Fixed-parameter tractability of the graph isomorphism and canonization problems

Frank Fuhlbrück

September 25, 2013

Fixed-parameter tractability of the graph isomorphism and canonization problems

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von: Frank Fuhlbrück
geboren am: 16.08.1986
in: Berlin

Gutachter: Prof. Dr. Johannes Köbler
Dr. habil. Oleg Verbitsky

eingereicht am: 25.9.2013 verteidigt am: 10.10.2013

Preface

With this thesis, I would like to take you on journey to the place where the graph isomorphism problem and parametrized complexity meet.

The ***graph isomorphism problem*** asks whether there is a bijection between the vertex sets of two graphs that maps edges to edges and non-edges to non-edges. Its importance for complexity theory arises from the fact that it is neither known to be in P nor known to be NP-complete. As the polynomial time hierarchy collapses if the graph isomorphism problem is NP-complete [Sch88], the latter option is regarded as unlikely.

As a close relative, the ***graph canonization problem*** asks for the computation of a canonical form, that is, a mapping that assigns to every graph an isomorphic graph such that any two isomorphic graphs are mapped to the same graph. Obviously, any canonical form can be used to decide the graph isomorphism problem and thus the isomorphism problem is polynomial time Turing reducible to the canonization problem. Whether a reduction in the converse direction is also possible, is an open question. Notwithstanding this, many practical graph isomorphism algorithms for restricted classes easily generalize to graph canonization algorithms.

Parameterized complexity theory is, among other things, a helpful tool to differentiate between levels of computational complexity within NP. It does so by basing the analysis of complexity not only on the size of the input as a string, but also on some other parameter like the number of variables in a boolean formula. The notion of tractability is represented by the class of ***fixed-parameter tractable*** problems, that is, problems for which there is an algorithm with polynomial runtime whose degree does not depend on the parameter.

Since graphs permit a great spectrum of possible parameterizations (degree distribution, size of substructures …) and as the complexity of the graph isomorphism problem is unsettled, parameterized complexity may help to describe hard and easy instances of the graph isomorphism problem. So it is natural to ask for which parameters the graph isomorphism problem is fixed-parameter tractable. Furthermore, we may ask the same for the graph canonization problem and if it turns out to be fixed-parameter tractable for some parameter, it is interesting to look at the asymptotic runtime differences between graph isomorphism and graph canonization (if there are any).

This thesis aims to provide a survey of the graph isomorphism and canonization problems in the context of fixed-parameter tractability and to describe most known results in detail. It is divided into three parts. The first part briefly introduces parameterized complexity with a strong focus on fixed-parameter tractability and defines the graph isomorphism and canonization problems. It further outlines how different flavors of both problems relate if we parameterize them by the same parameter and discusses some basic algorithms to which we will refer in the later chapters. The second part exhibits four (classes of) parameters for which graph isomorphism is known to be fixed-parameter tractable. Moreover, it investigates the usage of modular decompositions in combination with bounded parameters. Eventually, the

third part knots the loose ends together by assembling a comparison graph between various parameters which occurred in the earlier chapters and by listing additional results.

Acknowledgements

I am grateful to my wife Frederike for her inspiration, support, patience and some valuable hours of sleep as well as to our daughter Freya for the happiest year of my life so far. Furthermore, I thank my advisor Prof. Dr. Johannes Köbler and Sebastian Kuhnert for the helpful comments, especially those to my talks in this summer, Sebastian for annotating my Studienarbeit and Johannes for the remarks in and after the evaluation. Many thanks to Dr. Oleg Verbitsky, who gave helpful hints and further directions in his evaluation and spotted some additional errors and unclear passages (see second list on the next page, most of the changes are due to his remarks). Finally, I am thankful to Stephan Verbücheln for proofreading this thesis.

Changes with respect to my Studienarbeit

The chapters 0, 1, 2, 3 (there section 3.1), 4 (there section 3.2) and 9 (there chapter 4) were already part of my Studienarbeit. There are, however, several amendments and some changes as listed in the following table:

Chapter	Changes
0	only minor corrections
1	definitions of parameterized problems, FPT and fpt-reductions expanded to include sets of functions/relations (for parametrized Canonical Labeling and Graph Canonization), treewidth, figure 1.1 is new, definition of XL and para-L
2	section 2.3 is new, sections 2.1 and 2.4 (was 2.3) were expanded (corrections in 2.4.2)
3	section 3.4 is new, section 3.5 (former 3.4) was changed from Graph Isomorphism to Canonical Labeling, small changes in the introduction and some notation changes
4	only minor corrections
9	distance widths are now covered in detail and are thus removed from the overview

Changes *after* evaluation

- some notation fixes (product of permutation groups)
- fixed index error in the definition of the weft hierarchy
- added reference to [Mil79] in the beginning of chapter 2
- fixed first reduction step in the proof of lemma 2.20
- explicitly required $d \geq 2$ for everything related to the Weisfeiler-Lehman algorithm (naïve vertex refinement (= 1-dim WL) is not discussed in detail in this thesis)
- clarified the rule of Lagrange's theorem in section 3.1
- added "$\in \mathcal{C}$" in definition 4.1 (graph has to be in the desired class after modification)
- explicit comparison spw vs. pdw in lemma 8.3
- added lemma 8.12 (vc $\preceq$ $'\kappa$) and modified figure accordingly

Contents

IV Appendix

Part I.

Introduction

We bury the roots, …

0. Notation and basic definitions

Sets and numbers: We will use $\mathbb{Z}$ and $\mathbb{N}$ as the sets of integers and nonnegative integers (i.e $0 \in \mathbb{N}$) and $\mathbb{Z}_m$ stands for the quotient ring $\mathbb{Z}/m\mathbb{Z}$. By $[n,m]$ we refer to the discrete interval $\{i \in \mathbb{Z} \mid n \leq i \leq m\}$, while $[n]$ stands for $[1,n]$ and thus $[0] = \emptyset$. For any sets M and N, $\binom{M}{k}$ stands for the set of all k-elementary subsets of M, N^M for the set of functions from M to N, $\wp(M)$ denotes the power set of M and $M \triangle N = (N \setminus M) \cup (M \setminus N)$. We use double braces $\{\!\{\}\!\}$ to denote multisets and $a \in_k M$ to denote the multiplicity of a in a multiset M, i.e. $a \in_k M \Leftrightarrow \mathbf{I}_M(a) = k$ (where $\mathbf{I}_M$ is the generalized indicator function of M). If not stated otherwise $\log n$ means $\lceil \log_2 n \rceil$. For functions and binary relations dom and rng denote the sets of first and second components (i.e. range (image) and domain in the case of a function). Images of sets $A \subset \mathrm{dom}(f)$ under a function f are denoted by $f(A) = \{f(a) \mid a \in A\}$. We use $\mathbb{1}$ to denote the identity function.

Graphs: Unless stated otherwise all graphs in this work are undirected graphs without loops, i.e. a graph G is a pair $G = (V, E), E \subseteq \binom{V}{2}$, $V(G) = V$ and $E(G) = E$. The complement of G will be denoted by $\overline{G}$ and thus $\overline{G} = (V(G), \binom{V(G)}{2} \setminus E(G))$. For any set $V' \subseteq V(G)$, $G[V']$ denotes the induced subgraph on V', i.e. the graph $(V', E(G) \cap \binom{V'}{2})$. $G \setminus V'$ is a shorthand for $G[V(G) \setminus V']$. We use $+/-$ to denote the addition and deletion of vertices and edges: $G - v = G \setminus \{v\}$, $G - e = (V(G), E(G) \setminus \{e\})$, $G + e = (V(G), E(G) \cup \{e\})$, $G \pm e = (V(G), E(G) \triangle \{e\})$ for $v \in V(G), e \in \binom{V(G)}{2}$. Since we will only consider graphs with $\forall u, v \in V(G) : v \notin u$, this notion is well-defined. Let $\mathcal{G} = \{G \mid G \text{ is a graph and } \exists k \in \mathbb{N} : V(G) = [k]\}$. We will merely use this set as a domain for functions that map graphs to something. Attributes of vertices and edges are denoted as usual: $N_G(v) = \{u \in V(G), \{u, v\} \in E(G)\}$, $\deg_G(v) = |N_G(v)|$, $\mathrm{d}_G(u, v) = \min\{k \mid \exists v_1, \ldots, v_k : v_1 = v, v_k = u \wedge \forall i \in [1, k-1] : \{v_i, v_{i+1}\} \in E(G)\} - 1$. We drop the index G in most circumstances, when the graph in question is implied by context.

Paths and cycles will be regarded as subgraphs, but we will write "let $v_1, \ldots, v_l$ be a cycle C" or similar and mean $E(C) = \{\{v_i, v_{i+1}\} \mid i \in [l], v_{l+1} = v_1\}$. A ***leaf*** shall be a vertex with degree ***at most*** one.

The term ***coloring*** refers to any function from the vertices to some finite set and thus does not necessarily mean a proper coloring in the sense of the chromatic number. A (vertex) colored graph is thus a pair (G, c) where G is a graph and $c : V(G) \to C$ for some class of colors C. A vertex and edge colored graph is a pair (G, c) such that $c : (V(G) \cup E(G)) \to C$. An injective coloring will be called a ***labeling***. The set of all colored graphs on nonnegative integers is denoted by $\mathcal{G}_c$, i.e. $\mathcal{G}_c = \{(G, c) \mid G \in \mathcal{G}, c : V(G) \to V(G)\}$.

The ***adjacency matrix*** $\mathrm{adj}(G, c)$ of a colored graph $(G, c) \in \mathcal{G}_c$ with $|V(G)| = n$ is the matrix

$(A_{i,j})_{(i,j)\in[n]\times[n]}$ with

$$A_{i,j} = \begin{cases} c(i) & i = j \\ 1 & \{i,j\} \in E(G) \\ 0 & \text{else} \end{cases}.$$

For uncolored graphs G the adjacency matrix is $\mathrm{adj}(G) = \mathrm{adj}(G, c_0)$, where $c_0 : V(G) \to \{0\}$.

Groups: Sym_n denominates the symmetric group on $[n]$, i.e. the set of all bijections $[n] \to [n]$. By $\mathrm{Sym}(M)$ we refer to the symmetric group of an arbitrary set M. If $G_1, \ldots, G_k$ are permutation groups with $G_i \subseteq \mathrm{Sym}(V_i)$ and $V_i \cap V_j = \emptyset$ for $i \neq j$, we treat the product $\bigotimes_{i\in[k]} G_i$ as identical to the group

$$G = (\{f \in \mathrm{Sym}(V) \mid \forall i \in [k] \exists f_i \in G_i : f|_{V_i} = f_i\}, \circ)$$

where $V = \bigcup_{i\in[k]} V_i$. To denote a subgroup of G generated by $g_1, \ldots, g_k \in G$ we write $\langle g_1, \ldots, g_k \rangle$. By $\mathbb{1}$, we mean the identity element (for any kind of group). For a subgroup H of a group G, $|G : H|$ denotes the index of H in G.

Encoding as strings and model of computation: As usually we model classical (i.e. non-parametrized) decision problems as languages $L \subseteq \Sigma^*$ over some finite alphabet Σ, but we do not provide specific encodings for instances, if the encoding is not relevant and e.g. simply write $(G, k) \in L$ for a graph G and an integer k. For graphs we measure runtime with respect to $|V(G)| + |E(G)|$ which roughly corresponds to the size of adjacency lists. Hence by linear time algorithm, we mean one with runtime $\mathcal{O}(|V(G)| + |E(G)|)$, if its input is an encoded graph $w(G)$ with $|w(G)| = |V(G)| + |E(G)|$.

We model algorithms as ***random access machines*** (RAMs). RAMs consist of countably infinite series $(r_i)_{i\in\mathbb{N}}$ of registers and a finite state machine (FSM). Each register holds a non-negative integer of arbitrary size. One of the operations load, store, addition, subtraction or (rounded down) division by 2 together with a register and an address mode (direct/indirect) is assigned to every state of the finite state machine. Direct addressing refers to the content r_i of register i, whereas indirect addressing refers to r_{r_i}. The other operand is always the register r_0, called the accumulator, which will also hold the result of the computation. The transition function of the finite state machine maps each state to two successor states, one for the case $r_0 = 0$ (after the operation) and a second one for every other case. At the beginning of the computation the input word w is written to $r_1 \ldots r_{|w|}$ (via some bijection $s : \Sigma \to [|\Sigma|]$), while every other register is set to 0. After reaching a final state the RAM halts and puts out $w' = s^{-1}(r_1) \ldots s^{-1}(r_k)$ for $k = \min\{i > 0 \mid r_i = 0\} - 1$ (or whether $r_0 \neq 0$ in the case of decision problems). The runtime of a RAM is $t : \mathbb{N} \to \mathbb{N}$ such that $t(n)$ is the maximal number of steps the RAM needs to reach a final state over all inputs $w \in \Sigma^*$ with $|w| = n$.

Theorems and proofs: We use the symbol $\not\square$ in a theorem if we completely omit the proof. The ordinary $\square$ in a theorem means that the proof is either trivially (mostly used for corollaries) or contained in the text (and lemmas) preceding the theorem.

1. Parameterized complexity theory

Traditionally, time (and space etc.) complexity of a problem are measured with respect to the length of its instances encoded as strings over some finite alphabet Σ. For instance the problem of deciding whether a given graph G belongs to some class $\mathcal{C}$ may be seen as the language L which contains all adjacency lists of graphs in $\mathcal{C}$. To classify its complexity we try to find upper and lower bounds on the number of computation steps performed by deterministic or nondeterministic Turing machines, the maximal space needed during the computation by any of the aforementioned or the size and/or depth of different kinds of circuits which decide for a given $w \in \Sigma^*$, whether $w \in L$. Irrespective of the model of computation we choose, we state all these bounds as functions of $|w|$, i.e. functions of $|V(G)|$ and $|E(G)|$ if w is an adjacency list. Reductions are a common tool to prove lower bounds (sometimes conditional to assumptions like $\mathsf{P} \neq \mathsf{NP}$), but the class of functions to consider as valid reductions (e.g. logspace many-one reductions) is again defined by their usage of some resource as a function of the input.

Expressing complexity relative to the size of the input has several advantages, e.g. it is always well-defined and allows for the comparison of problems with very different nature. There are however cases where another ***parameter*** of the input, which is less agnostic about the nature of the problem to be decided, grasps the inherent complexity of the task more accurately. Let us for instance consider the problem Clique, i.e. the language L of (encoded) pairs (G, k) such that the graph G has a clique (complete subgraph) of size at least k. There are several graphs parameters $p(G)$ which limit the size of the maximal clique in G, like the maximum degree or the treewidth. If we consider the language L' of those $(G, k) \in L$ where G has maximum degree m, then $L' \in \mathsf{P}$, because we only need to enumerate and check all subsets of size at most $m + 1$.

Results of this kind have been stated for decades, but parameterized complexity theory as developed and initiated by Downey and Fellows (e.g. [DF95a; DF95b]) goes far beyond this. One of its merits is the analysis of the interaction between parameter and input size, which (among other results) leads to the definition of ***fixed-parameter tractability*** and the W-hierarchy. During the following introduction we will adopt most definitions from [FG06] (with the exception of the W-hierarchy), which in some technical aspects differ to those found in [DF99] and [Nie06]. Note that the bibliographic references of this introductory chapter do not point to the original publication of each result or definition (which would be [DF95a] in most cases), but to the most easily comprehensible parts of the three monographs [FG06; Nie06; DF99].

1.1. Parameterized problems

Parameterized complexity theory treats entities of problems and parameters called ***parameterized problems***. We first give a formal definition and discuss it afterwards.

Definition 1.1. Let Σ be a finite set (the alphabet). A function $\kappa : \Sigma^* \to \mathbb{N}$ is a ***parameter*** over Σ if $\kappa \in \mathsf{FP}$, i.e. it can be computed in polynomial time. A ***parameterized (decision) problem*** over Σ is a pair (L, κ), where $L \subseteq \Sigma^*$ is a language over Σ and κ is a parameter over Σ. Likewise, a ***parameterized function problem*** over Σ is a pair (f, κ), where $f : \Sigma^* \to \Gamma^*$ is a function and κ a parameter over Σ. Instead of a function f may also be a binary relation $f \subseteq \Sigma^* \times \Gamma^*$ or set of such functions/relations. We will call (L, κ) ((f, κ)) the problem L (or f) parameterized by κ. •

We use relations to formalize search problems (f-Vertex Cover,f-Graph Isomorphism) and sets of functions/relations to specify problems like Graph Canonization and Canonical Labeling, which allow multiple solutions for one input but require the outputs for different inputs to be consistent.

A possibly surprising part of definition 1.1 is, that a parameter κ has to be polynomial time computable. This seems to be an obstacle to a discussion of parameters like the feedback vertex number (section 4.2), which are NP-complete [Kar72]. We may however state a very similar problem, if a parameter κ is not known to be in FP.

Definition 1.2. Let Σ be a finite alphabet, $\kappa : \Sigma^* \to \mathbb{N}$, L a language over Σ and f a function over Σ. L (or f resp.) ***mediately parameterized by*** κ shall be the parameterized problem (L', κ') $((f', \kappa'))$, where

$$\kappa' : \Sigma^* \times \mathbb{N} \to \mathbb{N} \text{ such that } \kappa'((w,k)) = k\,,$$

$$L' = \{(w,k) \mid w \in L, \kappa(w) = k\}$$

and f' is the minimal relation $f' \subseteq (\Sigma^* \times \mathbb{N}) \times (\mathrm{rng}(f) \cup \{\mathtt{null}\})$ such that

$$((w,k), f(w)) \in f' \qquad \text{if } \kappa(w) = k \text{ and}$$

$$\{(w,k)\} \times \{f(w), \mathtt{null}\} \subseteq f' \qquad \text{if } \kappa(w) \neq k\,.$$

We extend this definition in an analogous manner to relations and sets of functions/relations. •

This is in fact close to the definition of a parameterized problem in [Nie06]. We will never explicitly use the term ***mediately parameterized*** in the later chapters, but silently switch to this definition, whenever a parameter is not known to be in FP. Using the definition from [FG06] as our main definition allows us not to handle an additional bogus argument k, when we consider e.g. color class size or rooted path distance width.

Mind that a parameter may be any computable function which includes functions we would not intuitively call a parameter, because they do not contain any further information about the input.

Example 1.3. Let w be a word in Σ^*. Then we define $\mathrm{size}(w) = |w|$ (for graphs G we set $\mathrm{size}(G) = |E(G)| + |V(G)|$) and $\mathrm{one}(w) = 1$. e.g.

1.2. Graph parameters

The implicit usage of definition 1.2 allows us to use the most liberal definition of a graph parameter. We will now give some examples of graph parameters.

Definition 1.4. We call any function $\kappa : \mathcal{G} \to \mathbb{N}$ a ***graph parameter*** (and assume it is 0 for words w that do not encode graphs). •

Example 1.5. The maximal degree $\max\deg(G) = \max_{v\in V(G)} \deg_G(v)$, the average degree $2|E(G)|/|V(G)|$ of a graph G, the radius $\mathrm{rad}(G) = \min_{u\in V(G)} \max_{v\in V(G)} \mathrm{d}(u,v)$ and the diameter $\mathrm{diam}(G) = \max_{u,v\in V(G)} \mathrm{d}(u,v)$ are graph parameters. e.g.

Definition 1.6 (tw: [RS86; RS84],stw: [See85])**.** Let G be a graph. A ***tree decomposition*** of G is a triple (B,T,r), where

- T is a tree and $r \in V(T)$ shall be its root,
- $B : V(T) \to \wp(V(G))$ maps vertices of T to so called ***bags*** such that $\bigcup \mathrm{rng}(B) = V(G)$,
- for all $e \in E(G)$ there is a $t \in V(T)$ such that $e \subseteq B(t)$ and
- for all $v \in V(G)$ the graph $T[B^{-1}(\{v\})]$ is connected.

A ***strong tree decomposition*** of G is a triple (B,T,r), where

- T is a tree and $r \in V(T)$,
- $B : V(T) \to \wp(V(G))$ such that $\mathrm{rng}(B)$ is a partition of $V(G)$ and
- for all $e \in E(G)$ $\exists t,u \in V(T)$ such that $e \subseteq B(t) \cup B(u)$ and $t = u$ or $\{t,u\} \in E(T)$.

We will use B_t instead of $B(t)$ in both cases from now on. The ***width*** of a tree decomposition (B,T,r) is $\max\{|B_t| \mid t \in V(T)\} - 1$, whereas the width of a strong tree decomposition (B,T,R) is $\max\{|B_t| \mid t \in V(T)\}$. By the ***treewidth*** $\mathrm{tw}(G)$ and ***strong treewidth*** $\mathrm{stw}(G)$ we mean the minimal width of any tree decomposition of G (or strong tree decomposition of G respectively). •

We remark that the -1 ensures $\mathrm{tw}(T) = \mathrm{stw}(T) = 1$ for a tree T.

Example 1.7. Let $G \in \mathcal{G}$ and $|V(G)| = n$. The ***spectrum*** $\mathrm{spec}(G)$ of G is the spectrum of its adjacency matrix $A = \mathrm{adj}(G)$ together with the eigenvalue multiplicities a multiset, i.e.

$$\lambda \in_k \mathrm{spec}(G) \Leftrightarrow k = \max\{l|\exists p' : p_A(t) = (t-\lambda)^l \cdot p'(t)\} \Leftrightarrow \dim(\{r \in \mathbb{R}^n | Ar = \lambda r\}) = k\,,$$

where p_A is the characteristic polynomial of A. As adjacency matrices are symmetric the geometric and algebraic multiplicities coincide. Two natural graph parameters are the size of the spectrum as a simple set and the (maximal) ***eigenvalue multiplicity*** $\mathrm{em}(G)$ of G, $\mathrm{em}(G) = \max\{k \mid \exists\lambda : \lambda \in_k \mathrm{spec}(G)\}$. e.g.

1.3. FPT and fpt-reductions

Definition 1.8. Let (L,κ) be a parameterized problem over some alphabet Σ. (L,κ) is ***fixed-parameter tractable***, if there is an algorithm that decides for all $w \in \Sigma^*$ whether $w \in L$ in at most $g(\kappa(w)) \cdot p(|w|)$ steps, where g is an arbitrary computable function and p is a fixed polynomial (i.e. not depending on $\kappa(w)$). The class of all such parameterized problems is denoted by FPT. We also use the term ***fixed-parameter tractable*** for parameterized function problems (f,κ) computable in time $g(\kappa(w)) \cdot p(|w|)$ (subject to the same conditions) and denote their class by FFPT. If f is a binary relation, the algorithm may output any w' such that $(w,w') \in f$, if f is a set of functions or relations we require $f \cap \mathsf{FFPT} \neq \emptyset$ to say that f is fixed-parameter tractable. •

Note that the name FFPT is not widely used notation (it is used for counting problems (i.e. $\mathrm{rng}(f) \subseteq \mathbb{N}$) in [CTW08]). A parameter κ can also be seen as a function problem. If κ mediately parameterized by κ itself is fixed-parameter tractable, we simply say $\kappa \in$ FFPT from now on.

Example 1.9. For any decidable language L the parameterized problem (L, one) is in FPT if and only if $L \in P$, because $f(\mathrm{one}(w))$ is a constant for any $w \in \Sigma^*$ and computable f. On the other hand, (L, size) is always fixed-parameter tractable, since its runtime is computable and can be hidden as a function of $\mathrm{size}(w)$. e.g.

When it comes to natural parameters and problems, the standard example for a parameterized problem in FPT is Vertex Cover.

Definition 1.10. Let G be a graph and $M \subseteq V(G)$. M is a ***vertex cover*** of G if for all edges $e \in E(G)$ the intersection $e \cap M$ is not empty. By vc we denote the size of the smallest vertex cover in G. M is a ***clique*** of G if $E(G[M]) = \binom{M}{2}$ and an ***independent set*** of G if $E(G[M]) = \emptyset$ (or, equivalently, if $V(G) \setminus M$ is a vertex cover of G). This leads to the definition of the following parameterized problems:

		Vertex Cover
Input	:	Graph G and $k \in \mathbb{N}$
Parameter	:	k
Question	:	Does G have a vertex cover of size at most k?

		f-Vertex Cover
Input	:	Graph G and $k \in \mathbb{N}$
Parameter	:	k
Output	:	A vertex cover of size at most k if one exists

		Independent Set(vc)
Input	:	Graph G and $k \in \mathbb{N}$
Parameter	:	$\lvert V(G)\rvert - k$
Question	:	Is there an independent set of G of size at least k?

		Independent Set
Input	:	Graph G and $k \in \mathbb{N}$
Parameter	:	k
Question	:	Is there an independent set of G of size at least k?

•

Lemma 1.11. *Vertex Cover, f-Vertex Cover and Independent Set(*vc*) are fixed-parameter tractable.*

Proof sketch. A graph has an independent set of size at least k if and only it has a vertex cover of size at most $|V(G)| - k$. Since any edge of a graph agrees on at least one vertex with every vertex cover, we may recursively choose an arbitrary edge, try to remove one of its members a time and repeat the procedure on both resulting graphs until we either found a vertex cover or have unsuccessfully deleted k vertices. The recursion tree of the algorithm has at most 2^k leaves and checking whether a set of vertices is a vertex cover can be done in linear time. □

We only sketched the idea of an algorithm, since we will discuss so called modification sets in section 4.1.3 and a vertex cover is nothing but an edgeless graph deletion set. It is easier

to see the difference between Independent Set(vc) and Independent Set after we discussed fpt-reductions, which we will do now. However, as this work is aimed at a single classical problem and only the parameters change, we single out a constraint of the definition in [FG06] and define it separately inspired by the discussion of the relationship between width parameters in [YBFT99].

Definition 1.12. Let κ and κ' be parameters over Σ and Σ' respectively and $f : \Sigma^* \to \Sigma'^*$. We say κ' ***f-covers*** κ ($\kappa \preceq_f \kappa'$) if there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\forall w \in \Sigma^* : \kappa'(f(w)) \leq g(\kappa(w))$. If $\Sigma = \Sigma'$ and f is the identity on Σ, we omit f. •

It may seem strange that κ and κ' switch sides relative to $\preceq$ and $\leq$ within the above definition. This is, however, consistent with the following definition of fpt-reducibility. We will discuss the purpose of the previous definition afterwards.

Definition 1.13. Let (L, κ) and (L', κ') be parameterized problems over Σ and Σ' respectively. We say (L, κ) is ***fpt (many-one) reducible*** to (L', κ') ($(L, \kappa) \leq^{\text{fpt-m}} (L', \kappa')$) via the ***fpt reduction function*** $f : \Sigma^* \to \Sigma'^*$ if

1. for all $w \in \Sigma^*$: $w \in L \Leftrightarrow f(w) \in L'$,
2. $(f, \kappa) \in$ FPT and
3. $\kappa \preceq_f \kappa'$.

•

The first two parts look similar to the definition of polynomial time many-one reducibility, whereas the third is a crucial property to ensure that FPT is closed under fpt-reductions.

Definition 1.14. Let (f, κ) and (f', κ') be parameterized function problems over Σ and Σ' respectively such that f and f' are functions. We say (f, κ) is ***fpt Turing reducible*** to (f', κ') ($(f, \kappa) \leq^{\text{fpt-T}} (f', \kappa')$) if there is an algorithm that, equipped with an oracle for (f', κ'), computes $f(w)$ for an instance $w \in \Sigma^*$ in time $g(\kappa(w)) \cdot p(|w|)$ (as in the definition of FPT) such that for all queries $h(w)$ made during the computation $\kappa \preceq_h \kappa'$ holds. The definition for relations and languages is analogous. If f and f' are sets of relations or functions, we define

$$(f, \kappa) \leq^{\text{fpt-T}} (f', \kappa') \Leftrightarrow \forall h' \in f' \exists h \in f : (h, \kappa) \leq^{\text{fpt-T}} (h', \kappa').$$

Whenever one of f and f' is a set of objects and the other is not, we replace the single object by the singleton set $\{f\}$ (or $\{f'\}$ respectively). •

We will use the reduction definition for sets when we reduce Graph Isomorphism to Graph Canonization in section 2.3.

Example 1.15. Independent Set(vc) $\leq^{\text{fpt-m}}$ Vertex Cover. Assume that the parameters are named κ and κ' respectively. As in an NP-completeness proof we choose the reduction function f that maps (G, k) to $(G, |V(G)| - k)$. Then f is computable in polynomial time and the first two conditions are fulfilled. We now look at the parameters and see $\kappa(G, k) = k$ and $\kappa'(f(G, k)) = \kappa'(G, |V(G)| - k) = |V(G)| - (|V(G)| - k) = k = \kappa(G, k)$. So κ' can be bounded from above by a function of κ that does not depend on the instance. If we tried to reduce Independent Set with the same f, we would get $\kappa'(f(G, k)) = |V(G)| - \kappa(G, k)$. This violates the third condition. e.g.

Lemma 1.16. FPT *is closed under fpt-reductions, i.e. if* $(A, \kappa) \leq^{\text{fpt-m}} (B, \kappa')$ *and* $(B, \kappa') \in$ FPT *then* $(A, \kappa) \in$ FPT.

Proof. Let $r(\kappa'(w))p(|w|)$ the runtime bound of some algorithm for (B, κ') according to the definition of FPT and f a reduction function for $(A, \kappa) \leq^{\text{fpt-m}} (B, \kappa')$. Further let g be an upper bound for κ' as guaranteed by $\kappa \preceq_f \kappa'$ and (h, p') be the runtime of an FFPT compliant algorithm computing the reduction function f. Then an algorithm for (A, κ) may first compute $f(w)$ for its input w and then use the algorithm for (B, κ'). This takes at most $h(\kappa(w))p'(|w|) + r(g(\kappa(w)))p(|f(w)|)$ computation steps. Since $|f(w)| \leq h(\kappa(w))p'(|w|)$ and $q(ab) \leq q(a)q(b)$ for a polynomial q, we obtain an overall runtime bound that is again polynomial in $|w|$. Furthermore all operations on the runtime are computable as well as all functions involved (r, h and g) and thus $(A, \kappa) \in$ FPT. □

Observe that the polynomial dependence on $|w|$ could be violated, if g could depend on w in another way than via κ. We close the discussion of fpt-reductions with a corollary. As already mentioned, this handles the case when A is a fixed classical problem (e.g. Graph Isomorphism) and only the parameter changes. See chapter 8 for examples of parameters that (do not) cover other parameters.

Corollary 1.17. *If $\kappa \preceq \kappa'$ and $(A, \kappa') \in$ FPT then $(A, \kappa) \in$ FPT.* □

1.4. The weft hierarchy

While P is the standard notion of tractability in the non-parameterized world, the polynomial time hierarchy captures increasing levels of intractability by alternating quantifiers. Similar hierarchies also exists for the parameterized world. The ***weft hierarchy*** (or W***-hierarchy***) classifies problems by their fpt-reducibility to certain problems for boolean formulas with "interwoven" (i.e. alternating) levels of $\bigwedge$ and $\bigvee$.

Definition 1.18. Let $I = (I_{(j,i)})_{j\in[t],i\in[d]}$ be a series of index sets containing integers in $[d]$ and let ψ be a mapping of integers i in $[d]$ to a disjunction of 2 literals $\psi_i = \lambda_1 \vee \lambda_2$ ($\lambda_j = \overline{x}$ (negation) or $\lambda_j = x$ for some variable x). We define the propositional formula $\phi_{I,\psi}$ as

$$\phi_{I,\psi} = \bigwedge_{i_1\in I_{1,1}} \bigvee_{i_2\in I_{1,i_1}} \cdots \bigvee_{i_t\in I_{t-1,i_{t-1}}} \psi_{i_t} \text{ if } t \text{ is even and } \phi_{I,\psi} = \bigwedge_{i_1\in I_{1,1}} \bigvee_{i_2\in I_{1,i_1}} \cdots \bigwedge_{i_t\in I_{t-1,i_{t-1}}} \psi_{i_t} \text{ else .}$$

An ***assignment*** of $\phi_{I,\psi}$ with ***weight*** k is a function from the variables of $\phi_{I,\psi}$ to the set {true, false} such that exactly k variables are set to true. Further, we define the parameterized problem Weighted t-Normalized Satisfiability as:

		Weighted t-Normalized Satisfiability
Input	:	$((I_{(j,i)})_{j\in[t],i\in[d]}, \psi, k)$, where I, ψ are as above, $k \in \mathbb{N}$
Parameter	:	k
Question	:	Is there an assignment of true/false to the variables x_j in $\phi_{I,\psi}$ such that exactly k variables are set to true and $\phi_{I,\psi}$ becomes true?

•

Definition 1.19. Let $t \in \mathbb{N} \setminus \{0\}$. The tth level of the W-hierarchy (W[t]) is

$$\{(L, \kappa) \mid (L, \kappa) \leq^{\text{fpt-m}} \text{Weighted } t\text{-Normalized Satisfiability}\}$$

•

This is the definition used in [Nie06], whereas [FG06] uses a descriptive definition via Fagin definability and the original definition (e.g. in [DF99]) used circuits. Since we focus on algorithmic aspects the most important classes of this hierarchy are $\mathsf{W}[0] = \mathsf{FPT}$ and $\mathsf{W}[1]$, i.e. if a parameterized problem is $\mathsf{W}[1]$-hard, and therefore intractable under current assumptions, we do not care "how intractable" it is. Observe that $\mathsf{W}[1]$ is defined via reducibility to the problem of deciding whether there is an assignment with k variables set to true for a given formula of the form

$$\bigwedge_{i \in [d]} \lambda_{i,1} \vee \lambda_{i,2} \,,$$

where each $\lambda_{i,j}$ is some literal. If we require the literals to be negative, the formula is ***antimonotone***, i.e. switching a variable from true to false cannot change the evaluation of the entire formula from true to false. These formulas directly correspond to a graph whose vertices are variables and whose edges are the sets of variables which appear together in one clause. An assignment of weight k corresponds to an independent set of size k, since we may only choose one variable per clause and one vertex per edge. It turns out that antimonotone formulas suffice to characterize $\mathsf{W}[1]$ (and $\mathsf{W}[t]$ in general, see e.g. [FG06, Theorem 7.29]), i.e. the problem Weighted t-Normalized Antimonotone Satisfiability is $\mathsf{W}[1]$-complete, which in turn also holds for Independent Set.

