G \setminus S$ has more connected components than G . For sets $W_1, W_2 \subseteq V(G)$, a set $S \subseteq V(G)$ *separates* W_1 from W_2 , if there is no path from a vertex in $W_1 \setminus S$ to vertex in $W_2 \setminus S$ in the graph $G \setminus S$. *separate*
separate W_1 from W_2

Lemma 2.16 *Let (T, β) be a tree decomposition of G , let $t_1 t_2 \in E(T)$, and let T_1 and T_2 be the two connected components of $T - t_1 t_2$. Then $\beta(t_1) \cap \beta(t_2)$ separates $\beta(V(T_1))$ from $\beta(V(T_2))$.*

Proof. First note that by (T2), $\beta(T_1) \cap \beta(T_2) \subseteq \beta(t_1) \cap \beta(t_2)$. Towards a contradiction, suppose there is a path P in $G \setminus (\beta(t_1) \cap \beta(t_2))$ from $\beta(V(T_1))$ to $\beta(V(T_2))$. Then P must contain an edge e from a vertex in $\beta(V(T_1)) \setminus \beta(V(T_2))$ to a vertex in $\beta(V(T_2)) \setminus \beta(V(T_1))$. By (T1) the edge e is covered by some node $s \in V(T)$: $e \subseteq \beta(s)$. But s is neither in T_1 nor in T_2 , a contradiction. \square

We say that a tree decomposition (T, β) is *small*, if all nodes $s, t \in V(T)$ with $s \neq t$ satisfy $\beta(s) \not\subseteq \beta(t)$. *small*

Lemma 2.17 *Every graph G has a small tree decomposition of width $\text{tw}(G)$.*

Proof. Let (T, β) be a tree decomposition of G satisfying

- (1) $w(T, \beta) = \text{tw}(G)$, and
- (2) $|V(T)|$ is minimal subject to 1.

Towards a contradiction, suppose there are nodes $s, t \in V(T)$ with $s \neq t$ satisfying $\beta(s) \subseteq \beta(t)$. By (T2) all nodes $u \in V(T)$ on the path from s to t satisfy $\beta(s) \subseteq \beta(u)$. Let s' be the neighbour of s on the path from s to t . Then $\beta(s) \subseteq \beta(s')$. Let T' be the tree obtained from T by contracting the edge ss' . For $u \in V(T')$ let

$$\beta'(u) = \begin{cases} \beta(s') & \text{if } u \text{ is the new node} \\ \beta(u) & \text{otherwise.} \end{cases}$$

Then (T', β') is a tree decomposition of G with $w(T', \beta') = w(T, \beta)$ and $|V(T')| < |V(T)|$, a contradiction to the choice of (T, β) . \square

The following corollary is an immediate consequence of Lemma 2.17.

Corollary 2.18 *Every graph G has a tree decomposition (T, β) with $|V(T)| \leq |V(G)|$.* \square

Corollary 2.19 *Any cycle C satisfies $\text{tw}(C) = 2$.*

Proof. By Example 2.4, we have $\text{tw}(C) \leq 2$. Towards a proof of $\text{tw}(C) \geq 2$, let (T, β) be a small tree decomposition of C . We may assume that (T, β) is non-trivial. Let $st \in E(T)$. By Lemma 2.16, $\beta(s) \cap \beta(t)$ separates C . Hence $|\beta(s) \cap \beta(t)| \geq 2$. Since $\beta(s) \not\subseteq \beta(t)$ we have that $|\beta(t)| > |\beta(s) \cap \beta(t)| \geq 2$. Hence $w(T, \beta) \geq 2$. \square

The following easy Lemma shows how we can restrict tree decompositions to subgraphs.

Lemma 2.20 *If (T, β) is a tree decomposition of G and H is a subgraph of G , then (T, β') is a tree decomposition of H , where $\beta'(t) := \beta(t) \cap V(H)$ for all $t \in V(T)$.* \square

Corollary 2.21 *If H is a subgraph of the graph G , then $\text{tw}(H) \leq \text{tw}(G)$.*

Lemma 2.22 *Let G be a non-empty graph. Then G has a vertex $v \in V(G)$ with $\deg_G(v) \leq \text{tw}(G)$.*

Proof. Let (T, β) be a small tree decomposition of G and let $t \in V(T)$ be a leaf. Since (T, β) is small, there exists a vertex $v \in \beta(t)$ such that $v \notin \beta(s)$ for all $s \in V(T)$ with $s \neq t$. Choose such a vertex v . By (T1) we have $N_G(v) \subseteq \beta(t)$ and hence $\deg_G(v) = |N_G(v)| \leq |\beta(t)| - 1 \leq \text{tw}(G)$. \square

Theorem 2.23 *Any graph G satisfies $|E(G)| \leq |V(G)| \cdot \text{tw}(G)$.*

Proof. Let $k = \text{tw}(G)$. We use induction on $|V(G)|$. If $|V(G)| = 0$, then $|E(G)| = 0$. If $|V(G)| > 0$, let $v \in V(G)$ be a vertex with $\deg_G(v) \leq k$, and let $H := G \setminus v$. (Such a vertex exists by Lemma 2.22.) Using the inductive hypothesis, we obtain

$$|E(G)| \leq |E(H)| + k \leq |V(H)| \cdot \text{tw}(H) + k \leq (|V(G)| - 1) \cdot k + k = |V(G)| \cdot k.$$

\square

2.2. Tree Width and Separators

In this section we will give an approximate characterisation of tree width that uses separators of small size. This characterisation is the basis of an algorithm (the ‘separator algorithm’) for computing tree decompositions, which we will discuss in Chapter 5.

Definition 2.24 *Let G be a graph and $W \subseteq V(G)$. A balanced W -separator (or simply a W -separator) in G is a set $S \subseteq V(G)$, such that every component C of $G \setminus S$ satisfies* *balanced W -separator*

$$|C \cap W| \leq \frac{|W|}{2}.$$

Lemma 2.25 *Let G be a graph of tree width $k := \text{tw}(G)$ and $W \subseteq V(G)$. Then there is a W -separator S with $|S| \leq k + 1$.*

Proof. Let (T, β) be a tree decomposition of G of width k . If there is an edge ts such that all components of $G \setminus (\beta(t) \cap \beta(s))$ contain at most $\frac{|W|}{2}$ elements of W , then $\beta(t) \cap \beta(s)$ is a W -separator.

Otherwise we orient the edges of T as follows. Let $e := st \in E(T)$ be an edge and let T_s, T_t be the two trees of $T - e$, such that $s \in V(T_s)$. Let C_1, \dots, C_l be the components of $G \setminus (\beta(t) \cap \beta(s))$. We orient the edge ts from t to s if the (unique) component C_i with $|C_i \cap W| > \frac{|W|}{2}$ satisfies $C_i \subseteq \beta(T_s)$, otherwise we orient the edge from s to t . (Note that by Lemma 2.16, C_i is completely contained in precisely one of the sets $\beta(T_s)$ and $\beta(T_t)$.) I. e. the edge is oriented in the direction of the larger part of W . Since T is a tree, there is a node $u \in V(T)$, such that all incident edges are oriented towards u .

Let u be such a node.

Claim. $\beta(u)$ is a W -separator.

Proof. Clearly, every component of $G \setminus \beta(u)$ is contained in a component of $G \setminus (\beta(u) \cap \beta(v))$ for every $v \in V(T)$ such that $vu \in E(T)$. Hence, no component C of $G \setminus \beta(u)$ contains more than $\frac{1}{2}$ of the elements of W , as otherwise one of the edges incident to u would have been oriented in the opposite direction. \dashv \square

Definition 2.26 *The separator width of a graph G is defined as* *separator width*

$$\text{sw}(G) := \min\{k \in \mathbb{N} \mid \text{for all } W \subseteq V(G) \text{ there is a } W\text{-separator } S \text{ with } |S| \leq k\}.$$

We now prove the main theorem of this section.

Theorem 2.27 *Any graph G satisfies*

$$\text{sw}(G) - 1 \leq \text{tw}(G) \leq 3 \cdot \text{sw}(G).$$

In Lemma 2.25, we have already proved the first inequality of the theorem. The second inequality is an immediate consequence of the next lemma.

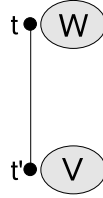
Lemma 2.28 *Let $G := (V, E)$ be a graph such that for all $W \subseteq V$ with $|W| = 2k + 1$ there is a W -separator S with $|S| \leq k$. Then $\text{tw}(G) \leq 3k$.*

Proof. We use induction on $|V(G)|$ to show the following claim.

Claim. If for every $W \subseteq V$ with $|W| = 2k + 1$ there is a W -separator S of order $|S| \leq k$, then for every $W \subseteq V$ with $|W| \leq 2k + 1$ there is a tree decomposition (T, β) of G such that

- the width of (T, β) is at most $3k$
- there is a node $t \in V(T)$ with $W = \beta(t)$.

Proof. If $|V| \leq 3k + 1$, then the decomposition (T, β) where $T = (\{t, t'\}, \{tt'\})$, $\beta(t) := W$, and $\beta(t') := V$ satisfies the claim.



So let $|V| > 3k + 1$ and let $W \subseteq V$ be such that $|W| \leq 2k + 1$. W.l.o.g. we assume that $|W| = 2k + 1$. Let S be a W -separator with $|S| \leq k$ and let C_1, \dots, C_n be the components of $G - S$. For $i \in [n]$ we let

$$\begin{aligned} G_i &:= G[C_i \cup S], \text{ and} \\ W_i &:= (W \cap C_i) \cup S. \end{aligned}$$

Hence, $|W_i| \leq \frac{|W|}{2} + k \leq 2k < |W|$, and therefore $|V(G_i)| < |V|$.

Furthermore, for all $W' \subseteq V(G_i)$ there is a W' -separator $|S'|$ in G_i with $|S'| \leq k$. (Choose any W' -separator in G and take its intersection with $V(G_i)$).

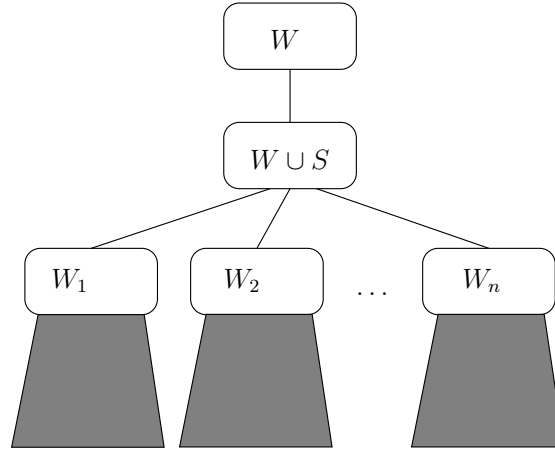
By induction hypothesis, there is a tree decomposition (T^i, β^i) of G_i , such that

- the width of (T^i, β^i) is at most $3k$ and
- there exists a node $t_i \in V(T^i)$ with $W_i := \beta^i(t_i)$.

We can now combine the n tree decompositions (T^i, β^i) to a tree decomposition of G by taking their disjoint union and adding two new nodes, s and s' , where the bag at s' is W , s is the unique neighbour of s' , the bag at s is $W \cup S$, and for every $i \in [n]$, s is connected by an edge to the node $t_i \in V(T^i)$.



Figure 2.7.: A non-chordal and a chordal graph.



□

2.3. Characterising Tree Width

Often it is helpful to have different views on tree width. In this section we give several alternative characterisations of tree width, including the historical characterisations (via *partial k -trees* and *simplicial decompositions*) mentioned in the introduction.

Let G be a graph and $C \subseteq G$ a cycle. An edge $uv \in E(G) \setminus E(C)$ is a *chord* of C (in G), if $\{u, v\} \subseteq V(C)$. A graph G is *chordal*, if every cycle of length at least 4 in G has a chord. Figure 2.7 shows a non-chordal and a chordal graph.

Chordal graphs enjoy several nice properties. In particular, they can be recursively decomposed into complete graphs, as we will see now.

A *simplicial decomposition* of a graph G is a pair (G_1, G_2) of subgraphs of G such that $G = G_1 \cup G_2$, the graph $G_1 \cap G_2$ is a complete graph, $V(G_1) \setminus V(G_2) \neq \emptyset$ and $V(G_2) \setminus V(G_1) \neq \emptyset$.

Examples 2.29 • The graph in Figure 2.8 is the simplicial sum of two of its subgraphs (circled in red and green, respectively).

- Neither of the two graphs in Figure 2.9 has a simplicial decomposition.

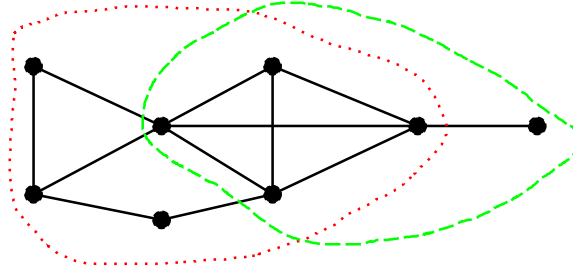


Figure 2.8.: This graph is the simplicial sum of two of its subgraphs (circled in red and green, respectively)

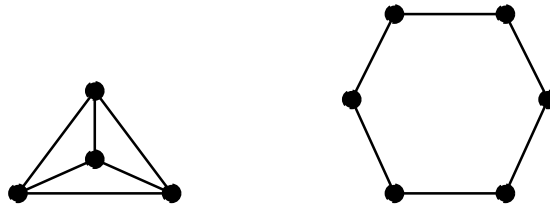


Figure 2.9.: Neither of these graphs has a simplicial decomposition.

Lemma 2.30 *Let G be a chordal graph. Then either G has a simplicial decomposition, or G is a complete graph.*

Proof. We show that if G is not complete, then G has a simplicial decomposition. If G is not connected, then this is obviously true. Hence we assume that G is connected. Since G is not complete, there are two vertices $u, v \in V(G)$ ($u \neq v$) with $uv \notin E(G)$. Let $X \subseteq V(G) \setminus \{u, v\}$ be a minimal $(\{u\}, \{v\})$ -separator of G . Let C_u be the connected component of $G \setminus X$ containing u , and let C_v be the connected component of $G \setminus X$ containing v . Let $G_1 := G[C_u \cup X]$ and $G_2 := G[V(G) \setminus C_u]$. We show that (G_1, G_2) is a simplicial decomposition of G . For this it suffices to show that $V(G_1) \cap V(G_2) = X$ is a clique in G . Indeed, if X is not a clique, then there are two vertices $x, y \in X$ ($x \neq y$) with $xy \notin E(G)$. By minimality of X , x has a neighbour $u_x \in C_u$ and y has a neighbour $u_y \in C_u$. Since C_u is connected, there is a path from u_x to u_y in C_u . Let P_u be such a path of minimal length. Analogously, x and y have neighbours $v_x \in C_v$ and $v_y \in C_v$, respectively, and there is a path P_v in C_v from v_x to v_y of minimal length. But now $V(P_u) \cup V(P_v) \cup \{x, y\}$ induces a cycle of length at least 4 without a chord, a contradiction to G being chordal. This proves that X is a clique, and hence (G_1, G_2) is a simplicial decomposition of G . \square

Together with Lemma 2.30, the following easy Lemma shows that we can recursively decompose chordal graphs into cliques.

Lemma 2.31 *Let G be a chordal graph and (G_1, G_2) a simplicial decomposition of G . Then G_1 and G_2 are chordal.* \square

Corollary 2.32 *Any chordal graph G has a tree decomposition whose bags are cliques in G .*

Proof. By Lemmas 2.30 and 2.31, either G is complete, or G has a simplicial decomposition (G_1, G_2) and G_1 and G_2 are chordal. If we recursively decompose G by simplicial decompositions, after a finite number of steps we obviously end up with complete graphs.

More precisely, if G is complete, then the trivial decomposition proves the statement. Otherwise, let (G_1, G_2) be a simplicial decomposition of G . Inductively, we may assume that each G_i has a tree decomposition (T_i, β_i) whose bags are cliques in G_i , for $i \in [2]$. Let $X := V(G_1) \cap V(G_2)$. If $X = \emptyset$, then we obtain the desired tree decomposition of G by connecting the tree decompositions (T_1, β_1) and (T_2, β_2) by an edge. Otherwise, since X is a clique, there is a node $t_i \in V(T_i)$ such that $X \subseteq \beta_i(t_i)$, for $i \in [2]$. We now obtain the desired tree decomposition of G by connecting the two tree decompositions (T_1, β_1) and (T_2, β_2) via the edge $t_1 t_2$. \square

For the characterisations of tree width, it is useful to consider tree decompositions of a particularly simple form.

Definition 2.33 *Let (T, β) be a tree decomposition of width k of G .*

- (T, β) is *thick*, if all $t \in V(T)$ satisfy $|\beta(t)| = k + 1$. *thick*
- (T, β) is *smooth*, if (T, β) is thick and, in addition, every edge $st \in E(T)$ satisfies $|\beta(s) \cap \beta(t)| = k$. *smooth*

Obviously, every smooth tree decomposition is thick and small.

Lemma 2.34 *If a graph G has a tree decomposition of width k , then G has a thick tree decomposition of width k .*

Proof. Let (T, β) be a tree decomposition of width k of G . Then there is a node $t_0 \in V(T)$ with $|\beta(t_0)| = k + 1$. Let $(T, \beta^0) := (T, \beta)$, choose t_0 to be the root of T and orient the edges of T away from t_0 .

Assume that (T, β^i) is a tree decomposition of G , such that all nodes $t \in V(T)$ at distance $\leq i$ from t_0 satisfy $|\beta^i(t)| = k + 1$. For each node s at distance $i + 1$ from t_0 that satisfies $|\beta(s)| < k + 1$, we add $(k + 1) - |\beta(s)|$ new nodes from the bag $\beta(t)$ to the bag $\beta(s)$, where t is the parent of s . In this way we obtain a tree

decomposition (T, β^{i+1}) of G where all nodes at distance at most $i+1$ from t_0 satisfy $|\beta^{i+1}(t_0)| = |\beta^{i+1}(s)| = k+1$.