Theorem 1.20. *Independent Set is $\mathsf{W}[1]$-complete (under fpt-reductions).* ☑

We omit the reduction type from now on, since we only consider fpt-reductions, when dealing with $\mathsf{W}[t]$-completeness.

Corollary 1.21. *The parameterized problem*

		Induced Subgraph Isomorphism
Input	:	$G, H \in \mathcal{G}$
Parameter	:	$\lvert V(G)\rvert + \lvert E(G)\rvert$
Question	:	Is G isomorphic to an induced subgraph of H?

is $\mathsf{W}[1]$-complete.

Proof. The $\mathsf{W}[1]$-hardness is obvious via a reduction from Independent Set (since there are no edges, the parameter is exactly the same). The other direction goes by a reduction to Independent Set. Let (G, H) be a candidate pair for Induced Subgraph Isomorphism, we construct a graph H_G such that $V(H_G) = V(G) \times V(H)$ and any two-element set $\{(u, v), (w, x)\}$ is in $E(H_G)$ if and only if it is ***not*** an isomorphism from $G[\{u, w\}]$ to $H[\{v, x\}]$. Now any independent set S of H_G is an isomorphism from $G[\mathrm{dom}(S)]$ to $H[\mathrm{rng}(S)]$ and any such partial isomorphism is an independent set. We output $(H_G, |V(G)|)$ as an instance for Independent Set and observe that the parameter never increases, while the size of H_G is polynomial in the size of G and H. □

Chen, Thurley and Weyer were able to show that this even holds for every unbounded class of graphs from which we may choose G.

Theorem 1.22 ([CTW08, Corollary 4])**.** *Let $\mathcal{C}$ be a family of graphs without an upper bound on the number of edges. Then the problem*

		$\mathcal{C}$ Induced Subgraph Isomorphism
Input	:	$G \in \mathcal{C}, H \in \mathcal{G}$
Parameter	:	$\lvert V(G)\rvert + \lvert E(G)\rvert$
Question	:	Is G isomorphic to an induced subgraph of H?

is W[1]*-complete.* ⍁

1.5. Other classes

Early parameterized results (e.g. [Luk82; Bod90]) usually do not handle all instances of a certain problem and study the role of the parameter, but discuss subproblems where all instances share the same parameter (partial k-trees and graphs with treewidth k). Classical problems of this kind are called ***slices*** in parameterized complexity. We now look at the class of parameterized problems, whose slices are decidable in polynomial time (in a uniform manner).

Definition 1.23. Let (L, κ) be a parameterized problem over Σ. The lth ***slice*** of (L, κ) is the language $(L, \kappa)_l = \{w \in \Sigma \mid \kappa(w) = l\}$. Then (L, κ) shall be contained in the class XP if there is a computable function A from $\mathbb{N}$ to a set of algorithms, such that for all $l \in \mathbb{N}$ the algorithm $A(l)$ decides $(L, \kappa)_l$ in polynomial time. •

Equivalently we can define XP as the class of parameterized problems (L, κ), for which there exists a an algorithm and a computable function f, such that for each $w \in \Sigma$ the algorithm decides whether $w \in L$ in at most $f(\kappa(w)) \cdot |w|^{f(\kappa(w))}$ computation steps. Problems in XP would actually also deserve the attribute "fixed-parameter tractable", since handling each slice separately is what is referred to as "fixing a value" in other areas of mathematics. While other names ("slowly increasing" is used in [EP99] to refer to FPT) might better distinguish between XP and FPT, we continue to use the standard naming.

Theorem 1.24. *For all $t < t' \in \mathbb{N}$:* $\mathsf{FPT} \subseteq \mathsf{W}[t] \subseteq \mathsf{W}[t'] \subseteq \mathsf{XP}$.

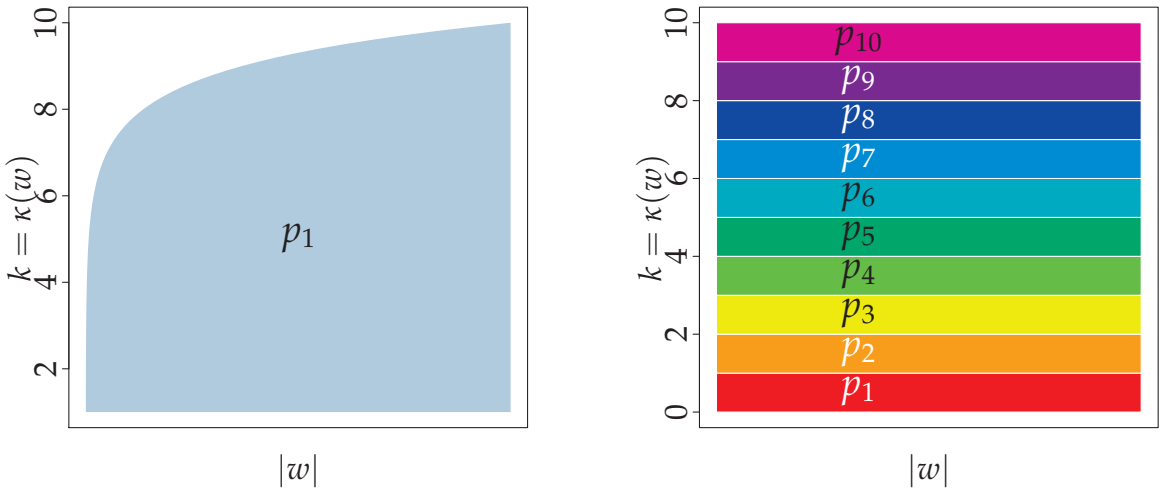
Proof sketch. W[t] is closed under fpt-reductions by definition and thus contains FPT. Further a formula ϕ is equivalent to $\bigwedge_{i\in[1]} \phi$ and $\bigvee_{i\in[1]} \phi$ and thus $\mathsf{W}[t] \subseteq \mathsf{W}[t']$. Finally we can decide whether $\phi \in$ Weighted t-Normalized Satisfiability$_l$, by testing all (i.e. $\mathcal{O}(|\phi|^l)$) assignments of weight l. □

On the other hand (by a parameterized diagonalization argument and the time hierarchy theorem, see e.g. [FG06, Corollary 2.26]) the outermost classes of this chain are different.

Theorem 1.25. $\mathsf{FPT} \subset \mathsf{XP}$ ⍁

Similar classes can be defined for space complexity. However, we do this using multi-tape Turing machines. We assume that every Turing machine has a non-writable input tape, a write-only output tape (i.e. the head may only stay or move to the right) and an arbitrary number of working tapes. The amount of space used by a machine on an input is the maximal number of used cells on all work tapes combined over all configurations during its execution. There are two classes which correspond to classical logarithmic space (L):

Figure 1.1. Illustration of the difference between FPT and XP: For a parameterized problem in FPT and algorithm with runtime bound $f(k)p(|w|)$ there is a polynomial $p_1(|w|) = |w| \cdot p(|w|)$ that bounds the runtime if $k \leq f^{-1}(|w|)$. With increasing $|w|$ instances with slightly larger κ become tractable (relative to $|w|$), these are the blue instances under the curve in the left picture. By contrast the right picture depicts the situation for problems in XP. For each possible value $\kappa(w) = k$ there is a polynomial p_k that bounds the runtime of some algorithm. But whenever we choose some series of instances such that $k \in \omega(|w|)$ the runtime cannot be bounded by any p_i, $i \in \mathbb{N}$



.

Definition 1.26. Let (L, κ) be a parameterized problem over Σ. The class XL contains (L, κ) if every slice $(L, \kappa)_l$ can be decided in logarithmic space by a uniform family of algorithms analogously to XP. (L, κ) is in para-L if there is an algorithm and a computable function f, such that for every $w \in \Sigma$ the algorithm decides whether $w \in L$ in at most $f(k) + \mathcal{O}(\log |w|)$ computation steps. •

With a similar argument as in the classical case (only polynomial many different configurations on logarithmic space) we get:

Lemma 1.27. para-L $\subseteq$ FPT *and* XL $\subseteq$ XP. ☑

We remark that FPT is also know as para-P and that X and para versions exist for a variety of classical complexity classes, see [FG03] for general information about the relation X vs. para and [EST12] for parameterized space complexity.

2. The graph isomorphism problem

Mathematicians usually define objects by the way they interact with other objects, not by by "what they are made of". There are multiple ways to define the set of real numbers and thus the number one can, for instance, be either a Dedekind cut or a Cauchy sequence (or rather equivalence classes of those). But under all circumstances it is the multiplicative identity. As a consequence two structures which only differ by the elements in there base sets are considered equivalent and the formal criterion for this equivalence is the existence of an ***isomorphism***, i.e. a bijective mapping that respects relations and operations. The ***isomorphism problem*** asks whether there is an isomorphism between two given structures. Since graphs are one of the most important class of structures in computing science, the isomorphism problem for graphs has a wide area of applications. Furthermore the isomorphism problem for explicitly given structures (i.e. all elements of relations are listed) is polynomial time reducible to graph isomorphism [Mil79]. Finally, the graph isomorphism problem plays an important role for complexity, as it is one of a few natural problems that is neither known to be efficiently decidable (i.e. in P) nor known to be NP-complete. A related problem is the ***graph canonization problem***. It demands the construction of a ***canonical form***, that is, a graph isomorphic to a given input graph such that for two isomorphic input graphs the output is identical. In this second introductory chapter we define graph isomorphisms and canonical forms as well as the corresponding parameterized problems and look at some basic algorithms from the viewpoint of parameterized complexity.

2.1. Graph isomorphisms and canonical forms

Definition 2.1 (Isomorphic graphs, isomorphism, automorphism)**.** For two Graphs G_1 and G_2, a function ϕ is an ***isomorphism*** from G_1 to G_2 ($G_1 \cong_\phi G_2$) if it is a bijection $\phi : V(G_1) \to V(G_2)$ such that

$$\forall \{u, v\} \in \binom{V(G_1)}{2} : \quad \{u, v\} \in E(G_1) \Leftrightarrow \{\phi(u), \phi(v)\} \in E(G_2)\,.$$

Two graphs G_1 and G_2 are ***isomorphic*** ($G_1 \cong G_2$) if there is an isomorphism ϕ such that $G_1 \cong_\phi G_2$. A function ϕ is an ***automorphism*** of G_1 if $G_1 \cong_\phi G_1$ and the automorphism group of G_1 ($\mathrm{Aut}(G_1)$) is the group

$$(\{\phi \mid G_1 \cong_\phi G_1\}, \circ)\,.$$

For any graph G the set $\{H \in \mathcal{G} \mid G \cong H\}$ is the ***isomorphism type*** of G. •

Colorings are a very handy tool to encode structural information of a graph (e.g. information about the neighborhood of a vertex). Many graph isomorphism algorithms (see e.g. chapter 4) rely on the fact that such structural information has to be preserved by isomorphisms between colored graphs. We now define this formally.

Definition 2.2 (Isomorphisms of colored Graphs)**.** Two vertex and edge colored Graphs (G_1, c_1) and (G_2, c_2) are ***isomorphic*** ($(G_1, c_1) \cong_\phi (G_2, c_2)$), the isomorphism being ϕ, if and only if their underlying graphs are isomorphic and the colors correspond: $G_1 \cong_\phi G_2$ and $\forall m \in (V(G_1) \cup E(G_1)) : c_1(m) = c_2(\phi(m))$. The usual case of vertex colored graphs is defined analogously, as well as ***automorphisms***, the ***automorphism group*** $\mathrm{Aut}(G_1, c_1)$ and the ***colored isomorphism type*** $\{H \in \mathcal{G}_c \mid (G_1, c_1) \cong H\}$ for both cases. •

See figures 2.1 and 2.2 for examples of (non-)isomorphic (colored) graphs.

Figure 2.1. Two isomorphic graphs, but two non-isomorphic colored graphs: The identity on $V(G_1)$ is an isomorphism between G_1 and G_2, but the coloring c_2 does not assign red to a single vertex, while $c_1(a)$ is red.

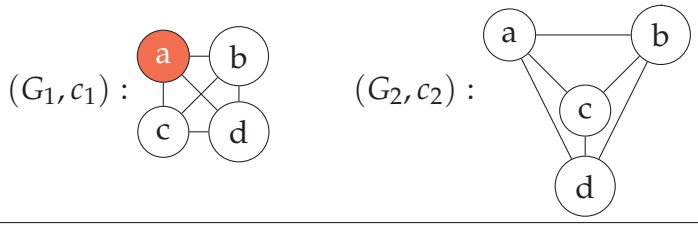
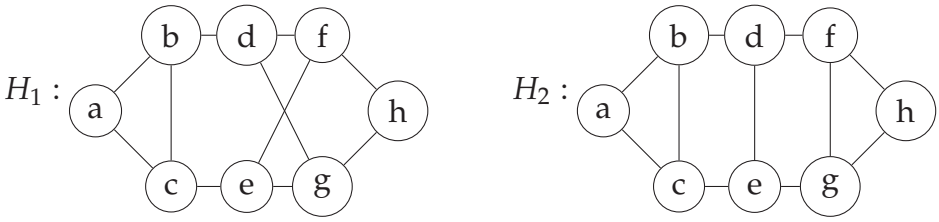


Figure 2.2. Two non-isomorphic graphs: In both graphs only a and h have degree 2, but in H_1 both a and h are simplicial (their neighborhood induces a clique), while in H_2 only a is simplicial.



Definition 2.3. Let $f : \mathcal{G} \to \Gamma^*$ be a function on graphs. We call f an ***invariant*** if for two isomorphic graphs $G_1, G_2 \in \mathcal{G}$ their values coincide, i.e. $f(G_1) = f(G_2)$. If the other implication $f(G_1) = f(G_2) \Rightarrow G_1 \cong G_2$ also holds, f is a ***complete*** invariant. •

We only discuss ***invariant graph parameters*** in this work and hence we will not mention the fact that a parameter is an invariant from now on, but simply assume it. Furthermore we may use $\kappa(G)$ for an arbitrary graph G (i.e. not require $G \in \mathcal{G}$), because any bijection $f : V(G) \to [|V(G)|]$ yields the same value $\kappa(G) = \kappa(([|V(G)|], \{f(e) \mid e \in E(G)\}))$.

Example 2.4. The eigenvalue multiplicity is an invariant. The adjacency matrices of two isomorphic graphs $G_1 \cong_\phi G_2$ in $\mathcal{G}$ can be transformed into each other by a permutation matrix P that corresponds to the isomorphism, i.e.

$$A_{G_2} = P^{-1} A_{G_1} P \quad \text{with} \quad P_{i,j} = \begin{cases} 1 & \phi(j) = i \\ 0 & \text{else} \end{cases} \text{ and thus}$$

$$A_{G_1} r = \lambda r \Leftrightarrow A_{G_1} P P^{-1} r = \lambda r \Leftrightarrow P^{-1} A_{G_1} P P^{1} r = P^{-1} \lambda r \Leftrightarrow A_{G_2} P^{-1} r = \lambda P^{-1} r$$

for all $r \in \mathbb{R}^{|V(G_1)|}$. So the bijection corresponding to P^{-1} maps eigenspaces of A_{G_1} to those of A_{G_2}, we know that there spectra coincide and thus $\mathrm{em}(G_1) = \mathrm{em}(G_2)$. e.g.

Definition 2.5. A function $\mathtt{C} : \mathcal{G} \to \mathcal{G}$ is ***canonical form*** if for any two graphs G_1 and G_2

$$\mathtt{C}(G_1) \cong G_1 \text{ and } (G_1 \cong G_2 \Rightarrow \mathtt{C}(G_1) = \mathtt{C}(G_2)) .$$

Let l be a labeling of a graph $G \in \mathcal{G}$ such that $\mathrm{rng}(l) = V(G)$. Then l is a ***canonical labeling*** of G w.r.t. $\mathtt{C}$ if

$$(V(G), \{l(e) \mid e \in E(G)\}) = \mathtt{C}(G) .$$

The corresponding canonical labeling relation is the set

$$\mathrm{cl}_{\mathtt{C}} = \{(G, l) \mid l \text{ is a canonical labeling of } G \text{ w.r.t. } \mathtt{C}\} .$$

We will use $\mathrm{cl}_{\mathtt{C}}(G)$ to denote the set of all canonical labelings of G,

$$\mathrm{cl}_{\mathtt{C}}(G) = \{l \mid (G, l) \in \mathrm{cl}_{\mathtt{C}}\} .$$

All these notions shall also be defined for colored graphs in an analogous manner. •

Whenever we construct a concrete canonical form, we drop the index $\mathtt{C}$. As a canonical labeling of a graph $G \in \mathcal{G}$ is a permutation of $V(G)$, we will interpret a labeling as a coloring (and denote it with Latin letters) or a permutation (and use Greek letters) depending on the context.

Lemma 2.6. *For any canonical form* $\mathtt{C}$*, any graph* $G \in$ *cGraphs and any* $l \in \mathrm{cl}_{\mathtt{C}}(G)$

$$\mathrm{cl}_{\mathtt{C}}(G) = \{l\phi \mid \phi \in \mathrm{Aut}\} .$$

Proof. Let l, l' be in $\mathrm{cl}_{\mathtt{C}}(G)$. Since $l(E(G)) = l'(E(G))$, $l^{-1}l'(E(G)) = E(G)$. Thus $\phi = l^{-1}l'$ is an automorphism and $l' = l\phi$. □

2.2. Parameterized isomorphism problems

In this section we directly define related parameterized problems and briefly discuss there relations among them if we parameterize by the same parameter.

Definition 2.7. Let κ be a graph parameter. We define ***Graph Isomorphism parameterized by*** κ as:

		Graph Isomorphism($\kappa = k$)
Input	:	(G_1, G_2, k), where G_1 and G_2 are graphs and $k \in \mathbb{N}$
Parameter	:	k
Question	:	$G_1 \cong G_2$ and $\kappa(G_1) = k$?

As discussed in section 1.1 ***Graph Isomorphism parameterized by*** κ shall be the following problem if $\kappa \in \mathsf{FP}$ is known:

		Graph Isomorphism(κ)
Input	:	A pair of graphs (G_1, G_2)
Parameter	:	$\kappa(G_1)$
Question	:	$G_1 \cong G_2$?

•

We will always speak about the graph isomorphism problem in later chapters, even though in most cases we actually handle the functional version:

Definition 2.8. Let κ be a graph parameter. Then ***f-Graph Isomorphism parameterized by*** κ shall be the following parameterized function problem:

		f-Graph Isomorphism($\kappa = k$)
Input	:	(G_1, G_2, k), where G_1 and G_2 are graphs and $k \in \mathbb{N}$
Parameter	:	k
Output	:	If $\exists \phi : G_1 \cong_\phi G_2$ and $\kappa(G_1) = k$: ϕ, else `null`.

As discussed in section 1.1 ***Graph Isomorphism parameterized by*** κ shall be the following problem if $\kappa \in \mathsf{FP}$ is known:

		f-Graph Isomorphism(κ)
Input	:	A pair of graphs (G_1, G_2)
Parameter	:	$\kappa(G_1)$
Output	:	If $\exists \phi : G_1 \cong_\phi G_2$: ϕ, else "not isomorphic"

•

We use `null` instead of "not isomorphic" or similar in the first case, because we don't actually know whether there is an isomorphism and we may interpret $\kappa(G_1) \leq k$ as bound on a resource, we were allowed to use. For the related problems, we only define the mediate version explicitly.

Definition 2.9. Let κ be a graph parameter and λ be a function $\mathcal{G}_c \to \mathbb{N}$. Then ***Colored Graph Isomorphism parameterized by*** λ shall be the parameterized problem:

		Colored Graph Isomorphism($\lambda = k$)
Input	:	$((G_1, c_1), (G_2, c_2), k)$, two colored graphs and $k \in \mathbb{N}$
Parameter	:	k
Question	:	$(G_1, c_1) \cong (G_2, c_2)$ and $\lambda((G_1, c_1)) = k$?

Whereas the ***Graph Automorphism parameterized by*** κ is:

		f-Graph Automorphism($\kappa = k$)
Input	:	(G, k), a graphs G and $k \in \mathbb{N}$
Parameter	:	k
Output	:	A set of generators for $\mathrm{Aut}(G)$ if $\kappa(G) = k$, else `null`

•

In the non-parameterized world all problems above are polynomial time equivalent via Turing reductions (see e.g. [Mat79; Hof82]), i.e. given an algorithm for one of those problems with runtime bound $r : \mathbb{N} \to \mathbb{N}$, we can find algorithms for all other problems with runtime bound $p + p \circ r$ (for some polynomial p). If we consider all problems above for the same parameter, this caries over to the parameterized world for many parameters.

From colored graphs to uncolored graphs There are some methods to turn a pair of colored graphs (G_1, c_1) and (G_2, c_2) ($|V(G_1)| = |V(G_2)|$) into uncolored graphs G_1' and G_2' such that

$$(G_1, c_1) \cong (G_2, c_2) \Leftrightarrow G_1' \cong G_2' .$$

One method attaches a graph B_v to each $v \in V(G_1) \cup V(G_2)$ such that for two vertices $u \in V(G_i), v \in V(G_j)$: $B_v \cong B_u \Leftrightarrow c_i(u) = c_j(v)$. The attached graphs B_v have to be chosen in a way that guarantees that no induced subgraph of either G_i is isomorphic to B_v (and no other additional local isomorphic induced subgraphs occur). A typical choice for B_v is therefore a large (i.e. $> |V(G_1)|$ vertices) clique or path together with some graph that encodes the color.

A second method (e.g. in [BC79]) adds $c_i(v)$ loops to each vertex v and subdivides every edge (including the new loops) by adding a new vertex in the middle. Yet another approach for encoding relations is described in [Gur97]. We may use it for colors, but it only works for Turing reductions: check that the sets of used colors in both graphs are the same and replace the colorings by relations ($R(u,v) \Leftrightarrow c(u) < c(v)$). Then encode pairs in the relation using a special arc graph and mark old vertices with two new adjacent leaves. Adding cliques whose size is not bounded by the graph parameter in question or adding an unbounded number of circles violates the third condition of fpt-reductions. We may, however, use the parameter itself to construct color encodings.

Example 2.10. To show that Colored Graph Isomorphism(max deg) is fpt-reducible to Graph Isomorphism(max deg) (the first max deg is formally a function on $\mathcal{G}_c$) we may choose the first method and first check $\max\deg(G_1) = \max\deg(G_2)$. Then we attach a path of length $c(v)$ to a $(\max\deg(G_1)+2)$-clique and this clique to the vertex v and go on for all other vertices of G_1 and G_2. Now each new clique can only be mapped to a new clique and each new clique has exactly two vertices of degree $\max\deg(G_1)+2$, one that is connected to the old vertex and one that is connected to a path that is separated from all other new cliques by the clique in question. Furthermore the maximal degree is only augmented by at most 1 and thus we described an fpt-reduction. e.g.

No such method which attaches multiple new connected vertices (and thus new uncovered edges) to every vertex can work if we parameterize Graph Isomorphism by the minimal vertex cover size vc. On the other hand, Graph Isomorphism$(\mathrm{vc} = k)$ is fixed-parameter tractable (see section 4.1.2) and its algorithm (4.3) can be easily modified to check for colors. Thus Colored Graph Isomorphism$(\mathrm{vc} = k) \in \mathsf{FPT}$ and therefore Colored Graph Isomorphism$(\mathrm{vc} = k) \leq^{\text{fpt-m}}$ Graph Isomorphism$(\mathrm{vc} = k)$. To my knowledge it is unknown whether Colored Graph Isomorphism$(\kappa = k) \leq^{\text{fpt-m}}$ Graph Isomorphism$(\kappa = k)$ holds for all graph parameters κ. A counterexample $\kappa \in \mathsf{FFPT}$ would yield Colored Graph Isomorphism$(\kappa = k) \notin \mathsf{FPT}$ and thus $\mathsf{P} \neq \mathsf{NP}$.

Question 2.11. Does Colored Graph Isomorphism$(\kappa = k) \leq^{\text{fpt-m}}$ Graph Isomorphism$(\kappa = k)$ hold for every $\kappa \in \mathsf{FFPT}$? **?**

Lemma 2.12. *f-Graph Isomorphism$(\kappa = k) \leq^{\text{fpt-T}}$ Colored Graph Isomorphism$(\lambda_\kappa = k)$ with $\lambda_\kappa((G,c)) = \kappa(G)$ for all $(G,c) \in \mathcal{G}_c$.*

Proof sketch. To compute an isomorphism from a graph G_1 to a graph G_2 (provided one exists), we assign the same color (say 1) to all vertices of both graphs and check whether a colored isomorphism exists. If there is none, we are done, else we pick a vertex $u \in V(G_1)$ and a vertex $v \in V(G_2)$ and assign the same new color (smallest unused) to both. We go on by checking for an isomorphism between both colored graphs. If there are not isomorphic, we assign the color 1 to v and choose another vertex in G_2 with color 1. Since both graphs are isomorphic, we finally succeed and find a parter for u. This procedure has to be repeated until all vertices $u \in V(G_1)$ are matched. Now the pairs of matching vertices form an isomorphism between G_1 and G_2. Finally, we observe that all queries to Colored Graph Isomorphism(λ_κ) are performed in a way such that $\kappa(G_1) = \lambda_\kappa((G_1, c_{1,i}))$ for the coloring $c_{1,i}$ in question and thus $\kappa \preceq_{f_i} \lambda_\kappa$ (f being the function that attaches $c_{j,i}$ to G_j for $j \in [2]$). Hence this procedure is an fpt Turing reduction. □

In a similar manner, using a tower of groups (see section 3.1), one obtains the following lemma.

Lemma 2.13. f-Graph Automorphism$(\kappa = k) \leq^{\text{fpt-T}}$ Colored Graph Isomorphism$(\kappa = k)$ ☑

Lemma 2.14. *Let $\kappa \in \mathsf{FFPT}$ be a graph parameter such that there are computable functions f and g with $\kappa(G) \leq g(\{\!\{\kappa(G[C]) \mid C \text{ is a component of } G\}\!\})$ and $\kappa(G[C]) \leq f(\kappa(G))$ for all connected components C of all graphs $G \in \mathcal{G}$. Then* Graph Isomorphism$(\kappa = k) \leq^{\text{fpt-T}}$ f-Graph Automorphism$(\kappa = k)$.

Proof sketch. We assume that the input graphs G_1 and G_2 for Graph Isomorphism$(\kappa = k)$ are vertex disjoint and connected, if they are not connected we handle components separately (backed by the existence of f, see lemma 2.28). If the input contains k as a candidate for $\kappa(G_1)$, we check $k = \kappa(G_1)$ and $k = \kappa(G_2)$ and reject if this check fails. Now we are able to construct the union graph $G = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$. Furthermore $\kappa(G) = g(\{\!\{\kappa(G_1), \kappa(G_1)\}\!\})$ and thus only depends on $\kappa(G_1)$. Hence we may actually compute $k' = \kappa(G)$ in fpt time and an oracle query to f-Graph Automorphism$(\kappa = k)$ with instance (G, k') will be always answered with a set of generators $\langle g_1, \ldots, g_l \rangle$ for $\mathrm{Aut}(G)$. We obtain

$$G_1 \cong G_2 \Leftrightarrow \exists i : g_i(V(G_1)) = V(G_2)\,,$$

because an automorphism cannot "partly" switch components. This completes the reduction. □

Note that we could have started with the colored version of each problem if the parameter for colored graphs ignores the colors (or at least does not inhibit the operations in the reduction). We conclude that the relations among the different flavors of the graph isomorphism/automorphism problem from the non-parametrized world remain intact when we deal with a single parameter, provided that we are able to reduce the colored problem to the uncolored case.

2.3. Parameterized canonization problems

The problems of computing canonizations and canonical labelings are actually an entire class of problems. If we apply canonizations in the context of Graph Isomorphism we do not care whether a canonical form has any specific property beside being canonical. This is why we modeled classes of parameterized function problems and the reductions among them in the specific way described in definitions 1.8 and 1.14. In this section we only define the directly parameterized versions explicitly and imply the definition for the mediately parameterized versions, but we use the mediate versions in lemmas and proofs, as they require a little more care in some cases. Again all reductions from uncolored to colored problems work analogously if we start with a colored problem whose parameter ignores colors.

Definition 2.15. Let $\kappa \in \mathsf{FP}$ be a graph parameter. ***Graph Canonization parameterized by*** κ (Graph Canonization(κ)) is the parameterized problem

$$(\{\mathsf{C} \mid \mathsf{C} \text{ is a canonical form}\}, \kappa)\,.$$

Further, we define ***Canonical Labeling parameterized by*** κ (Canonical Labeling(κ)) as

$$(\{\mathrm{cl}_{\mathsf{C}} \mid \mathsf{C} \text{ is a canonical form}\}, \kappa)\,.$$

Let Canonical Labeling Coset(κ) be defined as

$$(\{f \mid f(G) = (l, g_1, \ldots, g_n), \mathrm{Aut}(G) = \langle g_1, \ldots, g_n\rangle, l \in \mathrm{cl}_{\mathtt{C}}(G), \mathtt{C} \text{ is a canonical form}\}, \kappa)\,,$$

i.e. the problem of computing a representation for the coset containing all canonical labelings of a given graph. By Complete Invariant(κ) we denote the problem

$$(\{f : \mathcal{G} \to \Gamma^* \mid f \text{ is a complete invariant}\}, \kappa)\,.$$

Finally Graph Isomorphism Order(κ) shall be the problem

$$\left(\left\{ f : \mathcal{G}^2 \to \{-1,0,1\} \;\middle|\; \forall G,H,K \in \mathcal{G} : \begin{array}{l} f(G,H) = -f(H,G), \\ f(G,H) = 0 \Leftrightarrow G \cong H, \\ f(G,H) = f(H,K) = x \Rightarrow f(G,K) = x, \\ G \cong H \Rightarrow f(G,K) = f(H,K) \end{array} \right\}, \kappa_{\max}\right),$$

where $\kappa_{\max}(G_1, G_2) = \max\{\kappa(G_1), \kappa(G_2)\}$ for $(G_1, G_2) \in \mathcal{G}^2$. The definitions of the colored versions can be given analogously. •

Lemma 2.16. *Graph Isomorphism($\kappa = k$)* $\leq^{\text{fpt-T}}$ *Graph Canonization($\kappa = k$)* $\leq^{\text{fpt-T}}$ *Canonical Labeling($\kappa = k$)*

Proof. For the first reduction we query Graph Canonization($\kappa = k$) with each input graphs G_i, $i \in [1,2]$ and the candidate k for κ. If the query for G_1 is answered with `null`, we also return `null`. If not and the query for G_2 is answered with `null`, we return "not isomorphic", as κ is an invariant. In any other case we have two canonical forms and return "isomorphic" if and only if they are equal. The second reduction is obvious, we query with or own instance and apply the labeling that we got (if we got one). □

As a consequence we see that it is sufficient to handle the canonical labeling problem if we like to find upper bounds for the parameterized complexity of Graph Isomorphism. There are, however, situations were it is interesting to discuss both problems separately, even if an algorithm for both exists and the best known runtimes do not differ. This is the case if we like to indicate the transition from a Graph Isomorphism algorithm to a Canonical Labeling algorithm or if the algorithm for Graph Isomorphism is simpler and may serve as an introduction.

We now look at further reductions among the different canonical labeling problems, which are similar to the reductions for Graph Isomorphism.

Example 2.17. We continue example 2.10. Let f be the reduction function we described there for Colored Graph Isomorphism(max deg) $\leq^{\text{fpt-m}}$ Graph Isomorphism(max deg) (attaching widgets based on colors), we will use it to show Colored Canonical Labeling(max deg) $\leq^{\text{fpt-m}}$ Canonical Labeling(max deg). Observe that for an instance (G, c) the vertices of G are still present in $f(G, c)$. If we query for $f(G, c)$ and get l as answer, we simply return a condensed version of $l|_{V(G)}$, i.e. we sort the $v \in V(G)$ by $l(v)$ and return the positions $s : V(G) \to [|V(G)|]$. To see that s is a canonical labeling observe that the transformation from l to s describes the transformation from $\mathtt{C}(f(G, c))$ to $\mathtt{C}(f(G, c))[l(V(G))]$ where $\mathtt{C}$ is the canonical form corresponding to l. e.g.

Once more we cannot give a general scheme for arbitrary parameters and have to pose the following question.

Question 2.18. Does Colored Canonical Labeling$(\kappa = k) \leq^{\text{fpt-T}}$ Canonical Labeling$(\kappa = k)$ hold for every $\kappa \in$ FFPT? **?**

The following lemma can be seen as an analogon to lemma 2.13.

Lemma 2.19. *Canonical Labeling Coset*$(\kappa = k) \leq^{\text{fpt-T}}$ *Colored Canonical Labeling*$(\kappa = k)$.

Proof sketch. From the lemmas 2.13 and 2.16 we know f-Graph Automorphism$(\kappa = k) \leq^{\text{fpt-T}}$ Colored Graph Isomorphism$(\kappa = k) \leq^{\text{fpt-T}}$ Colored Canonical Labeling$(\kappa = k)$ and Canonical Labeling$(\kappa = k) \leq^{\text{fpt-T}}$ Colored Canonical Labeling$(\kappa = k)$ is obvious. □

Obviously, we have Complete Invariant$(\kappa) \leq^{\text{fpt-T}}$ Graph Canonization(κ), but the converse also holds for parameters with Colored Canonical Labeling$(\kappa = k) \leq^{\text{fpt-T}}$ Canonical Labeling$(\kappa = k)$.

Lemma 2.20 (similar in [Gur97])**.** *Canonical Labeling*$(\kappa = k) \leq^{\text{fpt-T}}$ *Colored Graph Isomorphism Order*$(\kappa = k) \leq^{\text{fpt-T}}$ *Colored Complete Invariant*$(\kappa = k)$

Proof sketch. For the first reduction let f be the isomorphism order function computed by our oracle for Colored Graph Isomorphism Order$(\kappa = k)$. Iteratively enlarging $\operatorname{dom}(l)$, construct partial canonical labelings l such that $(G, l \cup c_0(l))$ gets minimal w.r.t. f, where $c_0(l) : (V(G) \setminus \operatorname{dom}(l)) \to \{0\}$. That means, if v is the next vertex to be added to $\operatorname{dom}(l)$, then

$$\forall u \in V(G) \setminus \operatorname{dom}(l) : f((G, l_v \cup c_0(l_v)), (G, l_u \cup c_0(l_u))) \in \{0, 1\},$$

where $l_v = l \cup (v, |\operatorname{dom}(l)| + 1)$.

The second reduction is obvious because of our definition of Colored Graph Isomorphism Order$(\kappa = k)$ using $\kappa_{\max}$ in definition 2.15. □

2.4. Basic algorithms

The graph isomorphism algorithms we will investigate later use simple algorithms for certain base cases. To avoid later deviation, we will discuss them here briefly. Most of these basic algorithms rely on colors to encode structural information.

2.4.1. The Weisfeiler-Lehman algorithm

The 2-dimensional Weisfeiler-Lehman algorithm goes back to the 1960s [WL68; Wei76] and gave also rise to the study of so called cellular algebras. Later it was generalized to an arbitrary dimensionality ≥ 2. The related one-dimensional ***naïve vertex refinement*** (often called 1-dim WL) is the core building block in [BK79] to achieve an linear average time algorithm for Graph Isomorphism. For a detailed description of the Weisfeiler-Lehman algorithm, we refer to [CFI92].

Informally, the d-dimensional Weisfeiler-Lehman algorithm for $d \geq 2$ works as follows: color all

d-tuples of vertices by the (colored) isomorphism type of the graph they induce (observing the order of vertices in the tuple). Then exchange each vertex (one at a time) of the tuple by the same vertex and write down the colors (as a tuple) of the resulting tuples. Do this for all vertices and set the resulting multiset as the new color of the tuple. Repeat this procedure for all d-tuples until no new pair of tuples gets distinguished by a newly assigned color (partition refinement). Algorithm 2.1 is a formal description of this procedure. Note that we have to recolor with integers in each step to avoid exponential growth of the color names. Nevertheless the colors of former rounds and the renaming tables are stored to tell similar but different colorings apart (think of e.g. $c'(v) = c(v) + 1$).

Algorithm 2.1. d-dimensional Weisfeiler-Lehman algorithm for $d \geq 2$: ***d-dim WL***

Input : Colored graph $(G, c) \in \mathcal{G}_c$
Output : Multiset of d-dimensional stable coloring

subprocedure $\text{rep}(v{=}(v_1, \ldots, v_d), i, w)$
  $u \leftarrow v$; $u_i \leftarrow w$; **return** u
end
$i \leftarrow 0$
for $v{=}(v_1, \ldots, v_d) \in V(G)^d$
  $c_0(v) \leftarrow (c(v_1), \ldots, c(v_d), \text{adj}(G[\{v_i \mid i \in [d]\}]))$
end
do
  $i \leftarrow i + 1$
  for $v{=}(v_1, \ldots, v_d) \in V(G)^d$
    $c'_i(v) \leftarrow (c_{i-1}(v), \{\!\{(c_{i-1}(\text{rep}(v, 1, w)), \ldots, c_{i-1}(\text{rep}(v, d, w))) \mid w \in V(G)\}\!\})$
  end
  $s_i \leftarrow \text{sortLexico}(\text{rng}(c'_i))$ i.e. $s_i : [|\,\text{rng}(c'_i)\,|] \to \text{rng}(c'_i)$, $s_i(r) <_{\text{lex}} s_i(s) \Rightarrow r < s$
  for $v{=} \in V(G)^d$
    $c_i(v) \leftarrow s_i^{-1}(c'_i(v))$
  end
until $\forall\, (u, v) \in (V(G)^d)^2 : c_i(u) = c_i(v) \Leftrightarrow c_{i-1}(u) = c_{i-1}(v)$
return $\{\!\{(c_0(v), \ldots, c_i(v)) \mid v \in V(G)^d\}\!\}$

Lemma 2.21. *The d-dimensional Weisfeiler-Lehman algorithm (2.1) computes an invariant in polynomial time (at most $|V(G)|^d$ iterations of the do-until-loop).* ☐

Corollary 2.22. *Let $(G, c) \in \mathcal{G}_c$ be a colored graph. Then algorithm 2.2 returns in polynomial time a coloring f such that the colored isomorphism type of (G, f) is an invariant (if seen as a function of (G, c)).* ☐

Remark 2.23. An algorithm for a parameterized problem that has a runtime bound according to the definition of FPT can only contain the d-dimensional Weisfeiler-Lehman algorithm if d does not depend on the parameter. **!**

Algorithm 2.2. Vertex coloring from d-dimensional Weisfeiler-Lehman algorithm for $d \geq 2$: ***flattened d-dim WL***

Input : Colored graph $(G, c) \in \mathcal{G}_c$
Output : Vertex coloring of G

```
1  compute stable coloring c_i using d-dim WL
2  for w ∈ V(G)
3  |    for j ∈ [d]
4  |    |    f'_j(v) ← {{ c_i((v_1,...,v_d)) | (v_1,...,v_d) ∈ V(G)^d, v_j←w }}
5  |    end
6  |    f'(v) ← (f'_1(v),...,f'_i(v))
7  end
8  s ← sortLexico(rng(f'))
9  for w ∈ V(G)
10 |    f(w) ← (c(w), s^{-1}(f'(w)))
11 end
12 return f
```

2.4.2. Linear time algorithms for trees

For some graph classes like trees, the output of the d-dimensional Weisfeiler-Lehman algorithm is even a complete invariant for some fixed d. In the case of tree this even hold s for the naïve vertex refinement (1-dim WL). There are, however, simpler and faster algorithms in the case of trees. While the Weisfeiler-Lehman algorithm propagates adjacency information in all directions, it suffices for trees if we work from the leaves to the center. Algorithm 2.3 iteratively removes leaves and colors the sole neighbor of some leaves based on their colors in a consistent manner for two graphs. If its input are trees, its output is an isolated edge or vertex (the former center) colored according to the isomorphism type of the input graph and thus a kind of "relative complete invariant". The output keeps this property if the graphs are not trees, but it can, of course, have any number of vertices. Note that we can generalize this algorithm for more than two input graphs.

Lemma 2.24 ([AHU74, Theorem 3.2]). *A set of tuples $S = \{(a_{i,1}, \ldots, a_{i,l_i}) \mid i \in [k], a_{i,j} \in [m]\}$ can be sorted by a modified version of radix sort in time*

$$\mathcal{O}(m + \sum_{i=1}^{k} l_i)\,.$$

Lemma 2.25 ([AHU74, Theorem 3.3]). *Let $(G_1, c_1), (G_2, c_2) \in \mathcal{G}_c$ and let (G_1', c_1') and (G_2', c_2') be both graphs after the application of algorithm 2.3. Then each component of both (G_i', c_i') has either no leaves or at most two vertices and*

$$(G_1, c_1) \cong (G_2, c_2) \Leftrightarrow (G_1', c_1') \cong (G_2', c_2')\,.$$

Furthermore algorithm 2.3 can be implemented in linear time.

Proof sketch. The key observation for the correctness is to see that the check in line 18 ensures the consistence of the recoloring among both graphs. Each color of a vertex $v \in L_{i,l-1}$ corresponds to the isomorphism type of the tree formed of v and its adjacent components

Algorithm 2.3. Remove leaves and recolor [AHU74, Example 3.2]: ***removeLeavesAndRecolor***

Input : Colored graphs (G_1, c_1) and (G_2, c_2)
Output : Colored graphs where each component has either no leaves or at most two vertices

```
1  if (|V(G_1)|≠|V(G_2)|)
2  |   return fixed pair of non-isomorphic colored graphs
3  for i ∈ [2]
4  |   compute L_{i,0} ⊆ V(G_i) by iteratively removing vertices    //center for trees
5  |   |   with degree 1 from Components with at least 3 vertices
6  |   compute all L_{i,l} ← {v | min_{u∈L_{i,0}} d(u,v)=l}
7  |   h_i ← max {l | L_{i,l} ≠ ∅}
8  end
9  if (h_1 ≠ h_2)
10 |   return fixed pair of non-isomorphic colored graphs
11 for l←h_1 down to 1
12 |   for i ∈ [2]
13 |   |   for v ∈ L_{i,l-1}
14 |   |   |   c'_1(v)← (c(v), {{c(u) | u ∈ L_{i,l} ∩ N(v)}})
15 |   |   end
16 |   |   s_i ← sortLexico(c'_i(L_{i,l-1}))    //input as multiset, using lemma 2.24
17 |   end
18 |   if (s_1 ≠ s_2)
19 |   |   return fixed pair of non-isomorphic colored graphs
20 |   for i ∈ [2]
21 |   |   for v ∈ L_{i,l-1}
22 |   |   |   c_i(v)← s_i^{-1}(c'(v))    //s_i with duplicates removed
23 |   |   end
24 |   |   G_i ← G_i \ L_{i,l}
25 |   end
26 end
27 return (G_1, c_1) and (G_2, c_2)
```

in $G_i[F_{i,l}]$, where $F_{i,l} = \bigcup_{j\in[l,h_i]} L_{i,l}$. This correspondence only holds for vertices in the same distance to either "core" $L_{i,0}$, but for both graphs.

For the runtime, we look again at some fixed level l. Observe that the size of all color multisets combined is exactly $|L_{i,l}|$ and thus each vertex counts at most one time for the combined length of strings as input to the algorithm from lemma 2.24. The maximal color used for sorting (the original colors apart) is again $|L_{i,l}|$. To handle the original colors we could perform a global radix sort in each graph with the original colors and the smallest distance to a leave. This allows to use bogus colors in $[|L_{i,l-1}|]$ for the actual radix sort, but keep the original colors for the comparison in line 18. The loop starting in line 13 has to be implemented carefully for linear time: e.g. traverse $L_{i,l}$ and "push" the colors $c(u)$ ($u \in L_{i,l}$) to a list for each neighbor $v \in L_{i,l-1}$, instead of "pulling" them from v. □

While algorithm 2.3 does not compute a complete invariant if called with one instead of two input graphs, we have seen in the proof that the colors on each level are in some sense canonical. We can exploit this property to construct a linear time canonization algorithm for

trees in a quite obvious way. To achieve this goal, we add a second phase that works from the center to the leaves thereby propagating the canonical order of the vertices.

Algorithm 2.4. Canonical labeling of trees: ***treeCanonLab***

Input : Colored tree (T, c)
Output : Canonical labeling of (T, c)

```
1   //do not remove any vertices
2  (T,c)←removeLeavesAndRecolor((T,c))    //keep the levels L_l = L_{1,l} and h = h_1
3  choose bijection φ : L_0 → [|L_0|] such that ∀ u,v ∈ L_0 : φ(u)<φ(v) ⇒ c(u) ≤ c(v)
4  q ← |L_0|
5  p ← 0
6  for l ∈ 1 to h
7  |   r ← q
8  |   for i ∈ [p+1, q]
9  |   |   N_i ← N(φ^{-1}(i)) ∩ L_l
10 |   |   φ' : N_i → [r+1, r+|N_i|] such that ∀ u,v ∈ N_i : φ(u)<φ(v) ⇒ c(u) ≤ c(v)
11 |   |   φ ← φ ∪ φ'
12 |   |   r ← r + |N_i|
13 |   end
14 |   p ← q
15 |   q ← q + |L_l|
16 end
17 return φ
```

Corollary 2.26. *Algorithm 2.4 computes a canonical labeling for trees in linear time.*

Proof sketch. We already observed that the colors on each level correspond to the isomorphism type of the subtree rooted there and thus any order on equally colored vertices (and therefore subtrees) on each level produces the same graph if we ignore all levels closer to the center. Hence we may order the equally colored vertices on level l by the order of there sole neighbors in level $l-1$. Observe that we may exchange two vertices in the final labeling if and only if the tuple of colors on the path from the center is the same.

The runtime claim can be verified by seeing r,p and q as pointers and observing that we have to do bucket sorts with $|L_l|$ elements and integers in $|[L_{l-1}]|$ (assuming that L_l is already sorted according to c by ***removeLeavesAndRecolor***). □

2.4.3. Linear time isomorphism algorithm for colored cycles

The following lemma will be used in section 4.2.

Lemma 2.27 ([KS10])**.** *A graph isomorphism test for colored cycles (G_1, c_1) and (G_2, c_2) is doable in linear time.*

Proof. Assume both graphs are cycles of length k. Let $s_i : [k] \to V(G_i), i \in [2]$ be an arbitrary order on the vertices of each graph, such that $\{s_i(j), s_i(j+1)\} \in E(G_i)$ for $j \in [k-1]$. If G_1 and G_2 are isomorphic, the tuples of colors $t_i = (c_i(s_i(j)))_{j\in[k]}$ are identical up to reversal and

a cyclic shift. Using the Knuth-Morris-Pratt string-matching algorithm [KMP77] with t_1t_1 (2 concatenated copies of t_1) as the search string and t_2 and its reversed string t_2^R as patterns, we conclude that both graphs are isomorphic, if and only if t_2 or t_2^R is found in t_1t_1. This algorithm runs in time $\mathcal{O}(2(2|t_1| + |t_2|)) = \mathcal{O}(k)$ and dominates the runtime of the creation of the strings. □

2.4.4. Disconnected graphs

Some graph isomorphism algorithms assume that the input graphs are connected (e.g. in chapter 3). Two common approaches reduce Graph Isomorphism to Graph Isomorphism for connected graphs: taking the complement if the graphs are not connected or matching their components. Taking the complement is not a good idea in the parameterized world, as parameters like max deg are unbounded with respect to the original graph's parameter. Matching the components works, as long as the parameter of the components is bounded by some function of parameter of the entire graph.

Algorithm 2.5. Graph isomorphism for disconnected graphs: ***matchComponents***

Input : Graphs G_1 and G_2 and a function ***isoTest*** to call for each component
Output : Isomorphisms from G_1 to G_2 if one exists

```
1  φ ← ∅
2  for C ∈ {V | V is a connected component of G_1 }
3  |   for D ∈ {V | V is a not associated connected component of G_2 }
4  |   |   ψ ← isoTest(G_1[C], G_2[D])
5  |   |   if ψ ≠ "not isomorphic"
6  |   |   |   φ ← φ ∪ ψ
7  |   |   |   associate(C, D)
8  |   |   |   break
9  |   |   end
10 |   end
11 |   if C is not associated to any component
12 |   |   return "not isomorphic"
13 |   end
14 end
15 if ∃ not associated component D of G_2
16 |   return "not isomorphic"
17 else
18 |   return φ
19 end
```

Lemma 2.28. *For any graph parameter κ such that there is a computable function f with $\kappa(G[C]) \leq f(\kappa(G))$ for all connected components C of all graphs $G \in \mathcal{G}$, algorithm 2.5 provides an algorithm for the fpt Turing reduction*

$$\textsc{Graph Isomorphism}(\kappa) \leq^{\text{fpt-T}} \textsc{Connected Graph Isomorphism}(\kappa).$$

□

For canonical labelings the reduction is even easier, as we only have to sort the components by their size and their canonical form.

Lemma 2.29. *For any graph parameter κ such that there is a computable function f with $\kappa(G[C]) \leq f(\kappa(G))$ for all connected components C of all graphs $G \in \mathcal{G}$, we have*

$$\textsc{Canonical Labeling}(\kappa) \leq^{\text{fpt-T}} \textsc{Connected Canonical Labeling}(\kappa).$$

☑

Part II.

Parameterized problems in FPT

. . . watch the tree grow, . . .

3. Color multiplicity

Definition 3.1. Let (G, c) be a colored graph. Its color multiplicity $\mathrm{cm}(G, c)$ is defined as

$$\mathrm{cm}(G, c) = \max_{i \in \mathrm{rng}(c)} c^{-1}(i)\,,$$

i.e. the size of its biggest color class. •

The corresponding parameterized problems Colored Graph Isomorphism(cm) and Colored Canonical Labeling(cm) are the subject of this section and were tackled in the late 1970s and early eighties. Hence these first results concerning the fixed-parameter tractability of Graph Isomorphism and Canonical Labeling predate the introduction of fixed-parameter tractability by over a decade. The algorithms yielding the fixed-parameter tractability are of group-theoretic nature and work by a reduction to the colored graph automorphism problem, analogously to lemma 2.14. Clearly the color multiplicity is not increased by such a reduction.

Two papers make up the algorithm for the colored graph automorphism problem: László Babai's ***Monte Carlo Algorithms in Graph Isomorphism Testing*** [Bab79] introduces a probabilistic algorithm that constructs a tower of groups such that one of its level is the automorphism group and guesses coset representatives for the groups of the tower. In 1980 Furst, Hopcroft and Luks [FHL80] gave a deterministic algorithm to construct a set of coset representatives for towers of groups given a set of generators. In the rest of this section we will first discuss towers of groups in general, go into the algorithm in [FHL80] and explain the specific tower used by [Bab79] and how to construct a canonical labeling with this tower. Finally we will look at possible use cases of this result, by considering conditions which lead to bounded color class sizes.

3.1. Towers of groups

By ***coset*** we will refer to left cosets throughout this section, i.e. if H is a subgroup of G for every $g \in G$ the set $gH = \{gh \mid h \in H\}$ is a (left) coset of H in G. Briefly recall Lagrange's theorem and its proof:

Theorem 3.2 (Lagrange's theorem)**.** *For every finite group G and every subgroup H of G*

$$|H| \mid |G|, \quad \textit{particularly: } |G| = |G : H| \cdot |H|$$

Proof sketch. The cosets form a partition $\{C_i \mid 1 \leq i \leq k\}$ of G. Let $C_i = gH$ be a coset of H. Clearly $|gH| \leq |H|$ by the definition of gH. Assume $|gH| < |H|$, which means there are $h, h' \in H, h' \neq h$ such that $gh' = gh$. But then $h' = g^{-1}gh' = g^{-1}gh = h$. Thus $|C_i| = |H|$ for any i. Since $\sum_{i \in [k]} |C_i| = |G|$ we obtain the theorem. □

Lemma 3.3. *Let G be a group and H and G' subgroups of G, then*

$$|H : H \cap G'| \leq |G : G'| \quad .$$

Proof. Assume r and s are in different cosets of $H \cap G'$ in H. If r and s would be in the same coset of G' in G, then $r^{-1}s \in G'$ and since $r, s \in H$ $r^{-1}s \in G' \cap H$. This would contradict the assumption and thus the claim follows. □

The last part of proof of Lagrange's theorem now leads us to the notion of towers of groups. Observe that for a given coset C_i of a subgroup H in G and $a \in C_i$ the set $a^{-1}C_i$ equals H and thus $C_i = aH$. If we now fix a set R of coset representatives, every element of g can be written unambiguously as rh where $r \in R$ and $h \in H$. But why should we stop here? If K is a subgroup of H and S a set of coset representatives of K in H than $g = rsk$ where $s \in S$ and $k \in K$. This brings us to the concept of towers:

Definition 3.4 (Tower of groups)**.** For a group G with identity $\mathbb{1}$ a tuple of groups $(G_0, \ldots, G_k)$ is called a tower of groups for G, if

1. $G_0 = G$,
2. $G_k = \{\mathbb{1}\}$ and
3. $G_{i+1} \leq G_i$.

•

We will now state the observation above as a lemma:

Lemma 3.5. *Let $(G_0, \ldots, G_k)$ be a tower of groups and for $1 \leq i \leq k$ let R_i be a set of coset representatives for G_i in G_{i-1}. There is a unique representation $g = r_1 \ldots r_k$ ($r_i \in R_i$) for every $g \in G$.* □

Example 3.6. Consider the additive group of the quotient ring $\mathbb{Z}_{420}$. The subgroups $G_0 = \langle 2 \rangle, G_1 = \langle 10 \rangle, G_2 = \langle 30 \rangle, G_3 = \langle 210 \rangle, G_4 = \langle 0 \rangle$ form a tower of groups for $\langle 2 \rangle$, where $G_4 = \{0\}, G_3 = \{0, 210\}, G_2 = \{0, 30, 60, 90, 120, \ldots, 390, 420\}$. Choose a minimal ($\{0, 2, 4, 6, 8\}$ for G_1 in G_0) set of coset representatives for each $|G_i : G_{i+1}|$. 392 can now be written as $0 + 2 + 0 + 180 + 210$. e.g.

Example 3.7 ([FHL80])**.** Let G be a subgroup of Sym_n and $G_0 = G$. For $1 \leq i \leq n$ define

$$G_i = \{f \in \mathrm{Sym}_n \mid \exists (a_{i+1}, \ldots, a_n) : f(1, \ldots i, \ldots, n) = (1, \ldots, i, a_{i+1}, \ldots, a_n)\} \cap G \quad .$$

On each level choose some set of coset representatives R_i, e.g. choose the lexicographically minimal permutation from each coset. Choosing a representation for $g \in G$ now corresponds to mapping i to its image $g(i)$ for i from n to 1. e.g.

3.2. The sift-and-close-algorithm

3.2.1. sift …

The tower of groups approach may easily lead to algorithms, but there is still a fundamental question to solve: how to construct a set of coset representatives for each level of the tower? The algorithm in [FHL80] employs an idea also used in other two-phases algorithms like the

Knuth-Morris-Pratt algorithm [KMP77]: construct an auxiliary table in the same way you use it.

One use case of a tower of groups $(H_0, \ldots, H_k)$ for a subgroup H of G is to test for a given element of $g \in G$ whether it also lies in H and to compute a a coset representation of g. Given a set R_i of coset representatives for each level of the tower this is easy. Lemma 3.5 guarantees that if $g \in H$, then there is a unique representation $g = r_1 \ldots r_k$. To compute r_1 we only need to test which representative is equivalent to g ($r_1^{-1} g \in H_1$). If there is none, we know $g \notin H$. Otherwise $r_1^{-1} g = r_2 \ldots r_k$. Now recurse to compute $r_2 \ldots r_k$.

Note, that testing for $r_l^{-1} \ldots r_1^{-1} g \in H_l$ recursively does not yield a polynomial time algorithm, which is the reason why we need the ability to check whether $p \in H_l$ in polynomial time by some other means.

To ***construct*** a table of coset representatives in a similar way, we apply the procedure to elements $h = r_1 \ldots r_k$ of H itself. Since we know $h \in H$, a failure to find a coset representative equivalent to $r_{l-1}^{-1} \ldots r_1^{-1} h$ means that such a representative is missing. Using $r_{l-1}^{-1} \ldots r_1^{-1} h$ as this representative fills our gap. Thereby we define the representation of h as $r_1 \ldots r_l \mathbb{1}^{k-l}$. We summarize this procedure in algorithm 3.1 which we will call ***sift*** as in [FHL80]. The table T used in this algorithm is initialized by putting $\mathbb{1}$ into every of row of T.

Algorithm 3.1. The sift-algorithm [FHL80]

Input : Tower of groups $G_0, \ldots, G_k$, partially filled table of coset representatives T, $g \in G_0$
Output : Table T' with at most one additional element

```
1  for i ∈ [k] do
2  |   found ← false
3  |   for a ∈ T_i do
4  |   |   if (a^{-1} g ∈ G_i)
5  |   |   |   g ← a^{-1} g
6  |   |   |   found ← true
7  |   |   |   break
8  |   |   end
9  |   end
10 |   if (¬ found)
11 |   |   append(T_i, g)
12 |   |   break
13 |   end
14 end
```

Example 3.8. Continuing example 3.6, let us look at the following table, after some sifts. Each row is labeled with the corresponding index, which means in the end this row will have this number of entries.

$\lvert G_0 : G_1 \rvert$	0	168	72
$\lvert G_1 : G_2 \rvert$	0	350	
$\lvert G_2 : G_3 \rvert$	0	240	
$\lvert G_3 : G_4 \rvert$	0		

Again we consider $g = 392$: For $i = 1$, 392 is equivalent to 72 : $-72 + 392 = 320 = 32 \cdot 10$, so g becomes 320. Now 350 and 320 are equivalent, since their difference is 30 and $G_2 = \langle 30 \rangle$,

so $g \leftarrow 390(= -30)$. Finally 390 is inequivalent to both 0 and 240 since neither 390 nor 150 is a multiple of 210. So we add 390 to the third row.

$\lvert G_0 : G_1 \rvert$	0	168	72
$\lvert G_1 : G_2 \rvert$	0	350	
$\lvert G_2 : G_3 \rvert$	0	240	**390**
$\lvert G_3 : G_4 \rvert$	0		

e.g.

3.2.2. …and close

After the discussion of the sift algorithm one question naturally arises: which and how many group elements do we have to sift to get a complete table? There are two answers to this question: If we know all the indices $I_i = \lvert G_i : G_{i+1} \rvert$ or at least the order of G, we need to sift elements until $\lvert T_i \rvert = I_i$ and thus $\lvert G \rvert = \prod_{i \in [k]} \lvert T_i \rvert$.

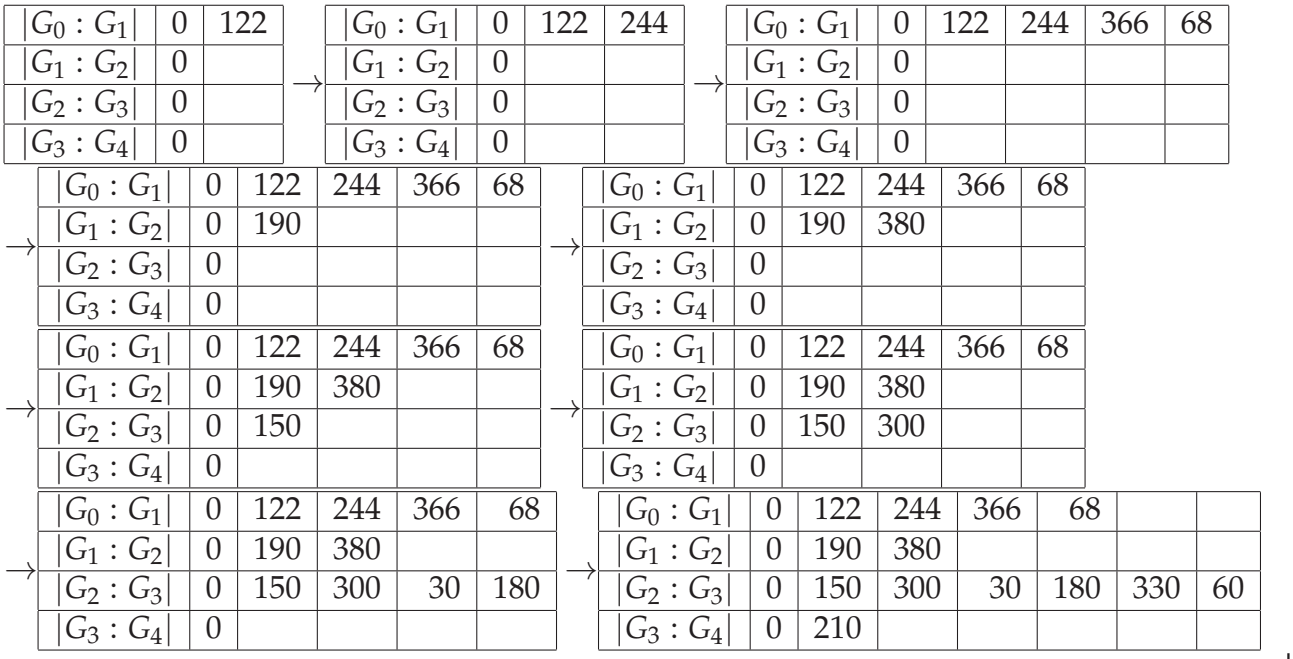
But we are far better off, if we know a set of generators $\langle g_1, \ldots, g_m \rangle$ for G. Since any element of G can be written as a product of generators and every generator has a unique representation after sifting all the generators, we may write any element of G as a product of coset representatives in T. This gives us a tuple $(r_1, \ldots, r_l)$ of coset representatives. Let $l(r) = i \Leftrightarrow r \in T_i$ be the level where r occurs. Then $(r_1, \ldots, r_l)$ is a representation in the sense of lemma 3.5 if and only if $(l(r_1), \ldots, l(r_l))$ is a strictly increasing tuple. Missing levels can be filled with $\mathbb{1}$ (which is in every level), so $l < k$ is no problem. If the level sequence is not strictly decreasing, there is a minimal l and a maximal i (depending on l) such that $l(r_{i-1}) \geq l(r_i) = l$ (a conflict on level l). Replacing their product $r_{i-1}r_i$ with its unique representation solves the problem, but for this purpose we need to sift all products rs with $r \in T, s \in T, l(r) \geq l(s)$. Observe that replacing $r_{i-1}r_i$ results in a product with one fewer conflict on level l. If we remove all conflicts on level l from right to left and continue on all levels $< l$, the product we obtain is in the form of lemma 3.5. Thus closing the table against the group operation is all we need to do (algorithm 3.2).

Algorithm 3.2. The close-algorithm [FHL80]

Input : Tower of groups $G_0, \ldots, G_k$, table of coset representatives T (generators sifted)
Output : complete table T of coset representatives

```
1 U ← T
2 while (U ≠ ∅)
3 |   T̃ ← T
4 |   for (r, s) ∈ U² ∪ U × T̃ ∪ T̃ × U
5 |   |   sift(T, rs)
6 |   end
7 |   U ← T \ T̃
8 end
9 return T
```

Example 3.9. For the last time we consider example 3.6. Since $G_0 = \langle 2 \rangle = \langle 122 \rangle$ sifting 122 and closing the table is all we need to do. The tables below show T before each iteration of the while loop in algorithm 3.2.



Algorithm 3.3. The sift-and-close-algorithm [FHL80]

Input : Tower of groups $G_0, \ldots, G_k$, with $G_0 = \langle g_1, \ldots, g_l \rangle$
Output : table T of coset representatives

```
1 T ← (T_1, ..., T_k), T_i ← ∅
2 for i ∈ [k]
3 |   append(T_i, 𝟙)
4 end
5 for i ∈ [l]
6 |   sift(T, g_i)
7 end
8 close(T)
9 return T
```

Theorem 3.10 ([FHL80]). *Let $(G_0, \ldots, G_k)$ be a tower of groups for a group $G = \langle g_1, \ldots, g_l \rangle$ and c be an upper bound for the number of steps needed to apply the group operation, to compute the inverse of a group element and to test whether an element is in G_i for $1 \leq i \leq k$. Then algorithm 3.3 computes a complete set of coset representatives in time $\mathcal{O}(c \cdot t^3)$ where $t = \sum_{i \in k} |G_{i-1} : G_i|$.*

Proof. It is easy to see that the sift procedure itself is correct, i.e. it creates a new entry if and only if its argument cannot be represented by the current table T. The sequence of as with positive test result in line 4 of algorithm 3.1 is a proper beginning of a unique representation. Thus we found a missing representative if and only if we cannot complete this representation up to level k.

While motivating the algorithm, we already stated that any representation by generators can be turned into a unique representation using coset representatives by representing the generators and rewriting pairs with nondecreasing levels from right to left. We replace each rs (with $l(s) \leq l(r)$) by the sequence $a_1, \ldots, a_k$ of as of the sift procedure (algorithm 3.1 line 4), actually omitting $a_1, \ldots, a_{l(s)-1}$. We can do this since for $r, s \in G_{l(s)-1} \Rightarrow rs \in G_{l(s)-1}$ and

thus $a_i = \mathbb{1}$ $(i \in [l(s) - 1])$. After this step the rightmost occurrence of a level $l(s)$ representative is at the position of r and we will eventually get a unique representation according to lemma 3.5. During the runtime analysis we will observe that all products of pairs of entries in the final table T have been sifted.

The final size of T is t, justified by lemma 3.5 and the order of G. To analyze the runtime, observe that in the close-phase (see algorithm 3.2), every element of the final Table T is at most once in the set U (each generator in line 1, every other product in line 7). So for every pair of elements (r, s) of the final T, either r is earlier in U than s or s earlier than r or both at the same time, so (r, s) is considered in exactly one iteration of the while loop and contained in exactly one set of line 4. Thus sift is called t^2 times. The sift procedure processes each element of the current T at most once, performing one inversion and one test of membership as well as one group operation, so its runtime is in $\mathcal{O}(3ct)$, which results in a total runtime of $\mathcal{O}(ct^3)$. □

3.3. Application to Graph Isomorphism(cm)

During this section we will look at Babai's work [Bab79] and discuss why the tower of groups used there is suitable to solve graph isomorphism for graphs of bounded color class size in polynomial time. This is to some extent anachronistic, but the conditions for applying Babai's probabilistic algorithm are nearly identical to those of the deterministic algorithm in [FHL80]. From now on (G, c) is a colored graph with $\mathrm{rng}(c) = [m]$, $C_i = c^{-1}(i)$ and $k = \mathrm{cm}(G, c)$.