It is easy to see that in this way, we finally obtain a thick tree decomposition of with k of G . \square

Lemma 2.35 *If a graph G has a tree decomposition of width k , then G has a tree decomposition of width k , that is both small and thick.*

Proof. Let (T, β) be a tree decomposition of width k of G . By Lemma 2.34 we may assume that (T, β) is thick. Choose (T, β) such that $|V(T)|$ is minimal with these properties. With this, an argument similar to the proof of Lemma 2.17 shows that (T, β) is small. \square

Lemma 2.36 *If a graph G has a tree decomposition of width k , then G has a smooth tree decomposition of width k .*

Proof. Let (T, β) be a tree decomposition of width k of G . By Lemma 2.35, we may assume that (T, β) is small and thick. For every edge $st \in E(T)$ with $\ell := |B_s \cap B_t| < k$, we add a path with new nodes $s_1, \dots, s_{k-\ell}$ between s and t , and we delete the edge st . Let $B_s = \{x_1, \dots, x_l, \dots, x_{k+1}\}$ and $B_t = \{x_1, \dots, x_l, y_{l+1}, \dots, y_{k+1}\}$. Let $B'_s := B_s$ and $B'_t := B_t$. Now define $B'_{s_i} := \{x_1, \dots, x_{k+1-i}, y_{k+2-i}, \dots, y_{k+1}\}$ for $i = 1, \dots, k-\ell$. In this way we obtain a smooth tree decomposition (T', β') of width k of G . \square

The following notion of *elimination ordering* goes back to Gauss Elimination of matrices (see [1]).

elimination ordering
elimination graph

Definition 2.37 *An elimination ordering of a graph G is a linear ordering (v_1, \dots, v_n) of $V(G)$. For $i \in [n]$ the i th elimination graph G_i of G is the graph defined as follows.*

$$G_1 := G, \\ G_{i+1} := (G_i - v_i) + \{uw \mid uv_i \in E(G_i), v_i w \in E(G_i)\}$$

The width of the elimination ordering is $\max_{i \in [n]} \deg_{G_i}(v_i)$.

elimination width, $\text{ew}(G)$

The elimination width of G , denoted by $\text{ew}(G)$, is the minimal width over all elimination orderings of G .

Intuitively, G_{i+1} is obtained from G_i by deleting the ‘smallest’ vertex v_i of G_i and inserting a clique onto the neighbours of v_i in G_i .

Example 2.38 *Figure 2.10 shows a graph G with vertex set $V(G) = \{1, 2, 3, 4, 5\}$, and elimination graphs G_1, \dots, G_5 , corresponding to the ordering $(1, 2, 3, 4, 5)$.*

Remark 2.39 *Any two graph G and H with $H \subseteq G$ satisfy $\text{ew}(H) \leq \text{ew}(G)$.*

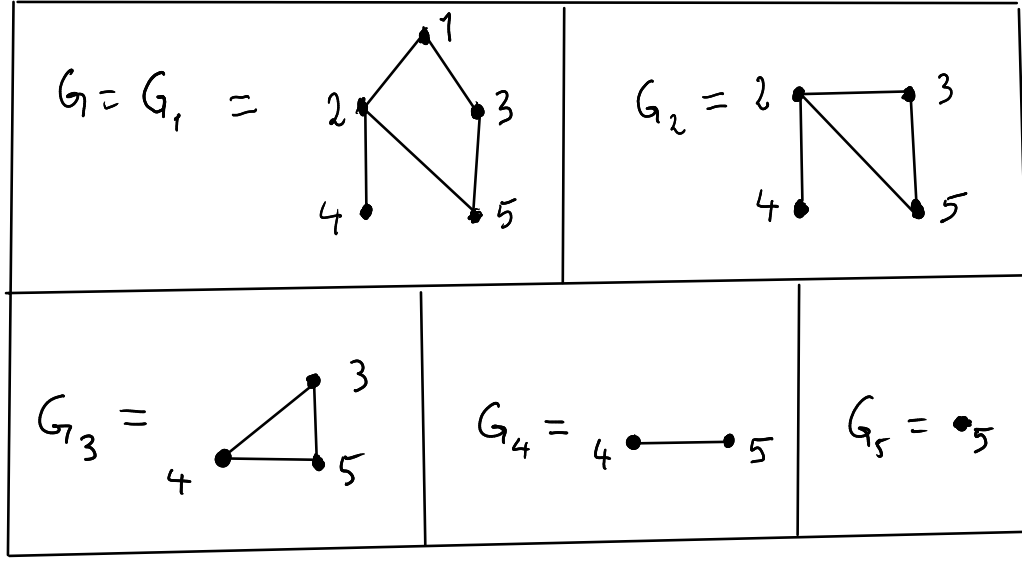


Figure 2.10.: A graph G with elimination graphs G_1, \dots, G_5 , corresponding to the ordering $(1, 2, 3, 4, 5)$ of $V(G)$ (cf. Example 2.38).

Proof. If (v_1, \dots, v_n) is an elimination ordering of G of width at most k , then the restriction of (v_1, \dots, v_n) to the vertices of H is an elimination ordering of H of width at most k . \square

Definition 2.40 Let $k \in \mathbb{N}$. The class of k -trees is a graph class recursively defined as follows. *k-tree*

(K1) K_k is a k -tree.

(K2) If G is a k -tree and $X \subseteq V(G)$ is a k -clique in G , then the graph G' obtained from G by adding a new vertex $v \notin V(G)$ to G and connecting v by an edge to each vertex of X is a k -tree.

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

Example 2.41 Figure 2.11 shows a 2-tree and a 3-tree.

Lemma 2.42 A graph G is a 1-tree if and only if G is a tree.

Proof. Let G be a 1-tree. Using induction, we show that G is a tree.

(K1): If $G = K_1$, then G obviously is a tree.

(K2): If G_0 is a 1-tree and G is obtained from G_0 by adding a new vertex v , making it adjacent to a single vertex $u \in V(G_0)$, then G_0 is a tree by induction and hence G is also a tree: G is connected and the new edge vu does not create a cycle.

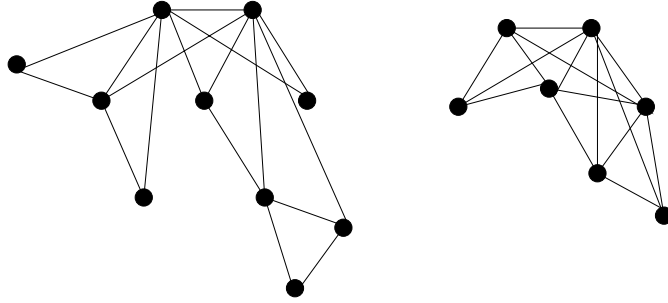


Figure 2.11.: A 2-tree and a 3-tree.

Conversely, let G be a tree. Let $K_1 = G_1 \subseteq G_2 \subseteq G_3 \subseteq \dots \subseteq G_n = G$ be an increasing sequence of subtrees of G with $|V(G_i)| = i$ for $i \in [n]$. The graph G_1 is a 1-tree by (K1), and if G_i is a 1-tree, G_{i+1} is also a 1-tree by (K2) (for $i \in [n-1]$). \square

Lemma 2.43 *Let $k \in \mathbb{N}$, let $G \neq \emptyset$ be a k -tree, and let $v \notin V(G)$ be a new vertex. Then the graph $G^v := (V(G) \cup \{v\}, E(G) \cup \{vu \mid u \in V(G)\})$, obtained from G by making v adjacent to every vertex of G , is a $(k+1)$ -tree.*

Proof. We use induction on the definition of k -trees.

(K1): If $G = K_k$, then $G^v = K_{k+1}$ is a $(k+1)$ -tree.

(K2): If G is a k -tree obtained from a k -tree G_0 by connecting a new vertex $u \notin V(G_0)$ to all vertices of a k -clique $X \subseteq V(G_0)$, then G^v can be obtained from $G_0^v := (V(G_0) \cup \{v\}, E(G_0) \cup \{vu \mid u \in V(G_0)\})$ by making u adjacent to all vertices of the clique $X \cup \{v\} \subseteq V(G_0^v)$. By the inductive hypothesis, G_0^v is a $(k+1)$ -tree, hence G^v is a $(k+1)$ -tree. \square

Corollary 2.44 *For $\ell, k \in \mathbb{N}$ with $\ell \leq k$, every ℓ -tree is a subgraph of some k -tree.*

Proof. Apply Lemma 2.43 $(k - \ell)$ times. \square

triangulation **Definition 2.45** *Let G be a graph. A triangulation of G is a chordal supergraph $G' \supseteq G$ with $V(G') = V(G)$.*

clique number **Definition 2.46** *The clique number of G , $\omega(G)$, is the cardinality of a maximal clique in G .*

We are now ready to give three alternative characterisations of tree width.

Theorem 2.47 *Let $k \in \mathbb{N}$. For any graph G the following statements are equivalent.*

- (1) $\text{tw}(G) \leq k$

- (2) G is a partial k -tree.
- (3) $\text{ew}(G) \leq k$
- (4) G has a triangulation G' such that $\omega(G') \leq k + 1$.

Proof. ‘1 \implies 2’: Let $\ell := \text{tw}(G) \leq k$. By Corollary 2.44 it suffices to show that G is a subgraph of an ℓ -tree. Let (T, β) be a smooth tree decomposition of width ℓ of G (which exists by Lemma 2.36). Define a graph G' by $V(G') := V(G)$ and

$$E(G') := E(G) \cup \{uv \mid \{u, v\} \subseteq \beta(t) \text{ for a node } t \in V(T)\}.$$

Obviously, (T, β) is also a tree decomposition of G' .

With the following claim we are done.

Claim: G' is an ℓ -tree.

Proof. T is a tree and thus, by Lemma 2.42, T is a 1-tree. We use induction on $|V(T)|$.

(K1): If $T = K_1 = (\{t\}, \emptyset)$, then G' is the complete graph with $\ell + 1$ nodes, and hence G' is an ℓ -tree.

(K2): For a subtree $T_0 \subseteq T$ let G'_{T_0} denote the subgraph of G' induced by $\bigcup_{t \in V(T_0)} \beta(t)$. Let T be obtained from T_0 by adding a new node $s \notin V(T_0)$ to T_0 , making s adjacent to some $K_1 \subseteq T_0$, and inductively assume that G'_{T_0} is an ℓ -tree. Then

$$V(G') = V(G'_{T_0}) \cup \beta(s) = V(G'_{T_0}) \cup \{v\}, \quad \text{for a node } v \notin V(G'_{T_0}),$$

where the last inequality holds because (T, β) is smooth. Moreover, we have

$$E(G') = E(G'_{T_0}) \cup \{uv \mid u \in \beta(s), u \neq v\},$$

and $\beta(s) \setminus \{v\}$ is an ℓ -clique in G'_{T_0} . Notice that an edge uv with $u \notin \beta(s)$ cannot exist, because otherwise uv is covered by a bag at a node $t_0 \in V(T_0)$, a contradiction to $v \notin V(G'_{T_0})$. Hence $G' = G'_T$ is an ℓ -tree. \dashv

‘2 \implies 3’: Let $G' \supseteq G$ be a partial k -tree. By Remark 2.39 it suffices to show that $\text{ew}(G') \leq k$. Let (v_1, \dots, v_n) be an ordering of $V(G')$, where the first k vertices v_1, \dots, v_k are the vertices of the K_k in the base case (K1) (in any ordering) and the ordering (v_{k+1}, \dots, v_n) is an ordering in which the remaining vertices are introduced according to the inductive step (K2) of the definition of k -trees. We claim that the inverse ordering (v_n, \dots, v_1) is an elimination ordering of G' of width k . If $n = k$, this is obviously true. Otherwise $n > k$, and each vertex v_i of the k -tree G' , for $i \in \{k+1, \dots, n\}$, was introduced by making it adjacent to the vertices of a k -clique in the k -tree $G'[v_1, \dots, v_{i-1}]$. Hence the i th elimination graph G'_i is

$G'_i = G'[v_1, \dots, v_{n-i+1}]$ and $\deg_{G'_i}(v_{n-i+1}) \leq k$ for all $i \in [n]$. This proves that $\text{ew}(G') \leq k$.

‘3 \implies 4’: Let (v_1, \dots, v_n) be an elimination ordering of G of width at most k , and for $i \in [n]$, let G_i be the i th elimination graph. Let $G' := \bigcup_{i \in [n]} G_i$. We show that G' is a triangulation of G satisfying $\omega(G') \leq k + 1$. Obviously, we have $G \subseteq G'$ and $V(G') = V(G)$. We first show that every cycle in G' has a chord. Let $C \subseteq G'$ be a cycle of length at least 4. Pick $i \in [n]$ with $v_i \in V(C)$ and $j > i$ for all $v_j \in V(C) \setminus \{v_i\}$. Let u and v be the neighbours of v_i in C . Then $\{u, v\} \subseteq V(G_{i+1})$, $v_i u \in E(G_j)$ for some $j \leq i$ and $v_i v \in E(G_{j'})$ for some $j' \leq i$. Hence $\{u, v\} \subseteq N_{G_i}(v_i)$ and $uv \in E(G_{i+1}) \subseteq E(G')$. Hence C has a chord in G' . Finally, we show that cliques in G' contain at most $k + 1$ vertices. Let $X \subseteq V(G')$ be a clique. Pick $i \in [n]$ with $v_i \in X$ and $j > i$ for all $v_j \in X \setminus \{v_i\}$. Then $X \subseteq V(G_i)$ and for every $x \in X \setminus \{v_i\}$, there is a $j \leq i$ such that $v_i x \in E(G_j)$. Hence $X \setminus \{v_i\} \subseteq N_{G_i}(v_i)$. Since $\deg_{G_i}(v_i) \leq k$ we have $|X| \leq k + 1$.

‘4 \implies 1’: Let G' be a triangulation of G . Since $G \subseteq G'$, it suffices to prove that $\text{tw}(G') \leq k$. Let (T, β) be a tree decomposition of G' whose bags are cliques in G' . Such a tree decomposition exists by Corollary 2.32. Since $\omega(G') \leq k + 1$, any bag of (T, β) contains at most $k + 1$ elements. Hence (T, β) is a witness for $\text{tw}(G') \leq k$. \square

3. Cops and Robbers

In this chapter, we introduce the *cops-and-robber game*, in which a number of cops must catch a robber on a given graph. This game provides a characterisation of tree width, and it is due to Seymour and Thomas [6].

Indeed, the game comes together with a *monotone* variant, in which the cops have to make sure that the robber space never increases during a play. We show that the minimum number of cops necessary to catch the robber in the monotone cops-and-robber game on a graph G is equal to $\text{tw}(G) + 1$. Furthermore, we show that the minimum number of cops necessary to catch the robber is the same for both the monotone and the non-monotone variant of the game. In the literature, this equality is often referred to as *monotonicity* of the cops-and-robber game. In order to prove monotonicity, we use the concept of *brambles*. Brambles are configurations in graphs that are obstructions for small tree width. As we will see, the *maximum* order of a bramble in a graph G is equal to $\text{tw}(G) + 1$ (i.e. to the *minimum* width of a tree decomposition plus 1). This *min-max characterisation* of tree width is particularly useful for determining lower bounds on the tree width of graphs.

3.1. Cops and Robbers

Let G be a graph and let $k \in \mathbb{N}$. Informally, the cops-and-robber game is played by two players, the *cop player* (or simply *the cops*) and the *robber player* (or simply *the robber*) on the graph G . The cop player controls k cops (where k is a parameter of the game) and the robber player controls the robber. Both the cops and the robber move on vertices of G . The cops choose a set X of at most k vertices. The cops can see the robber at all times. In each move, some of the cops fly in helicopters to new vertices. Meanwhile, the clever robber listens to the police radio and discovers where the cops will be landing, and quickly tries to escape by running arbitrarily fast along paths in G , not being allowed to run through a cop. The cop player's objective is to place a cop on the vertex occupied by the robber, while the robber player's goal is to elude capture forever.

Now we define the game formally.

Definition 3.1 *Let G be a graph. For an integer $k \in \mathbb{N}$ the cops and robber game $\text{CR}(G, k)$ is the game described below.*

*cops and robber game,
 $\text{CR}(G, k)$*

A *position* of the game $\text{CR}(G, k)$ is a pair (X, y) with $X \subseteq V(G)$, $|X| \leq k$ and $y \in V(G)$. A *play* is a sequence of positions. In the beginning of a play, the robber

selects a vertex $y \in V(G)$ and the *starting position* is (\emptyset, y) . In every *round* of the play, the players move from a position (X, y) to a new position as follows.

- (C) The cops select a set $X' \subseteq V(G)$ with $|X'| \leq k$.
- (R) The robber selects a vertex $y' \in V(G)$ such that there is a path from y to y' in the graph $G \setminus (X \cap X')$.

Now the new position is (X', y') . A play is *over*, once a position (X, y) with $y \in X$ is reached. In this case, the cop player wins. If such a position is never reached, the robber player wins.

Formally, a *play* can be modeled as a sequence $(X_0, y_0), (X_1, y_1), \dots$ of positions (X_i, y_i) obtained in successive rounds obeying the above rules, that is either infinite or the last position (X, y) satisfies $y \in X$. A *strategy for the cops* is a function S that associates with every partial play $(X_0, y_0), \dots, (X_i, y_i)$ a set $X_{i+1} = S((X_0, y_0), \dots, (X_i, y_i)) \in [V(G)]^{\leq k}$ yielding the next move for the cops. The strategy S is a *winning strategy*, if every possible play played according to S is winning for the cops. Strategies and winning strategies for the robber are defined similarly.

The *cop-width* of a graph G is defined as

$$\text{cw}(G) := \min \{k \in \mathbb{N} \mid \text{the cops have a winning strategy in } \text{CR}(G, k)\}.$$

Example 3.2 (1) Every graph G satisfies $\text{cw}(G) \leq |V(G)|$.

(2) Every tree T with $E(T) \neq \emptyset$ satisfies $\text{cw}(T) = 2$.

(3) Every cycle C satisfies $\text{cw}(C) = 3$.

(4) For $n \in \mathbb{N}$, the complete graph K_n satisfies $\text{cw}(K_n) = n$.

(5) For every $n \in \mathbb{N}^+$, we have $n \leq \text{cw}(G_{n \times n}) \leq n + 1$.

Proof. 2: Two cops can catch the robber by first moving to an arbitrary node, and then chasing the robber down a path to a leaf. The robber has a winning strategy against one cop: she picks an edge and always moves to the free end of the edge.

3: Three cops have a winning strategy as follows: one cop picks an arbitrary vertex and occupies it throughout the whole play. The remaining graph is a path and two cops can win on a path. The robber can win against two cops as follows. She chooses a vertex. If the cops do not threaten her, she does nothing. Otherwise she picks any cop-free vertex and moves there.

4: Obviously, $\text{cw}(K_n) \leq n$. The robber can win against $n - 1$ cops by always moving to a cop-free vertex.

5: The $n + 1$ cops can search the graph row by row (as if reading a text) in a

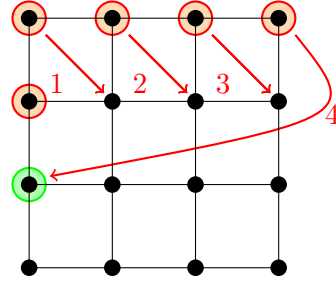


Figure 3.1.: Five cops have a winning strategy on a (4×4) -grid.

‘first-in-first-out’ manner. Starting with the upper left vertex, place one cop after the other on the vertices of the first row. You will use n cops for the first row and the $(n + 1)$ st cop is placed on the first vertex of the second row. Now remove the ‘oldest’ cop, i.e. the cop on the first vertex of the first row, and move him to the second vertex of the second row (cf. Figure 3.1).