3.3.1. Stabilizing the sets of equally colored edges

One way of characterizing automorphisms of a graph G is defining them as the set-wise stabilizers of the edgeset $E(G)$ regarding its action on $\binom{V(G)}{2}$, i.e. ϕ is an automorphism of G if and only if $\phi(E(G)) = E(G)$ where we interpret ϕ as a function from $\binom{V(G)}{2}$ to $\binom{V(G)}{2}$ such that

$$\phi(\{\{u_1, v_1\}, \ldots, \{u_k, v_k\}\}) = \{\{\phi(u_1), \phi(v_1)\}, \ldots, \{\phi(u_k), \phi(v_k)\}\} \quad .$$

Any vertex coloring $c : V(G) \to [m]$ defines an edge coloring $c' : E(G) \to \binom{[m]}{2}$ and because colored graph isomorphisms and automorphisms have to preserve c, they have to adhere to c', too. Using this we can again express automorphisms in another way: ϕ is an automorphism of G if and only if it stabilizes all edge sets of maximal subgraphs consisting of equally colored edges ($\phi(E_{i,j}) = E_{i,j}$, where $E_{i,j} = \{e \in E(G) \mid |e \cap C_i| = |e \cap C_j| \geq 1\}$).

Given these facts, we only miss one observation to build Babai's tower. Since we are interested in a tower for the automorphism group, we could easily get a set of coset representatives using the tower of example 3.7 and a set of generators of the automorphism group. But a set of generators is what we like to compute in the first place, so this approach seems circular. But a set of coset representatives for the tower $G_0, \ldots, G_l$ also includes coset representatives for all towers $G_i, \ldots, G_l$ for $i \in [l - 1]$. This enables us to put $\mathrm{Aut}(G, c)$ in the middle of a tower right above a tower like in example 3.7 (i.e. iteratively fixing vertices). Above $\mathrm{Aut}(G, c)$, we place a tower that "filters" all permutations by enforcing them to respect one additional set of equally colored edges $E_{i,j}$ in each level. But we do not even need to start with all

permutations, only those which stabilize all vertex color classes. Finding a set of generators for this set is easy: take 2 permutations for each color class C_i: a single cycle of length ($|C_i|$) and one transposition. Let us aggregate these thoughts into an example:

Example 3.11 (nearly the tower of [Bab79])**.** Let $G_0 = \bigotimes_{i\in[k]} \mathrm{Sym}(C_i)$ (seen as a subgroup of $\mathrm{Sym}(V(G))$), for $(i,j) \in [m]^2$:

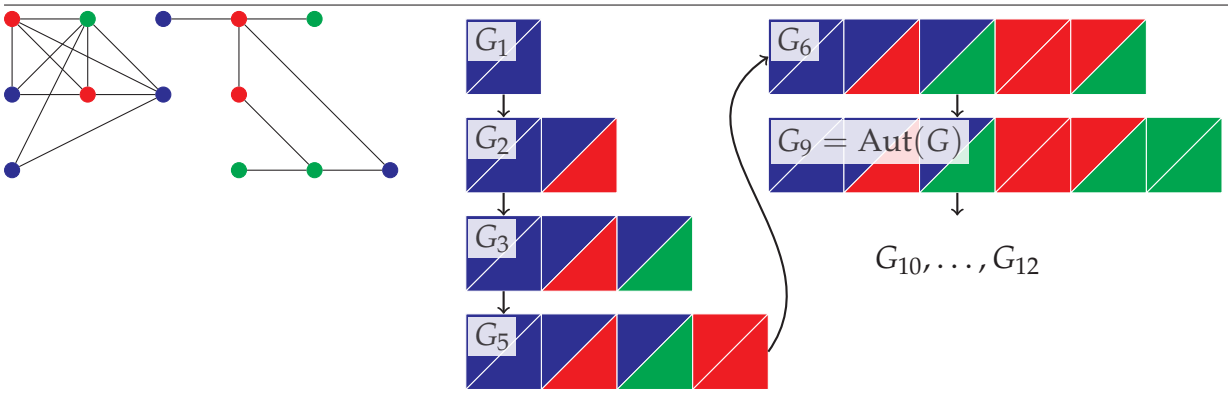
$$G_{m(i-1)+j} = G_{m(i-1)+j-1} \cap \{\phi \in \mathrm{Sym}(V(G)) \mid \phi'(E_{i,j}) = E_{i,j}\} \quad \text{and for } i \in [m]:$$

$$G_{m^2+i} = G_{m^2+i-1} \cap \{\phi \in \mathrm{Sym}(V(G)) \mid \forall v \in C_i : \phi(v) = v\} \quad .$$

e.g.

Note that this construction, as well as the final construction of [Bab79], contains useless steps (for undirected graphs), as $E_{i,j}$ and $E_{j,i}$ are treated separately. This eases the calculation of indices, but any real implementation for undirected graphs should skip the groups that are no proper subgroups (see figure 3.1, e.g. groups G_7 and G_8 are equal to G_6 and therefore left out).

Figure 3.1. A graph and a corresponding tower of groups (construction of example 3.11)



To apply algorithm 3.3 as a FPT-algorithm, we need to construct a tower with small indices, because their sum dominates the runtime. Since there are only at most k vertices in each color class, there are no more than $(k!)^2$ permutation on each $C_i \cup C_j$ which respect the colors. Furthermore, for any permutation ϕ in $G_{i-1} (i \in [m^2])$, the coset of G_i in G_{i-1} into which ϕ falls, is solely determined by its behavior on $C_i \cup C_j$ and thus $|G_{i-1} : G_i| \leq (k!)^2$. Obviously $|G_{i-1} : G_i| \leq k!$ for $i \in [m^2 + 1, m^2 + m]$, because there are at most $k!$ ways of setwise stabilizing a set of at most k elements. If our only goal is to get a FPT-algorithm, we are done. However $|G_0| = |\bigotimes_{i\in[k]} \mathrm{Sym}(C_i)| \leq (k!)^m$, so our upper bound leaves room for improvement. In [Bab79] Babai therefore inserted groups between each pair G_{i-1}, G_i which perform some kind of "lookahead":

Definition 3.12 (tower of [Bab79])**.** Let $H_0 = G_0$, $H_{2i} = G_i$ for $i \in [m^2]$ and $H_{2m^2+i} = G_{m^2+i}$ for $i \in [m]$ (each G_i as in example 3.11). Now define for $(i,j) \in [m]^2$:

$$H_{2(m(i-1)+j)-1} = \{\phi \in H_{2(m(i-1)+j)-2} \mid \exists \psi \in H_{2(m(i-1)+j)} \forall v \in V(G) \setminus C_i : \phi(v) = \psi(v)\}$$

•

Lemma 3.13. *Let $H_1, \ldots, H_{2m^2+m}$ be the tower of definition 3.12 and $k = \max_{i\in[k]} |C_i|$:*

$$|H_{2(m(i-1)+j)-2} : H_{2(m(i-1)+j-1)}| \leq k! \text{ and } |H_{2(m(i-1)+j)-1} : H_{2(m(i-1)+j)}| \leq k!$$

Proof. Lemma 3.3 allows us to restrict our attention to the groups $Q = \mathrm{Sym}(C_i) \otimes \mathrm{Sym}(C_j)$ (seen as a subgroup of $\mathrm{Sym}(C_i \cup C_j)$), $S = \{\phi \in Q \mid \phi(E_{i,j}) = E_{i,j}\}$ and $R = \{\phi \in Q \mid \exists \psi \in S : \phi|_{C_j} = \psi|_{C_j}\}$. Let $\psi, \phi \in Q$ and $\phi|_{C_j} = \phi|_{C_j}$. Then $\psi^{-1}\phi \in R$ and thus $|Q : R| \leq k!$ because $|\mathrm{Sym}(C_j)| \leq k!$. Take an arbitrary ϕ of a coset D of S in R. By the definition of R there is a $\psi \in S : \phi|_{C_j} = \phi|_{C_j}$. We can write D as $D = \phi\psi^{-1}S$ and $(\phi\psi^{-1})\big|_{C_j} = \mathbb{1}_{\mathrm{Sym}(C_j)}$, so there are at most $|\mathrm{Sym}(C_i)|$ many cosets of S in R, which yields the second part of the lemma. □

3.3.2. Algorithm, its correctness and runtime

Before stating the algorithm we only need to clarify how the basic operations in the sift procedure are performed. Computing the group operation (composition) and inverting a permutation is easy. This leaves us with the test for membership in algorithm 3.1, line 4. There are 3 kinds of this test: for any $(i, j) \in [m]^2$ simply check whether a given $\phi \in H_{2(m(i-1)+j)-1}$ stabilizes $E_{i,j}$ and is thus in $H_{2(m(i-1)+j)}$ (time complexity: $\mathcal{O}(k^2)$). For $\psi \in H_{2(m(i-1)+j)-2}$ we do the lookahead, by using a precomputed table of allowed restrictions to C_j. The complexity of this a lookup depends on the machine model: it is polynomial in k on a random access machine by e.g. using the factorial number system or base k encoding ($k \log k$ bits for each restriction) to compute the number of a register. On a Turing machine the time complexity would be $\mathcal{O}(k!)$. Checking whether a permutation fixes a color class pointwise is trivial and we conclude that the c from theorem 3.10 is $c = \mathcal{O}(k^2)$.

Algorithm 3.4. Graph isomorphism for graphs of bounded color class sizes.

Input : Colored Graphs (G_1, c_1) and (G_2, c_2) with color classes $C_i (1 \leq i \leq m)$
Output : Are (G_1, c_1) and (G_2, c_2) isomorphic?

```
1  if (G_1 is not connected)
2  |   G_1 ← complement(G_1)
3  |   G_2 ← complement(G_2)
4  (G, c) ← disjoint union of (G_1, c_1) and (G_2, c_2)
5  R ← set of generators for ⊗_{i∈[k]} Sym(C_i)
6  precompute the lookup tables for odd i in [m^2]
7  T ← (T_1, ..., T_{2m^2+m}), T_i ← ∅
8  for i ∈ [2m^2 + m]
9  |   append(T_i, 𝟙)
10 end
11 for g ∈ R
12 |   sift(T, g)
13 end
14 close(T)
15 for i ∈ [2m^2 + 1, 2m^2 + m]
16 |   for p ∈ T_i
17 |   |   if (p switches components)
18 |   |   |   return "isomorphic"
19 |   |   end
20 |   end
21 end
22 return "not isomorphic"
```

Theorem 3.14. *Algorithm 3.4 correctly decides whether 2 colored graphs (G_1, c_1) and (G_2, c_2) with color class sizes at most k are isomorphic in time $\mathcal{O}((k!)^3 k^2 |V(G_1)|^6)$ and hence* Colored Graph Isomorphism(cm) $\in$ FPT.

Proof. Group H_{2m^2+2} really is the automorphism group of (G, c) (line 4), because it is the intersection of stabilizers of the sets of equally colored edges. Since every element of a group can be uniquely represented by coset representatives (lemma 3.5) a complete table of coset representatives is a set of generators. So the loop starting in line 15 checks for every generator of a set of generators of the automorphism group whether it switches the component and thus whether any automorphism does this. From here correctness follows from the reduction of Graph Isomorphism to Graph Automorphism.

Precomputing a lookup-table for the lookahead groups takes time $\mathcal{O}(|V(G)|^2 (k!)^2)$. The sift-and-close algorithm with an upper bound on the basic operations in the sift-procedure of $\mathcal{O}(k^2)$ and a table size of at most $\mathcal{O}(|V(G)|^2 k!)$ takes times $\mathcal{O}((k!)^3 k^2 |V(G)|^6)$. This dominates all other part of the algorithm including the loop starting in line 15. This loop iterates over at most $\mathcal{O}(nk!)$ elements and each iteration takes time $\mathcal{O}(|V(G)|)$. □

3.4. Application to Canonical Labeling(cm)

In [Bab79] Babai asked for canonical labeling procedure based on his approach. This demand was soon fulfilled by [KL81]. As this preprint is not easily available today and the following articles (especially [BL83]) only describe more involved approaches, we refer to a talk by Luks [Luk10, 11–26]. Again (G, c) shall a colored graph with $\mathrm{rng}(c) = [m]$, $C_i = c^{-1}(i)$ and $k = \mathrm{cm}(G, c)$ throughout this section.

The central idea of algorithm 3.5 is to compute a lexicographically minimal adjacency string whose entries are sorted according to the pairs of colors of the incident vertices. This special sort order enables us to find this minimal string iteratively by first treating only the edges whose vertices both have the color 1, then additionally considering those with vertex colors $\{1, 2\}$ and so on. The tower of groups in algorithm 3.4 contains all the information we need for this undertaking, because all the coset representative in the table created from the tower precisely describe the possible actions on the edge sets with certain colors that stabilize the color classes.

Algorithm 3.5. Canonical labeling for graphs of bounded color class sizes.

Input : Colored Graph $(G, c) \in \mathcal{G}_c$ with color classes $C_i (1 \leq i \leq m)$
Output : A canonical labeling of (G, c)

```
1 sort (rename) all u, v ∈ V(G) such that c(u) < c(v) ⇒ u < v
2 T ← as in algorithm 3.4
3 t←𝟙
4 for i ∈ [m²]
5 |   choose r ∈ T_{2i−1}, s ∈ T_{2i} with minimal adj(trs(V(G)), trs(E_i(G)))
  |   |   where E'_i(G)←⋃_{h∈[i]} E_{x_h,y_h}(G) such that h←(x_h − 1)m + y_h
  |   |   and E_{x_h,y_h}(G)←{{u, v} ∈ E(G) | c(u)←x_h, c(v)←y_h}
6 |   t ← trs
7 end
8 return t
```

Theorem 3.15. *Algorithm 3.5 computes a canonical labeling for a colored graph $(G, c) \in \mathcal{G}_c$ with $\mathrm{cm}(G, c) = k$ in time $\mathcal{O}((k!)^3 k^2 |V(G)|^6)$ and therefore* Colored Canonical Labeling(cm) $\in$ FPT.

Proof. The runtime directly follows from theorem 3.14 as the sift-and-close-algorithm clearly dominates the additional loops. To see, why the computed labeling t is canonical, we analyze the algorithm for two isomorphic colored input graphs (G_1, c_1) and (G_2, c_2) in $\mathcal{G}_c$. By the definition of $\mathcal{G}_c$, we may set $V = V(G_1) = V(G_2) = [|V(G_1)|]$ and use it as the vertex set of both graphs. Let ϕ be the isomorphism between both graphs after sorting the vertices of each graph by their color. After sorting, the pairs of colors (x, y) corresponding to the ith level of each table T are equal for both graphs. For each graph G_j set $t_{j,0} = \mathbb{1}$, let $t_{j,i}$ be the value of t after the ith iteration of the for-loop starting in line 4 and set $E_{j,x,y} = Ex, y(G_j)$ for all pairs of colors (x, y) and $E'_{j,i} = E'_i(G_j)$ using the definition in the algorithm. Further let x_i and y_i be the colors such that $i = (x_i - 1)m + y_i$. The choice of two entries r and s in the ith round corresponds to the choice of a single coset representative $r_{j,i}$ in the simplified tower of example 3.11 and thus we will not discuss groups on odd levels explicitly. We will now

prove the following fact by induction on i:

$$\mathrm{adj}(t_{1,i}(V), t_{1,i}(E'_{1,i})) = \mathrm{adj}(t_{2,i}(V), t_{2,i}(E'_{2,i}))$$

For $i = 0$ both graphs in the claim become edgeless and since the vertices are already sorted according to colors in both graphs, their adjacency matrices are equal.

Assume $i > 0$ and thus the claim holds for all $h < i$. The coset membership on levels $2i - 1$ and $2i$ of each tower of groups is solely determined by the action on E_{j,x_i,y_i} (for $j \in [2]$). Especially, each $r_{j,i}$ is a member of the group $G_{j,2i-2}$ in each tower and thus stabilizes each $E'_{j,i-1}$. Hence we have $t_{j,i}(E'_{j,i-1}) = t_{j,i-1}(E'_{j,i-1})$ and thus

$$\mathrm{adj}(t_{1,i}(V), t_{1,i}(E'_{1,i-1})) = \mathrm{adj}(t_{2,i}(V), t_{2,i}(E'_{2,i-1}))\,.$$

This means that the algorithm had to choose each $r_{j,i}$ such that

$$\mathrm{adj}(r_{1,i}(V), r_{1,i}(E_{1,x_i,y_i})) = \mathrm{adj}(r_{2,i}(V), r_{2,i}(E_{2,x_i,y_i}))\,,$$

because its choice is not disturbed by earlier differences and $t_{2,i}\phi$ has unique representation in the table for G_1 and vice versa for $t_{1,i}\phi^{-1}$. Combining the last two equalities yields the desired claim and completes our proof. □

3.5. Consequences

For an example of an application for this theorem, see section 6.1, where we discuss the ***path distance width***.

Corollary 3.16. *Let $C : \mathcal{G} \to \{C \subseteq [n]^{[m]} \mid m, n \in \mathbb{N}, m \leq n\}$ a polynomial time computable function (in $|V(G)| + |E(G)|$) that assigns to every graph G a set of colorings $C(G)$ such that for any graph $H \in \mathcal{G}$ and any isomorphism $\phi : G \to H$ there is a bijection $b : C(G) \to C(H)$ such that for all $c \in C(G)$ ϕ is an isomorphism from (G, c) to $(H, b(c))$. Further let κ be a graph parameter that bounds* $\mathrm{mmcol}(C(G)) = \min_{c \in C(G)} \max_{i \in \mathrm{rng}(c)} |c^{-1}(i)|$. *Then* Canonical Labeling *is fixed-parameter tractable w.r.t. κ.*

Proof. Since C is computable in polynomial time, $C(G)$ has always polynomial size. Compute $C'(G) = \{c \in C(G) \mid \max_{i \in \mathrm{rng}(c)} |c^{-1}(i)| = \mathrm{mmcol}(C(G))\}$. For all $c \in C'(G)$ compute a canonical labeling for (G, c) according to algorithm 3.5 and choose a labeling l among these labelings that minimizes $adj(l(V(G)), l(E(G)))$. This can be done in time $\mathcal{O}(|C'(G)| f(\kappa(G)) |V(G)|^6)$ (for some f). It is easy to see that this labeling is canonical, as it is canonical for one of the colored graphs and the bijection b ensures that the set of canonized colored graphs is an invariant. □

Note that we only need to compute C' and thus C doesn't even need to be computable in polynomial time as long as C' is.

Example 3.17. Let $(G, c) \in \mathcal{G}_c$ be a colored graph. For a fixed $e \in \mathbb{N}$ and any $v = (v_1, \ldots, v_e) \in V(G)^e$ define $c_v(w) = (c(w), i)$ where $i = \max\{j \mid j = 0 \vee v_j = w\}$. For a fixed d run the d-dimensional Weisfeiler-Lehman algorithm with input (G, c_v) and flatten the colors to a vertex coloring f_v (algorithm 2.2). Now define

$$\mathrm{wlcm}_{d,e}(G, c) = \min_{v \in V(G)^e} \mathrm{cm}(G, f_v)\,.$$

e.g.

Corollary 3.18. *For any $d, e \in \mathbb{N}$ Colored Canonical Labeling$(\mathrm{wlcm}_{d,e})$ is fixed-parameter tractable.*

Proof. By individualizing every e-tuple (and not only every set of e elements) we ensure that the set of colorings does not depend on any choice of order. For any d, e there are only $|V(G)|^e$ colorings, which can be computed in polynomial time for a fixed d. Finally, we apply corollary 3.16. This can be done since the colored isomorphsim type of the flattened d-dimensional Weisfeiler-Lehman algorithm is an invariant (corollary 2.22) and thus yields the required bijection of colorings (corollary 3.16). □

4. Modification sets

Graph Isomorphism and many NP-complete graph problems are solvable in polynomial time for certain restricted graph classes $\mathcal{C}$. However, there is not always an inherent parameter to generalize such a result and turn it into a FPT-algorithm. For such problems and graph classes it might be interesting to consider sequences of modifications which turn an arbitrary graph G into a graph $G' \in \mathcal{C}$. By a series of modifications, we mean a tuple $(m_1, \ldots, m_k)$, $m_i \in \binom{V(G)}{2} \cup V(G)$, i.e. a tuple of operations of the form "delete vertex v", "add edge e" or "delete edge e". If we start with $G_0 = G$ and define for

$$i \in [k] : G_i = \begin{cases} G_{i-1} - m_i & m_i \in E(G_i) \cup V(G_i) \\ G_{i-1} + m_i & m_i \in E(\overline{G_i}) \end{cases},$$

we obtain $G' = G_k$, where all the modifications have been carried out in the specified order. Provided that all modifications are defined, e.g. we do not remove an edge after one of its vertices has been removed, the order of modifications is irrelevant. Furthermore additions and deletions of the same edge cancel each other and thus the effect of a series of modifications can be subsumed by a set of modifications:

Definition 4.1 (modification sets, modification numbers)**.** Let $\mathcal{C}$ be a class of graphs and G an arbitrary graph. A set M is a $\mathcal{C}$ ***modification set*** for G if there is a modification sequence $m = (m_1, \ldots, m_{|M|})$ from G to a graph $G' \in \mathcal{C}$ such that $M = \{m_i \mid i \in [|M|]\}$. We call such an M a $\mathcal{C}$ ***vertex deletion set*** if $M \subseteq V(G)$, a $\mathcal{C}$ ***edge deletion set*** if $M \subseteq E(G)$ and a $\mathcal{C}$ ***edge addition set*** if $M \subseteq E(\overline{G})$. The $\mathcal{C}$ ***modification number of*** G ($\mathcal{C}$-mn(G)) is the smallest k, such that there is a $\mathcal{C}$ modification set for G of size k. Analogously, we define the $\mathcal{C}$ ***vertex deletion number*** of G, the $\mathcal{C}$ ***edge deletion number*** of G and the the $\mathcal{C}$ ***edge addition number*** of G. •

Such a definition naturally raises the question, how to test whether there is a modification set of some size k. Thus, we consider the following parameterized problem:

		$\mathcal{C}$ Modification Set
Input	:	(G, k), where G is a graph and $k \in \mathbb{N}$
Parameter	:	k
Question	:	Is there a $\mathcal{C}$ modification set of G of size at most k?

Again, the similar problems $\mathcal{C}$ Vertex Deletion Set, $\mathcal{C}$ Edge Deletion Set and $\mathcal{C}$ Edge Addition Set can be defined in the same manner.

4.1. Finite set of forbidden induced subgraphs

[Cai96] presents a FPT-algorithm for $\mathcal{C}$ Modification Set and the related problems subject to the condition that there is a finite set $\mathcal{F}$ of graphs such that

$$\mathcal{C} = \{\text{graphs } G \mid \forall F \in \mathcal{F}\, \forall V' \subseteq V(G) : G[V'] \ncong \mathcal{F}\} \quad .$$

We call $\mathcal{F}$ the set of forbidden induced subgraphs of $\mathcal{C}$ and $\mathcal{C}$ the class of $\mathcal{F}$-free graphs ($\mathcal{C} = \mathcal{F}$-free).

Algorithm 4.1. $\mathcal{C}$ modification set, for $\mathcal{C} = \mathcal{F}$-free [Cai96]: ***modSet***

Input : Graph G and $k \in \mathbb{N}$
Output : A $\mathcal{C}$ modification set of G of size at most k or "none"

```
1  if G ∈ C
2  |    return ∅
3  end
4  F ← minForbSubgraph(G)
5  for m ∈ V(F) ∪ E(F) ∪ E(F̄)
6  |    if (m ∈ E(F̄))
7  |    |    G' ← G + m
8  |    else
9  |    |    G' ← G − m
10 |    end
11 |    if (modSet(G', k − 1) ≠ "none")
12 |    |    return ({m} △ modSet(G', k − 1))
13 |    end
14 end
15 return "none"
```

Cai's algorithm (algorithm 4.1) requires the construction of a minimal forbidden induced subgraph (the call to ***minForbSubgraph***). For the moment we state that this can be done in time $\mathcal{O}(|V(G)|^{d+1})$, $d = \max_{F \in \mathcal{F}} |V(F)|$ and look at an algorithm which achieves this bound (also from [Cai96]) after the following theorem.

Theorem 4.2 ([Cai96])**.** *Algorithm 4.1 finds a $\mathcal{F}$-free modification set (if one exists) of a given graph G of size at most k in time $\mathcal{O}(d^{2k}t)$, $d = \max_{F \in \mathcal{F}} |V(F)|$, where t is an upper runtime bound on the runtime to compute a minimal forbidden induced subgraph of a graph H with $|V(H)| \leq |V(G)|$.*

Proof. Any set returned by algorithm 4.1 is a modification set of size at most k, which can be shown inductively. Note that the symmetric difference in line 12 only cancels out additions and deletions of the same edge and therefore never increases the returned set.

For the other direction assume M is a modification set of size at most k. Then for any forbidden induced subgraph F of G there is an $m \in M$ which is a vertex, an edge or an edge of the complement of F (otherwise F would still be present in the modified G). Since algorithm 4.1 considers every way to modify one forbidden subgraph, it will eventually find M, if it does not return another modification set of size at most k.

All recursive calls to algorithm 4.1 (modSet) use $k - 1$ as the second parameter and because $|V(F) \cup E(F) \cup E(\overline{F})| \leq d^2$ holds, the entire call tree contains no more than d^{2k} calls to modSet. Each invocation of modSet only constructs a single minimal forbidden subgraph, which

dominates the runtime. All other operations can be implemented in constant time, thus the claimed runtime follows. □

Similar results follow for $\mathcal{C}$ Vertex Deletion Set (runtime only $\mathcal{O}(d^k t)$), $\mathcal{C}$ Edge Deletion Set and $\mathcal{C}$ Edge Addition Set. Theorem 4.2 with $\mathcal{F} = \{K_2\}$ ($K_2 = ([2], \{[2]\})$) also completes the proof for lemma 1.11 (Vertex Cover $\in$ FPT).

4.1.1. Find a minimal forbidden induced subgraph

As promised, we now consider the algorithm to find a minimal forbidden subgraph (algorithm 4.2). Note that this algorithm does not require the set of forbidden induced subgraphs $\mathcal{F}$ to be finite as long as testing for $G \in \mathcal{F}$-free can be done efficiently, so it can be used for e.g. chordal graphs and forests, too. The algorithm is built on the observation that if the removal of a vertex v does not make G $\mathcal{F}$-free, then $G - v$ contains a forbidden induced subgraph that is not bigger than the smallest forbidden induced subgraph of G.

Algorithm 4.2. Minimal forbidden induced subgraph [Cai96]: ***minForbSubgraph***

Input : Graph G and $k \in \mathbb{N}$
Output : A minimal forbidden induced subgraph F of G

```
1  if G ∈ C
2  |    return "none"
3  end
4  F ← G
5  for v ∈ V(G)
6  |    if (F − v ∉ C)
7  |    |    F ← F − v
8  |    end
9  end
10 return F
```

Lemma 4.3 ([Cai96])**.** *For any class $\mathcal{C} = \mathcal{F}$-free, algorithm 4.2 finds a minimal forbidden subgraph of a graph $G \notin \mathcal{C}$ in time $\mathcal{O}(|V(G)|t)$, where t is an upper runtime bound for the test $H \in \mathcal{F}$-free over all graphs $H : |V(H)| \leq |V(G)|$. If $\mathcal{F}$ is finite the runtime is bounded by $\mathcal{O}(|V(G)|^{d+1})$, where $d = \max_{F \in \mathcal{F}} |V(F)|$.*

Proof. Let F_{res} be the graph returned by algorithm 4.2. We will at first show that $F_{\text{res}} \notin \mathcal{C}$. If the condition in line 6 is never fulfilled, then $G = F_{\text{res}}$ and $G \notin \mathcal{C}$ is tested at the beginning. In any other case at least one vertex v was removed from F and we consider the last time this happened. Let F' be the F before the removal of v, so $F_{\text{res}} = F' - v$. Then v wouldn't have been removed if $F' - v = F_{\text{res}}$ would have been in $\mathcal{C}$ and thus $F_{\text{res}} \notin \mathcal{C}$.

To show that F_{res} is minimal, assume that it is not. This means there is a vertex $v \in V(F_{\text{res}})$, such that $F_{\text{res}} - v \notin \mathcal{C}$. Let F' be the F at the beginning of the iteration of the for-loop when v was considered. We know that $F' - v \in \mathcal{C}$, otherwise v would have been removed. But $F_{\text{res}} - v$ is an induced subgraph of $F' - v$ and any induced subgraph of $F_{\text{res}} - v$ is an induced subgraph of $F' - v$, too. Thus $F_{\text{res}} - v \in \mathcal{C}$, which contradicts the assumption.

Everything except the for-loop (which iterates $V(G)$ times) and the membership test takes constant time, so the total runtime is in $\mathcal{O}(|V(G)|t)$. If $\mathcal{F}$ is finite, any test $G \in \mathcal{C}$ is doable in time $t = \mathcal{O}(|V(G)|^d)$, $d = \max_{F\in\mathcal{F}} |V(F)|$. Simply run d nested loops that iterate over $V(G)$ and for each $H \in \mathcal{F}$, $|V(H)| = c$ test whether the graph induced by any c-tuple of vertices equals H (isomorphism that respects order of vertices). We do not need a real isomorphism test at his level since we implicitly iterate over all automorphisms of d-elementary subgraphs of G as well, by considering every d-tuple. □

4.1.2. Application to Graph Isomorphism

During this section we study an algorithm for Graph Isomorphism parameterized by the $\mathcal{F}$-free vertex deletion number as presented by Kratsch and Schweitzer in [KS10]. Notwithstanding this, we will consider a slightly modified version of this algorithm, that deals with modification sets instead of vertex deletion sets. This is somewhat more general and results for the subtypes of modification sets (including vertex deletion sets) follow easily.

One observation in the previous proof (lemma 4.3) already yields the core building block of algorithm 4.3: for every (minimal) forbidden induced subgraph F and every modification set M of a graph G:

$$|V(F) \cap V(M)| \geq 1 \text{ or } |E(F) \cap E(M)| \geq 1 \text{ or } |E(\overline{F}) \cap \overline{E}(M)| \geq 1 \quad ,$$
$$\text{where } V(M) = M \cap V(G), E(M) = M \cap E(G), \overline{E}(M) = M \cap E(\overline{G}) \quad .$$

These definitions depend on G and so we will always use them in a context where G is clear.

Luckily, if ϕ is an isomorphism from G_1 to G_2 and M is a $\mathcal{C}$ modification set of G_1, then ϕ maps M to a modification set M' of G_2 such that $\phi(V(M)) = V(M'), \forall e \in E(M) : \phi(e) \in E(M')$ and $\forall e \in \overline{E}(M) : \phi(e) \in \overline{E}(M')$. Thus, if F is a forbidden induced subgraph of G_2, then for every $m \in V(M) : \phi(m) \in V(F) \cup E(F) \cup \overline{E}(F)$. Now, an algorithm for Graph Isomorphism can fix a pair (m_1, m_2) in $M \times (V(F) \cup E(F) \cup \overline{E}(F))$, apply the pair to both graphs, recurse until the graph is $\mathcal{F}$-free and use a Graph Isomorphism algorithm for $\mathcal{F}$-free graphs. Finally it builds up the isomorphism during the recursive ascent.

However, if m_1 and m_2 are vertices, an isomorphism ψ from $G_1 - m_1$ to $G_2 - m_2$ may map neighbors of m_1 to nonneighbors of m_2. This is solved in [KS10] by a common technique: they encode (non-)adjacency as color. Because of the recursiveness of the algorithm, we need to handle colors as well and pass colored graphs to the Graph Isomorphism algorithm for $\mathcal{F}$-free graphs. So we need to encode old colors along with the adjacency information and define for any coloring $c : V(G) \to [l]$ and any $v \in V(G)$:

$$(c - v) : \text{dom}(c) \setminus \{v\} \to \text{rng}(c), \quad (c - v)(u) = \begin{cases} (c(u), 1) & \{u, v\} \in E(G) \\ (c(u), 0) & \{u, v\} \notin E(G) \end{cases} .$$

Again this definition depends on G. For the case where m_1 and m_2 are edges or edges of the complement, we only need to remember, where we changed the adjacency. Thus we define

$$(c \pm e) : \text{dom}(c) \to \text{rng}(c), \quad (c \pm e)(u) = \begin{cases} (c(u), 1) & u \in e \\ (c(u), 0) & u \notin e \end{cases} .$$

These thoughts lead to algorithm 4.3 and the following theorem about it:

Algorithm 4.3. Graph isomorphism for graphs with bounded $\mathcal{F}$-free modification sets [KS10]: ***graphIsoModSet***

Input : Colored graphs (G_1, c_1) and (G_2, c_2) and $k \in \mathbb{N}$
Output : Isomorphism ϕ from G_1 to G_2 or `null`

```
1  if (G1 ∈ C ∧ G2 ∈ C)
2  |    return colIsoClass(G1, G2)
3  if (G1 ∈ C ∨ G2 ∈ C)
4  |    return null
5  if (k=0)
6  |    return null
7  M ← modSet(G1, k)
8  if (M = "none")
9  |    return null
10 end
11 F ← minForbSubgraph(G2)
12 for (v1, v2) ∈ V(M) × V(F)
13 |    if (c1(v1)=c2(v2))
14 |    |    φ ← graphIsoModSet((G1 − v1, c1 − v1), (G2 − v2, c2 − v2), k − 1)
15 |    |    if (φ ≠ null )
16 |    |    |    return φ ∪ {(v1, v2)}
17 end
18 for (e1, e2) ∈ (E(M) × E(F)) ∪ (Ē(M) × E(F̄))
19 |    if (c1(e1)=c2(e2))
20 |    |    φ ← graphIsoModSet((G1 ± e1, c1± e1), (G2 ± e2, c2± e2), k − 1)
21 |    |    if (φ ≠ null )
22 |    |    |    return φ
23 end
24 return null
```

Theorem 4.4 ([KS10]). *Let $\mathcal{F}$ be a finite set of graphs and **colIsoClass** an $\mathcal{O}(|V(G)|^c)$-time algorithm that computes an isomorphism for $\mathcal{F}$-free graphs, if one exists and returns `null` otherwise. Then algorithm 4.3 correctly computes for two colored graphs (G_1, c_1) and (G_2, c_2) and $k \in \mathbb{N}$ whether (G_1, c_1) and (G_2, c_2) are isomorphic and have a $\mathcal{F}$-free modification set of size at most k in time $\mathcal{O}(k! d^{2k} |V(G)|^{d+1})$, where $d = \max(\{|V(F)| \mid F \in \mathcal{F}\} \cup \{c\}))$. Hence Graph Isomorphism parameterized by the $\mathcal{F}$-free modification number is fixed-parameter tractable. If the graphs are isomorphic the algorithm returns an isomorphism.*

Proof. The proof works by induction on the argument k. If $k = 0$ and either $G_1 \notin \mathcal{F}$-free or $G_2 \notin \mathcal{F}$-free, then algorithm 4.3 returns `null` in line 4 or 6. If both are $\mathcal{F}$-free the correctness follows from that of ***colIsoClass*** and the runtime is $\mathcal{O}(|V(G)|^c)$.

For $k > 0$ line 9 returns `null` if and only if the modification number of G_1 is greater than k (theorem 4.2) and a modification set can be computed in time $\mathcal{O}(d^{2k}|V(G)|^{d+1})$. This dominates the computation of the minimal forbidden induced subgraph F (lemma 4.3).

Both loops after line 11 combined iterate over all $m \in M$ and thus for one such m and and every isomorphism ψ, $\psi(m)$ is in either $V(F), E(F)$ or $E(\overline{F})$ if G_1 and G_2 are isomorphic. Thus there is an isomorphism if and only if there is an isomorphism that fixes one of the

pairs considered in one of the for-loops and so returning `null` otherwise is correct. Observe that $|V(F)| + |E(F)| + |E(\overline{F})| \leq d^2$ and thus both loops together have at most kd^2 iterations. As discussed earlier, recoloring correctly encodes the adjacency information for a vertex and individualizing one edge (of the complement) per graph ensures that an isomorphism maps one edge to the other. Because of this and the inductive hypothesis each returned ϕ (or $\phi \cup \{(v_1, v_2)\}$ resp.) is an isomorphism from (G_1, c_1) to (G_2, c_2). The inductive hypothesis also yields a runtime bound for every pass of each for-loop: $\mathcal{O}((k-1)!d^{4(k-1)}|V(G)|^{d+1})$. As this clearly dominates $\mathcal{O}(d^{2k}|V(G)|^{d+1})$ the overall runtime follows. □

4.1.3. Classes with forbidden subgraphs

As well as for forbidden induced subgraphs we might consider a class with forbidden subgraphs, regardless whether they are induced or not. It is easy to observe that a finite set $\mathcal{F}$ of forbidden subgraphs implies a finite set $\mathcal{F}'$ of forbidden induced subgraphs: if $F \in \mathcal{F}$ than G does not contain F as a subgraph if and only if it does not contain any induced subgraph F', such that $|V(F')| = |V(F)|$ and F is a subgraph of F'. Thus we define $\mathcal{F}' = \{F' \mid E(F') \supseteq E(F), V(F') = V(F)\}$ and conclude: the class of graphs that do not contain any $F \in \mathcal{F}$ as a subgraph is the class $\mathcal{F}'$-free.

There are, however, some reliefs if a class $\mathcal{C}$ of graphs is defined by forbidden subgraphs. It is clear that $\mathcal{C}$ edge addition sets are pointless, because a subset of a set is a subset of any superset. Furthermore $\mathcal{C}$ edge deletion sets M can be transformed into $\mathcal{C}$ vertex deletion sets of size at most $2|M|$: $M' = \{v \in e \mid e \in M\}$. Thus a bound on the $\mathcal{C}$ edge addition number, $\mathcal{C}$ edge deletion number or $\mathcal{C}$ modification number implies a bound on the $\mathcal{C}$ vertex deletion number, which is thus the only parameter we have to deal with (if we want to show tractability).

4.2. Feedback vertex set

Following theorem 4.4 one might ask, whether such a result can be generalized to cover graph classes whose set of forbidden (induced) subgraphs is infinite. Since graph isomorphism for bipartite graphs is GI-complete and bipartite graphs are those without odd cycles as subgraphs, no general FPT-result is possible unless Graph Isomorphism ∈ P. Nevertheless it is possible to study individual classes, which is what Kratsch and Schweitzer did in [KS10] for forests.

As outlined in section 4.1.3 we only need to consider vertex deletion sets for forests. Those sets are generally called ***feedback vertex sets*** and their minimal size in a graph G the ***feedback vertex number*** of G ($\mathrm{fvs}(G)$). This name comes from (directed) dependency graphs, where a vertex on a (directed) cycle depends on its own (previous) state.

Let us now list the ingredients needed to construct an algorithm like algorithm 4.3 for graphs with bounded feedback vertex number:

1. a small cycle (size bounded by $\mathrm{fvs}(G)$)
2. an efficient way to find such a small cycle
3. a FPT-algorithm to construct a feedback vertex set of size k, if one exists

Finding small cycles if they exist, is the easiest part: e.g. do a BFS for every vertex u and find an edge $\{v, w\}$ not in the BFS-tree such that $d(u, v) + d(u, w) + 1$ is minimized. The complexity of computing a feedback vertex set has been studied extensively: it belongs to Karp's classical NP-complete problems [Kar72] and also has been considered early by Downey and Fellows [DF95c] as an example for the usefulness of the notion of fixed-parameter tractability. We define the parameterized function problem as:

		f-Feedback Vertex Set
Input	:	(G, k), G is a graph, $k \in \mathbb{N}$
Parameter	:	k
Output	:	A feedback vertex set of size at most k, if one exists, else "none".

4.2.1. Find a feedback vertex set, ...

The most recent FPT-algorithm is due to [CFLLV08] and yields a runtime of $\mathcal{O}(5^k k|V(G)|^2)$ where k is the supposed feedback vertex number. We will only sketch the proofs of this result, as this somewhat deviates from our focus. The key idea is to compute a feedback vertex set M of a Graph G for the special case where G is partitioned into two induced forests: $V(G) = V_1 \cup V_2$, such that $G[V_1]$ and $G[V_2]$ are forests. Furthermore we require $M \subseteq V_1$ and thus any vertex in $v \in V_1$, that has two neighbors in the same component of V_2 ***must*** be in M, as v is part of a cycle C such that $V(C) \setminus \{v\} \subseteq V_2$. Deleting (or moving) leaves in $G[V_1]$ with at most one neighbor in V_2, the algorithm from lemma 4.5 only has to branch on vertices of V_1 which connect components of $G[V_2]$.

This algorithm is then applied in the ***iterative compression*** procedure due to [RSV04]. Iterative compression tries to find a vertex set $S = S_l$ (a feedback vertex set in our case) of size at most k in G, by considering an increasing sequence $G_1, \ldots, G_l = G$ of induced subgraphs of G and iteratively computing the set S_{i+1} of size at most k for G_{i+1} by shrinking the set $S_i \cup (V(G_{i+1}) \setminus V(G_i))$. $|V(G_{i+1}) \setminus V(G_i)|$ needs to be bounded by some c (often and here too: $c = 1$). Going from $S_i \cup (V(G_{i+1}) \setminus V(G_i))$ to S_{i+1} requires the exchange of at most $k + c$ by k vertices. The algorithm from lemma 4.5 is applied to compute for each $R \subseteq S_i$ a replacement of size $|R - 1|$. We will now discuss some details by proving the following lemma and theorem.

Lemma 4.5 ([CFLLV08]). *Let G be a graph, $k \in \mathbb{N}$ and $V_1, V_2 \subseteq V(G)$ such that $V_1 \cup V_2 = \emptyset$ and $G[V_1]$ and $G[V_2]$ are forests and $G[V_2]$ has l components. In $\mathcal{O}(2^{k+l}|V(G)^2|)$ we can decide whether G has a feedback vertex set $M \subseteq V_1$ of size at most k and, if there is one, compute it.*

Proof sketch. The proof goes by induction on k,l and $|V_1|$. If $k = 0$ we test whether G is a forest, if $l = 0$ or $|V_1| = 0$ we know $G = G[V_1]$ (or $G = G[V_2]$ resp.) is a forest.

Assume $k > 0, l > 0, |V_1| > 0$. If there are no vertices in V_1 that have two neighbors in V_2, we pick an arbitrary leaf v of $G[V_1]$. If it has at most one neighbor in G we may remove it from G. Otherwise all of its neighbors but at most one lie in one component of $G[V_2]$ and we may move it to V_2, since it is either a leaf of G or has a neighbor $v' \in V_1$ that also lies on all of its cycles. In either case we may recurse with an instance which has smaller $|V_1|$ and thus use the inductive hypothesis.

If there is a vertex v in V_1 that has two neighbors in the same component of V_2 it has to included in any feedback vertex set and we recurse with $G - v$. In any other case there is a vertex $v \in V_1$ that connects at least two components of $G[V_2]$. This vertex is either part of a feedback vertex set of size at most k or not. If it is, we obtain a feedback vertex set by computing one for $G - v$ and $k - 1$, if it is not we may move it to V_2 thereby reducing the number of components by at least one. We try the first option and if it fails, go for the second. This brings to recursive calls where either k or l is reduced by at least 1. Since computing the components of $G[V_2]$ is the most expensive part, which may take $|V(G)^2|$ steps and the only case where two recursive calls are needed reduces k or l, the runtime follows inductively. □

Note that this approach also works for graphs with multi-edges and loops. Vertices with loops are in any feedback vertex set and a vertex with a multi-edge to another vertex can be seen as having two times the same neighbor and thus two neighbors in one component.

Theorem 4.6 ([CFLLV08]). *f-Feedback Vertex Set is solvable in time $\mathcal{O}(5^k k|V(G)|^2)$ (for input (G, k)) and thus fixed-parameter tractable.*

Proof sketch. We will sketch the proof via iterative compression for a series of graphs $G_1, \ldots, G_{|V(G)|}$ such that $V(G_{i+1}) \setminus V(G_i) = \{v_i\}$, while [CFLLV08] uses a 2-approximation of $\text{fvs}(G)$ to construct a G_1 with $|V(G_1)| \geq |V(G)| - k$ and thus have only k iterative compression steps. In our setting, obviously $\text{fvs}(G_i) \leq k$ for $i \leq k$. For $i + 1 > k$, let S_i be the feedback vertex set of G_i and let $R_{i+1} = (S_i \cup \{v_{i+1}\}) \setminus S_{i+1}$ be set of vertices that will be removed from the feedback vertex set in the compression step and $K_{i+1} = (S_i \cup \{v_{i+1}\}) \cap S_{i+1}$ the set of those we keep. As we try to construct a feedback vertex set without R_{i+1}, $G[R_{i+1}]$ has to be a forest. The same holds for $G[N'_{i+1}]$, where $N'_{i+1} = V(G_{i+1}) \setminus (S_i \cup \{v_{i+1}\})$ is the set of candidates for new vertices in S_{i+1}. Now S_{i+1} is a feedback vertex set of G_{i+1} if and only if $S_{i+1} \setminus K_{i+1}$ is feedback vertex set for $G[V(G_{i+1}) \setminus K_{i+1}] = G[N'_{i+1} \cup R_{i+1}]$. Thus using the algorithm from lemma 4.5 with input $(N'_{i+1}, R_{i+1}, k - |K_{i+1}|)$ computes the missing part of our next feedback vertex set in time $\mathcal{O}(2^{2|R_{i+1}|}n^2)$. Since the input is completely defined by the choice of R_{i+1}, we need to test this for every possible R_{i+1}. By the binomial theorem this yields $\mathcal{O}((4+1)^k n^2)$ calls for each step $i \to i+1$ and thus an overall runtime of $\mathcal{O}(5^k|V(G)|^3)$ for the entire algorithm or $\mathcal{O}(5^k k|V(G)|^2)$ if the construction using the 2-approximation is applied. □

4.2.2. … ensure that the graphs have a short cycle …

Short cycles are the most demanding ingredient on our list. Luckily there is a theorem by Raman, Saurabh and Subramanian [RSS06] that guarantees short cycles in graphs with bounded feedback vertex number. However this result has one caveat: it only holds for graphs with minimum degree 3.

Theorem 4.7 ([RSS06]). *Let G be a Graph with feedback vertex set of size at most k and minimum degree at least 3. Either the girth of G is at most 6 or $V(G) \leq 2k^2$ (and the girth is at most $2 + 4 \log k$).*

Proof. We assume $|V(G)| > 2k^2$, so there is a feedback vertex set M of size k in G. Then $G' = G \setminus M$ is a forest. The key idea is to find a set $\{\{a_1, a_2\}, \{b_1, b_2\}\} \subseteq E(G') \cup \binom{V(G')}{1}$, $\{a_1, a_2\} \cap \{b_1, b_2\} = \emptyset$ such that there are $x \neq y : \{x, y\} = N(a) \cap N(B) \cap M$ (where

$N(\{u,v\}) = N(u) \cup N(v)$). Then $(a_1, x, b_1, b_2, y, a_2, a_1)$ or a subsequence of it is a cycle in G. We prove the existence of such sets by the pigeonhole principle and the fact that a forest may not contain too many vertices with degree ≥ 3. Any vertex of degree ≤ 1 in G' must have two neighbors in M, because the minimum degree in G is 3. The same holds for an edge in G' such that both of its vertices have degree 2, unless they share a common neighbor (and thus form a cycle with it). Let L be the set of leaves of G', $T = \{v \in V(G') \mid \deg_{G'}(v) = 2\}$ and $U = V(G') \setminus (L \cup T)$. Now $|U| < |L|$ and thus $|V(G')| < |T| + 2|L|$. Observe that $G'[T]$ is a disjoint union of paths. Thus we may construct a maximal matching R of $G'[T]$ that leaves only endpoints of paths of even length unmatched. We do this by rooting each component of $G'[T]$ and ensure that each unmatched vertex has its children in $V(G) \setminus T$. Then there is a injection from the unmatched vertices S to $L \cup U$ and thus $2|R| \geq |T| - |S| \geq |T| - (2|L|)$ and $2|L \cup R| \geq |T|$. We now replace $|T|$ in the inequality above and get $2|L \cup R| + 2|L| > |V(G')|$, which leads to

$$|L \cup R| > \frac{(|V(G)| - k)}{4} \geq \frac{k^2 - k/2}{2} > \binom{k}{2} \quad .$$

So two elements $\{a_1, a_2\}, \{b_1, b_2\} \in L \cup R$ share a pair of vertices in $\{x, y\} \in M$, which yields the desired constellation. If $|V(G)| \leq 2k^2$, the girth directly follows from an upper bound for the girth of $2 \log |V(G)|$ for graphs G with minimum degree 3 [EP62]. Since $k \geq 2$ for simple graphs with minimum degree three, this bound also includes the case where the girth is at most 6. □

Note, that a bound on the girth of $2k^2$ would also be sufficient for the purpose of fixed-parameter tractability. Furthermore observe that this result also holds for graphs with multi-edges and loops, since those are cycles of length two and one, respectively.

Willing to use the theorem above, we are forced to turn arbitrary input graphs into graphs with minimum degree 3. This can be done by ***reducing*** the graphs: progressively removing leaves and replacing vertices with degree 2 by and edge that connects their neighbors. In order to get rid of all vertices with degree 2, we may need to introduce multi-edges and loops. Both can be interpreted as weights (i.e. colors) on edges and vertices. A weighted graph G_w is thus a vertex and edge colored graph (G, w) with $\mathrm{rng}(w) \subseteq [0, \binom{V(G)}{2}]$. A cycle or path in a weighted graph may have the form $\ldots, v, w, v, \ldots$ or $\ldots, v, v \ldots$, if $w(e) \geq = 2$ (or $w(v) \geq 1$ resp.), the degree is defined as $\deg(v) = 2w(v) + \sum_{u \in N(v)} w(\{u, v\})$ and removing a vertex means $G_w - v = (G - v, w|_{\mathrm{dom}(w) \setminus \{v\}})$. Note that a simple graph G corresponds to the weighted graph $((V(G), \binom{V(G)}{2}), w)$, where $w(m) = 1 \Leftrightarrow m \in E(G)$. We denote the class of weighted graphs as $\mathcal{W} = \{G_w \mid G \in \mathcal{G}\}$.

We have to reduce graphs in a way, that is compatible with isomorphism, i.e. a pair of isomorphic graphs shall remain isomorphic. But looking on algorithm 4.3, we eventually want to fix pairs of vertices, which has to be done in reduced graphs as only those are guaranteed to contain short cycles. This leads to a problem if a fixed vertex in the reduced graph corresponds to many vertices in the original graph and we are left with too many choices. So what we actually want is a reduction ***function*** $R : \mathcal{W} \to \mathcal{W}$, such that $V(R(G)) \subseteq V(G)$ and for every $\phi : V(G_1) \to V(G_2)$:

$$G_1 \cong_\phi G_2 \Rightarrow R(G_1) \cong_{\phi|_{V(R(G_1))}} R(G_2) \quad .$$

Kratsch and Schweitzer solve this in a way, that does not impose an order of reduction steps and is yet well defined:

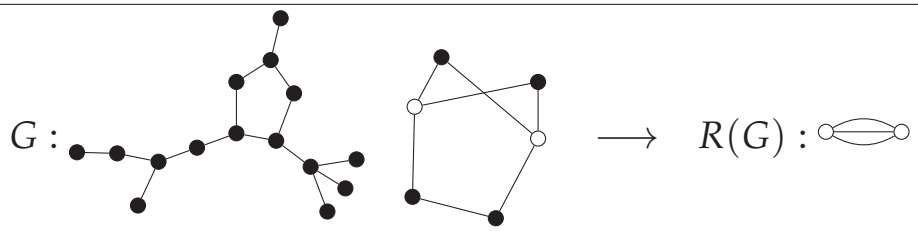
Definition 4.8 ([KS10])**.** Let G_w be a weighted graph. The reduction function R_S with respect to a set of vertices S is defined such that $R_S(G_w)$ is the result after the exhaustive application of the following rules:

1. remove a vertex v with $\deg(v) \leq 1$
2. remove a vertex contained in S
3. contract a vertex v with degree $\deg(v) = 2$: Let $\{u, x\}$ be the set of its neighbors ($u \neq v \neq x$). If $u \neq x$, increment $w(\{u, x\})$, else increment $w(u)$. Then remove v.
4. remove a vertex from a connected component with at most one cycle

Further we define R on unweighted graphs by the correspondence above and $R = R_\emptyset$. •

Note that the set S will be used to remove vertices that where fixed in a previous step and will not be considered, when we like to find the next pair of vertices to fix. The important addition by [KS10] is step 4. Without this step, e.g. a cycle C_n would reduce to a single vertex with a loop, but we were free to chose which vertex remains.

Figure 4.1. Exhaustive application of rules in definition 4.8: The reduction function R is well-defined and thus we know which vertices (here white) of G remain in $R(G)$.



Lemma 4.9 ([KS10])**.** *R_S of definition 4.8 is well-defined and thus for every pair of graphs G_1 and G_2 and every $\phi : V(G_1) \to V(G_2)$:*

$$G_1 \cong_\phi G_2 \Rightarrow R(G_1) \cong_{\phi|_{V(R(G_1))}} R(G_2) \quad .$$

Proof. The proof goes by the minimal criminal method: let $\#(G_w, S)$ be the number of (v, i) such that v may be removed from G_w using rule i (after other reductions steps). Clearly R_S is well-defined if $\#(G_w, S) = 0$. Now choose a pair (G_w, S) with minimal $\#(G_w, S)$ such that there exists two sequences of graphs $G_w = G_0, G_1, \ldots, G_l$, and $G_w = G'_0, G'_1, \ldots, G'_{l'}$ obtained by exhaustive application of the rules of definition 4.8. By the choice of G_w and S, $R_S(G'_1)$ and $R_S(G_1)$ are well-defined, thus it suffices to show that they are equal. Let v be the vertex considered during the step $G_w \to G_1$ and let v' likewise correspond to $G_w \to G'_1$. If none of the vertices was contracted, the other vertex is still present, thus v can be removed from G'_1 and v' from G_1. Assume that v has been contracted during the step $G_w \to G_1$. Now there three options for v':

1. v' has been removed from G_w: Then the degree of v has not been increased from G_w to G'_1, so the same rule as in $G_w \to G_1$ can still be applied. The degree of v' is also still the same in G_1 and it can be removed.
2. v' has been contracted during the step $G_w \to G'_1$ and is not adjacent to v: Then both vertices retain their respective neighbors in the other's graph, so both contractions are still possible.
3. v' has been contracted, but is adjacent to v: If v still has at least one neighbor $u \neq v$ in G'_1, its degree has not increased and it may still be contracted. The same holds for v' in

G_1. Otherwise v is a vertex with a single loop in G_1' and v' is a vertex with a single loop in G_1, as a neighbor $u : v' \neq u \neq v$ of one of them in G_w would become the neighbor of the other in G_1 and G_1' respectively. In this case they may be removed by rule 4.

The same argument with the same cases works for v' as well and thus G_1 and G_1' may be reduced to the same graph, which yields $R_S(G_1') = R_S(G_1)$. □

Clearly, every graph G with at most one cycle per component reduces to the weighted empty graph $((\emptyset, \emptyset), \emptyset)$. The converse is also true:

Lemma 4.10 ([KS10])**.** *Let G be a graph such that $R(G)$ is the empty graph. Then every component of G contains at most one cycle.*

Proof. Assume G has a component with at least two cycles. Without rules 2 and 4, no vertex on a cycle may be removed in a way, that decreases the degree of its neighbors. Take one cycle $v_1, \ldots, v_l$ in any component with two cycles. Then there is either a vertex disjoint path from this cycle to itself (w.l.o.g. $v_1, \ldots, v_j$ with $2 \neq j \neq l$) or a path to another vertex disjoint cycle (w.l.o.g. starting in v_1). Neither of the paths is affected by rule 1, so v_1 will always have degree ≥ 3 and thus cannot be removed. □

4.2.3. … and use them to fix a pair of vertices.

Algorithm 4.4. Graph isomorphism for graphs with bounded feedback vertex number [KS10]: ***graphIsoFVS***

Input : Graphs (G_1) and (G_2), injective partial function $\psi : V(G_1) \to V(G_2)$ and $k \in \mathbb{N}$
Output : Isomorphism ϕ from G_1 to G_2 with $\phi|_{\mathrm{dom}(\psi)} = \psi$ or `null`

```
1  G1' ← R_dom(ψ)(G1)
2  G2' ← R_rng(ψ)(G2)
3  if G1' is empty ∧ G2' is empty
4  |    return isoOneCyclePerComp(G1, G2, ψ)
5  if G1' is empty ∨ G2' is empty
6  |    return null
7  if k = 0
8  |    return null
9  M ← FVS(G1')
10 C ← shortCycle(G2')
11 for (v1, v2) ∈ M × V(C)
12 |    φ ← graphIsoFVS(G1, G2, ψ ∪ {(v1, v2)}, k − 1)
13 |    if φ ≠ null
14 |    |    return φ
15 end
16 return null
```

Now that we know, that we find short cycles in a reduced graph, we are able to state algorithm 4.4. This algorithm differs conceptually from algorithm 4.3 in two ways: we do not remove the fixed pairs of vertices and recolor, but save them as a partial isomorphism. The

Algorithm 4.5. Graph isomorphism for graphs with one cycle per component [KS10]: ***isoOneCyclePerComp***

Input : (Colored) graphs (G_1, c_1) and (G_2, c_2) with at most one cycle in each component, injective partial function $\psi : V(G_1) \to V(G_2)$
Output : Isomorphism ϕ from G_1 to G_2 with $\phi|_{\mathrm{dom}(\psi)} = \psi$ or `null`

```
1  if c_1 = c_2 = ∅
2  |   c_1 ← (V(G_1) → {1}); c_2 ← (V(G_2) → {1})
3  if ψ ≠ ∅
4  |   (v_1, v_2) ← arbitrary pair from ψ
5  |   if (c(v_1)=c(v_2))
6  |   |   return isoOneCyclePerComp((G_1 − v_1, c_1 − v_1), (G_2 − v_2, c_2 − v_2), ψ\{(v_1, v_2)})
7  |   else
8  |   |   return null
9  ((G_1, c_1), (G_2, c_2)) ← removeLeavesAndRecolor((G_1, c_1), (G_2, c_2))
10 return matchComponents((G_1, c_1), (G_2, c_2))
```

other difference is that in the base case ($R(G_1)$ and $R(G_2)$ are empty) the graphs are not always elements of the class under consideration (i.e. forests). This means we have to yield an isomorphism test for colored graphs with at most one cycle in each component (after removal of the fixed pairs), which is what algorithm 4.5 does. Let us now prove its correctness and runtime:

Lemma 4.11 ([KS10])**.** *Given two colored graphs (G_1, c_1) and (G_2, c_2) and an injective partial function $\psi : V(G_1) \to V(G_2)$ such that $G_1 \setminus \mathrm{dom}(\psi)$ and $G_2 \setminus \mathrm{rng}(\psi)$ contain at most one cycle in each component, algorithm 4.5 computes an isomorphism ϕ from (G_1, c_1) to (G_2, c_2) with $\phi|_{\mathrm{dom}(\psi)} = \psi$, if and only if one exists, in time $\mathcal{O}(|V(G_1)|^2)$.*

Proof. The adjacency information of all the vertices in $\mathrm{dom}(\psi)$ and $\mathrm{rng}(\psi)$ respectively is correctly encoded into colors, as discussed in section 4.1.2. Each step takes time $\mathcal{O}(|V(G_i)|)$ and since ψ is injective, there are at most $|V(G_i)|$ steps. The correctness of the removeLeavesAndRecolor-method (algorithm 2.3) was proved in lemma 2.25 as well as its runtime: $\mathcal{O}(|V(G_1) + E(G_1)|)$. Finally ***matchComponents*** (algorithm 2.5) takes time $\mathcal{O}(|V(G_1)|^2)$ and tests correctly whether there is a matching of isomorphic components between both graphs (2.28), since each component-wise isomorphism test is performable in linear time (for colored cycles, see lemma 2.27). □

We conclude this section with the discussion of runtime and correctness of algorithm 4.4:

Theorem 4.12 ([KS10])**.** *Let G_1 and G_2 be graphs, ψ an injective partial function $V(G_1)$ from $V(G_2)$ and $k \in \mathbb{N}$. Algorithm 4.4 decides in time $\mathcal{O}(|V(G_1)|^2)$, whether $\mathrm{fvs}(R_{\mathrm{dom}(\psi)}(G_1)) \leq k$, $\mathrm{fvs}(R_{\mathrm{rng}(\psi)}(G_2)) \leq k$ and $G_1 \cong_\phi G_2$ such that $\phi|_{\mathrm{dom}(\psi)} = \psi$ and if so computes an isomorphism ϕ. Graph Isomorphism parameterized by the feedback vertex number is thus fixed-parameter tractable.*

Proof. First observe that the reduction as defined in definition 4.8 ensures that a feedback vertex set of size l in $G_i, i \in [2]$ can be turned into a feedback vertex set of the same size in G_i': if a vertex is removed, the resulting graph has feedback vertex set of size $l - 1$ by the definition of feedback vertex sets and if a vertex is contracted, it has degree 2, so both of its

neighbors are on all of its cycles, too. Also observe that a reduced graph is a forest, if and only if it is empty.

Correctness: We now proceed and prove the correctness by induction on the argument k. For $k = 0$, $G_1' = R_{\mathrm{dom}(\psi)}(G_1)$ and $G_2' = R_{\mathrm{dom}(\psi)}(G_2)$ have to be forests and thus empty, otherwise the algorithm does correctly reject them until line 8. If they are empty, the algorithm goes to line 4 and calls isoOneCyclePerComp (algorithm 4.5), which correctly computes the result by lemma 4.11.

Assume $k > 0$. If exactly one of G_1' and G_2' is empty the algorithm correctly rejects them as in the case $k = 0$, the same holds for the case where both are empty, so assume that none of them is empty. In the same way as discussed for algorithm 4.3, for any isomorphism ϕ' from G_1' to G_2' and a feedback vertex set M of G_1' and any cycle C of G_2' the image of M intersects $V(C)$: $\phi'(M) \cap V(C) \neq \emptyset$. Thus for every such isomorphism ϕ' there is a pair $(v_1, v_2) \in M \times V(C) : \phi(v_1) = v_2$. Lemma 4.9 guarantees us that this also holds for isomorphisms ϕ from G_1 to G_2, since their restrictions to $V(G_1')$ are isomorphisms from G_1' to G_2'. So testing for each pair (v_1, v_2) yields an isomorphism if and only if one exits (subject to $\mathrm{fvs}(G_1') \leq k$). Finally we need to argue that recursing with parameter $k - 1$ is correct. This is also easy to see by lemma 4.9: Assume we call our algorithm with $\psi' = \psi \cup \{(v_1, v_2)\}$ instead of ψ. The order of reduction steps is irrelevant, so we may remove v_1 and v_2 last. Let G_i'' be the graph G_i' if ψ' is used. Now $G_i' - v_i = G_i''$ and since v_1 i in a feedback vertex set of size at most k in G_1', we know that $\mathrm{fvs}(G_1'') \leq k - 1$. The same holds for G_2' and v_2, provided that G_1 and G_2 are isomorphic and we picked a partial isomorphism ϕ'' up to now. Finally it is easy to see, that we will not return a function ϕ, that is not an isomorphism, as the actual construction of ϕ is only performed in the base case and we return it "as-is".

Runtime: The isomorphism test in the base case is doable in time $\mathcal{O}(|V(G)|^2)$ (lemma 4.11). Theorem 4.6 gives us a runtime of $\mathcal{O}(5^k|V(G)|)$ for the computation of a feedback vertex set, which dominates the time for finding a shortest cycle. The girth of $V(G_2')$ is at most $2 + 4\log(k)$ (theorem 4.7) and the size of the feedback vertex set M is bounded by k, so $M \times V(C)$ has size at most $2k + 4k\log(k)$. Since this size is clearly greater than 5 for $k \geq 1$, we assume that each recursive call has runtime $\mathcal{O}((2(k-1) + 4(k-1)\log(k-1))^{k-1})$, which yields runtime $\mathcal{O}((2k + 4k\log(k))^k)$ for the loop and dominates all other parts. Called with $\psi = \emptyset$ the algorithm yields fixed-parameter tractability of the Graph Isomorphism parameterized by the feedback vertex number (technically, we have to assure $\mathrm{fvs}(G_1) \leq k$ separately, because $\mathrm{fvs}(G_1') \leq k \not\Rightarrow \mathrm{fvs}(G_1) \leq k$). □

5. Modular decompositions

5.1. Cographs

Many graph algorithms are designed for connected graphs. This is no restriction since there are at least two simple ways of extending a graph algorithm for connected components to the entire graph: use the complement (which is connected) or test at most all pairs of components (one from each graph) until you found a matching. If on the other hand the graph is connected, but the complement is not, this yields a simple recursive algorithm for graph isomorphism, provided that isomorphism testing for the induced subgraphs which are connected and have a connected complement is easy. This idea is summarized in algorithm 5.1. Its name already implies the class to which this algorithm is dedicated: ***cographs***.

Definition 5.1. A graph G is a cograph if and only if $|V(G)| = 1$ or G or $\overline{G}$ is a disjoint union of at least 2 cographs. •

In other words, a graph is a cograph if line 9 is never executed in any of the recursive calls of algorithm 5.1. Another form to characterize cographs is to exclude the P_4, the path of length 3, as an induced subgraph.

Lemma 5.2 ([Sum73])**.** *A graph is a cograph if and only if it does not contain an induced subgraph which is isomorphic to the P_4.*

Proof. The complement of a P_4 is again a P_4 and so all vertices of the induced subgraph belong to the same connected component, so a graph which contains a P_4 as an induced subgraph is not a cograph.

If on the other hand a graph G is not a cograph it contains an induced subgraph G' such that G' and $\overline{G'}$ are connected. G' has a spanning tree T. If the length of the longest path in T is at most 1, $|V(G')| \leq 2$ and thus G' is a cograph. If the longest path has length 2, then T is a star and its central vertex is isolated in $\overline{G'}$. Both cases contradict the assumption and thus the longest path in T is at least 3 and thus G' and G contain a P_4 as an induced subgraph. □

Corollary 5.3. *Cograph Modification Set is fixed-parameter tractable.*

Proof. The class of cographs is identical to $\{P_4\}$-free and we can apply theorem 4.2. □

Observe that the naïve algorithm runs in polynomial time for cographs: Each recursive call handles a disjoint subset of $V(G) \times V(G')$, which is strictly contained in the set of the parental call and there are $|V(G)|^2$ leaves in the call tree. Since a directed tree with outgoing degrees of at least 2 for every internal vertex and $|V(G)|^2$ leaves has $\mathcal{O}(|V(G)|^2)$ vertices and detecting all components can be done in linear time, the overall runtime is polynomial.