Whenever you need a new cop, move the one that has been standing on the graph longest. Continuing like this you will finally catch the robber.

For the robber to win against $n - 1$ cops, notice that no matter how $n - 1$ cops are placed on the grid, there is always a cop-free ‘cross’ consisting of a cop-free column and a cop-free row. Since any two crosses have a non-empty intersection, the robber can win by always moving to a cop-free cross. \square

3.2. Brambles

We will determine the precise cop-width of $G_{n \times n}$ with the help of *brambles*.

Definition 3.3 Let G be a graph. A *bramble* in G is a collection $\mathcal{B} \subseteq 2^{V(G)}$ of subsets of $V(G)$ satisfying

(B1) Every $A \in \mathcal{B}$ is connected in G

(B2) Every two elements $A, A' \in \mathcal{B}$ touch, i.e. $A \cap A' \neq \emptyset$ or there is an edge $e \in E(G)$ satisfying $e \cap A \neq \emptyset$ and $e \cap A' \neq \emptyset$.

A set $X \subseteq V(G)$ satisfying $X \cap A \neq \emptyset$ for all $A \in \mathcal{B}$ is called a *hitting set* for \mathcal{B} . The order of \mathcal{B} is defined as $\text{ord}(\mathcal{B}) := \min \{ |X| \mid X \text{ hitting set for } \mathcal{B} \}$. The *bramble number* of G is defined as

$$\text{bn}(G) := \max \{ \text{ord}(\mathcal{B}) \mid \mathcal{B} \text{ is a bramble in } G \}.$$

Example 3.4 Let $n \in \mathbb{N}$.

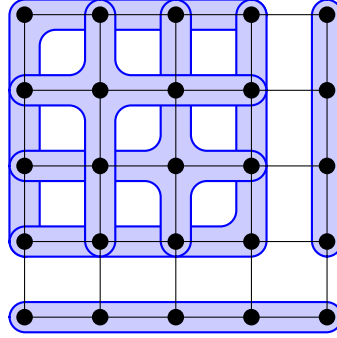


Figure 3.2.: A (5×5) -grid with the bramble sets $C_{1,1}, C_{2,2}, C_{3,3}, C_{4,4}, R$, and S .

- (1) In the complete graph K_n the collection of all singleton sets $\{\{v\} \mid v \in V(K_n)\}$ is a bramble of order n .
- (2) In the grid $G_{n \times n}$ we define the sets

$$\begin{aligned} C_{i,j} &:= \{(i, k) \mid k \in [n-1]\} \cup \{(k, j) \mid k \in [n-1]\} \text{ for } i, j \in [n-1], \\ R &:= \{(n, k) \mid k \in [n]\}, \text{ and} \\ S &:= \{(k, n) \mid k \in [n-1]\}. \end{aligned}$$

Then $\mathcal{B} := \{C_{i,j} \mid i, j \in [n-1]\} \cup \{R, S\}$ is a bramble of order $n+1$ in $G_{n \times n}$. See Figure 3.2.

Proof. We only prove the second statement. For a hitting for \mathcal{B} of size $n+1$, take the vertices of the main diagonal plus one vertex of S . To see that there is no hitting set of smaller size, note first that picking $n-2$ vertices from the subgrid $G_{(n-1) \times (n-1)}$ will leave one of the sets $C_{i,j}$ untouched. Hence $n-1$ vertices are necessary for hitting all the sets $C_{i,j}$, and $\text{ord}(\mathcal{B}) = n+1$. \square

Brambles can be seen as particularly nice winning strategies for the robber:

Remark 3.5 Every graph G satisfies $\text{bn}(G) \leq \text{cw}(G)$.

Proof. Let $k \in \mathbb{N}$. We show that $\text{bn}(G) > k$ implies $\text{cw}(G) > k$. Let \mathcal{B} be a bramble in G of order at least $k+1$. Then the robber wins $\text{CR}(G, k)$ using the following strategy. Whenever the cops move to a set $X \in [V(G)]^{\leq k}$ the robber chooses a set $A \in \mathcal{B}$ with $A \cap X = \emptyset$ and moves to a vertex $y \in A$. \square

Example 3.6 For every $n \in \mathbb{N}$ the grid $G_{n \times n}$ satisfies $\text{cw}(G_{n \times n}) = n+1$.

Proof. We already know that $n+1$ cops can catch the robber. Conversely, $\text{cw}(G_{n \times n}) \geq n+1$ holds by Example 3.4 using Remark 3.5. \square

3.3. Monotonicity of the Cops-and-Robber Game

For a subset $X \subseteq V(G)$ and a vertex $y \in V(G) \setminus X$ we let $\text{comp}(X, y)$ denote the connected component of $G \setminus X$ containing y . If $y \in X$ we let $\text{comp}(X, y) := \emptyset$. For a position (X, y) of $\text{CR}(G, k)$, we call $\text{comp}(X, y)$ the *robber space with respect to X* .

Definition 3.7 *Let G be a graph. For an integer $k \in \mathbb{N}$ the monotone cops and robber game $\text{MCR}(G, k)$ is defined as $\text{CR}(G, k)$, where, in addition, the cops have to make sure that the robber space never increases. Formally, in every position (X, y) of $\text{MCR}(G, k)$ with $y \in V(G) \setminus X$, the cops may only choose sets $X' \subseteq V(G)$ with $|X'| \leq k$ where*

(M) the robber space $\text{comp}(X, y)$ is a connected component of $G \setminus (X \cap X')$.

The monotone cop-width of a graph G is defined as

$$\text{mcw}(G) := \min \{k \in \mathbb{N} \mid \text{the cops have a winning strategy in } \text{MCR}(G, k)\}.$$

Obviously, if the cops have a winning strategy for $\text{MCR}(G, k)$, then they have a winning strategy $\text{CR}(G, k)$, and hence every graph G satisfies $\text{cw}(G) \leq \text{mcw}(G)$. In [6], Seymour and Thomas proved that the converse also holds. We will prove this as the main theorem of this section, along with the equivalence to tree width and bramble number.

We will make use of a fundamental graph theoretic result by Karl Menger. See [3] for a proof.

Theorem 3.8 (Menger 1927) *Let G be a graph and $A, B \subseteq V(G)$. Then*

$$\begin{aligned} & \min \{ |S| \mid S \subseteq V(G) \text{ separates } A \text{ from } B \text{ in } G \} \\ &= \max \{ k \in \mathbb{N} \mid \text{there are } k \text{ pairwise vertex-disjoint paths from } A \text{ to } B \}. \end{aligned}$$

Lemma 3.9 *Let \mathcal{B} be a bramble in a graph G and let $X, Y \subseteq V(G)$ be hitting sets for \mathcal{B} . Then there exist $\text{ord}(\mathcal{B})$ pairwise vertex-disjoint paths from X to Y .*

Proof. Observe that every set $S \subseteq V(G)$ separating X from Y is a hitting set for \mathcal{B} as well. Hence $|S| \geq \text{ord}(\mathcal{B})$, and the Lemma follows using Menger's Theorem. \square

Theorem 3.10 *Every graph G satisfies $\text{tw}(G) + 1 \leq \text{bn}(G)$.*

Proof. Let $\text{bn}(G) = k$. We will show that G has a tree decomposition of width at most $k - 1$. Let \mathcal{B} be a bramble in G . We say that a tree decomposition (T, β) is \mathcal{B} -good, if all $t \in V(T)$ satisfy

$$\text{if } \beta(t) \text{ is a hitting set for } \mathcal{B} \text{ then } |\beta(t)| \leq k.$$

We prove the following stronger statement.

(*) For every bramble \mathcal{B} in G there exists a \mathcal{B} -good tree decomposition of G .

Given (*), we can simply choose $\mathcal{B} = \emptyset$. Then every subset of G is a hitting set for \mathcal{B} and hence a \mathcal{B} -good tree decomposition witnesses $\text{tw}(G) \leq k - 1$.

Towards a contradiction, assume that (*) does not hold. Let \mathcal{B} be a bramble in G of maximum cardinality such that no \mathcal{B} -good tree decomposition exists. (Such a bramble exists, because the cardinality of every bramble in G is bounded by $2^{|V(G)|}$.) Let X be a hitting set for \mathcal{B} of minimum cardinality $|X| = \text{ord}(\mathcal{B}) =: \ell$.

Claim: For every connected component Y of $G \setminus X$ the graph $G[X \cup Y]$ has a tree decomposition (T, β) satisfying

- (T, β) is \mathcal{B} -good, and
- there exists a node $t \in V(T)$ with $X = \beta(t)$.

The Claim implies (*), because the tree decompositions guaranteed by the Claim can be glued together at the nodes with bag X to obtain the desired tree decomposition of G .

Proof. Let Y be a connected component of $G \setminus X$.

Case 1: There exists a set $A \in \mathcal{B}$ such that Y and A do not touch. Then the tree decomposition consisting of K_2 with the two bags X and $Y \cup N_G(Y)$ has the desired properties.

Case 2: For every $A \in \mathcal{B}$ the sets A and Y touch. Then $\mathcal{B}' := \mathcal{B} \cup \{Y\}$ is a bramble with $|\mathcal{B}'| > |\mathcal{B}|$. Due to maximality of $|\mathcal{B}|$, there exists a \mathcal{B}' -good tree decomposition (T, β') . Since (T, β') is not \mathcal{B} -good, there exists a node $s \in V(T)$ with $|\beta'(s)| > k$ such that $\beta'(s)$ is a hitting set for \mathcal{B} . Note that $\beta'(s)$ cannot be a hitting set for \mathcal{B}' and hence $\beta'(s) \cap Y = \emptyset$.

We will now transform the tree decomposition (T, β') into a tree decomposition for $G[X \cup Y]$ satisfying the conditions of the Claim. In particular, we will replace the bag $\beta'(s)$ by X . This is done as follows.

By Lemma 3.9 there are ℓ pairwise vertex-disjoint paths P_1, \dots, P_ℓ from X to $\beta'(s)$ (see Figure 3.3).

For $i \in [\ell]$ let x_i be the first vertex of P_i . Then $X = \{x_1, \dots, x_\ell\}$. Since $\beta'(s) \cap V(Y) = \emptyset$ the set X separates Y from $\beta'(s)$ and for every $i \in [\ell]$ we have $V(P_i) \cap Y = \emptyset$.

For $i \in [\ell]$ choose $t_i \in V(T)$ with $x_i \in \beta'(t_i)$, and let $U_i \subseteq V(T)$ be the set of nodes of the path from s to t_i in T (see Figure 3.4). For every node $t \in V(T)$ let

$$\beta(t) = (\beta'(t) \cap (X \cup Y)) \cup \{x_i \mid t \in U_i, i \in [\ell]\}.$$

We now show that (T, β) is a \mathcal{B} -good tree decomposition of $G[X \cup Y]$. For this, we first verify conditions (T1) and (T2) using the fact that (T, β') is a tree decomposition of G .

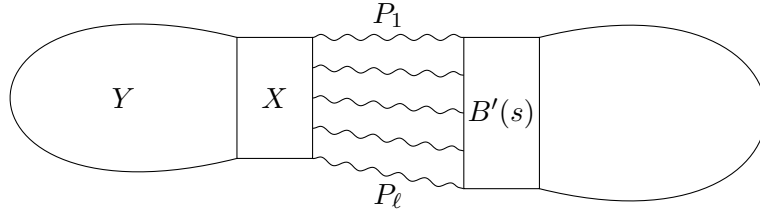


Figure 3.3.: The graph G in the proof of Theorem 3.10, Case 2. Here, $G - X$ has two connected components.

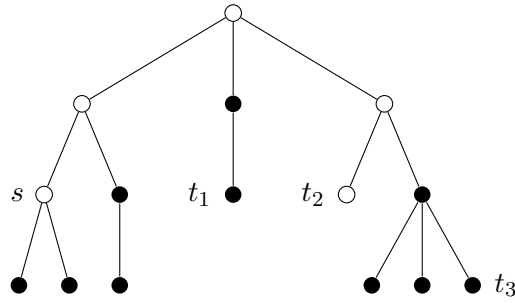


Figure 3.4.: Proof of Theorem 3.10: The tree T with the nodes t_1 , t_2 and t_3 , covering x_1 , x_2 and x_3 , respectively. The white nodes show the set U_2 .

(T1): Let $e \in E(G[X \cup Y]) \subseteq E(G)$. Choose $t \in V(T)$ such that $e \subseteq \beta'(t)$. Then $e \subseteq \beta(t) \cap (X \cup Y) \subseteq \beta(t)$.

(T2): Every vertex $v \in Y$ satisfies $\beta^{-1}(v) = (\beta')^{-1}(v)$ and hence $\beta^{-1}(v)$ is non-empty and connected. Every $v \in X$, say $v = x_i$, satisfies $\beta^{-1}(v) = (\beta')^{-1}(x_i) \cup U_i$. Both sets $(\beta')^{-1}(x_i)$ and U_i are connected in T , and they have a non-empty intersection, because $t_i \in (\beta')^{-1}(x_i) \cap U_i$. Therefore $(\beta')^{-1}(x_i) \cup U_i$ is connected and non-empty and (T, β) satisfies (T2).

Before we show that (T, β) is \mathcal{B} -good, observe that

(a) $|\beta'(t)| \geq |\beta(t)|$ for all $t \in V(T)$.

(Either $\beta(t) \subseteq \beta'(t)$, or there exists a vertex $x_i \in \beta(t) \setminus \beta'(t)$ for some $i \in [\ell]$. But then x_i replaces a vertex on the path P_i , and since for $i \neq j \in [\ell]$ the paths P_i and P_j do not share vertices, the observation follows.)

Now let $t \in V(T)$ with $|\beta(t)| > k$. Our goal is to show that $\beta(t)$ is not a hitting set for \mathcal{B} . By (a) we have $|\beta'(t)| > k$. Moreover, $\beta(t) \cap Y \neq \emptyset$, because $\beta(t) \subseteq X \cup Y$ and $|X| \leq k$. Hence

(b) $\beta'(t) \cap Y \neq \emptyset$.

Since (T, β') is \mathcal{B}' -good, $\beta'(t)$ is not a hitting set for \mathcal{B}' . By (b), $\beta'(t) \cap V(Y) \neq \emptyset$, so there exists a set $A \in \mathcal{B}$ such that $\beta'(t) \cap A = \emptyset$. If we can show that $\beta(t) \cap A = \emptyset$, then $\beta(t)$ is not a hitting set for \mathcal{B} and we are done.

Assume that $\beta(t) \cap A \neq \emptyset$. Then $x_i \in \beta(t) \cap A$ for some $i \in [\ell]$. But this implies that t lies on the path from s to t_i in T . Thus

- $A \cap \beta(t_i) \neq \emptyset$, and
- $A \cap \beta(s) \neq \emptyset$ (because $\beta(s)$ is a hitting set for \mathcal{B}).

Since A is connected, by Lemma 2.9 the set $(\beta')^{-1}(A)$ is connected in T , and hence $t \in (\beta')^{-1}(A)$ and $\beta'(t) \cap A \neq \emptyset$, a contradiction. This proves that (T, β) is a \mathcal{B} -good, completing the proof of the Claim. \dashv

Property (*) follows immediately from the Claim, completing the proof. \square

Now we are ready for the main theorem of this section.

Theorem 3.11 *Let $k \in \mathbb{N}$. For every graph G the following are equivalent:*

- (1) $\text{tw}(G) + 1 \leq k$
- (2) $\text{mcw}(G) \leq k$
- (3) $\text{cw}(G) \leq k$

(4) $\text{bn}(G) \leq k$

Proof. ‘1 \Rightarrow 2’: Let (T, β) be a tree decomposition of G of width $k-1$. We describe a winning strategy S for the cops in $\text{MCR}(G, k)$. Choose a root $r \in V(T)$. Intuitively, the cops start by moving to $\beta(r)$ and then they chase the robber down a branch of T by moving to one bag after the other following the robber, and they finally catch her in a leaf.

By (T2), for every vertex $v \in V(G)$ and for every node $t \in V(T)$ with $v \notin \beta(t)$ there is a unique connected component $C_{v,t}$ of $T \setminus t$ such that $\beta^{-1}(v) \subseteq C_{v,t}$. We let $s_{v,t} \in N_T(t) \cap C_{v,t}$ denote the neighbour of t in $C_{v,t}$.

Formally, we begin with letting $S() = \emptyset := X_0$. The robber chooses a vertex $y_0 \in V(G)$, and we let $S(X_0, y_0) = \beta(r)$. If r is the only node of T , then $y_0 \in \beta(r)$ and we are done.

Otherwise, assume that the play is in position (X_i, y_i) with $y_i \notin X_i$. If $X_i = \beta(t)$ for some $t \in V(T)$ we let $S((X_0, y_0), \dots, (X_i, y_i)) = \beta(s_{y_i, t})$. For all other partial plays $(X_0, y_0), \dots, (X, y)$ with $y \notin X$ we let $S((X_0, y_0), \dots, (X, y)) = X$.

Using Lemma 2.16 it is straightforward to see that for every play $(X_0, y_0), (X_1, y_1), \dots$ played according to S and for every $i > 0$ the following hold:

- there is a node $t_i \in V(T)$ with $X_i = \beta(t_i)$,
- $\text{dist}_T(r, t_i) = i - 1$, and
- the robber space $\text{comp}(X_i, y_i)$ satisfies $\text{comp}(X_i, y_i) \subseteq \beta(T_{t_{i+1}}) \setminus \beta(t_i)$, and $\text{comp}(X_i, y_i)$ is a connected component of $G \setminus (X_i \cap X_{i+1}) = G \setminus (\beta(t_i) \cap \beta(t_{i+1}))$.

Hence S is a monotone winning strategy.

The implication ‘2 \Rightarrow 3’ is obvious, ‘3 \Rightarrow 4’ is proved in Remark 3.5, and ‘4 \Rightarrow 1’ is proved in Theorem 3.10. \square

4. Computing Tree Decompositions

Many computational problems that are NP-hard in general can be solved efficiently when restricted to graphs of bounded tree width. All problems considered in Chapter 1 are of this kind. Typically, such algorithms are based on dynamic programming ‘bottom-up’ along the tree decomposition. Hence we first need to compute a tree decomposition of the input graph.

In this chapter we discuss algorithms for computing tree decompositions. We begin with a negative result, that we mention without a proof.

Theorem 4.1 (Arnborg, Corneil und Proskurowski) *The Problem*

TREE WIDTH
Input: graph G , $k \in \mathbb{N}$.
Problem: Is $\text{tw}(G) = k$

is NP-complete.

Hence, determining the tree width of a graph is infeasible in general.

Definition 4.2 *A class \mathcal{C} of graphs has bounded tree width, if there is an integer $k \in \mathbb{N}$, such that all graphs $G \in \mathcal{C}$ satisfy $\text{tw}(G) \leq k$.* *bounded tree width*

If we restrict our input graphs to a class of bounded tree width, then efficient algorithms for computing tree width and tree decompositions (of small width) exist.