Algorithm 5.1. Naïve isomorphism test for cographs: ***naïveIsoCoGr***

Input : Graphs G_1 and G_2
Output : Are G_1 and G_2 isomorphic?

```
1  if (|V(G1)|≠|V(G2)|)
2  |   return "not isomorphic"
3  if (|V(G1)|=|V(G2)|=1)
4  |   return "isomorphic" V(G1)× V(G2)
5  if (G1 is connected)
6  |   G1 ← complement(G1)
7  |   G2 ← complement(G2)
8  if (G1 is connected)
9  |   return (sophisticated algorithm(G1, G2))
10 φ ← ∅
11 for C ∈ {V | V is a connected component of G1 }
12 |   for D ∈ {V | V is a not associated connected component of G2 }
13 |   |   ψ ← naïveIsoCoGr(G1, G2)
14 |   |   if ψ ≠ "not isomorphic"
15 |   |   |   φ ← φ ∪ ψ
16 |   |   |   associate(C, D)
17 |   |   |   break
18 |   |   end
19 |   end
20 |   if C is not associated to any component
21 |   |   return "not isomorphic"
22 |   end
23 end
24 if ∃ not associated component D of G2
25 |   return "not isomorphic"
26 else
27 |   return φ
28 end
```

However using the so called ***cotree*** a linear time algorithm can check two cographs for isomorphism, see [CPS85]. We will not discuss cotrees separately, since the cotree is the modular decomposition tree of a cograph.

Corollary 5.4. *Graph Isomorphism parameterized by the cograph modification number is fixed-parameter tractable.*

Proof. Graph Isomorphism is in P for cographs, Cograph Modification Set is fixed-parameter tractable and thus the claim directly follows from theorem 4.4. □

As a final note on cographs we remark that not all problems are easy to solve if restricted to cographs: for instance subgraph isomorphism remains NP-complete [Dam91].

5.2. Modules and the uniqueness of the modular decomposition

Modules and the modular decomposition where devised multiple times with a variety of names. One of the earliest occurrences dates back to 1967 when Tibor Gallai [Gal67] introduced modules as "geschlossene Punktmengen" (closed vertex sets) and defined a unique decomposition via quasi-maximal modules. The "join" by Gert Sabidussi (introduced in 1961, [Sab61]) generalized the lexicographic product of graphs and was accompanied by notion of representation that used modules (practically the inverse of a join), but was not a unique representation. The term modular decomposition itself is used in the way below at least since [Spi83]. A good overview on algorithmic aspects of modular decompositions is given by [HP10].

Definition 5.5 (module, quasi-maximal, strong)**.** Let G be a graph. We say a vertex $v \in V(G)$ can ***distinguish*** a pair of vertices $\{u, w\} \subseteq V(G)$ if $\{u, v\}$ in $E(G)$ and $\{w, v\} \notin E(G)$. A set of vertices $M \subseteq V(G)$ is called a ***module*** of G if no vertex outside of M can distinguish any pair of vertices in M, i.e. if for all $u \in V(G) \setminus M$ and all $v, w \in M$

$$\text{either } \{\{u, v\}, \{u, w\}\} \subseteq E(G) \text{ or } \{\{u, v\}, \{u, w\}\} \cap E(G) = \emptyset \quad .$$

We will call a module M ***quasi-maximal*** if $M \neq V(G)$ and for all modules $M' \neq V(G)$: $M \not\subset M'$. The term ***strong module*** refers to a module M such that for all modules M' either $M \subseteq M'$, $M' \subseteq M$ or $M \cap M' = \emptyset$. •

The notion of modules generalizes the concept of connected components. Since decomposing a graph G in a canonical way is easy if either G or $\overline{G}$ are not connected (we take the set of components (of the complement)), we will focus on graphs were both G and $\overline{G}$ are connected. To build up some intuition about modules, we start with a simple lemma showing that induced paths also play an important role for modules.

Lemma 5.6. *Let G be a graph and $P = (\{v_i \mid i \in [l]\}, \{\{v_i, v_{i+1}\} \mid i \in [l-1]\})$ with $l \geq 4$ be an induced path of G, i.e. $P = G[V(P)]$. Then $V(P)$ is a subset of a module M of G if M contains two vertices v_i and v_j, $1 \leq i < j \leq l$.*

Proof. Let $\{v_i, v_j\} \subseteq M$ for some module M of G. For $2 \leq i < j \leq l$, the vertex v_{i-1} distinguishes $\{v_i, v_j\}$ and likewise v_{j+1} for $1 \leq i < j \leq l-1$. Hence $v_{i-1} \in M$ and $v_{j+1} \in M$ respectively. If we iteratively replace j by $j+1$ and i by $i-1$, we see that $\{v_i \mid i \in [i] \cup [j, l]\} \subseteq M$. But unless $j = i+2$, v_{i+1} distinguishes $\{v_i, v_j\}$ and we iteratively apply this again. If $j = i+2$, we assume w.l.o.g. that v_{i-1} exists (because $l \geq 4$) and hence v_{i+1} distinguishes $\{v_{i-1}, v_i\}$. Hence $M = V(P)$. □

While partitioning $V(G)$ into modules is not sufficient to obtain a unique decomposition, using quasi-maximal modules yields a unique decomposition, provided that G and $\overline{G}$ are connected. We will prove this now by a series of lemmas.

Lemma 5.7 ([Gal67, (2.7)])**.** *Let M_1 and M_2 be modules of G. If $M_1 \cap M_2 \neq \emptyset$, then $M_1 \cup M_2$ and $M_1 \cap M_2$ are also modules. If the sets $M_i \setminus M_{3-i}, i \in [2]$ are both not empty, then they are modules, too.*

Proof. Assume there is a vertex $v \in V(G) \setminus (M_1 \cap M_2)$ that distinguishes $\{u, v\}$. Than v is not in both M_1 and M_2, but u and w are, thus either M_1 or M_2 would not be a module. The same holds for a vertex $v \in V(G) \setminus (M_1 \cup M_2)$ that distinguishes $\{u, w\}$ ($u \in M_1, w \in M_2$): since there is some $x \in M_1 \cap M_2$, v would distinguish either $\{x, u\}$ or $\{x, w\}$ and therefore contradict the assumption.

For the second part assume $v \notin M_1 \setminus M_2$ distinguishes some $\{u, w\} \subseteq M_1 \setminus M_2$. We know $v \in M_1 \cap M_2$, otherwise M_1 would not be a module. Now assume w.l.o.g. $\{u, v\} \in E(G)$. Since there is an $x \in M_2 \setminus M_1$ and M_2 is a module, we know $\{u, x\} \in E(G)$. But since M_1 and M_2 are a modules, we conclude $\{x, w\} \in E(G)$ and $\{w, v\} \in E(G)$, so v does not distinguish $\{u, w\}$. In the same way we conclude that $M_2 \setminus M_1$ is a module. □

Lemma 5.8 ([Gal67, (2.8)])**.** *Let M_1 and M_2 be modules of a graph G such that G and $\overline{G}$ are connected and $M_1 \neq V(G) \neq M_2$, then $M_1 \cup M_2 \neq V(G)$.*

Proof. If either module is contained in the other one, the claim directly follows, so we assume $M_1 \setminus M_2$ and $M_2 \setminus M_1$ are not empty. Now M_2 and $M_1 \setminus M_2$ are disjoint modules by lemma 5.7, so $\{u, v\} \in E(G')$ for each pair $(u, v) \in (M_1 \setminus M_2) \times M_2$, for either $G' = G$ or $G' = \overline{G}$. If $V(G) = M_1 \cup M_2$, then either in G or in $\overline{G}$ there is no path from vertices in M_1 to vertices in $M_1 \setminus M_2$. Thus not both graphs can be connected, which yields the desired contradiction. □

Lemma 5.9 ([Gal67, (2.9)])**.** *If M is a quasi-maximal module of a graph G such that G and $\overline{G}$ are connected and $|V(G)| \geq 2$, then M is strong.*

Proof. Assume there is another module N such that M and N overlap, i.e. $M \cap N \neq \emptyset$, $N \setminus M \neq \emptyset$ and $N \setminus M \neq \emptyset$. By lemma 5.7, we know that $N \setminus M$ and $M \cup (N \setminus M) = M \cup N$ are modules and lemma 5.8 guarantees us that $M \cup N \subsetneq V(G)$ and thus M was not quasi-maximal. □

Theorem 5.10 ([Gal67, (2.10)])**.** *For any graph G such that G and $\overline{G}$ are connected and $|V(G)| \geq 2$ and any vertex $v \in V(G)$ there is a exactly one quasi-maximal module M_v that contains v. Therefore there is a unique partition of $V(G)$ into quasi-maximal modules.*

Proof. If $v \in M$ for some module M, either M is quasi-maximal (let $M' = M$ in this case) or contained in some quasi-maximal module M'. Thus v is contained in at least one strong Module $M' \neq V(G)$ (by lemma 5.9). Let S be the family of strong modules $M \neq V(G)$ which contain v. Since $M \cup N \supseteq \{x\}$ for all $M, N \in S$, either $M \subseteq N$ or $N \subseteq M$ by the definition of modules. Thus $\subseteq$ is a linear order on S and we can take its unique maximum M_v, which must be the only quasi-maximal module containing v, since any such module must be in S and a subset of M_v. □

Note that the last proof differs from [Gal67], since we do not discuss comparability graphs here and the proof there is closely related to them.

Definition 5.11. For any graph G we define the ***quasi-maximal modular partition*** of G as

- $\{V(G)\}$, if $|V(G)| \leq 1$,
- the set of components of G, if G is not connected,
- the set of components of $\overline{G}$, if $\overline{G}$ is not connected,
- the set of quasi-maximal modules of G in any other case.

If D is the quasi-maximal modular partition of G, the ***quotient graph*** of G shall be the graph $\text{quot}(G) = (D, E_D)$, where

$$\{M_1, M_2\} \in E_D \Leftrightarrow \exists v_1 \in M_1, v_2 \in M_2 : \{v_1, v_2\} \in E(G) \quad .$$

We call G a ***prime graph*** (or ***prime*** as an attribute) if G and its complement are connected and its quasi-maximal modular partition D consists only of modules $\{v\}, v \in V(G)$. •

Corollary 5.12. *The quasi-maximal modular partition of a graph is uniquely defined.* □

Theorem 5.10 and its proof directly lead to a simple polynomial time algorithm for the computation of the quasi-maximal modular partition (algorithm 5.2): Observe that we can turn any candidate set $S \subseteq V(G)$ into a module by recursively adding vertices v to S that distinguish any $\{u, w\} \subseteq S$. This procedure is called ***closeUnderDistinguishers*** in algorithm 5.2 and clearly runs in linear time. By theorem 5.10 there is exactly one quasi-maximal module M_v for each vertex v and since every module $M \neq V(G)$ is a subset of a quasi-maximal module, it is a subset of M_v. So for any candidate set $S \subseteq M_v$ its closure under distinguishers S' is still a subset of M_v, but if $v \in S$ and $S \nsubseteq M_v$, then $S' = V(G)$ otherwise we would have found a second quasi-maximal module containing v. This brings us to the add-and-close approach of algorithm 5.2 (for-loop starting in line 11) and the following lemma:

Algorithm 5.2. Naïve computation of the quasi-maximal modular partition

Input : Graph G
Output : The quasi-maximal modular partition of G

```
1  if (|V(G)|=1)
2  |    return {V(G)}
3  if (G or Ḡ is not connected)
4  |    return the components of G (or its complement resp.)
5  P ← ∅
6  for (v ∈ V(G))
7  |    if (marked(v))
8  |    |    continue
9  |    M ← {v}
10 |    marked(v) = true
11 |    for (u ∈ V(G))
12 |    |    if (marked(u))
13 |    |    |    continue
14 |    |    M' ← M ∪ {u}
15 |    |    closeUnderDistinguishers(M')
16 |    |    if (M'≠ V(G))
17 |    |    |    for w ∈ (M'\ M) marked(w) = true  end
18 |    |    |    M = M'
19 |    end
20 |    P ← P ∪ {M}
21 end
22 return P
```

Lemma 5.13. *Algorithm 5.2 computes a quasi-maximal modular partition of a graph G in time $\mathcal{O}(|V(G)|^4)$.*

Proof. The correctness was mostly discussed above. Observe that each vertex is marked only once and thus we do not add modules to P multiple times, so no membership test has to be performed in line 20 if we implement P as a list.

Computing the components and the components of the complement can be done in linear time. The outer loop is dominated by the inner loop, which in turn is dominated by the closure operation. Both loop over all vertices and the close operation is performed in time $\mathcal{O}(|V(G)| + |E(G)|) = \mathcal{O}(|V(G)|^2)$, which provides the claimed runtime. □

As prime graphs and quotient graphs will play a fundamental role in the next section, we now state and prove and important lemma about the relationship between graphs and their quotient graphs:

Lemma 5.14 ([CHM81])**.** *If G is a connected graph whose complement is also connected, then* $\text{quot}(G)$ *is prime.*

Proof. Assume $\text{quot}(G)$ is not a prime graph. Let $\{M_1, \dots, M_k\} \neq V(\text{quot}(G))$ be a module of $\text{quot}(G)$ such that $M_i \neq M_j$ for $i \neq j$ and $k \geq 2$. Thus $M_1, \dots, M_k$ are quasi-maximal modules of G and their union $M' = \bigcup\{M_1, \dots, M_k\}$ is a proper subset of $V(G)$ by lemma 5.8. Take any vertex $v \in V(G) \setminus M'$ and assume v distinguishes $\{u, w\} \subseteq M'$. Clearly $u \in M_i, w \in M_j$ for $i \neq j$, since all M_i are modules. Let M_v be the quasi-maximal module which contains v. Now M_v distinguishes $\{M_i, M_j\}$ and we reach our awaited contradiction. □

5.3. The modular decomposition tree and its computation

Definition 5.15 (modular decomposition tree)**.** For a graph G we define its modular decomposition tree MD as the inclusion minimal rooted tree $T = (V, E, r)$, such that $r = V(G)$, $r \in V$ and for all $M \in V$

$$D_M \subseteq V \text{ and } \forall N \in D_M : \{M, N\} \in E,$$

where D_M is the quasi-maximal modular partition of $G[M]$. If G is a cograph, we call $\text{MD}(G)$ its ***cotree***. •

Theorem 5.16. *The modular decomposition tree can be computed in polynomial time.*

Proof. Let G be a graph. Its modular decomposition tree has $|V(G)|$ leaves and no inner vertices of degree smaller than three, so $\text{MD}(G)$ has at most $2|V(G)|$ vertices. To compute the children (and to test if there are any) of any vertex in $\text{MD}(G)$ a call to algorithm 5.2 is sufficient. So an $\mathcal{O}(|V(G)|^6)$-time algorithm is easily achievable. □

While this result is sufficient for the purposes of this work, the algorithm sketched here is extremely inefficient. In fact, linear time algorithms are known since the 1990s (see again [HP10] for a good overview), but even a discussion of the simple algorithm in [TCHP08] would deviate us from graph isomorphism. Nevertheless, we state the result.

Theorem 5.17 (e.g. [TCHP08])**.** *The modular decomposition tree can be computed in linear time.* □

5.4. Application to Graph Isomorphism

We are now ready to assemble the parts in this section and turn to the application of modular decompositions to the graph isomorphism problem. To achieve this we return to the naïve isomorphism algorithm for cographs (algorithm 5.1). Instead of the partition into components (of the complement) we use the quasi-maximal modular partition. But now we have to solve an additional problem. The components of a graph are independent, because they are all connected in the same way. But the quasi-maximal modules form the quotient graph, whose adjacency information has to be respected by a graph isomorphism algorithm. Let us assume that we have an Colored Graph Isomorphism algorithm for a certain graph class $\mathcal{C}$ which contains all prime graphs that occur in the modular decompositions of the input graphs G_1 and G_2. Then we are able to solve Graph Isomorphism for the entire graphs, if we can compute colors that represent the isomorphism type of each quasi-maximal module, color the vertices of the quotient graph with them and use the algorithm for the fixed restricted class $\mathcal{C}$. Luckily, these colors do not have be canonical, which brings us to algorithm 5.3 and following theorem about it.

Theorem 5.18. *Let $\mathcal{C}$ be a class of graphs for which f-Colored Graph Isomorphism is in computable in time $r(|V(H_1)|)$ for $H_1, H_2 \in \mathcal{C}$ and G_1 and G_2 be graphs such that all graphs associated to prime vertices of the modular decomposition of G_1 and G_2 are in $\mathcal{C}$. Then algorithm 5.3 computes an isomorphism from G_1 to G_2, if and only if they are isomorphic, in time $\mathcal{O}(r(|V(H_1)|)|V(G_1)|^4)$.*

Proof. The runtime analysis is the same as for the naïve cograph isomorphism algorithm (5.1), with the exception that we do not compute components, but the quotient graph (which is also possible in linear time by theorem 5.17) and there is an additional factor for the isomorphism test in line 26.

The correctness proof goes by induction on the height of the call tree. Since the only function (and thus isomorphism) is returned for two graphs with just a single vertex, the base case (height 0) is correct.

Assume that the height is at least 1. Then the nested loops test every pair of modules (and thus pairs of vertices in the quotient graph) $(M_1, M_2) \in V(Q_1) \times V(Q_2)$ for isomorphism and colors M_1 and M_2 with the same color, if $G[M_1]$ and $G[M_2]$ are isomorphic. If Q_1 and Q_2 are either both cliques or independent sets of vertices of the same size, the only thing we have to do is to count the numbers of occurrences of each color and check if they match. If exactly one of them is a clique or if the sizes are different, no isomorphism is possible. In any other case we use the algorithm for colored graphs in $\mathcal{C}$. An isomorphism ψ is returned if and only the graphs induced by the modules are isomorphic. If no isomorphism is returned, G_1 and G_2 are not isomorphic, because the modular decomposition tree is an invariant (up to isomorphism), so assume ψ is returned. By the nature of modules this defines an isomorphism on the entire graph: Let $M_i, N_i \in V(Q_i), i \in [2]$. Further assume $\psi(M_1) = M_2$ and $\psi(N_1) = N_2$. Since ψ is an isomorphism $\{M_1, N_1\} \in E(Q_1)$ if and only if $\{M_2, N_2\} \in E(Q_2)$. Furthermore for $m_i \in M_i$ and $n_i \in N_i$: $\{m_i, n_i\} \in E(G_i)$ if and only if $\{M_i, N_i\} \in E(Q_i)$ by the nature of modules. Thus the image of m_1 under an isomorphism ϕ from G_1 to G_2 does only depend on an isomorphism from $G_1[M_1]$ to $G_2[M_2]$ and such an isomorphism ϕ_{M_1,M_2} was precomputed within the nested loops. □

Observe that algorithm 5.3 is completely agnostic about the algorithm ***coloredIsoClass*** used

Algorithm 5.3. Graph isomorphism via modular decomposition: ***modDecIso***

Input : Graphs G_1 and G_2
Output : Isomorphism $\phi : G_1 \cong_\phi G_2$ or "not isomorphic"

```
1  if (|V(G1)|≠|V(G2)|)
2  |   return "not isomorphic"
3  if (|V(G1)|=|V(G2)|=1)
4  |   return V(G1) × V(G2)
5  maxUsedColor ← 0
6  Q1 ← quot(G1)
7  Q2 ← quot(G2)
8  if ((|V(Q1)|≠|V(Q2)|) ∨ exactly one of Q1 and Q2 is prime)
9  |   return "not isomorphic"
10 for M1 ∈ V(Q1)
11 |   c1(M1) ← maxUsedColor + 1
12 |   for M2 ∈ V(Q2)
13 |   |   φ_{M1,M2} ← modDecIso(G1[M1], G2[M2])
14 |   |   if (φ_{M1,M2} ≠ "not isomorphic")
15 |   |   |   if (∃ c : c2(M2)=c)
16 |   |   |   |   c1(M1) ← c2(M2)
17 |   |   |   |   break
18 |   |   |   else
19 |   |   |   |   c2(M2) ← c1(M1)
20 |   |   |   |   maxUsedColor ← c1(M1)
21 |   end
22 end
23 if (¬ (|c1|=|c2|=|V(Q1)|))
24 |   return "not isomorphic"
25 if (Q1 is prime)
26 |   ψ ← coloredIsoClass((Q1, c1), (Q2, c2))
27 else
28 |   ψ ← trivialColorComparison((Q1, c1), (Q2, c2))
29 if (ψ = "not isomorphic")
30 |   return "not isomorphic"
31 φ ← ⋃_{(M1,M2)∈ψ} φ_{M1,M2}
32 return φ
```

in line 26 to compute an isomorphism for colored graphs of some fixed class $\mathcal{C}$. Thus this algorithm also works for classes $\mathcal{C} = \{G \in \mathcal{G} \mid \kappa(G) \leq k\}$ defined by bounded parameters. This gives us the possibility to combine modular decompositions and arbitrary graph parameters (especially those which punish certain dense graphs for which Graph Isomorphism is actually easy). We conduct this combination by considering the parameter of the prime graphs of the modular decomposition instead of the whole graph:

Definition 5.19 (prime parameter)**.** Let κ be any graph parameter $\mathcal{G} \to \mathbb{N}$ and G an arbitrary graph. The ***prime parameter*** of G corresponding to κ (denoted $'\kappa$) is defined as

$$'\kappa(G) = \max\{\min\{\kappa(P), \kappa(\overline{P})\} \mid V(P) \in V(MD(G)) \wedge P \text{ is prime}\} \quad .$$

If G is a cograph, we define $'\kappa(G) = 0$. •

Corollary 5.20 (of theorem 5.18). *GRAPH ISOMORPHISM parameterized by $'\kappa$ is fpt Turing reducible to GRAPH ISOMORPHISM parameterized by κ.* □

Remark 5.21. [Cou96] defines a parameter called ***modular width***. If we call the number of vertices of a graph its ***order***, then our name for the modular width is ***prime order***, i.e. the maximum order of any prime graph appearing in the modular decomposition of a graph. **!**

Remark 5.22. Algorithm 5.3 also constitutes an unparametrized polynomial time Turing reduction from GRAPH ISOMORPHISM to PRIME GRAPH ISOMORPHISM. But for this purpose there is an even simpler many-one reduction. Let G be a graph with $|V(G)| > 3$. Replace all edges in a G with a P_4 and all non-edges with a P_5 and call the result $\hat{G}$. Clearly only the old vertices have degree $|V(G)| - 1$ and any two vertices lie on an induced P_4 or P_5. If two vertices of an induced P_k ($k \geq 4$) belong to a module M of $\hat{G}$, then $V(P_k) \subseteq M$. Thus the constructed graph is prime and $\forall u, v \in V(G) : d_{\hat{G}}(u, v) = 4 \Leftrightarrow \{u, v\} \in E(G)$. **!**

We defer a detailed analysis of the relations between different (prime) parameters to section 8.2 and only provide an example for now.

Example 5.23. Let

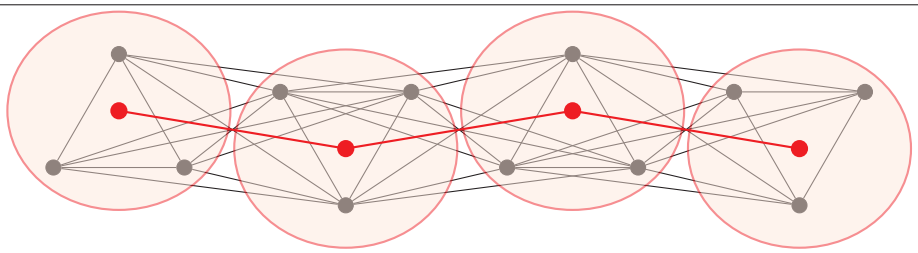
$$G_{n,m} = ([mn], \{\{i, j\} \mid \exists l : \{i, j\} \subseteq [ln + 1, (l + 2)n - 1]\})$$

be the graph obtained by replacing each vertex in a path P_m ($m \geq 4$) with a clique K_n and connecting all vertices of adjacent cliques (see figure 5.1). Then the root R of $\mathrm{MD}(G_{n,m})$ is isomorphic to the P_m and thus $\mathrm{fvs}(R) = 0$. Furthermore all other vertices of $\mathrm{MD}(G_{n,m})$ are either parallel or serial and thus the prime feedback vertex number of $G_{n,m}$ is 0 ($'\mathrm{fvs}(G_{n,m}) = 0$). On the other hand $\mathrm{fvs}(G_{n,m}) \geq m(n-2) - 1$, since not a single triangle in any copy of the K_n may remain to turn $G_{n,m}$ into a forest. In the same way we get:

treewidth	$\mathrm{tw}(G_{n,m} = 2n$	$'\mathrm{tw}(G_{n,m}) = 1$
maximum degree	$\max\deg(G_{n,m}) = 3n - 1$	$'\max\deg(G_{n,m}) = 2$

e.g.

Figure 5.1. The graph $G_{n,m}$ of example 5.23 with $n = 3$ and $m = 4$



5.5. Application to CANONICAL LABELING

During the analysis of algorithm 5.3 we mentioned that the coloring of modules that encodes their isomorphism type does not need to be canonical. Well, that was only half of the truth as algorithm 5.3 can be further simplified if we reduce to COLORED CANONICAL LABELING instead of COLORED GRAPH ISOMORPHISM. Furthermore, if we have canonical labelings of the prime graphs, we can also recursively compute canonical labelings for all subtrees of the modular decomposition tree. This is what algorithm 5.4 does.

Algorithm 5.4. Canonical labeling via modular decomposition: ***modDecCL***

Input : Graphs G
Output : Canonical labeling of G

```
1  if (|V(G)|=1)
2  |    return V(G) × {1}
3  Q ← quot(G)
4  for M ∈ V(Q)
5  |    l_M ← modDecCL(G[M])
6  |    c'(M) ← (l_M(M), l_M(E(G[M])))
7  end
8  s ← sortLexico(rng(c'))    // input as a set
9  for M ∈ V(Q)
10 |    c(M) ← s^{-1}(c'(M))
11 end
12 if (Q is prime)
13 |    l' ← coloredCLClass((Q, c))    //canonical labeling for prime graphs
14 else
15 |    l' : V(Q) → [|V(Q)|] such that c(M) < c(N) ⇒ l'(M) < l'(N)
16 m ← 0
17 for i ∈ [|V(Q)|]
18 |    M ← l'^{-1}(i)
19 |    l_M ← l_M + m
20 |    l ← l ∪ l_M
21 |    m ← m + |M|
22 end
23 return l
```

Theorem 5.24. *Algorithm 5.4 computes a canonical labeling for an input graph G in time $\mathcal{O}(r(G)|V(G)|^3)$ where $r(G)$ is the maximal runtime for* ***coloredCLClass*** *over all prime graphs appearing in* $\mathrm{MD}(G)$. *Hence* CANONICAL LABELING *parameterized by $'\kappa$ is fpt Turing reducible to* CANONICAL LABELING *parameterized by κ.*

Proof. The correctness is easily to see. The first for-loop computes canonical labelings and canonical forms of all modules of the quasi-maximal modular decomposition of the input graph G and hence we have local canonical labelings and a canonical colors of each module based on the canonical forms. Sorting and the second for-loop turns these colors into integers. These are used to compute a "global" canonical order l', i.e. one of the modules. This is done by either the call to ***coloredCLClass*** in line 13 for prime graphs Q or a simple sort if Q is complete or edgeless. In either case, if another labeling $\hat{l}$ corresponds to the same canonical form $\mathtt{C}$ as l', then for two modules $M \neq N$ of G with $l'(M) = \hat{l}(N)$ we know that $G[M] = G[N]$ and there is an automorphism ϕ of Q such that $\phi(M) = N$. This ensures that the canonical form for G corresponding to the labeling constructed in the last for-loop does not depend on the choice of $l' \in \mathrm{cl}_{\mathtt{C}}(Q, c)$. The correctness of this last for-loop itself is obvious as it just computes a union of constantly shifted labelings, respecting the "local" and "global" canonical orders.

The runtime analysis is again similar to that of algorithm 5.3, but this time we only have $|V(G)|$ leaves in our call tree, which is equivalent to the modular decomposition tree of G.

$V(G)^2$ clearly bounds all operations, except the call to ***coloredCLClass*** and the recursive calls. □

Iterative algorithm: In most interesting scenarios for this algorithm the prime graphs are smaller than the modules and thus the same holds for their canonical forms. Hence it may be beneficial to sort canonical forms of prime graphs in this case. We can rewrite algorithm 5.4 in a way similar to the linear time canonical labeling isomorphism for trees (algorithms 2.3 and 2.4). The result is algorithm 5.5. Its calls to ***coloredCLClass*** corresponds to the computation of the multisets representing the isomorphism type of a subtree in the for-loop starting in line 13 of algorithm 2.3. Observe that the sort here runs over all modules on one level to compute consistent colors for all subtrees of $MD(G)$ whose root has the same distance to $V(G)$ in $MD(G)$. This also shows that our runtime analysis above was everything but rigorous, but as all algorithms in this chapter should be seen as reductions or schemes and a tight upper bound depends on the parameter, we will not give a more precise analysis.

Algorithm 5.5. Canonical labeling via modular decomposition (iterative)

Input : Graph G
Output : canonical labeling of G

```
1  T←MD(G)
2  L_0←{V(G)}
3  compute all L_l ← {M ∈ V(T) | d_T(V(G), M)=l}
4  h ← height(T)
5  for M ∈ (V(G) choose 1)
6  |    c(M)←1
7  end
8  for l←h down to 1
9  |    for M ∈ L_{l−1}
10 |    |    Q ← quot(G[M])
11 |    |    if (Q is prime)
12 |    |    |    φ_M ← coloredCLClass(Q, c|_{L_l ∩ N(M)})
13 |    |    else
14 |    |    |    φ_M : V(Q) → [|V(Q))|] such that c(N) < c(O) ⇒ φ_M(N) < φ_M(O)
15 |    |    c'(M) ← (φ_M(M), φ_M(E(G[M])))
16 |    end
17 |    s ← sortLexico(c'(L_{l−1}))    //input as multiset
18 |    for M ∈ L_{l−1}
19 |    |    c(M)← s^{−1}(c'(M))    //s with duplicates removed
20 |    end
21 end
22 compute a global canonical labeling as in algorithm 2.4
23 combine the global canonical labeling with each local φ_M as in the for-loop
     starting in line 17 of algorithm 5.4
```

6. Distance widths

Motivated by the study of treewidth, in [YBFT99] Yamazaki, Bodlaender, de Fluiter and Thilikos introduced similar parameters that take distances into account and thereby reduce the number of decompositions to consider. Doing this, they were able to prove that Graph Isomorphism is fixed-parameter tractable if parameterized by the so called rooted path distance with or the rooted tree distance width. This chapter is devoted to these results as well as some generalizations. We start by giving basic definitions.

Definition 6.1 ([YBFT99])**.** Let G be a graph. The ***path distance decomposition*** with respect to a root set R of G is the tuple of ***distance levels*** $N = (N_0, \dots, N_l)$, such that $R = N_0$ and $\{N_i \mid i \in [0, l]\}$ is a partition of $V(G)$ by the minimal distance to R, i.e.

$$N_i = \{v \in V(G) \mid \min_{u \in R} \mathrm{d}(u, v) = i\} \quad .$$

We use $|N|$ to denote the number of levels $l + 1$. The width of N is the maximal size of any level N_i (including R). The ***path distance width*** of G ($\mathrm{pdw}(G)$) is the minimal width of a path distance decomposition over all root sets $R \subseteq V(G)$. The minimal width over all root sets R with size $|R| = 1$ will be called ***rooted path distance width*** of G ($\mathrm{rpdw}(G)$). We set $\mathrm{rpdw}(G) = |V(G)|$, if there is no rooted path distance decomposition, i.e. if G is not connected.

A ***tree distance decomposition*** of a G is a strong tree decomposition (B, T, r) (see definition 1.6) such that if M is the path distance decomposition of T with root set $\{r\}$ and N the path distance decomposition of G with root bag B_r then

$$\forall i \in [0, |N| - 1] : N_i = \bigcup_{t \in M_i} B_j \, .$$

The width of a tree distance decomposition shall be its width as a strong tree decomposition. The ***(rooted) tree distance width*** $(\mathrm{r})\mathrm{tdw}(G)$ is defined analogously to $(\mathrm{r})\mathrm{pdw}(G)$. •

We remark that [YBFT99] and [Ota12] only define the widths parameters for connected graphs, which leaves multiple options for the definition in the disconnected case. Our definition requires the existence of a ***single*** decomposition, which is consistent with treewidth and strong treewidth. Another natural option would be the maximum over all components. If we only consider the fixed-parameter tractability of Graph Isomorphism, the difference between both definitions is irrelevant, but it may influence the degree of the polynomial in $|V(G)|$.

6.1. Rooted path distance width

Corollary 6.2 (of corollary 3.16)**.** *Canonical Labeling parameterized by the rooted path distance width is fixed-parameter tractable.*

Proof. Coloring vertices according to their distance to a single vertex v can be done in linear time and there are only $|V(G)|$ rooted path distance decompositions of a connected graph G. Thus we need to apply algorithm 3.5 at most $|V(G)|$ times with color classes of size at most k to compute a canonical labeling for a connected input graph G with $\text{rpdw}(G) = k$. □

However, using the algorithm for bounded color classes is fairly inefficient. We will thus discuss the algorithm from [YBFT99], which is also a good preparation for the other results. Algorithm 6.1 is an isomorphism testing algorithm, but we will construct an analogous canonization algorithm later on.