The following theorem, which we do not prove here, implies that on graph classes of bounded tree width, we can compute tree decompositions of minimal width in polynomial time.

Theorem 4.3 (Bodlaender) *There is an algorithm, that given a graph G , computes a tree decomposition of G of width $k := \text{tw}(G)$ in time*

$$2^{O(k^3)} \cdot |V(G)|.$$

We present two algorithms for computing tree decompositions here. The first one is based on winning strategies for the cops, and the second one uses separators in graphs of bounded tree width. While the first algorithm is in the (parameterised) complexity class XP (with parameter the tree width) and computes the tree width exactly, the second one is even in FPT, but computes the tree width only up to a constant factor.

4.1. Tree Decompositions from Winning Strategies

Definition 4.4 A strategy S for the cops in the game $\text{MCR}(G, k)$ is strictly monotone, if all positions (X, y) of $\text{MCR}(G, k)$ with $X' := S(X, y)$ satisfy

$$X' \cap \text{comp}(X, y) \neq \emptyset.$$

This means that in every round the cops must move to at least one vertex in the robber space.

Lemma 4.5 Let G be a graph and let $k \in \mathbb{N}$.

- (1) If the cops have a winning strategy in the game $\text{MCR}(G, k)$, then the cops have a strictly monotone winning strategy in the game $\text{MCR}(G, k)$.
- (2) If the cops have a winning strategy in the game $\text{CR}(G, k)$, then the cops have a winning strategy, in which they catch the robber in at most $|V(G)|$ rounds.

Proof. (1) If the cops have a winning strategy in $\text{MCR}(G, k)$, then by 3.11 we have $\text{tw}(G) + 1 \leq k$. Let (T, β) be a small tree decomposition of G of width $w(T, \beta) = \text{tw}(G)$. (which exists by Lemma 2.17), rooted at some node. It is easy to see that we may assume that for all $t \in V(T)$ the set $\beta(T_t) \setminus \beta(t)$ is a connected component of $G \setminus \beta(t)$. Let S be the winning strategy, that we obtain by following the tree decomposition (T, β) , as in the proof of Theorem 3.11. This strategy is strictly monotone, because (T, β) is small, and it uses only $\text{tw}(G) + 1 \leq k$ cops.

(2) This follows from Theorem 3.11 and Part (1). \square

Theorem 4.6 There is an algorithm, that, given a graph G and an integer $k \in \mathbb{N}$, computes a strictly monotone winning strategy for k cops in time $|V(G)|^{O(k)}$, if such a winning strategy exists. Otherwise it outputs $\text{cw}(G) > k$.

Proof. Let $G = (V, E)$ and $n := |V|$. For $i \geq 0$ let

$$W_i := \{(X, y) \in [V]^{\leq k} \times V \mid \text{in position } (X, y) \text{ the cops can win in } \leq i \text{ rounds with a strictly monotone strategy}\}.$$

Then all $(X, y) \in [V]^{\leq k} \times V$ satisfy

- $(X, y) \in W_0 \iff y \in X$, and
- $(X, y) \in W_{i+1} \iff y \in X$ or there is a set $X' \in [V]^{\leq k}$ with $\text{comp}(G \setminus X, y) = \text{comp}(G \setminus (X \cap X'), y) =: R$ and $X' \cap R \neq \emptyset$, such that all $y' \in R$ satisfy: $(X', y') \in W_i$.

With this, we can compute W_{i+1} from W_i in time $n^{O(k)}$. The cops win, if and only if all $y \in V(G)$ satisfy $(\emptyset, y) \in W_n$. Moreover, for every position we can compute a suitable set X and hence a strictly monotone winning strategy. \square

Theorem 4.7 *There is an algorithm, that, given a graph G , computes a tree decomposition of G of width $\text{tw}(G)$ in time $|V(G)|^{O(\text{tw}(G))}$.*

Proof. For $k = 1, 2, \dots$, test whether $\text{cw}(G) \leq k$ and compute a strictly monotone winning strategy. From this compute the desired tree decomposition. \square

Corollary 4.8 *Let \mathcal{C} be a class of graphs of bounded tree width. Then there is a polynomial time algorithm that, given a graph $G \in \mathcal{C}$, computes a tree decomposition of G of width $\text{tw}(G)$.* \square

Note that in the running time of Theorem 4.7, the tree width contributes to the exponent of the size of the graph. (It shows that the problem belongs to the parameterized complexity class XP.) It is actually possible to ‘separate’ the contribution of the tree width from the size of the graph, as can be seen in Bodlaender’s Theorem (Theorem 4.3). (It shows that the problem belongs to the parameterized complexity class FPT – which is a subclass of the XP that is considered to contain the problems that are ‘efficiently’ solvable in the sense of parameterised complexity, see [4].)

4.2. Tree decompositions from separators

In this section we will present an algorithm for computing tree decompositions, that is based on separators. In principle, the proof of Lemma 2.28 already yields such an algorithm. However, computing balanced separators is computationally difficult. Therefore, we will now use separators, which do not separate a vertex set W exactly into parts of size at most one half of W .

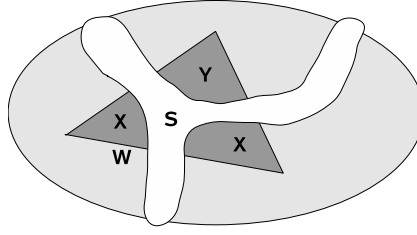
Definition 4.9 *Let G be a graph and let $W \subseteq V(G)$. A nearly balanced separation of W is a triple (X, S, Y) where $X, Y \subseteq W$ and $S \subseteq V(G)$, such that*

nearly balanced separation

- (1) $W = X \cup (S \cap W) \cup Y$,
- (2) S separates X and Y , and
- (3) $0 < |X| < \frac{2}{3}|W|$ and $0 < |Y| < \frac{2}{3}|W|$.

The order of (X, S, Y) is $|S|$.

order of a separation



Lemma 4.10 *Let G be a graph and $k := \text{tw}(G) \geq 2$. Furthermore, let $W \subseteq V(G)$ with $|W| \geq 2k + 3$. Then there exists a nearly balanced separation of W of order $\leq k + 1$.*

Proof. Let S be a W -separator in G of order $k + 1$, as exists by Lemma 2.25. Let C_1, \dots, C_m be the connected components of $G \setminus S$ and let $W_i := C_i \cap W$ for all $i \in [m]$. Hence, $|W_i| \leq \frac{|W|}{2}$ for all $i \in [m]$. In particular, we have $m \geq 2$, because $W \setminus S = \bigcup_{i=1}^m W_i$ and

$$|W \setminus S| \geq |W| - (k + 1) > \frac{|W|}{2}, \quad \text{as } |W| \geq 2k + 3.$$

W.l.o.g. we assume that $|W_1| \geq |W_2| \geq \dots \geq |W_m|$. Let $i \in [m]$ be minimal with

$$\sum_{j=1}^i |W_j| > \frac{|W \setminus S|}{3}.$$

(This is well-defined, as $\sum_{j=1}^m |W_j| = |W \setminus S| > \frac{|W \setminus S|}{3}$.) We define

$$X := \bigcup_{j=1}^i W_j \quad \text{and} \quad Y := \bigcup_{j=i+1}^m W_j.$$

It follows that $|X| > \frac{|W \setminus S|}{3}$ and therefore

$$|Y| = |W \setminus S| - |X| < \frac{2}{3}|W \setminus S| \leq \frac{2}{3}|W|.$$

Claim 1: $|X| \leq \frac{2}{3}|W|$ and $|Y| > 0$.

Proof. If $i = 1$, then

$$|X| = |W_1| \leq \frac{|W|}{2} \leq \frac{2}{3}|W|.$$

So let $i > 1$. Then $|W_i| \leq |W_1| \leq \frac{|W \setminus S|}{3}$ for all $i \in [m]$, and therefore

$$|X| = \sum_{j=1}^i |W_j| = \sum_{j=1}^{i-1} |W_j| + |W_i| \leq \frac{|W \setminus S|}{3} + \frac{|W \setminus S|}{3} = \frac{2}{3}|W \setminus S| \leq \frac{2}{3}|W|.$$

Furthermore

$$|Y| = |W \setminus S| - |X| \geq \frac{1}{3}|W \setminus S| > 0.$$

–

□

We state the next lemma without proof. It is based on Mengers Theorem (Theorem 3.8), and for computing a W -separation it extends standard network flow techniques – e.g. the algorithm of Ford and Fulkerson for computing a maximal set of pairwise disjoint paths between two vertex sets.

Lemma 4.11 *The problem*

Input: Graph $G, k \in \mathbb{N}$ and $W \subseteq V(G)$ with $|W| = 3k + 1$.
Problem: Compute a W -separation of order $\leq k + 1$, if it exists.

can be solved in time $\mathcal{O}(3^{3k} \cdot k \cdot |E(G)|)$.

Using Lemma 4.11, we can now present an algorithm for computing tree decompositions.

Theorem 4.12 *There exists an algorithm which, given a graph G as input, computes a tree decomposition of G of width at most $4 \cdot \text{tw}(G) + 1$ in time $2^{\mathcal{O}(k)} |V(G)|^2$.*

Proof. We first present an algorithm which, given a graph $G = (V, E)$ and an integer $k \in \mathbb{N}$, either computes a tree decomposition of G of width $\leq 4 \cdot k + 1$ or determines that $\text{tw}(G) > k$. For this, we call the algorithm TDEC in Figure 4.1 with parameters $\text{TDEC}(G, \emptyset, k)$.

We first show the correctness of the algorithm. The proof of correctness is analogous to the proof of Lemma 2.28. For all W_i in line 13:

$$|W_i| = |W' \cap C_i| + |S| \leq \lfloor \frac{2}{3}(3k + 1) \rfloor + k + 1 \leq 2k + k + 1 \leq 3k + 1.$$

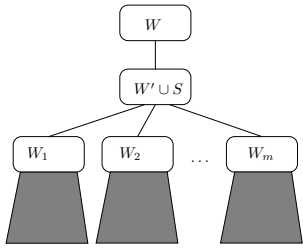
The correctness of the algorithm now follows easily from this. Furthermore, $|W' \cup S| \leq 4k + 2$ and therefore the width of the tree decomposition returned by the algorithm is at most $4k + 1$.

Towards the running time, we assume w.l.o.g. that $|S| = k + 1$. Let $n := |V(G)|$. The running time can then be estimated by the following recurrence $T(n)$:

$$T(n) := \begin{cases} \mathcal{O}(k) & \text{if } n \leq 4k + 2 \\ \max_{m, n_1, \dots, n_m} \left(\sum_{i=1}^m T(n_i) \right) + \mathcal{O}(3^{3k} \cdot k \cdot n) & \text{otherwise} \end{cases}$$

where $n_i := |V(G_i)|$ and the maximum is taken over all $m \geq 2$ and $n_1, \dots, n_m < n$ with $\sum_{i=1}^m (n_i - (k + 1)) = n - (k + 1)$.

```

1: procedure TDEC( $G, W, k$ )
    Graph  $G := (V, E)$ ,  $k \in \mathbb{N}$ 
     $W \subseteq V(G)$  with  $|W| \leq 3k + 1$ .
2:   if  $|E| > k \cdot |V|$  then
3:     stop with output  $\text{tw}(G) > k$  ▷ See Theorem 2.23
4:   end if
5:   if  $|V| \leq 4k + 2$  then
6:     return trivial tree decomposition of  $G$ 
7:   end if
8:   Choose  $W' \supseteq W$  with  $|W'| = 3k + 1$ 
9:   if there is a nearly balanced  $W'$ -separation  $(X, S, Y)$  of order  $\leq k + 1$  then
10:    Let  $C_1, \dots, C_m$  be the connected components of  $G \setminus S$ .
11:    for  $i = 1, \dots, m$  do
12:       $G_i := G[C_i \cup S]$ 
13:       $W_i := (C_i \cap W') \cup S$ 
14:       $(T_i, (\beta_i(t))_{t \in V(T_i)}) \leftarrow \text{TDEC}(G_i, W_i, k)$ 
15:    end for
16:    Return the following tree decomposition:
      
17:   else
18:     stop with output  $\text{tw}(G) > k$  ▷ There is no separation
19:   end if
20: end procedure

```

Figure 4.1.: The algorithm TDEC

Now let $T'(l') = T(l' + (k + 1))$. Then

$$T'(n') := \begin{cases} \mathcal{O}(k) & \text{if } n' \leq 3k + 1 \\ \max_{m, n'_1, \dots, n'_m} \left(\sum_{i=1}^m T'(n'_i) \right) + \mathcal{O}(3^{3k} \cdot k \cdot (n' + k + 1)) & \text{otherwise} \end{cases}$$

where we now maximise over all $m \geq 2$ and $n'_1, \dots, n'_m < n'$ such that $\sum_{i=1}^m n'_i = n'$.

It follows that $T'(n') \leq c \cdot 3^{3k} \cdot k \cdot (n')^2$, for a suitable constant c , and therefore $T(n) = T'(n - (k + 1)) \leq c \cdot 3^{3k} \cdot k \cdot n^2$.

To compute a tree decomposition of width at most $4 \cdot \text{tw}(G) + 1$, we run $\text{TDEC}(G, \emptyset, k)$ for $k = 0, 1, \dots$ counting upwards until we get the first success. \square

We recall Bodlaender's theorem, stating that a tree decomposition of optimal width can be computed in time $2^{\mathcal{O}(k^3)} \cdot n$. The algorithm consists of two parts. In the first part, an approximate tree decomposition (i.e. of non-optimal width) is computed. Based on this tree decomposition, an optimal tree decomposition is then computed by dynamic programming. In principle, one could run the second part of Bodlaender's algorithm on the tree decompositions found by our algorithm.

Even though Bodlaender's algorithm is much faster than ours in terms of the dependence on the graph size, for practical purposes the cubic dependency of the exponent in Bodlaender's algorithm is a serious problem, in addition to its immense technicalities which make implementing it a difficult task.

We close this section by citing a Theorem recently proved in [2]. This theorem allows us to compute tree decompositions of approximately optimal width with both a single-exponential dependence on the tree width and linear in the number of vertices of the input graph.

Theorem 4.13 *There is an algorithm that for an input n -vertex graph G and integer $k > 0$, in time $2^{\mathcal{O}(k)} \cdot n$ either outputs that the tree width of G is larger than k , or gives a tree decomposition of G of width at most $5k + 4$.*

5. Algorithms on Tree Decompositions

5.1. Dynamic programming on graphs of bounded tree width

As a first example we will solve 3-COL on graph classes of bounded tree width.

3-COL
Input: Graph G , $k \in \mathbb{N}$.
Problem: Is G 3-colorable?

Theorem 5.1 *Let \mathcal{C} be a class of graphs of bounded tree width. Then 3-COL can be decided in linear time on \mathcal{C} .*

Proof. Assume that all graphs in \mathcal{C} have tree width at most $k \in \mathbb{N}$. Let $G \in \mathcal{C}$ be given and let $n := |V(G)|$. We first compute a small tree decomposition (T, β) of width at most k of G in linear time. This can be done using Bodlaender's algorithm (Theorem 4.3). We choose an arbitrary node $r \in V(T)$ as root.

The algorithm computes, starting from the leaves, for all $t \in V(T)$ the following information.

Col(t): Set of all proper 3-colourings of $G[\beta(t)]$.

ExCol(t): Set of all colourings from $Col(t)$, which can be extended to a valid 3-colouring of $G[\beta(T_t)]$.

Obviously, G is 3-colourable if, and only if, $ExCol(r) \neq \emptyset$.

The data structures $Col(t)$ and $ExCol(t)$ can be computed as follows. Let $t \in V(T)$. The set $Col(t)$ contains all proper 3-colourings of $G[\beta(t)]$, i.e. all functions from $\beta(t)$ to $\{1, 2, 3\}$, which are proper colourings. There are at most 3^{k+1} functions from $\beta(t)$ to $\{1, 2, 3\}$, and for each of these we can decide in time $\mathcal{O}((k+1) \cdot k)$ whether it is a proper colouring. Therefore, the running time of this step is $\mathcal{O}(3^{k+1} \cdot (k+1)k)$.

If t is a leaf of T , then $ExCol(t) = Col(t)$. Otherwise, let t_1, \dots, t_m be the children of t . Suppose that for $i \in [m]$, the set $ExCol(t_i)$ has already been computed. We compute $ExCol(t)$ as follows. For all $c \in Col(t)$ we test whether for every $i \in [m]$ there is a colouring $c_i \in ExCol(t_i)$ such that $c(v) = c_i(v)$ for all $v \in B_t \cap B_{t_i}$. If this is the case, we add c to $ExCol(t)$, otherwise we drop it.

Clearly, every colouring in $ExCol(t)$ can be extended to a colouring of $G[\beta(T_t)]$. Furthermore, for every proper 3-colouring c of $\beta(T_t)$, its restriction $c_{\beta(t)}$ to $\beta(t)$ is a proper colouring and hence in $ExCol(t)$.

The time needed to compute $ExCol(t)$ can be estimated as $3^{k+1} \cdot 3^{k+1} \cdot m \cdot k$.

In total, for each edge in $E(T)$ the algorithm requires time $\mathcal{O}(3^{2(k+1)} \cdot k)$. As (T, β) is small, we have $|E(T)| \leq |V(T)| \leq |V(G)|$ and therefore a total running time of $2^{\mathcal{O}(k)} \cdot n$. Since k is fixed, this proves that the problem can be solved in linear time. \square

Definition 5.2 *The number*

$$\chi(G) := \min\{k : G \text{ is } k\text{-colourable}\}$$

is called the chromatic number of G .

Lemma 5.3 *Every graph G satisfies $\chi(G) \leq \text{tw}(G) + 1$.*

Proof. We use induction on $|V(G)|$. Let (T, β) be a small tree decomposition of G of width $k := \text{tw}(G)$. If $|V(G)| \leq k + 1$, then there is nothing to show. Otherwise, $|V(G)| > k + 1$. Let t be a leaf of T with parent s and let $v \in \beta(t) \setminus \beta(s)$. (Such a vertex exists as (T, β) is small.) By induction hypothesis, there is a $k + 1$ -colouring $c: V(G) \setminus \{v\} \rightarrow [k + 1]$ of $G \setminus v$. As $|\beta(t)| \leq k + 1$, there is a colour i such that $c(u) \neq i$ for all $u \in \beta(t) \setminus \{v\}$. Then $c': V(G) \rightarrow [k + 1]$ with $c'(u) := c(u)$ for all $u \neq v$ and $c'(v) := i$ is a proper $(k + 1)$ -colouring of G . \square

This immediately implies the next theorem.

Theorem 5.4 *Let \mathcal{C} be a class of graphs of bounded tree width. The problem*

MINCOL
<i>Input:</i> G Graph.
<i>Problem:</i> Compute $\chi(G)$

can be solved in linear time on \mathcal{C} .

Proof. Let k be an upper bound on the tree width of $G \in \mathcal{C}$. By Lemma 5.3, $\chi(G) \leq k + 1$ for all $G \in \mathcal{C}$. For every $i \leq k + 1$ we can decide if G is i -colourable in the same way as for 3-colourability in Theorem 5.1. \square

The colouring problem demonstrates an important type of algorithms for solving problems on graph classes of small tree width. Many other problems can be solved in the same way. For this, it is often useful to convert tree decompositions into a simpler form.

Definition 5.5 *A tree decomposition (T, β) of a graph G is nice, if it satisfies the following conditions:*

(1) T is a rooted tree where every node has at most two children.

(2) Every node $t \in V(T)$ is of one of the following types:

Leaf: t is a leaf of T and $|\beta(t)| = 1$.

Introduce (+): t has exactly one child s and $\beta(s) \subseteq \beta(t)$ and $|\beta(t)| = |\beta(s)| + 1$.

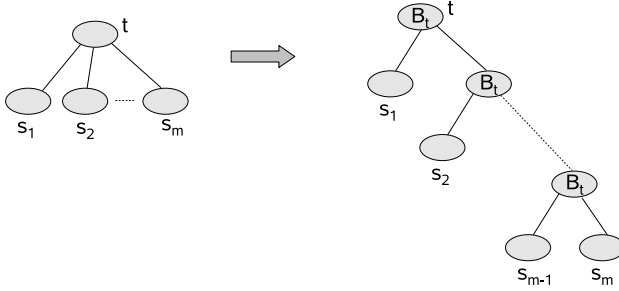
Forget (-): t has exactly one child and $\beta(t) \subseteq \beta(s)$ and $|\beta(t)| = |\beta(s)| - 1$.

Join (2): t has exactly two children t_1, t_2 and $\beta(t) = \beta(t_1) = \beta(t_2)$.

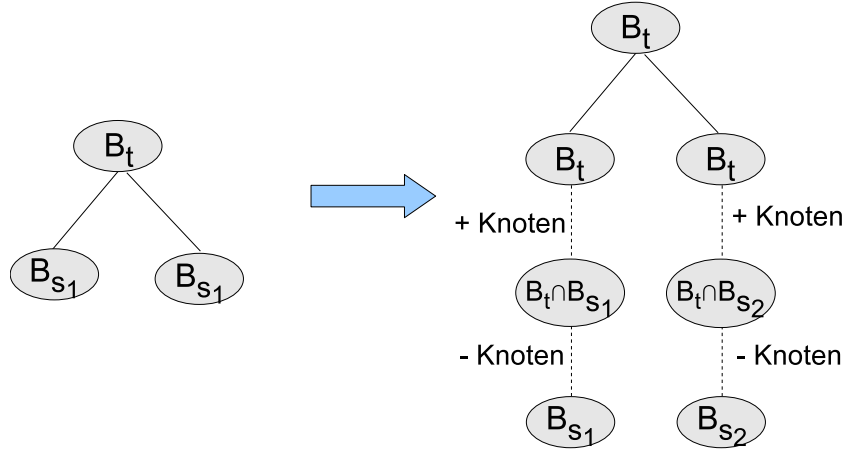
Lemma 5.6 Every non-empty graph G has a nice tree decomposition of width $k := \text{tw}(G)$ which can be computed in time $2^{\mathcal{O}(k^3)}|E(G)|$.

Proof. As a first step we compute a tree decomposition of G of width $\text{tw}(G)$ using Theorem 4.3. We then apply the following transformations as long as possible. As a result we obtain a nice tree decomposition.

Step 1: If there is a node with more than two children, we replace it by a chain of nodes with two children as follows:

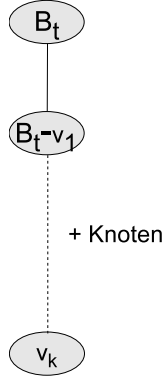


Step 2: In the second step we deal with nodes with exactly 2 children. Let t be a node with children s_1, s_2 such that $B_{s_1} \neq \beta(t)$ or $\beta(s_2) \neq \beta(t)$. We create two new children t_1, t_2 of t labelled by $\beta(t)$. Starting with t_1 we successively remove through a sequence of +-nodes all $v \in \beta(t) \setminus \beta(s_1)$ until we reach a node s labelled $\beta(t) \setminus \beta(s_1)$. We then add a chain of --nodes to s which successively add the vertices in $\beta(s_1) \setminus \beta(t)$. We proceed in the same way for t_2 . The following figure demonstrates this step.



Step 2': We proceed in a similar way with nodes having exactly one child.

Step 3: Finally, we replace the leaves by a sequence of $+$ -nodes as follows. A leaf t with $\beta(t) := \{v_1, \dots, v_k\}$ is replaced by a path $P := (t_1, \dots, t_k)$ with $\beta(t_i) := \{v_i, \dots, v_k\}$.



□

Many problems on graphs of small tree width can now be solved in the following way.

General dynamic programming approach.

Input. Graph G of tree width k .

- (1) Compute a nice tree decomposition (T, β) of G .
- (2) Starting with the leaves, for all $t \in V(T)$ compute a list \mathcal{L}_t of “partial solutions”.

(3) Read off the answer from the partial solutions of the root.

In most cases, the running time of such algorithms is bounded by $f(k) \cdot |V(G)|$, for some function $f: \mathbb{N} \rightarrow \mathbb{N}$. Often, f is of the form $2^{p(k)}$ for a polynomial p .

Recall. The Hamiltonian-Path Problem is the problem to decide for a given graph G whether G contains a Hamiltonian-cycle, i.e. a cycle that contains every vertex.

Theorem 5.7 *The problem*

HAMILTONIAN-CYCLE

Input: Graph G .

Problem: Does G contain a Hamiltonian-cycle?

can be decided in time $2^{\text{tw}(G)^{O(1)}} \cdot |V(G)|$. In particular, if \mathcal{C} is a class of graphs of bounded tree width, then HAMILTONIAN-CYCLE can be solved in linear time on \mathcal{C} .

Proof. Let G be a graph. We may assume that $|V(G)| \geq 3$. In the first step we compute a nice tree decomposition (T, β) of G . Our first aim is to compute partial solutions ('partial Hamilton cycles') at each node $t \in V(T)$. Towards this, let

$$M_t := \left\{ \mathcal{P} : \begin{array}{l} \mathcal{P} \text{ is a collection of paths in } \beta(T_t) \text{ with endpoints in } \beta(t) \text{ and inner} \\ \text{vertices in } \beta(T_t) \setminus \beta(t), \text{ such that the paths in } \mathcal{P} \text{ are pairwise disjoint} \\ \text{except that vertices in } \beta(t) \text{ may be endpoints of two paths in } \mathcal{P}, \\ \text{and } \beta(T_t) = \bigcup_{P \in \mathcal{P}} V(P). \end{array} \right\}.$$

Notice that if $C \subseteq G$ is a Hamilton cycle, then the restriction of C to $G[\beta(T_t)]$ is a member of M_t . We cannot keep track of M_t , because our running time would be exponential in $|V(G)|$. To avoid this, we define an equivalence relation on M_t as follows. For $\mathcal{P}, \mathcal{Q} \in M_t$ we let $\mathcal{P} \sim \mathcal{Q}$ if and only if all pairs $x, y \in \beta(t)$ satisfy: there is a path P with endpoints x, y in \mathcal{P} if, and only if, there is a path P' with endpoints x, y in \mathcal{Q} .

Clearly, it suffices to keep track of the equivalence classes only. We do this by using 'pattern graphs'. A *pattern graph* at $t \in V(T)$ is a graph G with $V(G) = \beta(t)$ and $\deg_G(v) \leq 2$ for all $v \in V(G)$. For convenience, we allow pattern graphs to have double (parallel) edges and loops. Here two parallel edges contribute 2 to the degree of each of their ends. A loop also contributes 2 to the degree of its end. Moreover, cycles in pattern graphs can have length 2 or 1.

In a first step, for every node $t \in V(T)$ we compute the set

$$\text{Pat}_t := \{G \mid G \text{ is a pattern graph at } t\}.$$

For $t \in V(T)$ and $\mathcal{P} \in M_t$, let $G(\mathcal{P})$ be the graph with vertex set $\beta(t)$, where two vertices $u, v \in \beta(t)$ are joined by an edge if and only if there is a path from x to y in \mathcal{P} . Clearly, $G(\mathcal{P})$ is a pattern graph at t .

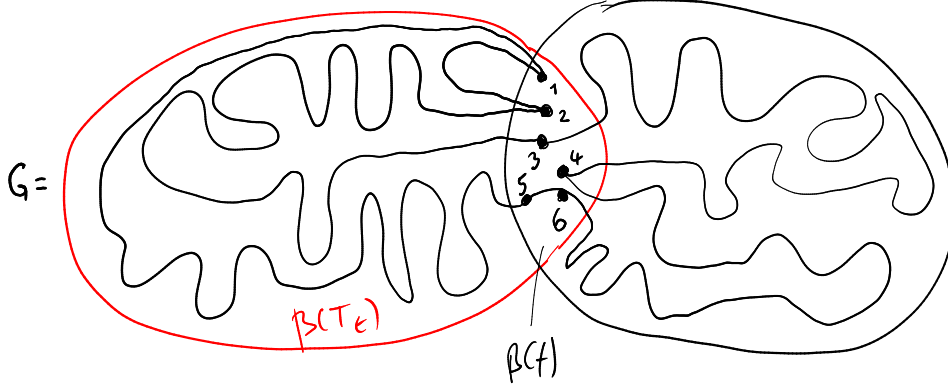


Figure 5.1.: A graph with a Hamilton cycle intersecting bag $\beta(t)$

As an example, Figure 5.1 shows a graph with a hamiltion cycle C and bag $\beta(t) = \{1, 2, 3, 4, 5, 6\}$. The restriction of C to $\beta(T_t)$ is a member \mathcal{P} of M_t and the corresponding pattern graph $G(\mathcal{P})$ has vertex set $\{1, 2, 3, 4, 5, 6\}$ and edge set $\{12, 15, 23, 56\}$.

Any two elements $\mathcal{P}, \mathcal{Q} \in M_t$ satisfy $\mathcal{P} \sim \mathcal{Q}$, if and only if $G(\mathcal{P}) = G(\mathcal{Q})$. We let $\mathcal{L}_t := \{G(\mathcal{P}) \mid \mathcal{P} \in M_t\} \subseteq \text{Pat}_t$. Then \mathcal{L}_t containg precisely one patter graph for each equivalence class of M_t . Now we show how to compute the sets \mathcal{L}_t . Let $t \in V(T)$.

Case 1: t is a leaf. Then $\beta(t) = \{v\}$ for a vertex $v \in V(G)$ and $\mathcal{L}_t = \{(\{v\}, \emptyset)\}$.

Case 2: t is an introduce node. Let s be the child of t and let $v \in \beta(t) \setminus \beta(s)$. Then

$$\mathcal{L}_t = \{G \in \text{Pat}_t \mid G[\beta(s)] \in \mathcal{L}_s \text{ and the edges of } G \text{ incident to } v \text{ are in } E(G)\}.$$

Case 3: t is a forget node. Let s be the child of t and let $v \in \beta(s) \setminus \beta(t)$. Then

$$\mathcal{L}_t = \{(G-v)+xy \in \text{Pat}_t \mid G \in \mathcal{L}_s, \deg_G(v) = 2, vv \notin E(G) \text{ and } vx, vy \in E(G)\}.$$

Case 4: t is a join node. Let s_1 and s_2 be the children of t . Then

$$\mathcal{L}_t = \{G \in \text{Pat}_t \mid G = G_1 \cup G_2 \text{ for } G_1 \in \mathcal{L}_{s_1} \text{ and } G_2 \in \mathcal{L}_{s_2}\}.$$

At the root $r \in V(T)$, we simply test whether there is a pattern graph $G \in \mathcal{L}_r$ that is a cycle on $\beta(r)$. If this is the case, we accept, otherwise we reject. (This is correct because cycles may have length 1 or 2.)

□

6. Monadic Second Order Logic

6.1. Monadic second order logic

In this chapter we introduce monadic second order logic. Monadic second order logic extends first order logic (predicate logic) by the possibility to quantify over subsets of the universe. The success of this logic is due to the fact that it is strong enough to express numerous important algorithmic problems, and it still has good algorithmic properties.

Relational structures

Definition 6.1 A (relational) signature σ is a finite set of so-called relation symbols. Every relation symbol $R \in \sigma$ has an arity $\text{ar}(R) \in \mathbb{N}$, $\text{ar}(R) \geq 1$. A σ -structure $\mathcal{A} := (A, R_1^{\mathcal{A}}, \dots, R_k^{\mathcal{A}})$ consists of a set A , the universe of \mathcal{A} , and a relation $R^{\mathcal{A}} \subseteq A^{\text{ar}(R)}$ for all $R \in \sigma$. We write $\bar{a} \in R^{\mathcal{A}}$ or $R^{\mathcal{A}}\bar{a}$, if the tuple $\bar{a} \in A^{\text{ar}(R)}$ is in the relation $R^{\mathcal{A}}$.

Recall that a directed graph $G = (V(G), E(G))$ consists of a finite vertex set $V(G)$ and a set $E(G) \subseteq V(G) \times V(G)$ of directed edges.

Example 6.2 Let σ_{graph} be the signature that consists of a binary relation symbol E . σ_{graph}

- Every directed graph (V, E) can be seen as a σ_{graph} -structure $\mathcal{G} = (V, E^{\mathcal{G}})$.
- Every (undirected) graph (V, E) can be seen as a σ_{graph} -structure $\mathcal{G} = (V, E^{\mathcal{G}})$, where $E^{\mathcal{G}}$ is symmetric and irreflexive.

Definition 6.3 A hypergraph $H = (V(H), E(H))$ consists of a finite set $V(H)$ of vertices and a set $E(H) \subseteq 2^{V(H)}$ of subsets of $V(H)$, called the hyperedges of H . hypergraph
hyperedge

In particular, every undirected graph is a hypergraph (where all hyperedges have two elements).

Example 6.4 Let σ_{HG} be the signature, that consists of two unary relation symbols VERT and EDGE and a binary relation symbol I . Every hypergraph $\mathcal{H} = (V, E)$ can be seen as a σ_{HG} -structure $\mathcal{H} = (H, \text{VERT}^{\mathcal{H}}, \text{EDGE}^{\mathcal{H}}, I^{\mathcal{H}})$, where:

- $H := V \cup E$,

- $VERT^{\mathcal{H}} := V$, $EDGE^{\mathcal{H}} := E$ und
- $I^{\mathcal{H}} := \{(v, e) \mid v \in V, e \in E \text{ and } v \in e\}$ (the incidence relation).

Since every graph can be seen as a hypergraph, we now have a second way of representing graphs as relational structures. We call this the *hypergraph representation* of a graph.

We will consider relational structures. For algorithms that receive relational structures as input, we define the *size* of a relational structure in a way that captures the length of a (reasonable) encoding of the structure as a string.

Definition 6.5 *The size of a σ -structure \mathcal{A} is*

$$\|\mathcal{A}\| := |\sigma| + |A| + \sum_{R \in \sigma} |R^{\mathcal{A}}| \cdot ar(R).$$

Monadic Second Order Logic Let var_{11} and var_{12} be countably infinite sets of *variables* with $var_{11} \cap var_{12} = \emptyset$. The elements of var_{11} are called *individual variables* (or *first-order variables*) and we usually denote them by lower case letters x, y, z, x_1, x_2, \dots . The elements of var_{12} are called *set variables* (or *second-order variables*) and we usually denote them by capital letters X, Y, Z, X_1, X_2, \dots .

Let σ be a signature. The set of all σ -formulas of monadic second order logic, $MSO[\sigma]$, is inductively defined as follows.

(1) The rules for first-order logic:

- (a) $x = y \in MSO[\sigma]$ for all $x, y \in var_1$,
- (b) $Rx_1 \dots x_k \in MSO[\sigma]$ for all $R \in \sigma$ with $ar(R) = k$ and $x_1, \dots, x_k \in var_1$,
- (c) $Yx \in MSO[\sigma]$ for all $x \in var_1$ and $Y \in var_2$
- (d) If $\varphi, \psi \in MSO[\sigma]$, then also $(\varphi \vee \psi), (\varphi \wedge \psi), \neg\varphi \in MSO[\sigma]$,
- (e) If $\varphi \in MSO[\sigma]$, then also $\exists x\varphi, \forall x\varphi \in MSO[\sigma]$, for all $x \in var_1$.

(2) The rules for set quantifiers:

- (a) If $\varphi \in MSO[\sigma]$, then also $\exists X\varphi \in MSO[\sigma]$ and $\forall X\varphi \in MSO[\sigma]$ for all $X \in var_2$.

Formulas of type (1a), (1b) and (1c) are also called *atomic σ -formulas*. The set of all formulas of monadic second-order logic is defined as $MSO := \bigcup_{\sigma \text{ signature}} MSO[\sigma]$. We also write $\varphi \rightarrow \psi$ for $\neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi$ for $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.

The set $free(\varphi)$ of *free variables* of $\varphi \in MSO$ is defined as usual, where

- $free(\exists X\psi) = free(\forall X\psi) := free(\psi) \setminus \{X\}$.

Intuitively, $\text{free}(\varphi)$ is the set of those variables of φ , that are not within the scope of a quantifier. Note that $\text{free}(\varphi)$ can contain both individual variables and set variables. We sometimes use the notation $\varphi(Y_1, \dots, Y_l, x_1, \dots, x_k)$, if

$$\{Y_1, \dots, Y_l, x_1, \dots, x_k\} \subseteq \text{free}(\varphi).$$

A formula $\varphi \in \text{MSO}[\sigma]$ without free variables is called a *sentence*.

sentence

The individual variables are interpreted by elements of the universe, while the set variables are interpreted by subsets of the universe. More formally, the semantics of MSO is defined like the semantics of first-order logic with the following extension: Yx means: „the element that interprets x is contained in the set that interprets Y “, and $\varphi := \exists Y \psi$ means: „there exists a subset of the universe such that ψ is satisfied if Y is interpreted by this subset“ (after all free variables in $\text{free}(\varphi)$ have already been interpreted). Analogously, $\varphi := \forall Y \psi$ means: „if Y is interpreted by an arbitrary subset, then ψ is satisfied“ (after all free variables in $\text{free}(\varphi)$ have already been interpreted).

In the sequel we will primarily work with $\text{MSO}[\sigma]$ -sentences, but especially in inductive arguments we will also have to deal with free variables.