6.1.1. Isomorphism test

Algorithm 6.1. Graph isomorphism for graphs of bounded rooted path distance width [YBFT99]

Input : Connected graphs G_1,G_2
Output : Are G_1 and G_2 isomorphic?

```
1  if (|V(G1)|≠|V(G2)|)
2  |    return "not isomorphic"
3  N←null
4  for v ∈ V(G1)
5  |    N' ← pathDistanceDec(G1, {v})
6  |    if (width(N') < width(N) ∨ N = null )
7  |    |    N ← N'
8  end
9  for v ∈ V(G2)
10 |    M ← pathDistanceDec(G2, {v})
11 |    if (width(N) ≠ width(M) ∨ |N| ≠ |M|)
12 |    |    continue
13 |    I_{|N|-1} ← {φ | G1[N_{|N|-1}] ≅_φ G2[M_{|N|-1}]}
14 |    for i←|N| − 2 down to 0
15 |    |    I_i ← {φ | ∃ ψ ∈ I_{i+1} : G1[N_i ∪ N_{i+1}] ≅_{φ∪ψ} G2[M_i ∪ M_{i+1}]}
16 |    end
17 |    if (|I_0| ≠ 0)
18 |    |    return "isomorphic"
19 end
20 return "not isomorphic"
```

The following lemma is important to understand that linear time means $\mathcal{O}(\text{pdw}(G)|V(G)|)$.

Lemma 6.3 ([Ota12]). *For any graph G:* $\max \deg(G) \leq 3\,\text{pdw}(G) - 1$.

Proof. Assume the is a $v \in V(G)$ with $\deg(v) \geq 3k$ for some $k \in \mathbb{N}$. Let $P = (N_0, \ldots, N_l)$ a path distance decomposition and $v \in N_i$ and thus $N(v) \subseteq N_{i-1} \cup N_i \cup N_{i+1}$ (let $N_{-1} = \emptyset$). Now at least one $N_j, j \in [i-1, i+1]$ has size $k+1$, because $|N(v) \cup \{v\}| = 3k+1$. This means $\text{pdw}(G) \geq k+1$, which yields the claim. □

Theorem 6.4 ([YBFT99]). *Algorithm 6.1 computes whether two connected input graphs G_1 and G_2 are isomorphic in time $\mathcal{O}((k+1)!^2|V(G_1)|^2)$, where $k = \mathrm{rpdw}(G_1)$.*

Proof. To prove the correctness, we observe that ϕ is an isomorphism from G_1 to G_2 if and only if there exist rooted path distance decompositions $N = (N_0, \ldots, N_l)$ of G_1 and $M = (M_0, \ldots, M_l)$ of G_2 such that

$$\forall i \in [0, l] : G_1[N_i] \cong_{\phi|_{N_i}} G_2[M_i] \text{ and } G_1[N_i \cup N_{i+1}] \cong_{\phi|_{N_i \cup N_{i+1}}} G_2[M_i \cup M_{i+1}]\,.$$

This corresponds to the characterization of isomorphism as stabilizers of the sets of equally colored edges as given in section 3.3.1. Clearly, we may color two isomorphic graphs identically by distances to one vertex of a pair of fixed vertices if we consider all such pairs. This is what the loops of algorithm 6.1 do. Each set I_i contains the set of isomorphisms $\phi : G_1[N_i] \cong_\phi G_2[M_i]$ that are extendible, that is there is a ψ:

$$G[N_i \cup \cdots \cup N_l] \cong_{\phi\cup\psi} G[M_i \cup \cdots \cup M_l]\,,$$

which can easily be proved inductively. Thus we can rewrite I_0 as

$$I_0 = \{\phi|_{N_0} \mid G_1 \cong_\phi G_2\}\,,$$

which is empty if and only if there is an isomorphism from G_1 to G_2.

By lemma 6.3 the first loop runs in time $\mathcal{O}(k|V(G_1)|^2)$ and the computation of the rooted path distance decomposition in every iteration of the second outer loop takes time $\mathcal{O}(k|V(G)|)$. Inspecting all $k!$ bijections from N_i to M_i for compatibility as isomorphisms with a set I_{i+1}, which has at most $k!$ members, is doable in $\mathcal{O}((k!^2)k^2) = \mathcal{O}((k+1)!^2)$ time. Doing this at most $|V(G)|$ times (maximal number of distance levels) clearly dominates any other part of the second outer for-loop, which in turn runs in $\mathcal{O}((k+1)!^2|V(G_1)|^2)$ time and dominates the first loop. □

Computing an isomorphism As opposed to most other parts of this work algorithm 6.1 only decides Graph Isomorphism without computing an isomorphism. This can be done with a single loop in $\mathcal{O}(|V(G_1)|(\mathrm{rpdw}(G_1)+1)!)$ time: simply choose $\phi_0 \in I_0$ and $\phi_{i+1} \in I_{i+1}$ that is compatible with ϕ_i ($G_1[N_i \cup N_{i+1}] \cong_{\phi_i \cup \phi_{i+1}} G_2[M_i \cup M_{i+1}]$). Since all edges are in or between some M_i and M_{i+1} (and N_i resp.) the choice of ϕ_{i+1} does not need to consider any $\phi_j, j < i$ as their compatibility is already implied by the choice of ϕ_i. We conclude that $\phi = \bigcup_{i\in[0,|N|-1]} \phi_i$ is an isomorphism from G_1 to G_2.

6.1.2. Canonical labelings

The idea behind algorithm 6.1 can be used to compute a canonical labeling of a graph G. Instead of constructing extendible partial isomorphisms, we now construct extendible partial canonical labelings. Because we will consider other kinds of root sets later and we have already seen the global structure (for-loop over possible root sets) of an isomorphism algorithm, we only canonize a single path distance decomposition in algorithm 6.2. Because a path distance decomposition N of a graph G is a coloring defined by its root set, we may see a canonical form of the pair (G, N) as a canonical form of the colored graph (G, c_N), where $c^{-1}(i) = N_i$ for all $i \in [0, |N| - 1]$.

Algorithm 6.2. Canonical labeling of path distance decompositions

Input : Graph $G \in \mathcal{G}$ with path distance decomposition N
Output : Canonical labeling of (G, N)

1 rename all $v \in V(G)$ such that $u \in N_i, v \in N_j \wedge i < j \Rightarrow u < v$
2 $N_{|N|} \leftarrow \emptyset$
3 $I_{|N|} \leftarrow \emptyset$
4 **for** $i \leftarrow |N| - 1$ **down to** 0
5 | $a_i \leftarrow \min \{\text{adjStr}(G, N_i, \psi, N_{i+1}, \psi') \mid \psi \in \text{Sym}(N_i), \psi' \in I_{i+1}\}$
6 | $I_i \leftarrow \{\phi \in \text{Sym}(N_i) \mid \exists\, \phi' \in I_{i+1} : \text{adjStr}(G, N_i, \phi, N_{i+1}, \phi') = a_i\ \}$
7 **end**
8 Choose arbitrary $\phi_0 \in I_0$
9 **for** $i \leftarrow 1$ **to** $|N - 1|$
10 | choose $\phi_i \in I_i$ such that $\text{adjStr}(G, N_{i-1}, \phi_{i-1}, N_i, \phi_i) = a_{i-1}$
11 **end**
12 **return** $\bigcup_{i \in [0,|N|-1]} \phi_i$

13 **subprocedure** $\text{adjStr}(G, A, \phi, B, \phi')$
14 | $c(v) \leftarrow phi'(N(v) \cap B)$
15 | $\{v_1, \ldots, v_l\} \leftarrow A$ sorted according to ϕ
16 | **return** $(c(v_1), \ldots, c(v_l), e_{1,2}, \ldots, e_{l-1,l})$, where $e_{i,j} = \mathbf{I}_{E(G)}(\{v_i, v_j\})$
17 **end**

Theorem 6.5. *Algorithm 6.2 computes a canonical labeling for an input pair (G, N), a graph G together with a path distance decomposition N of G, in time $\mathcal{O}((k+1)!^2|V(G)|)$, where k is the width of N.*

Proof. We use all names from the algorithm. The correctness will be shown using backward induction, i.e. we show for all $i \in [0, |N| - 1]$ that

$$\bigcup_{j \in [i,|N|-1]} \phi_j \text{ is canonical for } \left(G\left[\bigcup_{j \in [i,|N|-1]} N_j\right], (N_i, \ldots, N_{|N|-1}) \right) = (G'_i, N'_i)\,.$$

If $i = |N| - 1$, we only have to choose those $\phi \in \text{Sym}(N_i)$ that minimize the special adjacency string computed by ***adjStr***. Since the last two arguments are $\emptyset$, this return value of ***adjStr*** is simply the adjacency matrix and thus using the minimum is clearly canonical. Hence I_i is the canonical labeling coset and we do not care which of its elements is chosen in the second for-loop.

For $i < |N| - 1$ we may now inductively assume that I_{i+1} is the canonical labeling coset containing the extendible canonical labelings for N_{i+1}. We know that $\bigcup_{j \in [i+1,|N|-1]} \phi_j$ labels canonically for any choice of $\phi_{i+1} \in I_{i+1}$ by the inductive hypothesis. Hence we are free to choose a ϕ_{i+1} that fulfills an additional minimization requirement. Now let ϕ_i and ϕ_{i+1} be chosen as in the second for-loop, i.e. a_i gets minimal for (ϕ_i, ϕ_{i+1}). Observe that the possible colorings in line 14 have to be the same for two isomorphic graph-decomposition pairs as I_{i+1} is a canonical labeling coset. Thus vertex orders in N_i (corresponds to A in the subprocedure) bijectively correspond for two isomorphic graphs and only the adjacency matrix decides about the minimum string, which is thus an invariant. By putting all first components of minimal (ϕ_i, ϕ_{i+1}) pairs in I_i, we again ensure that we have free choice on the next

level, since I_i is the canonical labeling coset.

The runtime analysis is nearly identical to that of algorithm 6.1, except that there are of course no nested for-loops. The runtime for an isomorphism check and the computation of a adjacency string is both $\mathcal{O}(k^2)$ and $|\operatorname{Sym}(N_i) \times \operatorname{Sym}(N_{i+1})|$ is still $k!^2$. □

Corollary 6.6. *A canonical labeling for a graph G can be computed in time $\mathcal{O}((k+1)!^2|V(G)|^2)$, where $k = \operatorname{rpdw}(G)$.* □

6.2. Connected and clustered path distance width

6.2.1. Connected path distance width

In [Ota12] Otachi was able to extend theorem 6.4 to the entire class of connected root sets and furthermore all the classes of root sets with at most c components, where c is an absolute constant, i.e. not considered a parameter in the sense of parameterized complexity. In this section we will outline these ideas with a strong focus on the enumeration procedure in [Ota12].

Definition 6.7 ([Ota12])**.** The ***c*-connected path distance width** of a graph G, c-cpdw(G), is the least width of a path distance decomposition with root set R, where $G[R]$ has at most c components. If there is no such decomposition (G has more than c components), then we set c-cpdw$(G) = |V(G)|$. We simply write ***connected path distance width*** (cpdw) for the 1-cpdw. •

Once we can enumerate all path distance decompositions of width at most k whose root set has at most c components, we are able to apply a slightly modified version of algorithm 6.1 to solve Graph Isomorphism for graphs with bounded c-cpdw. Note that a potential parameter k has to be part of the input, as deciding, whether a graph G has c-cpdw $\leq k$ is NP-hard ([Ota12, Theorem 3.2]) and thus we cannot compute c-cpdw(G) for any c. The enumeration algorithm (algorithm 6.3) recursively adds vertices with a path to at least one member of an initial root set R by considering the neighborhood. Actually, we do not need to enumerate ***all*** path distance decompositions in question, as we could return whenever we found one in line 5 and reduce k to the size of the minimal root set already returned during the for-loop. However, this would considerably complicate the analysis of the algorithm, so we refrain from these optimizations.

Theorem 6.8 ([Ota12, Theorem 3.6])**.** *For any graph G and $R \subseteq V(G)$ algorithm 6.3 computes the set*

$$P = \{\textit{path dist. dec. } N \mid \operatorname{width}(N) \leq k, \forall v \in N_0 \exists \textit{ path } v, \ldots, u \textit{ in } G[N_0] : u \in R\}$$

in time $\mathcal{O}(|V(G)|^{\frac{(2k)!}{k!}})$.

Proof. First observe that a set R with $|R| + |N(R)| > 2k$ cannot be a subset of a root set R' of a path distance decomposition whose width is at most k, because each vertex in $N(R) \cup R$ is in either R' or $N(R')$. Furthermore if there is a path $v_1, \ldots, v_l$ in G, then $v_{i+1} \in N(v_i)$ for all $i \in [l-1]$. Thus the for-loop and its recursive call test all simple paths whose first vertex is its only vertex in R and whose addition to R does not violate the size bound.

Algorithm 6.3. Enumerate path distance decompositions by adding neighboring vertices [Ota12]: ***enumPDD***

Input : Graph G, initial root set R and $k \in \mathbb{N}$
Output : Set of all path distance decompositions N of width at most k whose root set N_0 contains only vertices with a path (in $G[N_0]$) to at least one vertex in R

```
1  if |R| > k ∨ |R| + |N(R)\ R| > 2k
2  |    return ∅
3  N ← pathDistanceDec(G, R)
4  if (width(N) ≤ k)
5  |    P ← {N}
6  else
7  |    P ← ∅
8  for v ∈ (N(R) \ R)
9  |    P ← P ∪ enumPDD(G, R ∪ {v}, k)
10 end
11 return P
```

We prove the runtime by observing, that the for-loop is passed at most $2k - |R|$ times, since this is the maximal number of vertices we can add. Furthermore $|R|$ increases by 1 in each recursive level (of which there are thus at most k). We conclude that there are no more than $\mathcal{O}(\prod_{i=k}^{2k-1} i) = \mathcal{O}(\frac{(2k-1)!}{k!})$ recursive calls. Note that the union in line 9 takes only constant time, if we allow duplicates and use a linked list. Since computing a path distance decomposition takes only linear, i.e. $\mathcal{O}(k|V(G)|)$ time (lemma 6.3), the overall runtime follows by multiplication of these runtimes. □

Given this enumeration algorithm, all we need to do is choosing the correct initial sets. If the root set of a path distance decomposition of width at most k has at most c components, then all vertices in the root set can be reached from one of c vertices. If we started algorithm 6.3 with the set of these c vertices, it would thus include the aforementioned path distance decomposition in its output set. Thus running algorithm 6.3 for all sets $R \subseteq V(G), |R| = c$ yields all path distance decomposition of size at most k with at most c components. Because path distance decompositions directly correspond to colorings, we conclude in the same way as above:

Corollary 6.9 (of corollary 3.16)**.** *Graph Isomorphism parameterized by the c-connected path distance width is fixed-parameter tractable.* □

Again, using towers of groups is inefficient here and we slightly modify algorithm 6.1 to obtain a similar result.

Theorem 6.10 ([Ota12])**.** *Algorithm 6.4 decides whether G_1 and G_2 are isomorphic and both have c-connected path distance width at most k in time $\mathcal{O}((2k)!(k+1)!|V(G_1)|^{c+1})$.*

Proof. The correctness can be easily proved in the same manner as 6.4. The second outer for-loop clearly dominates the runtime. It is passed $|V(G_1)|^c$ times and contains $\mathcal{O}(\frac{(2k-1)!}{k!})$ iterations of the inner for-loop (see the proof of theorem 6.8), as well as the computation of all c-connected path distance decompositions of width at most k (in time $\mathcal{O}(|V(G_1)|\frac{(2k)!}{k!})$).

Algorithm 6.4. Graph isomorphism for graphs of bounded c-connected path distance width [Ota12]

Input : Graphs G_1,G_2 and $k \in \mathbb{N}$
Output : Are G_1 and G_2 isomorphic and c-cpdw$(G_1) \leq k$?

```
1  if (|V(G1)| ≠ |V(G2)|)
2  |   return false
3  N ← ∅
4  for R ∈ (V(G1) choose c)
5  |   if (enumPDD(G1, R, k) ≠ ∅)
6  |   |   N ← some element of enumPDD(G1, R, k)
7  |   |   break
8  end
9  if (N = ∅)
10 |   return false
11 for R ∈ (V(G2) choose c)
12 |   for M ∈ enumPDD(G2, R, k)
13 |   |   if (width(N) ≠ width(M) ∨ |N| ≠ |M|)
14 |   |   |   continue
15 |   |   I_{|N|-1} ← {φ | G1[N_{|N|-1}] ≅_φ G2[M_{|N|-1}]}
16 |   |   for i ← |N| − 2 down to 0
17 |   |   |   I_i ← {φ | ∃ ψ ∈ I_{i+1} : G1[N_i ∪ N_{i+1}] ≅_{φ∪ψ} G2[M_i ∪ M_{i+1}]}
18 |   |   end
19 |   |   if (|I_0| ≠ 0)
20 |   |   |   return true
21 |   end
22 end
23 return false
```

We know from the proof of theorem 6.4 that testing two path distance decompositions for isomorphisms takes time $\mathcal{O}(k!^2k^2|V(G_1)|)$. Their product is

$$\mathcal{O}\left(k!^2k^2|V(G_1)| \cdot \frac{(2k-1)!}{k!}\right) = \mathcal{O}((2k!)(k+1)!|V(G_1)|)$$

and clearly dominates the other parts of the second outer for-loop, so we proved the claim. □

Corollary 6.11. *If a graph G has c-connected path distance width at most k, a canonical labeling of G can be computed in time $\mathcal{O}((2k)!(k+1)!|V(G)|^{c+1})$.*

6.2.2. Clustered path distance width

The restriction of the root sets to connected subgraphs can be slightly generalized. Because of lemma 6.3, we know that the maximal degree in a graph with path distance width k is $3k-1$. Thus any connected root set is the subset of the set of all vertices with distance at most k to a single vertex v and the latter set has size smaller than $2(3k-1)^k$, i.e. it is bounded by a function on k. Generalizing this observation to root sets with c components is easy.

Definition 6.12. Let N be a path distance decomposition of a graph G. The ***c*-cluster width** of N is the minimal k such that the width of N is at most k and the root set N_0 may be partitioned into c (possibly empty) sets $C_1, \ldots, C_c$ such that for all $i \in [c]$ the graph $G[C_i]$ has diameter at most k. If such a partition is not possible, the ***c*-cluster width** of N is $|V(G)|$. The ***c*-clustered path distance width** of G, c-clpdw(G), is the minimal c-cluster width over all path distance decompositions N. •

There is a tiny technical difference between this approach and the connected path distance width. Instead of excluding certain path distance decompositions, we make them "more expensive" if they require bigger root clusters.

We are now able to replace the procedure ***enumPDD*** in algorithm 6.4 by algorithm 6.5 and state an analogous theorem.

Algorithm 6.5. Enumerate path distance decompositions with bounded c-cluster width

Input : Graph G, initial root set R and $k \in \mathbb{N}$
Output : Set of all path distance decompositions N of $|R|$-cluster width at most k

```
1  for v ∈ R
2  |    R_v ← {u ∈ V(G) | d(u,v) ≤ k}
3  end
4  R' ← ⋃_{v∈R} R_v
5  P ← ∅
6  for N_0 ∈ ℘(R')
7  |    N ← pathDistanceDec(G, N_0)
8  |    if (width(N) ≤ k)
9  |    |    P ← P ∪ {N}
10 end
11 return P
```

Theorem 6.13. *Algorithm 6.4 with* ***enumPDD*** *replaced by algorithm 6.5 decides whether input graphs G_1 and G_2 are isomorphic and both have c-clustered path distance width at most k in time $\mathcal{O}(2^{2(3k-1)^k}(k+1)!^2|V(G_1)|^{c+1})$.* □

We pay a high prize for this generalization in regard to the dependence on the parameter, but the degree with respect to $V(G_1)$ is not increased. We will take up this issue again in remark 8.6.

6.3. Rooted tree distance width

While a path distance decomposition is essentially a coloring and thus designing algorithms with runtime according to the definition of FPT means finding ways to enumerate the root sets, tree distance decompositions cannot be seen as colorings, because there may be multiple possible mappings between the children of two bags. Luckily, the algorithms for rooted path distance width in [YBFT99] were already designed with regard to rooted tree distance decompositions. Many aspects carry over in slightly more complicated manner. For instance the maximum degree of a graph with bounded tree distance width is no longer bounded, but

the average degree (and the number of edges) still is. Note that we only consider connected graphs in this section.

Lemma 6.14. *Let G be a graph with tree distance width at most k. Then $|E(G)| < 2k|V(G)|$.*

Proof. The proof works by induction on $|V(T)|$ where (B, T, r) is a tree distance decomposition of G. In a single bag B_i are at most k vertices and thus fewer than $k|B_i|$ edges. In a tree distance decomposition with at least two bags, one bag B_i corresponds to a leaf in the tree. The inductive hypothesis yields fewer than $2k(|V(G)| - |B_i|)$ edges in $G[V(G) \setminus B_i]$. Combined with no more than $k|B_i|$ edges in $G[B_i]$ and $k|B_i|$ edges between B_i and its adjacent bag, there are less than $2k|V(G)|$ edges in G. □

We remark that the proof of the fact that the average degree is bounded by $2\,\mathrm{tw}(G)$ is similar (especially easy, if we use the characterization as partial k-trees (see chapter 9)).

A path distance decomposition is completely defined by its root set, whereas we are left with a choice whether to branch or not to branch in some situations during the construction of a tree distance decomposition with a given root set, particularly we can always choose a path as our tree, so any path distance decomposition directly corresponds to a tree distance decomposition. If, however, we choose to branch wherever possible, each root bag defines exactly one tree distance decomposition (up to isomorphism of the underlying tree).

Lemma 6.15. *For a connected graph G and a root bag $R \subseteq V(G)$ there is a unique (up to isomorphism) tree distance decomposition (B, T, r) with $B_r = R$, called the **minimal** tree distance decomposition with root bag R, such that for all $t \in V(T)$ the graph $G\left[\bigcup_{u \in V(T_t)} B_u\right]$ is connected, where T_t is the partial tree rooted in t, i.e. $T[S \cup \{t\}]$, where S is the set of vertices not reachable from r in $T - t$. Furthermore all tree distance decompositions (B', T', r') of G with $B'_{r'} = R$ have at least the same width as (B, T, r).*

Proof. Uniqueness can be shown by induction on $|V(G)|$ and is of course fulfilled for $|V(G)| \leq 1$. For $|V(G)| > 1$ assume there is another minimal tree distance decomposition (B', T', r') of G with $B'_{r'} = R$. Let C be a component of $G[V(G) \setminus R]$. Then $C = \bigcup_{u \in V(T_t)} B_u = \bigcup_{u' \in V(T'_{t'})} B_{u'}$ for some children t, t' of r, r' in T or T' respectively. Otherwise either an edge in G would not be represented in the bags (if the component is not entirely contained in those unions of bags) or at least one of the decompositions would not be minimal (if one union of bags contains two or more components). But since there is only one way of representing $G[C]$ as a minimal tree distance decomposition (by the inductive hypothesis) and C was chosen arbitrarily, (B, T, r) and (B', T', r') are isomorphic (B and B' can be seen as colorings of T and T' respectively).

Finally we show that the width is minimal by comparing (B, T, r) with an arbitrary tree distance decompositions (B', T', r') of G with $B'_{r'} = R$. It is not hard to see that once $G_t = G\left[\bigcup_{u \in V(T'_t)} B_j\right]$ is not connected for some $t \in V(T')$ and no other vertex u in T'_t has this property, we can replace T'_t with minimal path distance decompositions rooted in $C \cap B_t$ for each component C of G_t. This does not increase the width and if we do this iteratively from the leaves to the root, (B', T', r') becomes minimal and thus equal to (B, T, r). □

The proof of lemma 6.15 already indicates that the minimal tree distance decomposition for a given root bag R can be computed from the leaves of the decomposition to the root,

i.e. in decreasing order of distance. A naïve approach would lead to the computation of components of $\bigcup_{j \geq i} N_i$ for each distance i, where N is a path distance decomposition with $N_0 = R$. However, if we somehow remember which vertices are connected to vertices in the same component in a greater distance to the root bag, we can achieve linear time. Algorithm 6.6 from [YBFT99] solves this problem by introducing arbitrary phony edges that allow us to restrict our attention to two consecutive distance levels.

Algorithm 6.6. Minimal tree distance decomposition for a given root bag [YBFT99]: ***minTreeDistDec***

Input : Connected graph G, and $R \subseteq V(G)$
Output : Minimal tree distance decomposition (B, T, r) with $B_r = R$

```
1  N←pathDistanceDec(G, R)
2  N_|N| ←∅
3  T←(∅, ∅)
4  for i ← |N| − 1 down to 0
5  |   for C ∈ {V | V is a connected component of G[N_i ∪ N_{i+1}]}
6  |   |   t ← max V(T) + 1
7  |   |   B_t ← C ∩ N_i
8  |   |   make G[B_t] connected by adding phony edges e ⊆ B_t
9  |   |   V(T) ← V(T) ∪ {t}
10 |   |   E(T) ← E(T) ∪ {{t, u} | B_u ⊆ C ∩ N_{i+1}}
11 |   end
12 end
13 return (B, T, max V(T))
```

Lemma 6.16 ([YBFT99])**.** *For a connected graph G and a set $R \subseteq V(G)$ algorithm 6.6 computes the minimal tree decomposition with root bag R in time $\mathcal{O}(|E(G)|)$.*

Proof sketch. We prove the correctness by induction on the number of distance levels $|N|$ for the path distance decomposition N of G with $N_0 = R$. Since G is connected, the algorithm outputs a tree T with $V(T) = \{1\}$ and B such that $B_1 = V(G)$ if $R = V(G)$. So we assume $|N| \geq 2$ from now on. Let C be a component of $G' = G[V(G) \setminus R]$. The phony edges added in line 8 cannot cross components of G' as the newly constructed bag is part of a component by its definition. This relies on older phony edges, but may easily be proved inductively. Thus all bags contained as subsets in C are computed in the same way during the outer for loop for $i = |N| - 1$ down to $i = 1$, as it would have been the case if we computed a tree distance decomposition of $G'[C]$ with root set $C \cap N_G(R)$ instead and thus $G'[C]$ is correctly decomposed by the inductive hypothesis. Since G is connected, every component of G' has a neighbor in R and thus the bags in the first level of a minimal tree distance decomposition are all of the form $C \cap N_G(R)$, where C is a component of G'. Thus the entire construction is correct.

To prove the runtime, we observe that each distance level of the path distance decomposition N is considered twice during the algorithm and each edge lies between at most two such levels. So each edge is treated at most twice. To make the graph connected, we only need to introduce at most $|N_i| - 1$ phony edges for each distance level N_i and thus no more than $|V(G)|$ edges at all. Since G is connected, it has still only $\mathcal{O}(E(G))$ edges after the addition of all phony edges. We insert phony edges by e.g. creating a star or using breadth first search

to get a set of proper endpoints. Because all operation within the inner for-loop have linear runtime with respect to $G[N_i \cup N_{i+1}]$, the overall runtime follows. □

6.3.1. Isomorphism test

Now we have all tools we need to assemble algorithm 6.7 for graphs of bounded rooted tree distance width in a similar way as algorithm 6.1 for bounded rpdw. As there is no a priori bijection between bags of two tree decompositions (even if the underlying trees are isomorphic), we need a subprocedure that matches child bags. Lemma 6.15 helps us to achieve this: each child bag corresponds to a connected component and thus the subprocedure can be designed like algorithm 2.5.

Theorem 6.17 ([YBFT99]). *Algorithm 6.7 solves* Graph Isomorphism *parameterized by the rooted tree distance width in time* $\mathcal{O}((k+1)!^2|V(G_1)|^3)$ *for two connected input graphs* G_1 *and* G_2*, where* $k = \mathrm{rpdw}(G_1)$*. Therefore* Graph Isomorphism *parameterized by the rooted tree distance width is fixed-parameter tractable.*

Proof. **Correctness**: The structure of the two outermost for-loops is similar to algorithm 6.1 and the correctness of this approach can be proved analogously to theorem 6.4. We use the same names as in the algorithm and analyze one iteration of the second of the outermost for-loops. Let t be in N_h and $u \in M_h$. Now we prove the following statement for arbitrary t, u by backward induction on h (i.e. along the loop starting in line 18):

$$I_{t,u} = \left\{ \phi|_{B_t} \;\middle|\; G_1[\bigcup_{p \in V(T_t)} B_p] \cong_\phi G_2[\bigcup_{q \in V(U_u)} C_q] \;\wedge\; \exists \chi : ((T_t, t) \cong_\chi (U_u, u) \wedge \forall p \in V(T_t) : \phi(B_p) = C_{\chi(p)}) \right\} .$$

This is trivial for the leaf bags of the tree decomposition, so we assume $h < |N| - 1$. If $(T_t, t) \ncong (U_u, u)$ then either the number of the children of t and u differ or any mapping between the children maps a child p of t onto a child q of u such that $(T_p, p) \ncong (U_q, q)$ and thus $I_{p,q} = \emptyset$ by the inductive hypothesis. But then the condition of line 35 is always violated by at least one pair (p, q), so $I_{t,u}$ will also be empty, since the subprocedure ***matchChildren*** rejects all candidates ϕ.

Hence we now change our assumption to $(T_t, t) \cong_\chi (U_u, u)$. Let ϕ be an isomorphism from $G_1[\bigcup_{p \in V(T_t)} B_p]$ to $G_2[\bigcup_{q \in V(U_u)} C_q]$ that maps bags according to χ. Now χ defines a matching between the children of t and u and if p is a child of t the set $I_{p,\chi(p)}$ contains $\phi|_{B_p}$ by the inductive hypothesis. Thus the subprocedure ***matchChildren*** accepts $\phi|_{B_t}$ and it is included in $I_{t,u}$.

Finally, we assume $\phi' \in I_{t,u}$ for some fixed ϕ'. Then it was accepted by ***matchChildren***, which implies that there is a matching χ' between the children of t and u and corresponding partial isomorphisms on level $h+1$. We use the inductive hypothesis to extend χ' to an isomorphism χ from (T_t, t) to (U_u, u) and ϕ' to an isomorphisms ϕ from $G_1[\bigcup_{p \in V(T_t)} B_p]$ to $G_2[\bigcup_{q \in V(U_u)} C_q]$. This finalizes our induction and we use the proved claim for $I_{r,s}$ to conclude that the algorithm is correct.

Algorithm 6.7. Graph isomorphism for graphs of bounded rooted tree distance width [YBFT99]

Input : Connected graphs G_1,G_2
Output : Are G_1 and G_2 isomorphic?

```
1  if (|V(G1)|≠|V(G2)|)
2  |   return "not isomorphic"
3  (B,T,r)←(null ,null ,null )
4  for v ∈ V(G1)
5  |   (B',T',r') ← minTreeDistDec(G1,{v})
6  |   if (width((B',T',r')) < width((B,T,r)) ∨ T = null )
7  |   |   (B,T,r) ← (B',T',r')
8  end
9  N ← pathDistanceDec(T,{r})
10 for v ∈ V(G2)
11 |   (C,U,s) ← minTreeDistDec(G2,{v})
12 |   if (width((B,T,r)) ≠ width((C,U,s)) ∨ T ≇ U)
13 |   |   continue
14 |   M ← pathDistanceDec(U,{s})
15 |   for (t,u) ∈ N_{|N|-1} × M_{|N|-1}
16 |   |   I_{t,u} ← {φ | G1[B_t] ≅_φ G2[C_u]}
17 |   end
18 |   for h←|N| − 2 down to 0
19 |   |   for (t,u) ∈ N_h × M_h
20 |   |   |   if (|N_T(t)∩ N_{h+1}| ≠ |N_U(u)∩ M_{h+1}|)
21 |   |   |   |   I_{t,u} ← ∅
22 |   |   |   |   continue
23 |   |   |   for bijections φ : B_t → C_u
24 |   |   |   |   I_{t,u} ← I_{t,u} ∪ matchChildren(G1,G2,(B,T,r),(C,U,s),M,N,h,t,u,φ)
25 |   |   |   end
26 |   |   end
27 |   end
28 |   if (|I_{r,s}| ≠ 0)
29 |   |   return "isomorphic"
30 end
31 return "not isomorphic"

32 subprocedure matchChildren(G1,G2,(B,T,r),(C,U,s),M,N,h,t,u,φ)
33 |   for p ∈ N_T(t)∩ N_{h+1}
34 |   |   for unmatched q ∈ N_U(u)∩ M_{h+1}
35 |   |   |   if (∃ ψ ∈ I_{p,q} : G1[B_t∪ B_p] ≅_{φ∪ψ} G2[C_u∪ C_q])
36 |   |   |   |   match ← match ∪ {(p,q)}
37 |   |   end
38 |   end
39 |   if (|match|=|N_T(t)∩ N_{h+1}|^2)
40 |   |   return {φ}
41 |   else
42 |   |   return ∅
43 end
```

Runtime: The test of the condition in line 35 takes time $\mathcal{O}((k!)4k^2)$ ($k!$ tests, whether the union of functions is an isomorphism on graphs with at most $2k$ vertices) and is performed at most $|N_T(t) \cap N_{h+1}|^2$ times. Each pair of bags in the distance levels N_{h+1} and M_{h+1} respectively will be considered $k!$ times as children to be possibly matched during the for-loop starting in line 19: once for each bijection between their parents (line 23). Thus each iteration (for level h) of the loop starting in line 18 takes time $\mathcal{O}(k!^2k^2|N_{h+1}|^2)$ and the entire loop runs in $\mathcal{O}(k!^2k^2|V(G_1)|^2)$. Since there is again one rooted tree distance decomposition for each vertex of G_2, we have proved the claimed runtime. □

6.3.2. Canonical Labelings

Similar to section 6.1.2, we will subsequently outline how to find a canonical labeling for a pair of a graph G and (a root bag of) a minimal tree distance decomposition of it. Technically, we will interpret the membership in the root bag as a color that has to be preserved. What makes the canonization of such pairs harder than the canonization of path distance decompositions, is the number of different orders of child bags and that there is no upper bound on the number of children of a bag. This is also the reason why we only encode the root bag as color, as opposed to each level for the path distance decompositions.