Example 6.6 Let $\varphi_{vc}(X)$ be the following σ_{graph} -formula:

$$\varphi_{vc}(X) := \forall y \forall z (Eyz \rightarrow Xy \vee Xz).$$

Then any graph G and any subset $S \subseteq V(G)$:

$$G \models \varphi_{vc}(S) \iff S \text{ is a vertex cover of } G.$$

Example 6.7 Let $k \in \mathbb{N}$. Obviously, a graph G is k -colourable, if and only if there are subsets V_1, \dots, V_k of $V(G)$, such that every vertex of G is on precisely one of the sets V_1, \dots, V_k , and neighbouring vertices are not contained in the same set. This can be expressed in MSO as follows. The following σ_{graph} -sentence expresses that a graph is k -colourable.

$$\text{col}_k := \exists X_1 \dots \exists X_k \forall y \forall z \left(\bigvee_{1 \leq i \leq k} X_i y \wedge \bigwedge_{1 \leq i < j \leq k} \neg(X_i y \wedge X_j y) \wedge \bigwedge_{1 \leq i \leq k} (Eyz \rightarrow \neg(X_i y \wedge X_i z)) \right)$$

Then all graphs G satisfy: $G \models \text{col}_k \iff G$ is k -colourable.

Example 6.8 Let φ_{conn} be the following σ_{graph} -sentence:

$$\varphi_{\text{conn}} := \exists x (x = x) \wedge \forall X ((\exists y Xy \wedge \forall y \forall z (Xy \wedge Eyz \rightarrow Xz)) \rightarrow \forall y Xy).$$

Then all graphs G satisfy: $G \models \varphi_{\text{conn}} \iff G$ is connected.

„Connectivity“ cannot be expressed in first-order logic. Hence MSO is strictly more expressive than FO. (For a detailed introduction into this subject, we warmly recommend the lecture *Logik in der Informatik*).

Tree decompositions of structures The notion of tree decomposition can be extended to relational structures.

tree decomposition of a structure

Definition 6.9 A tree decomposition of a σ -structure \mathcal{A} is a pair (T, β) , where T is a tree and $\beta(t) \subseteq A$ for all $t \in V(T)$, such that

- (1) for every $a \in A$, the set $B^{-1}(a) := \{t \in V(T) \mid a \in \beta(t)\}$ is non-empty and connected in T and
- (2) for every $R \in \sigma$ and all $\bar{a} := a_1 \dots a_{ar(R)} \in R^{\mathcal{A}}$ there is a $t \in V(T)$ with $\{a_1, \dots, a_{ar(R)}\} \subseteq \beta(t)$.

tree width of a structure

The width $w(T, \beta)$ is defined as $\max\{|\beta(t)| \mid t \in V(T)\} - 1$ and the tree width $\text{tw}(\mathcal{A})$ is the minimal width of a tree decomposition of \mathcal{A} .

Figure 6.1 shows the structure $\mathcal{A} := (\{1, \dots, 9\}, E^{\mathcal{A}}, P^{\mathcal{A}}, R^{\mathcal{A}})$, where $E^{\mathcal{A}}$ is represented by directed edges, $P^{\mathcal{A}} := \{1, 5\}$ and $R^{\mathcal{A}} := \{(i, j, l) \mid \{i, j, l\} = \{3, 7, 9\}\}$.

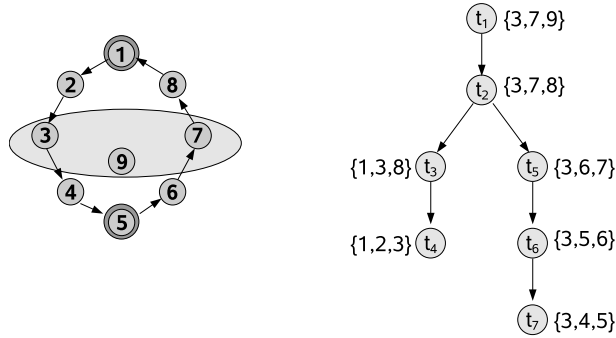


Figure 6.1.: A structure with a corresponding tree decomposition.

We now give an equivalent characterisation of tree width of structures.

underlying graph, Gaifman graph

Definition 6.10 Let σ be a signature and \mathcal{A} a σ -structure. The underlying graph (also called Gaifman graph) of \mathcal{A} is the graph $\mathcal{G}(\mathcal{A})$ with

- $V(\mathcal{G}(\mathcal{A})) := A$ and
- $\{a, b\} \in E(\mathcal{G}(\mathcal{A}))$, if $a \neq b$ and there is a relation symbol $R \in \sigma$ of arity $r := ar(R)$ and a tuple $\bar{a} := a_1, \dots, a_r \in R^{\mathcal{A}}$ with $a = a_i$ and $b = a_j$ for $1 \leq i, j \leq r$.

Hence the underlying graph $\mathcal{G}(\mathcal{A})$ is obtained from \mathcal{A} by inserting an edge between two elements, whenever they occur together in a tuple of some relation in \mathcal{A} . In this way we transfer graph notions like *distance*, *neighbours*, *paths* from the

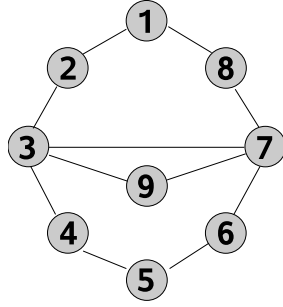


Figure 6.2.: The underlying graph of the structure shown in Figure 6.1.

underlying graph to the structure. E. g. the distance between two elements $a, b \in A$ is their distance in $\mathcal{G}(\mathcal{A})$. Figure 6.2 shows the underlying graph of the structure of Figure 6.1.

The following Lemma is not hard to prove.

Lemma 6.11 *Any structure \mathcal{A} has the same tree decompositions as its underlying graph $\mathcal{G}(\mathcal{A})$. In particular, $\text{tw}(\mathcal{A}) = \text{tw}(\mathcal{G}(\mathcal{A}))$.*

6.2. MSO on trees

In this section we will study the evaluation of MSO-formulas on trees.

The primary objective of the following chapter will be to show the following two important results.

- The problem that, given a graph G and a sentence $\varphi \in \text{MSO}$ as input, asks whether $G \models \varphi$ can be decided in time $2^{\text{tw}G^{O(1)}} \cdot |V(G)|$. This theorem was first proved by Courcelle and implies linear time decision procedures for a large class of computational problems on graphs of bounded tree-width.
- For every sentence $\varphi \in \text{MSO}$ and all $k \in \mathbb{N}$ it is decidable, whether there is a graph of tree width at most k such that $G \models \varphi$. This theorem is due to Seese and will play a special rôle later on.

We will now show the corresponding results for trees, and in the following chapter we generalise them to arbitrary graphs of bounded tree width.

6.2.1. Tree automata

Definition 6.12 *Let Σ be a finite alphabet. An ordered, Σ -labelled binary tree, or simply Σ -tree, is a tuple $T := (V, E_1, E_2, \lambda)$, where*

- $(T, E_1 \cup E_2)$ is a rooted binary tree, where all nodes have at most one E_1 child and at most one E_2 child and
- $\lambda : V \rightarrow \Sigma$ is a labelling function.

We call a child t_1 of a node t with $(t, t_1) \in E_1$ the 1-child of t and a child t_2 with $(t, t_2) \in E_2$ the 2-child of t . Note that there can be nodes with a 2- but no 1-child and vice versa.

tree automaton **Definition 6.13** A (bottom-up) tree automaton is a tuple $\mathcal{A} := (Q, \Sigma, \Delta, F)$, where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\Delta \subseteq (Q \dot{\cup} \{\perp\}) \times (Q \dot{\cup} \{\perp\}) \times \Sigma \times Q$, with $\perp \notin Q$, is the transition relation and
- $F \subseteq Q$ is the set of final states.

deterministic tree automaton \mathcal{A} is deterministic, if Δ is (the graph of) a function from $(Q \dot{\cup} \{\perp\}) \times (Q \dot{\cup} \{\perp\}) \times \Sigma$ to Q .

run **Definition 6.14** A run ρ of a tree automaton $\mathcal{A} := (Q, \Sigma, \Delta, F)$ on a Σ -tree $T := (V, E_1, E_2, \lambda)$ is a function $\rho : V \rightarrow Q$, such that all $t \in V$ satisfy the following.

- If t is a leaf, then $(\perp, \perp, \lambda(t), \rho(t)) \in \Delta$.
- If t is a node with only a 1-child t' then $(\rho(t'), \perp, \lambda(t), \rho(t)) \in \Delta$.
- If t is a node with only a 2-child t' then $(\perp, \rho(t'), \lambda(t), \rho(t)) \in \Delta$.
- If t has a 1-child t_1 and a 2-child t_2 then $(\rho(t_1), \rho(t_2), \lambda(t), \rho(t)) \in \Delta$.

accepting run The run ρ is accepting, if $\rho(r) \in F$ for the root r of T .

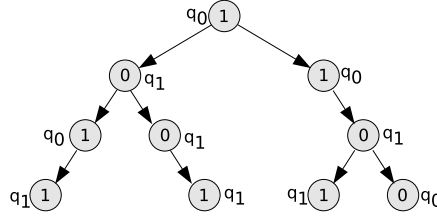
A tree automaton \mathcal{A} accepts a Σ -tree \mathcal{T} , if there is an accepting run of \mathcal{A} on \mathcal{T} .

accepted language The language accepted by \mathcal{A} , denoted by $\mathcal{L}(\mathcal{A})$, is the class of all Σ -trees accepted by \mathcal{A} .

Example 6.15 Let $\mathcal{A} := (Q, \Sigma, \Delta, F)$ be a tree automaton with $Q := \{q_0, q_1\}$, $\Sigma := \{0, 1\}$, $F := \{q_0\}$ and

$$\begin{aligned} \Delta := & \cup \{(\perp, \perp, i, q_i) : i \in \{0, 1\}\} \\ & \cup \{(q_i, \perp, j, q_k), (\perp, q_i, j, q_k) : i + j \equiv k \pmod{2}\} \\ & \cup \{(q_i, q_l, j, q_k) : i + l + j \equiv k \pmod{2}\} \end{aligned}$$

The language accepted by \mathcal{A} consists of all $\{0, 1\}$ -trees which contain an even number of 1s. The following figure shows an accepting run of \mathcal{A} on a tree.



Definition 6.16 Let Σ be a finite alphabet. A (tree) language is a class of Σ -trees. A language \mathcal{L} is regular, if $\mathcal{L} = \mathcal{L}(\mathcal{A})$ for a tree automaton \mathcal{A} .

tree language

regular tree language

We denote the class of all Σ -trees by \mathcal{T}_Σ .

We state the following closure properties of tree languages without proof.

Theorem 6.17 The class of regular tree languages over an alphabet Σ is closed under union, intersection, complementation and projection. Furthermore every regular tree language can be accepted by a deterministic tree automaton.

More precisely:

- (1) If $\mathcal{L}_1, \mathcal{L}_2$ are regular Σ -tree languages, then so are $\mathcal{L}_1 \cup \mathcal{L}_2$ and $\mathcal{L}_1 \cap \mathcal{L}_2$.
- (2) If \mathcal{L} is a regular Σ -tree language, then so is $\mathcal{T}_\Sigma \setminus \mathcal{L}$.
- (3) If $\mathcal{L} \subseteq \mathcal{T}_{\Sigma_1 \times \Sigma_2}$ is a regular tree language over the alphabet $\Sigma_1 \times \Sigma_2$, then the language

$$\left\{ (T, E_1, E_2, \lambda) \in \mathcal{T}_{\Sigma_1} : \begin{array}{l} \text{there is a } \Sigma_1 \times \Sigma_2\text{-tree } T' := (T, E_1, E_2, \lambda') \in \mathcal{L}, \\ \text{such that all } t \in V \text{ satisfy:} \\ \text{If } \lambda(t) = a, \text{ then there is a } b \in \Sigma_2 \text{ with } \lambda'(t) = (a, b). \end{array} \right\}$$

is regular.

- (4) For every non-deterministic tree automaton \mathcal{A} there is a deterministic tree automaton \mathcal{B} accepting the same language.

Furthermore, all transformations are effective, i.e. the corresponding automaton can be computed given automata for the original language. For intersection, union and projection, the size of the resulting automata is polynomial in the size of the original automata, for complementation and determinisation it is exponential.

6.2.2. MSO on Trees

Let Σ be a finite alphabet. A Σ -tree $\mathcal{T} := (V, E_1, E_2, \lambda)$ can be encoded by a structure $\mathcal{A}_\mathcal{T} := (V, E_1, E_2, (P_a)_{a \in \Sigma})$ over the signature $\sigma(\Sigma) := \{E_1, E_2, P_a \mid a \in \Sigma\}$ in the obvious way, where $P_a := \{t \in V \mid \lambda(t) = a\}$. In this way Σ -tree-languages can be defined by logical formulas. We will usually not distinguish between a Σ -tree \mathcal{T} and the corresponding structure $\mathcal{A}_\mathcal{T}$ and will simply write $\mathcal{T} \models \varphi$ for an MSO-sentence φ .

Definition 6.18 Let Σ be a finite alphabet and let $\sigma := \sigma(\Sigma)$. Let $\varphi \in \text{MSO}[\sigma]$ be a sentence. The tree language $\mathcal{L}(\varphi)$ defined by φ is defined as

$$\mathcal{L}(\varphi) := \{\mathcal{T} \mid \mathcal{T} \text{ } \Sigma\text{-tree and } \mathcal{T} \models \varphi\}.$$

Example 6.19 (1) The following MSO-sentence defines the class of all $\{0, 1\}$ -trees such that every node labelled by a 1 has a successor labelled by 0.

$$\forall x(P_1x \rightarrow \exists y[(E_1xy \vee E_2xy) \wedge P_0y]).$$

(2) The sentence in Example (1) even is a first-order formula. The next formula defines the class of all $\{0, 1\}$ -trees such that for every node t labelled by a 1 there is a node s in the subtree rooted at t which is labelled by 0.

$$\forall x \left(P_1x \rightarrow \forall Z \left((Zx \wedge \forall y \forall z ((Zy \wedge (E_1yz \vee E_2yz)) \rightarrow Zz)) \rightarrow \exists y (Zy \wedge P_0y) \right) \right)$$

The formula states that whenever x is interpreted by a node labelled by 1, then every set Z which contains x and with every node also its successors must contain a node labelled by a 0. In particular, the subtree rooted at x must contain a 0 node.

The next theorem says that the MSO-definable tree languages are exactly the regular tree languages. We do not prove it here.

Theorem 6.20 A Σ -tree language is regular if, and only if, it is MSO-definable. Furthermore, for every given MSO-formula we can effectively construct an equivalent tree automaton and vice versa.

Corollary 6.21 Let Σ be a finite alphabet and $\sigma := \sigma(\Sigma)$.

The MSO-Model-Checking Problem on Σ -trees

$\text{MC}(\text{MSO})$ <i>Input:</i> Σ -tree \mathcal{T} , $\varphi \in \text{MSO}[\sigma]$. <i>Problem:</i> $G \models \varphi?$
--

is decidable in time $f(|\varphi|) \cdot \mathcal{O}(\|\mathcal{T}\|)$, for a computable function f .

Proof. First we transform φ into an equivalent deterministic Σ -tree automaton according to Theorem 6.20. Then we run the automaton on the input \mathcal{T} and we check whether it accepts \mathcal{T} .

The first step is effective, hence it is computable in time $f(|\varphi|)$ for some computable function. It is easy to see that one can check in linear time, whether an automaton accepts a tree. \square

From Corollary 6.21 we immediately obtain the following.

Corollary 6.22 *Let Σ be a finite alphabet and $\sigma := \sigma(\Sigma)$. For every MSO[σ]-formula φ , the problem*

MC(φ)
Input: Σ -tree \mathcal{T} .
Problem: Does $\mathcal{T} \models \varphi$?

is decidable in linear time.

For example, Corollary 6.22 immediately implies that for fixed $k \in \mathbb{N}$, there is a linear time algorithm that decides whether a tree has an independent set of size k . Similar statements follow for other graph problems.

Lemma 6.23 *Let Σ be a finite alphabet. There is a polynomial time algorithm which, given an automaton $\mathcal{A} := (Q, \Sigma, \Delta, F)$ as input, decides whether $\mathcal{L}(\mathcal{A}) = \emptyset$.*

Proof. We will present a very simple such algorithm running in time $\mathcal{O}(|Q| \cdot |\Delta|)$. A more clever implementation of the same idea yields an algorithm with running time $\mathcal{O}(|Q|)$.

The algorithm builds up a set $L \subseteq Q$ of states q such that for all $q \in L$ there is a Σ -tree T and a run ρ of \mathcal{A} on T such that $\rho(w) = q$ for the root w of T . The algorithm terminates and outputs “accept” as soon as L contains a final state.

We initialise $L := \{q \in Q : (\perp, \perp, a, q) \in \Delta \text{ for } a \in \Sigma\}$. As long as there is a $q \in Q \setminus L$ and a transition $(q_1, q_2, a, q) \in \Delta$ such that $a \in \Sigma$ and $q_1, q_2 \in L \cup \{\perp\}$, add q to L . If L contains a final state, stop and output $\mathcal{L}(\mathcal{A}) \neq \emptyset$. If no more states can be added to L and L does not contain a final state, then return $\mathcal{L}(\mathcal{A}) = \emptyset$.

We show next that L only contains states for which there is a tree T and a run ρ of \mathcal{A} on T such that $\rho(w) = q$ for the root w of T . This is obvious for the states added to L at the beginning, and follows easily by induction for the other states added later.

Furthermore, it is easily seen that the algorithm runs in polynomial time. \square

Corollary 6.24 *Let Σ be a finite alphabet and let $\sigma := \sigma(\Sigma)$. For every $\varphi \in \text{MSO}[\sigma]$ it is decidable whether there is a Σ -tree T such that $T \models \varphi$.*

Proof. On input φ we first construct an equivalent tree-automaton \mathcal{A}_φ using Theorem 6.20 and then test whether $\mathcal{L}(\mathcal{A}_\varphi) = \emptyset$. \square

7. Courcelle's Theorem

In this chapter we generalise the results proved in the previous chapter from trees to classes of graphs of bounded tree width. The main idea is to reduce the model-checking problem for MSO-formulas on graphs of bounded tree width to the case of trees. Towards this aim, we show that

- (1) tree decompositions can be encoded in labelled trees and
- (2) MSO-formulas on graphs can be rewritten in such a way that they speak about the tree decomposition of a graph instead, or – more precisely – about its encoding in a labelled tree.

7.1. Coding Tree Decompositions in Labelled Trees

In this section we will show how to encode graphs of bounded tree width by labelled trees. Towards this aim, we will first consider a variant of tree decompositions in which the bags are tuples of vertices rather than sets. This can be achieved by ordering the elements of a bag arbitrarily. Furthermore, we will require that the arity of all these tuples is the same and that the decomposition tree is binary.

Definition 7.1 *Let $k \in \mathbb{N}$. An ordered tree decomposition of width k of a graph G is a pair $(T, (\bar{b}_t)_{t \in V(T)})$, where T is a rooted, binary tree and $\bar{b}_t \in V(G)^k$, such that (T, β) , with $\beta(t) := \{b_1, \dots, b_k\}$ for $\bar{b}_t := b_1, \dots, b_k$, is a tree decomposition of G .* *ordered tree decomposition*

It is easily seen that every tree decomposition of width k can be converted into an ordered tree decomposition of G of width k in linear time. Together with Bodlaender's algorithm (Theorem 4.3) this implies the following lemma.

Lemma 7.2 *Let $k \in \mathbb{N}$ and let G be a graph with $\text{tw}(G) \leq k$. An ordered tree decomposition of G of width at most k can be computed in time $2^{\mathcal{O}(k^3)} \cdot |V(G)|$.*

Let G be a graph and $\mathcal{D} := (T, (\bar{b}_t)_{t \in V(T)})$ be an ordered tree decomposition of G of width k . We will encode \mathcal{D} as a labelled tree $\mathcal{T} := \mathcal{T}(G, \mathcal{D})$, such that the graph G – or rather its tree decomposition \mathcal{D} – can be reconstructed from \mathcal{T} .

The underlying tree of $\mathcal{T} := (T, \lambda)$ is simply the tree T of \mathcal{D}^1 . Let $t \in V(T)$ and $\bar{b}_t := b_1, \dots, b_k$. We define

$$\lambda(t) := (\text{eq}(t), \text{overlap}(t), \lambda_E(t))$$

where $\text{eq}(t), \text{overlap}(t), \lambda_E(t)$ are defined as follows:

- $\text{eq}(t) := \{(i, j) \mid i, j \in [k] \text{ and } b_i = b_j\}$.
- If t is the root of T then $\text{overlap}(t) := \emptyset$. Otherwise let s be the parent of t in T and let $\bar{b}_s := a_1, \dots, a_k$. We define

$$\text{overlap}(t) := \{(i, j) \mid i, j \in [k] \text{ and } b_i = a_j\}.$$

- $\lambda_E(t) := \{(i, j) \mid i, j \in [k] \text{ and } b_i b_j \in E(G)\}$.

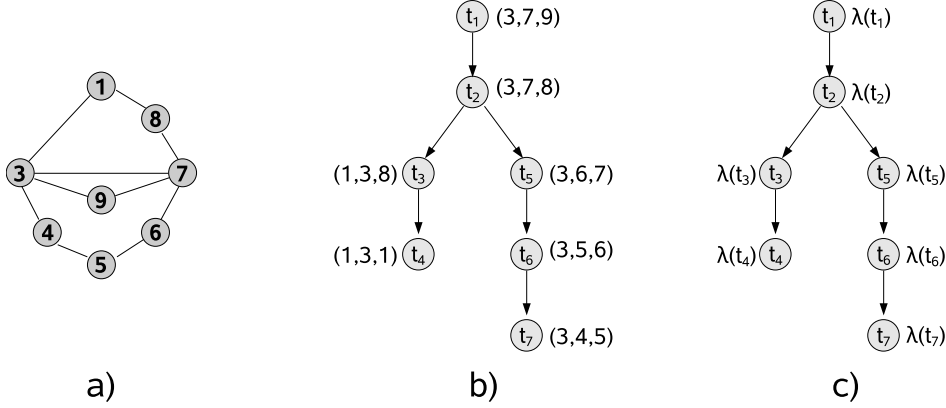
Intuitively, the individual parts of the labelling have the following meaning. The relation $\text{eq}(t)$ encodes which entries of \bar{b}_t refer to the same vertices in G . The relation $\text{overlap}(t)$ encodes the equalities between entries of \bar{b}_t and entries of the tuple of the parent of t . Finally, $\lambda_E(t)$ encodes the edges of the graph. We will see that this information is sufficient to reconstruct the encoded tree decomposition and the original graph G . It is easily seen that the labels of the nodes are letters of the finite alphabet

$$\Sigma_k := 2^{[k]^2} \times 2^{[k]^2} \times 2^{[k]^2}.$$

(Recall that we use the abbreviation $[k] := \{1, \dots, k\}$.) Note that the alphabet Σ_k depends on the arity k of the tuples in the ordered tree decomposition. In the sequel, we will usually drop the index k and assume that it is always the maximal tree width of the graphs we consider. It will always be clear from the context.

Example 7.3 *Let G be the graph displayed in Part a) of the following figure. Part b) gives an ordered tree decomposition \mathcal{D} of G .*

¹Formally, we require that in Σ -trees the children of each node are ordered, i.e. we have distinguished 1- and 2-children. For this, we can simply order the children of every node in T arbitrarily. To simplify notation we will implicitly assume an ordering among the children of each node.



Note that in tuple t_4 , vertex 1 is repeated such that the tuple has arity 3.

Part c) illustrates an encoding of \mathcal{D} in a Σ -tree. The underlying tree is the same, only the labelling changes. For instance, the labelling $\lambda(t_4)$ is as follows:

$$\lambda(t_4) := (\text{eq}(t_4), \text{overlap}(t_4), \lambda_E(t_4))$$

with

$$\begin{aligned} \text{eq}(t_4) &:= \{(1, 3), (3, 1), (1, 1), (2, 2), (3, 3)\} \\ \text{overlap}(t_4) &:= \{(1, 1), (3, 1), (2, 2)\} \\ \lambda_E(t_4) &:= \{(1, 2), (2, 1), (2, 3), (3, 2)\}. \end{aligned}$$

$\text{eq}(t_4)$ states that the entries 1 and 3 encode the same vertex. $\text{overlap}(t_4)$ states that in position 1 of t_3 we have the same vertex as in position 1 and 3 of t_4 and similarly for position 2 in t_3 and t_4 . Finally, $\lambda_E(t_4)$ encodes the edge relation.

The next step is to rewrite MSO-formulas on graphs so that they speak about Σ -trees instead of graphs. For every $\text{MSO}[\sigma_{\text{graph}}]$ -formula φ we will define an $\text{MSO}[\sigma(\Sigma_k)]$ -formula $\hat{\varphi}$ such that all graphs G with $\text{tw}(G) \leq k$ satisfy: if \mathcal{D} is an ordered tree decomposition of G of width k and $\mathcal{T} := \mathcal{T}(G, \mathcal{D})$ is the corresponding Σ -tree, then

$$G \models \varphi \iff \mathcal{T} \models \hat{\varphi}.$$

Let $\mathcal{D} := (T, (\bar{b}_t)_{t \in V(T)})$ and $\mathcal{T}(G, \mathcal{D}) = (T, \lambda)$, where T is the rooted binary tree and λ is the labelling function. We encode sets $S \subseteq V(G)$ by tuples $\bar{U}(S) := (U_1(S), \dots, U_k(S)) \in (2^{V(T)})^k$, with

$$U_i(S) := \{t \in V(T) \mid b_i \in S \text{ for } \bar{b}_t := b_1 \dots b_k\}.$$

Hence, in $U_i(S)$ we collect all nodes of T such that position i of the tuple \bar{b}_t contains an element of S . Correspondingly, a single vertex $v \in V(G)$ is encoded by $\bar{U}(\{v\})$. We simply write $\bar{U}(v)$ for $\bar{U}(\{v\})$.

Example 7.4 Let us again consider the graph in Example 7.3. We have

$$\overline{U}(7) := (\emptyset, \{t_1, t_2\}, \{t_5\})$$

and

$$\overline{U}(\{3, 5, 6\}) := (\{t_1, t_2, t_5, t_6, t_7\}, \{t_3, t_4, t_5, t_6\}, \{t_6, t_7\}).$$

Clearly, not every tuple (U_1, \dots, U_k) encodes sets $S \subseteq V(G)$ of vertices. The following lemma provides a condition for such tuples to encode sets of vertices.

Lemma 7.5 Let $\overline{U} := (U_1, \dots, U_k)$ be a tuple of sets $U_i \subseteq V(T)$. There is a set $S \subseteq V(G)$ with $\overline{U} = \overline{U}(S)$ if, and only if, for all $s, t \in V(T)$ and all $i, j \in [k]$ the following conditions are satisfied.

- (1) If $(i, j) \in \text{eq}(t)$, then $(t \in U_i \iff t \in U_j)$.
- (2) If t is a child of s and $(i, j) \in \text{overlap}(t)$, then $(t \in U_i \iff s \in U_j)$.
If $S = \{v\}$, then in addition to (1) and (2) the following conditions must hold.
- (3) If $t \in U_i$ and $t \in U_j$, then $(i, j) \in \text{eq}(t)$.
- (4) If t is a child of s , $t \in U_i$ and $s \in U_j$, then $(i, j) \in \text{overlap}(t)$.
- (5) $\bigcup_{i \in [k]} U_i$ is connected and non-empty.

Proof. Clearly, every tuple $\overline{U}(S)$ for a set $S \subseteq V(G)$ satisfies the Conditions (1) and (2) and for single vertex $a \in V(G)$, the set $\overline{U}(a)$ also satisfies (3) – (5).

Conversely, let $\overline{U} = (U_1, \dots, U_k)$ be a tuple satisfying Conditions (1) and (2). Let

$$S := \{v \in V(G) \mid v = b_i \text{ for an } i \in [k], t \in U_i \text{ and } \bar{b}_t := b_1, \dots, b_k\}.$$

We show that $\overline{U}(S) = \overline{U}$. For this we have to show that every $j \in [k]$ satisfies $U_j(S) = U_j$.

" \supseteq ": Let $t \in U_j$. Then by definition of S we have $(\bar{b}_t)_j \in S$, hence $t \in U_j(S)$.

" \subseteq ": Suppose $U_j(S) \not\subseteq U_j$. Then there is a node $s \in U_j(S) \setminus U_j$. By the definition of S there is a node $t \in V(T)$ and an index $i \in [k]$, such that $b := (\bar{b}_t)_i = (\bar{b}_s)_j$ and $t \in U_i$. Since $\beta^{-1}(b)$ is connected in T , we may assume that either $s = t$ holds, or s is a child of t or s the parent of t .

If $s = t$, then $(i, j) \in \text{eq}(t)$, a contradiction to Condition (1). If t is a child of s , then $(i, j) \in \text{overlap}(t)$, a contradiction to Condition (2). The case of s being a child of t is similar.

Now let \overline{U} be a tuple satisfying Conditions (1) – (5). Let S be defined as above. By Condition (5), $S \neq \emptyset$. Suppose that S contains more than one element. Then there exist $i, j \in [k]$ and $s, t \in V(T)$ with $b_t := (\bar{b}_t)_i \neq (\bar{b}_s)_j =: b_s$ and $b_t, b_s \in S$, i.e. $t \in U_i$ and $s \in U_j$. Since $\bigcup_{i=1}^k U_i$ is connected in T , by (5) we can choose i, j, s, t in such a way that that $s = t$ or s is a child of t or t is a child of s . But then (3) (or (4), respectively) implies that $b_t = b_s$, a contradiction to our assumption. \square

The previous lemma allows us to translate MSO-formulas on graphs to equivalent MSO-formulas on encodings of ordered tree decompositions by Σ -trees. Vertices $v \in V(G)$ and sets $S \subseteq V(G)$ will be encoded by tuples \bar{U} satisfying the conditions in the lemma. For this, we first need formulas $elem(\bar{U})$ and $set(\bar{U})$, stating that a tuple $\bar{U} := (U_1, \dots, U_k)$, with $U_i \subseteq V(T)$, encodes a vertex $v \in V(G)$ or a set $S \subseteq V(G)$. I.e., we need formulas such that for all graphs G with an ordered tree decomposition \mathcal{D} :

$$\begin{aligned} \mathcal{T}(G, \mathcal{D}) \models elem(\bar{U}) &\iff \text{there exists } v \in V(G) \text{ with } \bar{U} = \bar{U}(v) \\ \mathcal{T}(G, \mathcal{D}) \models set(\bar{U}) &\iff \text{there exists } S \subseteq V(G) \text{ with } \bar{U} = \bar{U}(S). \end{aligned}$$

For this, we formalise the Conditions (1) – (5) by MSO-formulas. For instance, Condition (1) can be formalised by

$$\varphi_1(X_1, \dots, X_k) := \forall x \bigwedge_{i,j \in [k]} \left(\bigwedge_{\substack{a=(eq, overlap, \lambda_E) \in \Sigma \\ (i,j) \in eq}} P_a x \rightarrow (X_i x \leftrightarrow X_j x) \right)$$

and Condition (4) by

$$\varphi_4(X_1, \dots, X_k) := \forall x \forall y \left(Exy \rightarrow \bigwedge_{i,j \in [k]} \left((X_i y \wedge X_j x) \rightarrow \bigvee_{\substack{a=(eq, overlap, \lambda_E) \in \Sigma \\ (i,j) \in overlap}} P_a y \right) \right).$$

The formulas φ_2, φ_3 expressing Conditions (2) and (3) are similar. Only Condition (5) is more complex, as this is not a local property of the tree but requires connectivity. Let

$$\varphi_{connset}(Z) := \forall Y \forall Z \left(\underbrace{\exists y Y y \wedge \exists z Z z \wedge \forall x (X x \leftrightarrow (Y x \vee Z x))}_{X=Y \cup Z} \wedge \underbrace{\forall x \neg (Y x \wedge Z x)}_{Y \cap Z = \emptyset} \right) \rightarrow \underbrace{\exists y \exists z (Y y \wedge Z z \wedge E y z)}_{\text{ex. edge between } Y \text{ and } Z}$$

The formula $\varphi_{connset}(Z)$ says that X is (non-empty and) connected (cf. Example 6.8).

Using this formula we can express Condition (5) as follows.

$$\varphi_5(X_1, \dots, X_k) := \exists U \left(\underbrace{\forall x (U x \leftrightarrow \bigvee_{1 \leq i \leq k} X_i x)}_{U = \bigcup_{1 \leq i \leq k} X_i} \wedge \underbrace{\varphi_{connset}(U)}_{U \text{ connected}} \right)$$

Now, we can define the formulas set and $elem$ as follows:

$$\begin{aligned} set(\bar{X}) &:= \varphi_1(\bar{X}) \wedge \varphi_2(\bar{X}) \\ elem(\bar{X}) &:= \bigwedge_{i=1}^5 \varphi_i(\bar{X}) \end{aligned}$$

Lemma 7.6 *Let $k \in \mathbb{N}$. For every $\text{MSO}[\sigma_{\text{graph}}]$ -formula $\varphi(X_1, \dots, X_r, y_1, \dots, y_s)$ one can effectively construct an $\text{MSO}[\sigma(\Sigma_k)]$ -formula $\hat{\varphi}(\bar{X}_1, \dots, \bar{X}_r, Y_1, \dots, Y_s)$, such that all graphs G with $\text{tw}(G) \leq k$, all ordered tree decompositions \mathcal{D} of G of width k and all $S_1, \dots, S_r \subseteq V(G), v_1, \dots, v_s \in V(G)$ satisfy*

$$G \models \varphi(S_1, \dots, S_r, v_1, \dots, v_s) \iff \mathcal{T}(G, \mathcal{D}) \models \hat{\varphi}(\bar{U}(S_1), \dots, \bar{U}(S_r), \bar{U}(v_1), \dots, \bar{U}(v_s)).$$

Proof. We translate φ by structural induction. Let $\mathcal{D} := (T, (\bar{b}_t)_{t \in V(T)})$ and $\mathcal{T} := \mathcal{T}(G, \mathcal{D}) = (T, \lambda)$.

- If $\varphi(y_1, y_2) := Ey_1y_2$ then

$$\hat{\varphi}(\bar{Y}_1, \bar{Y}_2) := \exists x \bigvee_{i,j \in [k]} \left(Y_{1,i}x \wedge Y_{2,j}x \wedge \bigvee_{\substack{a=(\text{eq}, \text{overlap}, \lambda_E) \in \Sigma \\ (i,j) \in \lambda_E}} P_ax \right).$$

Note that if $G \models \varphi(v_1, v_2)$, i.e. if there is an edge $v_1v_2 \in E(G)$, then this edge must be contained in a bag $\bar{b}_t := b_1 \dots b_k$ of \mathcal{D} , for some $t \in V(T)$. Hence, $v_1 = b_i, v_2 = b_j$, for some $1 \leq i, j \leq k$. To satisfy $\hat{\varphi}$ we can choose $x = t$. Then $(i, j) \in \lambda_E(t)$ and $t \in U_i$, for $\bar{U} = \bar{U}(v_1)$, and also $t \in U'_j$, for $\bar{U}' = \bar{U}(v_2)$.

- The case of $\varphi(y_1, y_2) := y_1 = y_2$ is analogous.
- If $\varphi(X, y) := Xy$ we let

$$\hat{\varphi}(\bar{X}, \bar{Y}) := \exists x \bigvee_{i \in [k]} (Y_ix \wedge X_ix).$$

- Boolean combinations are translated literally.
- If $\varphi := \exists y\psi$ we define

$$\hat{\varphi} := \exists Y_1 \dots \exists Y_k (\text{elem}(Y_1, \dots, Y_k) \wedge \hat{\psi}).$$

- Finally, if $\varphi := \exists X\psi$ we define

$$\hat{\varphi} := \exists X_1 \dots \exists X_k (\text{set}(X_1, \dots, X_k) \wedge \hat{\psi}).$$

- Universal quantification is translated analogously.

□

7.2. Courcelle's Theorem

We now have all the ingredients to prove the following theorem, due to Bruno Courcelle.

Theorem 7.7 (Courcelle) *The problem*

MC(MSO)
Input: Graph G , $\varphi \in \text{MSO}[\{E\}]$.
Problem: Decide $G \models \varphi$?

can be solved in time $f(|\varphi|, \text{tw}(G)) \cdot |V(G)|$, for a computable function f .

Proof. Let G and φ be given.

- (1) We first compute an ordered tree decomposition \mathcal{D} of G of width $k := \text{tw}(G)$ and based on this the tree $\mathcal{T} := \mathcal{T}(G, \mathcal{D})$.
- (2) We then translate φ to $\hat{\varphi}$ using Lemma 7.6.
- (3) Finally, we decide whether $\mathcal{T} \models \hat{\varphi}$.

The first step can be done in time $2^{\text{tw}(G)^{\mathcal{O}(1)}} \cdot |G|$. It is not hard to see that the translation in step (2) can be computed in time $|\varphi| \cdot 2^{\text{tw}(G)^{\mathcal{O}(1)}}$. By Corollary 6.21, step (3) can be done in time $f(\hat{\varphi}) \cdot \mathcal{O}(|\mathcal{T}|) = f'(\varphi, \text{tw}(G)) \cdot \mathcal{O}(|V(G)|)$ for some computable functions f, f' (computability of f follows from Theorem 6.20). \square

Corollary 7.8 *Let \mathcal{C} be a class of graphs of bounded tree width. For every fixed MSO $[\sigma_{\text{graph}}]$ -sentence φ the problem*

MC(φ)
Input: Graph $G \in \mathcal{C}$.
Problem: $G \models \varphi$?

is decidable in linear time.

Example 7.9 *Let col_k be the sentence of Example 6.7, characterising k -colourable graphs. Corollary 7.8 immediately implies The problem k -COL (for fixed $k \in \mathbb{N}$) is solvable in linear time on graph classes of bounded tree width (cf. Theorem 5.1).*

Corollary 7.10 *The MSO Optimisation Problem on graphs*

OPT(MSO)
Input: graph G , $\varphi(X) \in \text{MSO}[\{E\}]$.
Problem: Determine $\text{opt}\{|S| \mid S \subseteq V(G), G \models \varphi(S)\}$

is solvable in time $f(|\varphi|, \text{tw}(G)) \cdot |V(G)|$, for a computable function f .

Proof. The idea is to use the same methods as in the proof of Courcelle's Theorem (Theorem 7.7). In addition, we use bookkeeping on the size of an optimal solution on the subgraphs (similar to the proof of Theorem 1.12). \square

Example 7.11 The problems MAXIS, MAXCLIQUE (determine the maximal size of a clique), MINCOL (determine the chromatic number), MINVC (determine the minimal size of a vertex cover), MINFVS (determine the minimal size of a feedback vertex set) can be expressed as $\text{MSO}[\{E\}]$ optimisation problems and hence they are solvable in linear time on graph classes of bounded tree width (by Corollaries 7.8 and 7.10).

Modifying Corollary 7.10 slightly, it is easy to prove the following.

Corollary 7.12 *The problem*

Input: Graph G , $\varphi(X) \in \text{MSO}[\{E\}]$,
 $k \in \mathbb{N}$.
Problem: Is there a set $S \subseteq V(G)$ such
that $G \models \varphi(S)$ and $|S| \leq k$?

is solvable in time $f(|\varphi|, \text{tw}(G)) \cdot |V(G)|$, for a computable function f .

Example 7.13 The problems CLIQUE, INDEPENDENTSET, VERTEXCOVER and FVS can be expressed in $\text{MSO}[\{E\}]$ and hence they are solvable in linear time on graph classes of bounded tree width (by Corollary 7.12).

Theorem 7.14 (Courcelle's Theorem for structures) *The MSO Model Checking Problem on relational structures*

MC(MSO)
Input: σ -structure \mathcal{A} , $\varphi \in \text{MSO}$.
Problem: $\mathcal{A} \models \varphi$?

is decidable in time $f(|\varphi|, \text{tw}(\mathcal{A})) \cdot |\mathcal{A}| + \mathcal{O}(\|\mathcal{A}\|)$, for a computable function f .

Proof. Given \mathcal{A} and φ , we first test whether the relation symbols that occur in φ also occur in σ . If not, we reject. Otherwise we construct the reduct \mathcal{A}' , that arises from \mathcal{A} by forgetting all relations of \mathcal{A} that do not occur in φ . These steps can be computed in time $\mathcal{O}(\|\mathcal{A}\|)$. If we require that \mathcal{A} and φ use the same signature, then we can cancel this summand in the running time.

The remaining part of the theorem is proved like Courcelle's Theorem for graphs, with the following modification. Instead of encoding only the edge relation (in λ_E), we now encode all relations of the structure in the labelling of the tree. \square

Analogously, the statements corresponding to Corollaries 7.12 and 7.10 hold for structures.

Corollary 7.15 *The problem HAM is expressible in $\text{MSO}[\{\text{VERT}, \text{EDGE}, I\}]$ and hence it is decidable in linear time on graph classes of bounded tree width.*

Proof. Let \mathcal{C} be a graph class with tree width bounded by k . We encode the graphs $G \in \mathcal{C}$ using the hypergraph representation G_I over the signature $\{\text{VERT}, \text{EDGE}, I\}$, as in Example 6.4. By Lemma 6.11, the graph G_I has the same tree width as G , and there is a $\{\text{VERT}, \text{EDGE}, I\}$ -sentence φ_{HAM} characterising precisely the graphs that contain a hamilton cycle. This allows us to apply Theorem 7.14 and hence HAM is solvable in linear time on graphs of bounded tree width. \square

7.3. Seese's Theorem

Finally, we prove that $\text{MSO}[\{E\}]$ is decidable on graphs of bounded tree width. (Recall that in general, first-order logic – and hence also MSO – is undecidable. The question whether first-order logic is decidable was one of David Hilbert's famous 23 problems. In the 1940s, Alonzo Church and, independently, Alan Turing proved it to be undecidable.)

Theorem 7.16 (Seese) *Let $k \in \mathbb{N}$. The following problem*

Input: Sentence $\varphi \in \text{MSO}[\{E\}]$.
Problem: Is there a graph G with $\text{tw}(G) \leq k$ and $G \models \varphi$?

is decidable.

We already know almost all of the ingredients of the proof. Given φ , we first construct a formula $\hat{\varphi}$ according to Lemma 7.6. Then we test whether $\hat{\varphi}$ holds in a Σ_k -tree, where Σ_k is the corresponding alphabet. This is possible by Corollary 6.24. The only problem is, that not every Σ_k -tree is the encoding of an ordered tree decomposition of a graph. Hence we need an $\text{MSO}[\sigma(\Sigma_k)]$ -sentence that holds in a Σ_k -tree \mathcal{T} if and only if \mathcal{T} encodes a tree decomposition of a graph. Towards this, we first show the following lemma.

Lemma 7.17 *Let $k \in \mathbb{N}$ and $\mathcal{T} = (T, \lambda)$ a Σ_k -tree. There is a graph G and an ordered tree decomposition \mathcal{D} of G of width k with $\mathcal{T} = \mathcal{T}(G, \mathcal{D})$ if and only if \mathcal{T} satisfies the following conditions.*

- (1) *Every $t \in V(T)$ satisfies:*
 - (a) *$\text{eq}(t)$ is an equivalence relation on $[k]$, i. e. all $i, i', j \in [k]$ satisfy: If $(i, j) \in \text{eq}(t)$ and $(i', j) \in \text{eq}(t)$, then $(i, i') \in \text{eq}(t)$ and $(j, i) \in \text{eq}(t)$.*

- (b) $\lambda_E(t)$ is symmetric, i. e. if $(i, j) \in \lambda_E(t)$, then also $(j, i) \in \lambda_E(t)$.
 - (c) $\text{eq}(t)$ is consistent with $\lambda_E(t)$, i. e. if $(i, j) \in \lambda_E(t)$ and $(i, i') \in \text{eq}(t)$, then also $(i', j) \in \lambda_E(t)$.
- (2) All nodes $s, t \in V(T)$, such that t is a child of s , satisfy:
- (a) $\text{eq}(t)$ and $\text{eq}(s)$ are consistent with $\text{overlap}(t)$, i. e. if $(i, j) \in \text{overlap}(t)$ and $(i, i') \in \text{eq}(t)$, then also $(i', j) \in \text{overlap}(t)$. If $(j, j') \in \text{eq}(s)$, then also $(i, j') \in \text{overlap}(t)$.
 - (b) $\text{overlap}(t)$ and $\lambda_E(t)$, as well as $\lambda_E(s)$, are consistent, i. e. if $(i, j), (i', j') \in \text{overlap}(t)$ and $(i, i') \in \lambda_E(t)$, then also $(j, j') \in \lambda_E(s)$. Analogously, if $(j, j') \in \lambda_E(s)$, then also $(i, i') \in \lambda_E(t)$.

Proof. Obviously, for every graph G with an ordered tree decomposition \mathcal{D} of width k , the Σ_k -tree $\mathcal{T}(G, \mathcal{D})$ satisfies the conditions. Conversely, let $\mathcal{T} := (T, \lambda)$ be a Σ_k -tree that satisfies Conditions (1) and (2). We define a graph $G(\mathcal{T})$ as follows.

For all $i \in [k]$ and $t \in V(T)$ let $[t]_i \subseteq (V(T) \times [k])$ the smallest set that contains (t, i) and

- with (t, i) also (t, j) , if $(i, j) \in \text{eq}(t)$, and
- with (t, i) also (s, j) , if s is the parent of t and $(i, j) \in \text{overlap}(t)$.
- with (t, i) also (s, j) , if s is a child of t and $(j, i) \in \text{overlap}(s)$.

It is easy to see that Conditions (1) and (2) imply, that the $[t]_i$ are equivalence classes of an equivalence relation on $(V(T) \times [k])$, such that

- all $t \in V(T)$ and $(i, j) \in \text{eq}(t)$ satisfy $[t]_i = [t]_j$.
- if s the parent of t in $V(T)$ and $(i, j) \in \text{overlap}(t)$, then $[t]_i = [s]_j$.
- $\{s : (s, j) \in [t]_i \text{ for a } j \in [k]\}$ is connected in T , for all $t \in V(T)$ and $i \in [k]$.

Let

$$V(G(\mathcal{T})) := \{[t]_i : i \in [k], t \in V(T)\}.$$

We add an edge $\{[t]_i, [s]_j\}$ to $E(G(\mathcal{T}))$, if there exist $t' \in V(T)$ and $i', j' \in [k]$ with $(t', i') \in [t]_i$, $(t', j') \in [s]_j$ and $(i', j') \in \lambda_E(t')$.

Now let $\mathcal{D} := (T, (\bar{b}_t)_{t \in V(T)})$ with $\bar{b}_t := b_1 \dots b_k$ and $b_i := [t]_i$. Then \mathcal{D} is an ordered tree decomposition of $G(\mathcal{T})$, because obviously, every edge is covered, and connectivity follows immediately from the above argument. Moreover, we have $\mathcal{T} = \mathcal{T}(G, \mathcal{D})$. \square

With this we can now prove Seese's Theorem.

Proof of Theorem 7.16. It is easy to see that there is an $\text{MSO}[\sigma(\Sigma_k)]$ -sentence ϑ , that holds in a Σ_k -tree \mathcal{T} if and only if \mathcal{T} satisfies Conditions (1) and (2) of the previous lemma.

On input φ we first construct $\hat{\varphi}$ according to Lemma 7.6. Then we test whether there is a Σ_k -tree \mathcal{T} , such that $\mathcal{T} \models \hat{\varphi} \wedge \vartheta$. By Corollary 6.24, this is decidable. If such a tree \mathcal{T} exists, then \mathcal{T} encodes a graph G (and an ordered tree decomposition of G of width at most k), such that $G \models \varphi$. If no such tree exists, then no graph G with $\text{tw}(G) \leq k$ satisfies φ . \square

A. Graphs: Basic notions and terminology

\mathbb{Z} , \mathbb{N} , and \mathbb{N}^+ denote the sets of integers, nonnegative integers and positive integers, respectively. We sometimes use the abbreviation $[k] := \{1, \dots, k\}$. For a set M we let $2^M := \{X \mid X \subseteq M\}$ denote the *power set* of M . Let M be a set and let k be a non-negative integer. We denote the collection of all k -element subsets of M by $[M]^k$, and we denote the collection of all subsets of M with at most k elements by $[M]^{\leq k}$.

Graphs, Subgraphs and Minors A *graph* $G = (V(G), E(G)) = (V, E)$ consists of a finite set V and a subset $E \subseteq [V]^2$. The elements of V are called *vertices* of G and the elements of E are called *edges* of G . Thus graphs are always finite and simple, where simple means that they have no loops or parallel edges. For brevity we often denote edges by vw instead of $\{v, w\}$. If $e = vw \in E(G)$, then we say that v, w are the *endvertices* of e , and that v and w are *neighbours*. We also say that v and w are *incident* to e , and that v and w are *adjacent*. A graph $G' = (V', E')$ is a *subgraph* of the graph $G = (V, E)$, denoted by $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. If $G' \subseteq G$, we also say that G' is a *supergraph* of G . If $G' \subseteq G$, we also say that G *contains* G' . Let $G = (V, E)$ be a graph and let $X \subseteq V(G)$. The subgraph of G *induced* by X is the graph $G[X]$ with vertex set $V(G[X]) = X$ and edge set $E(G[X]) = \{e \in E(G) \mid e \subseteq X\}$.

A graph G' is an *induced subgraph* of G , if $G' = G[X]$ for some subset $X \subseteq V(G)$. Let $G = (V, E)$ be a graph and let $X \subseteq V(G)$. By $G \setminus X$ we denote the graph $G[V(G) \setminus X]$. For a singleton set $X = \{v\}$ we write $G \setminus v$ instead of $G \setminus \{v\}$. For a set $F \subseteq E(G)$ we let $G - F := (V(G), E(G) \setminus F)$, and for an edge $e \in E(G)$ we let $G - e := G - \{e\}$. Let G be a graph and let $x, y \in V(G)$. We say that x and y are *adjacent*, if $\{x, y\} \in E(G)$. For a vertex $x \in V(G)$ we let $N_G(x) := \{y \in V(G) \mid \{x, y\} \in E(G)\}$ be the set of all neighbours of x (in G). The *degree* of a vertex $v \in V(G)$ is the number $\deg_G(v) := |N(v)|$. Let G, H be graphs. The *union* $G \cup H$ of G and H is defined by $V(G \cup H) := V(G) \cup V(H)$ and $E(G \cup H) = E(G) \cup E(H)$, and the *intersection* $G \cap H$ is defined analogously. If $I = G \cup H$ and $V(G) \cap V(H) = \emptyset$ (and thus $E(G) \cap E(H) = \emptyset$), then we say that I is the *disjoint union* of G and H . Let G be a graph, and $e = \{u, v\} \in E(G)$. The graph G/e is the graph obtained by *contracting* edge e in G , defined as $V(G/e) := (V(G) \setminus e) \cup \{a\}$, for some new $a \notin V(G)$ and $E(G/e) := (E(G) \setminus \{\{x, y\} \in E(G) \mid x \in e\}) \cup \{\{x, a\} \mid \{x, u\} \in E(G)\}$.

graph

vertices

edges

neighbour

adjacent

subgraph

supergraph

induced subgraph

degree

union

intersection

disjoint union

minor $E(G)$ or $\{x, v\} \in E(G)\}$. A graph H is a *minor* of G , denoted by $H \preceq G$, if H can be obtained from a subgraph G' of G by a (possibly empty) sequence of edge contractions.

empty graph The *empty graph* (i. e. the graph with no edges and no vertices) is denoted by \emptyset .
complete graph For every finite set V , we let $K[V]$ be the *complete graph* with vertex set V and edge set $[V]^2$. For $n \in \mathbb{N}$, we let $K_n := K[[n]]$ (in particular, $K_0 = \emptyset$), and for $n, m \in \mathbb{N}^+$
complete bipartite graph we let $K_{m,n}$ be the *complete bipartite graph* with vertex set $[m+n]$ and edge set
clique, k-clique $\{ij \mid i \in [m], j \in [m+1, n]\}$. A *clique* in a graph G is a set $W \subseteq V(G)$ such that $G[W]$ is a complete graph. A *k-clique* in G is a clique W in G of size $|W| = k$. An
independent set *independent set* in G is a set $W \subseteq V(G)$ such that $E(G[W]) = \emptyset$.

We usually do not distinguish between isomorphic graphs.

path **Paths and Connectivity** A *path* is a graph $P = (V, E)$, such that $V = \{x_0, \dots, x_n\}$ is a set of pairwise distinct vertices and $E := \{\{x_0, x_1\}, \dots, \{x_{n-1}, x_n\}\}$. We often simply write $P = x_0, \dots, x_n$ and we say that P is a path *from* x_0 *to* x_n , or that P *connects* x_0 and x_n . Let G be a graph. A path *in* G is a subgraph of G that is a path. The *length* of a path P is the number of the edges of P . The *distance* of two nodes x and y in G , denoted by $\text{dist}_G(x, y)$, is the length of a shortest path in G from x to y . If x and y are not connected by a path, so we let $d_G(x, y) := \infty$. A subset $X \subseteq V(G)$ is *connected*, if $X \neq \emptyset$ and any two vertices in X are connected by a path with vertices in X . A subset $X \subseteq V(G)$ is a *connected component*, if X is connected and \subseteq -maximal with this property. G is *connected*, if $V(G)$ is connected.

cycle **Cycles, Forests and Trees** A *cycle* is a graph $C = (V, E)$, such that $V = \{x_0, \dots, x_n\}$ is a set of at least three pairwise distinct vertices and

$$E = \{\{x_0, x_1\}, \dots, \{x_{n-1}, x_n\}, \{x_n, x_0\}\}.$$

length of a cycle The *length* of a cycle C is the number of edges of C . A graph that does not contain
forest a cycle (as a subgraph) is called *forest*. A vertex of a forest is also called *node*. A
tree *tree* is a connected forest. Let W be a forest. A node $x \in V(W)$ is called a *leaf*, if x has at most one neighbor in W .

rooted tree **Rooted trees** A *rooted tree* is a tree T with a distinguished node $r \in V(T)$, called the *root* of T . Let T be a rooted tree with root r , and let $t, s \in V(T)$. The node s is
child, parent a *child* of t (and t the *parent* of s), if t is the (unique) neighbor of s on the (unique) path from s to r . Let T be a rooted tree with root r , and let $t, s \in V(T)$. The node
predecessor, successor s is a *predecessor* of t (and t a *successor* of s), if t is on the (unique) path from s to r . A *branch* of a rooted tree is a path from the root to a leaf. Let T be a rooted
branch tree with root r , and let $s \in V(T)$. By T_s we denote the subtree of T rooted at s that is induced by all nodes of T whose path to the root contains s .

A.0.1. Directed Graphs

A *directed graph*, or *digraph*, is a pair $D = (V(D), E(D))$, where $V(D)$ is the finite *vertex set* and $E(D) \subseteq V(D)^2$ is the set of (*directed*) *edges*. Note that we allow digraphs to have loops. Edges of a digraph are ordered pairs of vertices, which we usually denote by vw instead of (v, w) . For an edge $e = vw$ we call w the *head* and v the *tail* of e . We use standard graph theoretic terminology for digraphs, without going through it in detail. To avoid the awkward term “subdigraph”, *subgraphs* of a digraph are understood to be digraphs as well. Paths and cycles in a digraph are always meant to be directed; otherwise we call them “paths or cycles of the underlying undirected graph”. We sometimes turn an undirected graph G into a directed graph D with $V(D) = V(G)$ by *orienting* the edges of G in some way, i.e. for an edge $\{u, v\} \in E(G)$ we choose an ordering (either u, v or v, u) of u and v and we add the corresponding directed edge (either (u, v) or (v, u) – but not both) to $E(D)$.

directed graph, digraph

orienting edges

Bibliography

- [1] Hans L. Bodlaender. Treewidth: Structure and algorithms. In Giuseppe Prencipe and Shmuel Zaks, editors, *SIROCCO*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer, 2007.
- [2] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshantov, and Michal Pilipczuk. An $O(c^k n)$ 5-approximation algorithm for treewidth. In *FOCS*, pages 499–508. IEEE Computer Society, 2013.
- [3] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- [4] J. Flum. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- [5] Neil Robertson and Paul D. Seymour. Graph minors. II algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
- [6] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58:22–33, 1993.