Luckily for us, [DTW12] shows that Canonical Labeling(tdw $= k$) is in XL (see definition 1.26) by providing a logspace canonical labeling algorithm for graphs of bounded tree distance width using a modification of Lindell's isomorphism order [Lin92]. But this recursive algorithm has a call tree with height $\Theta(\log |V(G)|)$ and inner vertices of degree up to $\text{tdw}(G)!$ and hence no runtime bounded by a polynomial in $|V(G)|$ with degree not depending on $\text{tdw}(G)$. Nevertheless the isomorphism order computed within this algorithm enables us to sort the children of a bag and to turn the top-down logspace algorithm from [DTW12] into a bottom-up algorithm. We will only be able to use it for bounded rpdw, as the enumeration of all root bags of size k is possible in XL, but not in FPT.

Let $(G, (B, T, r))$ be a pair of a graph G and a minimal tree distance decomposition (B, T, r) of G and assume that the width of (B, T, r) is k. Assume there is a canonical form $\mathtt{C}$ for graph-decomposition pairs, whose decomposition tree has height less than the height of T. Further let $c1, \dots, c_l$ be the children of r in T. Assume we have a set S_i for each $c_i, i \in [l]$ that contains the canonical labelings ϕ of $G([\bigcup_{t \in T_{c_i}} B_t], B_{c_i})$ w.r.t. $\mathtt{C}$, where T_{c_i} is the subtree of T rooted in c_i as in lemma 6.15.

We now fix a labeling $\chi|_{B_r} : B_r \to [|B_r|]$ of B_r and describe how to construct a labeling χ of (G, B_r). To achieve this, we first define a sort order $\trianglelefteq$ on the children of r (using the ideas from [DTW12]).

Definition 6.18. We use the names and notation from the above paragraphs. Let $c_1, \dots, c_l$ be the order of the children of r after we sorted them according to $\trianglelefteq$, then $i < j$ if:

1. $n_i < n_j$, where n_i is the set $B_r \cap N(B_{c_i})$ seen as a bit field ($n_i \in \{0,1\}^{|B_r|}$) whose entries are sorted according to $\chi|_{B_r}$ or $n_i = n_j$ and
2. $a_i < a_j$, where a_i is the minimal adjacency matrix of $G[B_{c_i} \cup (B_r \cap N(B_{c_i}))]$ (sorted by $\chi|_{B_r}$ and ϕ) over all $\phi \in S_i$ or $a_i = a_j$ and
3. the canonical form $\mathtt{C}(G[\bigcup_{t \in V(T_{c_i})} B_t], B_{c_i})$ has a lexicographically smaller adjacency matrix then the canonical form corresponding to T_{c_j}.

(Read this as if a left parenthesis is placed after every "or".) •

Using this, we now define a continuation of the given restriction and furthermore a canonical form over all restrictions.

Definition 6.19. Arbitrarily choose some $\phi_i \in S_i$ for $i \in [l]$. Then χ shall be defined w.r.t. $\chi|_{B_r}$ as (+ denotes a constant shift)

$$\chi = \chi|_{B_r} \cup \bigcup_{i\in[l]} (\phi_i + s_{i-1}), \quad \text{where } s_i = \begin{cases} |B_r| & i = 0 \\ |\bigcup\{B_t \mid t \in V(T_{c_i})| & i \in [l] \end{cases}.$$

Let $\chi_{\min}$ be a labeling χ constructed in this way such that the adjacency matrix of the graph $(\chi(V(G)), \chi(E(G)))$ gets minimal. We now assume that the canonical form $\mathtt{C}$ mentioned above was recursively constructed such that

$$\mathtt{C}(G, B_r) = ((\chi_{\min}(V(G)), \chi_{\min}(E(G))), \chi_{\min}(B_r)) .$$

•

Lemma 6.20. $\mathtt{C}$ *is a canonical form for pairs* (G, B_r)*, where* G *is a graph and* $B_r \subseteq V(G)$*.*

Proof. Let (B, T, r) be the unique minimal tree distance decomposition of G with root set B_r (up to trees isomorphic to T). We prove the lemma by induction on the height of T. If T has height 0 then r has no children in T and $B_r = V(G)$. Thus $\chi_{\min}$ minimizes $\mathrm{adj}(G)$ over all $\phi \in \mathrm{Sym}(V(G))$, which makes $\mathtt{C}(G, B_r)$ canonical.

Now assume that the height of T is greater than 0 and thus $\mathtt{C}$ canonical for (G_i, T_{c_i}, r), where $c_1, \ldots, c_l$ are the children of r in T and $G_i = G[\bigcup_{t\in V(T_{c_i})} B_t]$. Let (H, C_r) be an isomorphic pair, i.e. there is a ϕ such that $G \cong_\psi H$ and $\psi(B_r) = C_r$. Furthermore, let (C, T, r) the corresponding minimal tree distance decomposition (note that we reuse T and r). Let $\chi|_{B_r}$ and $\chi'|_{C_r}$ be labelings of B_r and C_r respectively, such that $\chi|_{B_r} = \chi'|_{C_r} \circ \psi|_{B_r}$. Assume that the children $c_1, \ldots, c_l$ of r are sorted according to $\trianglelefteq$ for (G, B_r) with labeling $\chi|_{B_r}$ and $d_1, \ldots, d_l$ are the children of r sorted according to $\trianglelefteq$ for (H, C_r) with labeling $\chi'|_{C_r}$ and define $H_i = H[\bigcup_{t\in V(T_{d_i})} C_t]$. We will now show that the following propositions hold:

$$\psi(B_r \cap N(B_{c_i})) = C_r \cap N(C_{d_i}) \tag{one}$$

$$G[V(G_i) \cup (B_r \cap N(c_i))] \cong_\phi H[V(H_i) \cup (C_r \cap N(d_i))]$$
$$\text{such that} \quad \phi|_{B_r\cap N(B_{c_i})} = \psi|_{B_r\cap N(B_{c_i})} \quad \text{and} \quad \phi(B_{c_i}) = \phi(C_{d_i}) \tag{two}$$

$$T_{c_i} \cong T_{d_i} \tag{three}$$

The first proposition directly follows from the first condition in the definition of $\trianglelefteq$ and the connection between $\chi|_{B_r}$ and $\chi'|_{C_r}$. Since we assumed (G, B_r) and (H, C_r) to be isomorphic and we have mapped the neighborhood of B_{c_i} in the root bag to the one of C_{d_i}, using the first proposition and condition two of $\trianglelefteq$ sorts isomorphism types of $G[B_{c_i} \cup (B_r \cap N(c_i))]$ (and similar for H) consistently with the third condition (membership in S_i ensures this), proposition two follows. Proposition three also follows easily from condition three of $\trianglelefteq$.

Now assume χ' is defined for (H, C_r) analogously to χ (using the indices of $d_1, \ldots, d_l$). Looking at the definition of χ', it is now easy to see that we may alter the isomorphism ψ such that $\psi(B_{c_i}) = C_{d_i}$ (using ϕ from proposition two), subsequently alter the bag function C such

that $\psi(B_{c_i}) = C_{c_i}$ and finally redefine χ' using the indices of $c_1, \ldots, c_l$, all without changing the result of the application of χ' to (H, C_r). By the inductive hypothesis and the condition $\chi|_{B_r} = \chi'|_{C_r} \circ \psi|_{B_r}$, the labelings χ and χ' (old and new) produce the same canonical form if applied to (G, B_r) and (H, C_r), respectively. So ψ describes a one-to-one mapping of canonical forms and their lexicographic minima therefore coincide, making $\mathtt{C}$ a canonical form. □

The canonical form $\mathtt{C}$ directly translates into an algorithm. The only thing we have to do to achieve FPT-time is to replace the sets S_i by some subset of them, but since we only use restrictions of their elements for the sort order and only a single element for the construction of a labeling, this can be done easily. Algorithm 6.8 is the result of these thoughts and we will prove the following theorem about it.

Theorem 6.21. *The algorithm 6.8 computes a canonical labeling w.r.t.* $\mathtt{C}$ *from definition 6.19 for a pair* (G, R) *in time* $\mathcal{O}((k+1)!^2|V(G)|^3)$ *where k is the width of the minimal tree distance decomposition of a graph G with root set R. Hence* Canonical Labeling(rpdw) *is fixed-parameter tractable.*

Proof. If we ignore the change from the canonical labeling cosets S_i (see beginning of the section) to the sets I_t, the function ***canon*** together with the for-loop starting in line 7 directly implements the definition of $\mathtt{C}$, provided it did so for all subtrees. For the leaves of T (as defined in line 1) only the adjacency matrix is computed by ***canon*** and thus the entire loop computes the canon for leaf bags. So it remains to show that we can use the sets I_t instead of canonical labeling cosets. Since the for-loop starting in line 7 iterates over all possible labelings of a single bag, we ensure that all possible restrictions of canonical labelings are available, when the bag in question is considered again as a child bag of its parent. We underpin this by the observation that we could construct the canonical labeling coset by recursively combining each $\phi \in I_t$ with combination of members in I'_u over all children u of t (keep in mind that I'_u depends on ϕ).

The runtime of most parts is annotated in the comments of algorithm 6.8. We single out the sort calls in line 5. Because each bag is child bag exactly once and the canon of a a graph induced by the bags of a subtree has length at most $|V(G)|^2$, we need $\mathcal{O}(k|V(G)|^3)$ operations to sort (using radix sort) in line 5 over all iterations of the two outermost for-loops. The runtime of the other operations can be easily bounded from above by multiplying the annotated runtimes.

A canonical labeling for a graph G can be computed by choosing the labeling minimizing $\mathtt{C}(G, \{v\})$ over $v \in V(G)$. □

The previous proof clearly underpins that the costly part of algorithm 6.8 is its usage of large adjacency matrices. So it seems natural to seek for an application of the principles used in algorithm 2.3, i.e. not using a real "global" canon, but a mapping of isomorphic subtrees to integers that is only valid on each distance level w.r.t. the root bag. Given a minimal tree distance decomposition (B, T, r), the idea behind algorithm 6.9 is to keep all parts of the order and the canon that do not recursively depend on $\mathtt{C}$ and to compute a minimal color of a vertex $t \in V(T)$ over all possible labelings of B_t instead of the canon corresponding to the subtree T_t. We now prove the following theorem about it.

Theorem 6.22. *The algorithm 6.9 computes a canonical labeling for a pair* (G, R) *in time* $\mathcal{O}((k+1)!^2|V(G)|)$ *where k is the width of the minimal tree distance decomposition of a graph G with root set R.*

Algorithm 6.8. Canonical labeling for minimal tree distance decompositions

Input : Connected graph G and $R \subseteq V(G)$
Output : Canonical labeling of (G, R)

```
1  (B, T, r) ← minTreeDistDec(G, R)
2  N ← pathDistanceDec(T, {r})
3  for i←|N| − 1 down to 0
4  |   for t ∈ N_i    //O(|V(G)|) iterations (both outer loops combined)
5  |   |   presort u ∈ N_T(t) ∩ N_{i+1} according to C(u)
6  |   |   C(t) ← null
7  |   |   for bijective φ ∈ [|B_t|]^{B_t}    //O(k!) iterations
8  |   |   |   (ψ, a) ← canon(i, t, φ)
9  |   |   |   if (a < C(t) ∨ C(t) = null )
10 |   |   |   |   C(t) ← a
11 |   |   |   |   I_t ← {ψ}
12 |   |   |   else if (a = C(t))
13 |   |   |   |   I_t ← I_t ∪ {ψ}
14 |   |   end
15 |   end
16 end
17 return some element from I_r

18 subprocedure canon(i, t, φ)    //O(k!k²|V(G_t)|²)
19 |   for u ∈ N_T(t) ∩ N_{i+1}    //O(c) iterations, c = |N_T(t) ∩ N_{i+1}|
20 |   |   n_u ← string according to the first condition of ⊴    //O(k²)
21 |   |   a_u ← min {adj(H) | ∃φ' H=(φ'(V'), φ'(E(G[V']))), V'=B_u ∪ (B_t ∩ N_G(B_u)),
   |   |       ∃ψ ∈ I_u : φ'=φ ∪ (ψ|_{B_u} + |φ|)}    //O(k!k²)
22 |   |   I'_u ← {ψ ∈ I_u | adj(H) = a_u, (adj(H) computed as above)}    //O(k!k²)
23 |   end
   |   //O(ck²) (lemma 2.24):
24 |   sort u ∈ N_T(t) ∩ N_{i+1} according to (n_u, |I_u|, a_u, |I'_u|, C(u)) (using the list from line 5)
25 |   s ← |φ|
26 |   for u ∈ N_T(t) ∩ N_{i+1}    //O(|V(G_t)|) operations
27 |   |   ψ_u ← arbitrary element from I'_u
28 |   |   φ ← φ ∪ (ψ_u + s)
29 |   |   s ← s + |ψ_u|
30 |   end
31 |   G_t ← G[⋃_{u∈V(T_t)} B_u]
32 |   a ← adj(φ(V(G_t), φ(E(G_t))    //O(|V(G_t)|²)
33 |   return (φ, a)
34 end
```

Proof. We use the names and notation from algorithm 6.9. The correctness proof is basically a combination of the proofs for theorems 6.21 and 6.5 (canonical labeling for path distance decompositions), lemma 2.25 and corollary 2.26 (linear time canonical labeling for trees). Whenever two vertices t_1 and t_2 in the same level N_i have the same $\text{inv}(t_1) = \text{inv}(t_2)$ in line 12, their bags are isomorphic and any child bags connect to subsets of the respective bags that are invariant under isomorphism. If the jth child bag of t_1 has neighbors $V' \subseteq B_{t_1}$ in

Algorithm 6.9. Canonical labeling for minimal tree distance decompositions

Input : Connected graph G and $R \subseteq V(G)$
Output : Canonical labeling of (G, R)

1 $(B, T, r) \leftarrow \text{minTreeDistDec}(G, R)$; $N \leftarrow \text{pathDistanceDec}(T, \{r\})$
2 **for** $i \leftarrow |N| - 1$ **down to** 0
3 | **for** $t \in N_i$ //$\mathcal{O}(|V(G)|)$ iterations (both outer loops combined)
4 | | $\text{inv}(t) \leftarrow$ `null`
5 | | **for** bijective $\phi \in [|B_t|]^{B_t}$ //$\mathcal{O}(k!)$ iterations
6 | | | $a \leftarrow \text{invar}(i, t, \phi)$
7 | | | **if** $(a < \text{inv}(t) \vee \text{inv}(t) =$ `null` $)$
8 | | | | $\text{inv}(t) \leftarrow a$
9 | | | | $I_t \leftarrow \{\phi\}$
10 | | | **else if** $(a = \text{inv}(t))$
11 | | | | $I_t \leftarrow I_t \cup \{\phi\}$
12 | | **end**
13 | **end**
14 | $s \leftarrow \text{sortLexico}(\text{inv}(N_i))$ (also sorting N_i) //$\mathcal{O}(k^2(|N_i| + |N_{i+1}|))$ (lemma 2.24)
15 | **for** $t \in N_i$ (in sort order)
16 | | $\text{isot}(t) \leftarrow s^{-1}(\text{inv}(t))$
17 | | add t to presort list $L_{t'}$ (t' is parent of t in the rooted tree (T, r))
18 | **end**
19 **end**
20 **return** $\text{canlab}(0, r, 0, \phi)$ where $\phi \in I_r$
21 **subprocedure** $\text{invar}(i, t, \phi)$ //$\mathcal{O}(k!k^2|V(G_t)|)$
22 | **for** $u \in N_T(t) \cap N_{i+1}$ //$\mathcal{O}(c)$ iterations, $c = |N_T(t) \cap N_{i+1}|$
23 | | $n_u \leftarrow$ string according to the first condition of $\trianglelefteq$ //$\mathcal{O}(k^2)$
24 | | $a_u \leftarrow \min \{\text{adj}(H) \mid \exists \phi'\ H{=}(\phi'(V'), \phi'(E(G[V'])))$, $V'{=}B_u \cup (B_t \cap N_G(B_u))$, $\exists \psi \in I_u : \phi'{=}\phi \cup (\psi + |\phi|)\}$ //$\mathcal{O}(k!k^2)$
25 | | $I'_u \leftarrow \{\psi \in I_u \mid \text{adj}(H) = a_u,$ (adj(H) computed as above$)\}$ //$\mathcal{O}(k!k^2)$
26 | **end**
27 | sort $u \in N_T(t) \cap N_{i+1}$ according to $(n_u, a_u, \text{isot}(u))$ using L_t (see line 17)
28 | $u_1, \ldots, u_l \leftarrow N_T(t) \cap N_{i+1}$ (as sorted above)
29 | $a \leftarrow (\text{adj}(\phi(B_t), \phi(E(G[B_t]))), (n_{u,1}, a_{u,1}, \text{isot}(u_1)), \ldots, (n_{u,l}, a_{u,l}, \text{isot}(u_l)))$
30 | **return** (ϕ, a)
31 **end**
32 **subprocedure** $\text{canlab}(i, t, p, \phi)$
33 | $\text{invar}(i, t, \phi)$ without the last three lines
34 | $\phi \leftarrow \phi + p$ //constant shift
35 | $p \leftarrow p + |\phi|$
36 | **for** $u \in N_T(t) \cap N_{i+1}$
37 | | $\psi \leftarrow$ arbitrary element from I'_u
38 | | $\psi \leftarrow \text{canlab}(i + 1, u, p, \psi)$
39 | | $p \leftarrow p + |\psi|$
40 | | $\phi \leftarrow \phi \cup \psi$
41 | **end**
42 | **return** ϕ
43 **end**

t_1's bag then the jth child bag of t_2 has neighbors $\pi(V') \subseteq B_{t_2}$ where π is the isomorphism between the bags. Furthermore, if $\text{inv}(t_1) = \text{inv}(t_2)$, then the interaction a_u (line 24) between each jth child bag and B_{t_1}/B_{t_2} is identical as well the isomorphism type $\text{isot}(u)$. From here on we argue as in lemma 2.25 (regarding the recoloring, i.e. isot) and theorem 6.5 regarding extendible canonical labelings.

Again, the runtime is mostly annotated in the algorithm. Observe that the call to ***sortLexico*** in the main algorithm has only overhead k^2 compared to algorithm 2.3 and that the sorting step in the subprocedure ***invar*** only has to sort elements with size bounded by k^2, because we start with a presorted list each time and use a stable sort procedure for the other two positions afterwards. □

Corollary 6.23. *For two graphs G_1 and G_2 with rooted tree distance width k and $|V(G_1)| = |V(G_2)|$, we can compute a canonical labeling (thus decide* Graph Isomorphism*) of each graph in time $\mathcal{O}((k+1)!^2|V(G_1)|^2)$.* □

We close this section with a question, asking whether this finding and the result of [DTW12] can be further generalized to meet both of the requirements logspace and weak interdependence between parameter and size simultaneously.

Question 6.24. Is Canonical Labeling(rpdw) in para-L? **?**

6.4. c-connected d-separating tree distance width

In the same way as the rooted path distance width can be generalized to the c-connected path distance width, we would like to generalize the rooted tree distance width. There is, however, a difference that prevents this plan. While a level in a path distance decomposition has only two neighboring levels, a bag in a tree distance decomposition may have any number of child bags. Thus the crucial termination condition in line 1 of algorithm 6.3, which ensures that the first two levels combined have size smaller than $2k$, is not valid for arbitrary connected root bags. But we can easily state a similar root bag enumeration algorithm, if we require the root bag to have only d child bags for a fixed integer d. Then every appearance of $2k$ in the proof of theorem 6.10 just has to be replaced by $(d+1)k$, where k is the candidate for the connected path distance width, or a similar parameter for tree distance decompositions, respectively. Noticing that the only appearance of d is as a factor before k, we may even let d depend on k, that is, choose a ***function*** $d : \mathbb{N} \to \mathbb{N}$ instead of a fixed integer.

We are now close to the definition of such a parameter, but the condition "has at most $d(k)$ children in its minimal tree distance decomposition" is a bit cumbersome. Luckily, we have the nice characterization of minimal tree distance decompositions from lemma 6.15. (B, T, r) is a minimal tree distance decomposition if for all $t \in V(T)$ the graph $G[\bigcup_{j \in V(T_t)} B_j]$ is connected. Hence the number of children of a vertex $t \in V(T)$ is exactly the number of components in the graph $G[\bigcup_{j \in V(T_t) \setminus \{t\}} B_j]$. If t is the root r then this graph is nothing but $G \setminus B_r$, which gives us a shorter characterization for our desired criterion.

Definition 6.25. Let d be any function $d : \mathbb{N} \to \mathbb{N}$, $c \in \mathbb{N}$ and let G be a graph. The ***c-connected d-separating tree distance width*** of G ((c,d)-cstdw(G)) is the minimal k such that there is a tree distance decomposition (B, T, r) of width k such that $G[B_r]$ has at most c components and $G \setminus B_r$ has at most $d(k)$ components or $|B_r| \leq c$. •

The last alternative condition allows our new parameter to cover the rooted tree distance width.

Theorem 6.26. *Let d be any computable function $d : \mathbb{N} \to \mathbb{N}$ and let $c \in \mathbb{N}$. Then* Graph Isomorphism$((c,d)\text{-cstdw} = k)$ *and* Canonical Labeling$((c,d)\text{-cstdw} = k)$ *are fixed-parameter tractable.*

Proof. Since d is a computable function, $d(k)$ is computable in a computable time (e.g. use a counter) which only depends on k. Hence we simply compute $d(k)$ for an input pair (G,k) (for Canonical Labeling) and use this value instead of k in line 1 of algorithm 6.3. Of course we also change all computations of a path distance decomposition to minimal tree distance decompositions. Instead of calling the algorithm with neighborhoods of maximal sizes $2k-1$ to k, the maximal size of the neighborhood of R goes from $(d(k)+1)k-1$ to $d(k)k$ during a descent in the recursive call tree. Hence the overall runtime of the modified algorithm 6.3 is $\mathcal{O}(\frac{(d(k)k+k)!}{(d(k)k)!}|V(G)|)$, because a minimal tree distance decomposition is also computable in $\mathcal{O}(k|V(G)|)$ (lemmas 6.16 and 6.14). As usual an isomorphism or canonical labeling algorithm iterates over the enumerated decompositions and our proof is completed. □

7. Tree-depth

We already discussed a variety of tree-like structures and graph parameters accompanied by them. In this chapter we will treat yet another parameter related to trees, the ***tree-depth***. The tree-depth may be defined in many ways: e.g. via so called ordered colorings/vertex rankings [KMS95; Bod+98], via centered colorings [NO06] and so called tree-depth decompositions [BDK12; NO06].

Bouland, Dawar and Kopczyński [BDK12] construct a canonization algorithm for tree-depth decompositions using an extension of Lindell's isomorphism order for trees [Lin92] and apply this to graphs of bounded tree-depths. This approach only works because tree-depth decompositions are computable in FPT-time w.r.t the tree-depth and because there are (in some sense) not to many of them for a given input graph. These constraints determine the structure of this chapter. At first, we give some definitions of tree-depth and show their equivalence, then we explore the fixed-parameter tractability of the Tree-Depth Decomposition problem via a forbidden subgraph characterization of graphs with bounded tree-depth and finally, we turn our attention to the number of choices during the construction of a tree-depth decomposition and a Canonical Labeling algorithm using the isomorphism order mentioned above.

7.1. Some equivalent definitions

Definition 7.1. Let G be a graph and let $F = (V(F), E(F), (r_1, \dots, r_l))$ be a ***rooted forest*** with components $C_1, \dots, C_l$ of $(V(F), E(F))$, $V(F) = V(G)$ and $r_i \in C_i$ for all $i \in [l]$. Further let $\mathrm{subt}(F, t)$ be the tree $(S \cup \{t\}, F[S \cup \{t\}])$, where $t \in C_i$ and S is the set of vertices not reachable from r_i in $F[C_i] - t$. The ***closure*** of F is the graph

$$\mathrm{clos}(F) = (V(F), \{\{t, u\} \in V(F) \mid \exists i \exists \text{simple path } r_i, \dots, t, \dots, u \text{ in } F\}) .$$

We call F a ***tree-depth decomposition*** of G if for all $t \in V(F)$ the graph $G[V(\mathrm{subt}(F, t))]$ is connected and

$$E(G) \subseteq E(\mathrm{clos}(F)) .$$

The height of F is the the maximal length of a simple root-leaf path over all C_i. The ***tree-depth*** of G ($\mathrm{td}(G)$) is the minimal height over all tree-depth decompositions of G plus 1. •

The connected-subtrees-criterion will be helpful, but it is not necessary for the definition of tree-depth.

Lemma 7.2. *Let G be a graph and $F = (V(F), E(F), (r_1, \dots, r_l))$ be a rooted forest with height $k - 1$ and $V(G) = V(F)$ such that*

$$E(G) \subseteq E(\mathrm{clos}(F)) .$$

Then $\mathrm{td}(G) \leq k$.

Proof. Analogously to lemma 6.15 (minimal tree distance decompositions), we may alter an arbitrary rooted forest whose closure is a supergraph of G such that it becomes a tree-depth decomposition. If for some $t \in V(F)$ the graph $G[V(\text{subt}(F,t))]$ is not connected, we choose the lowest such t, i.e. such that $G[V(\text{subt}(F,t'))]$ is connected for all $t' \in V(\text{subt}(F,t)$. Since the subtree for any child t' of t is connected in G, one such child is not incident to t in G, otherwise $G[V(\text{subt}(F,t))]$ would be connected, too. So we simply make all non-incident children t' of t children of the parent of t (this parent exists, because each component has its own root). If it is iteratively applied in decreasing distance to the root of each component, this procedure eliminates all violations of the condition requiring connected subtrees, without increasing the height of each rooted tree. □

We look at yet another possible definition of tree-depth, that does not show a relation to trees at the first glance. But the subsequent proof shows that the "recursion tree" of this inductive definition directly corresponds to a tree-depth decomposition.

Lemma 7.3 ([NO06]). *Let G be a graph with connected components $C_1, \dots, C_l$ then*

$$\text{td}(G) = \begin{cases} 1 & |V(G)| = 1 \\ 1 + \min_{v \in V(G)} \text{td}(G - v) & l = 1 \wedge |V(G)| > 1 \\ \max_{i \in [l]} \text{td}(G[C_i]) & l > 1 \end{cases} .$$

Proof. We will prove this by induction on $\text{td}(G)$ for an arbitrary graph G. If $\text{td}(G) = 1$ then there is a tree-depth decomposition of G that has height 0 and thus each of its components is a single vertex. Hence each of its components has tree-depth 1 and the proposition is fulfilled either for the first or the third case.

Assume that $\text{td}(G) > 1$ and that G is disconnected. Then a tree-depth decomposition of height $\text{td}(G) - 1$ of G is a tree-depth decomposition for all of its components $C_1, \dots, C_l$, if properly restricted. So $\text{td}(G[Ci]) \leq \text{td}(G)$ for all $i \in [l]$. If, on the other hand, $\text{td}(G[Ci]) < \text{td}(G)$ for all $i \in [l]$, then we could replace the decomposition of all components by one with height less than $\text{td}(G) - 1$ and thus at for at least one component C_i we have $\text{td}(G[Ci]) = \text{td}(G)$.

If G is connected this argument works nearly analogous over all possible roots $r \in V(G)$ of a tree-depth decomposition F. If we remove the root r, $G - r$ has components $C_1, \dots, C_l$ and we already know that $\text{td}(G - r) = \max_{i \in [l]} \text{td}(G[C_i])$. Each component corresponds to a child of r and thus we can use a tree-depth decomposition of minimal height for each component and height of F is $\max_{i \in [l]} \text{td}(G[C_i]) = \text{td}(G - r)$. If we now choose r such that $\text{td}(G - r)$ is minimal, we get $\text{td}(G) = \text{td}(G - r) + 1$. □

7.2. Computation of tree-depth and decompositions

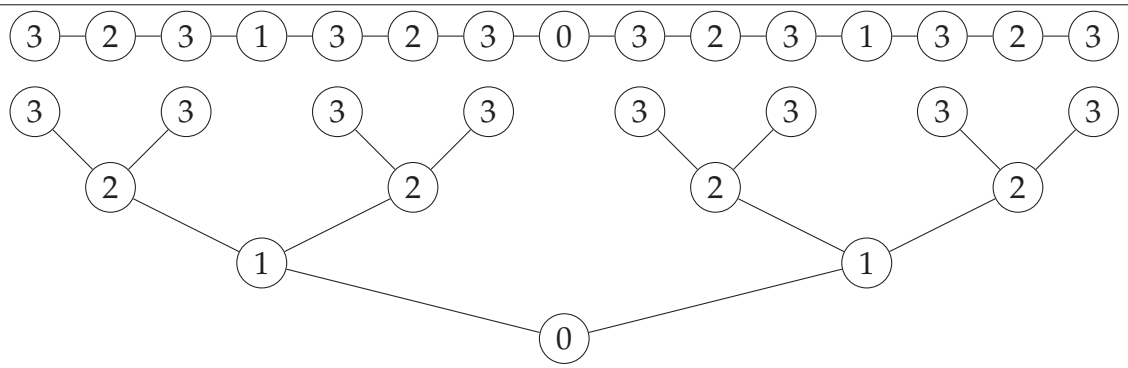
Despite the very simple notion of tree-depth there is unfortunately no known ***simple*** algorithm to compute it in FPT-time. There are, however, two linear time algorithms to test for $\text{td}(G) = k$ for a fixed k. The approach from [Bod+98] works by using graph minors and the Robertson–Seymour theorem [RS04]. In contrast to this, the algorithm from [NO12] is constructive, but requires Courcelle's theorem [Cou90], which states that properties definable in first-order logic can be tested in linear time on graphs of bounded treewidth (actually for

arbitrary structures, not only graphs). Luckily, such a formula is constructed in [NO12, Exercise 6.6]. Instead of this formula, we will use a result from [DGT12], which ensures that the graphs of tree-depth k can be characterized by a set of forbidden subgraphs with at most $2^{2^{k-1}}$ vertices. This easily translates into a first order formula and has a fundamental further application discussed in the next section.

This section is structured in reverse order of the preceding paragraph, i.e. we will first look at the forbidden subgraph characterization, then discuss Courcelle's theorem and why we can even use it ($\mathrm{tw}(G) \leq \mathrm{td}(G)$) and conclude with the theorem from [NO12].

7.2.1. Characterization via forbidden subgraphs

Figure 7.1. A path of length 14 and a tree-depth decomposition of it: vertices are colored with their depth in the tree-depth decomposition.



Lemma 7.4 ([NO06, Equation 6.2])**.** *Let G be a graph having a path P_n of length $n-1$ as its subgraph. Then* $\mathrm{td}(G) \geq \log(n) - 1$.

Proof. If $n \in [4]$ then the proposition holds even without the final -1, so assume n is minimal such that it does not hold. Then $\mathrm{td}(G) \geq \mathrm{td}(G[C])$, where C is the component of G that contains the P_n, so we can assume that G is connected. Using the second case of lemma 7.3 we see that whichever vertex v we remove from G, at least one component of $G - v$ contains a $P_{n'}$ such that $n' \geq \lfloor \frac{n}{2} \rfloor$. If n does not have the form $2^l + 1$ for some $l \in \mathbb{N}$ then $\log(n) = \log(n') + 1$ and as $\mathrm{td}(G) \geq \mathrm{td}(P_{n'}) + 1 \geq \log(n') - 1 + 1$ the claim follows. If $n = 2^l + 1$ then n' cannot have this form, because $n' = 2^{l-1}$ and $n \geq 6$. Thus we repeat the proof for n' such that n' is a power of 2 and the stronger claim $\mathrm{td}(P_{n'}) \geq \log(n)$ and use this stronger result together with $\log(n) = \log(n') + 2$. □

The last lemma already brings us one kind of forbidden subgraphs (see figure 7.1 for an example) for graphs of bounded tree-depth, but it also ensures that other forbidden subgraphs cannot be to far away from each other in a component that contains no long path. This observation is one of the key building blocks of the following theorem.

Theorem 7.5 ([DGT12])**.** *Let G be a graph such that* $\mathrm{td}(G) > k$*, where $k \in \mathbb{N}$, then G contains a connected subgraph F such that* $|V(F)| \leq 2^{2^{k-1}}$ *and* $\mathrm{td}(F) > k$.

Proof. For $k = 1$ the minimal forbidden graph F is the K_2, i.e. a single edge. If $k = 2$ a graph with $\mathrm{td}(G) > 2$ either a contains a path or a cycle of length 3, otherwise it would be a disjoint

union of stars. Now let k be minimal such that the theorem does not hold. Since the subgraph relation is transitive, we may assume that G is connected and $\mathrm{td}(G) = k+1$, otherwise we argue for a subgraph with this property. If G contains a path of length $2^k - 1$ as a subgraph, we may choose this path as our forbidden subgraph F by lemma 7.4 (stronger argument for n'), otherwise G has diameter $2^k - 2$.

Because k is a minimal counterexample, G contains a subgraph F' with $|V(F')| \leq 2^{2^{k-2}}$ and $\mathrm{td}(F') \geq k$. Set $\{v_1, \ldots, v_l\} = V(F')$. For each v_i, $i \in [l]$ the tree-depth decreases by at most one, if we delete it and hence $\mathrm{td}(G - v_i) \geq k$. We use the minimality of k again and conclude that there are subgraphs F_i of $G - v_i$ such that $\mathrm{td}(F_i) \geq k$ and $|V(F_i)| \leq 2^{2^{k-2}}$.

Now look at the union graph $F = (V(F') \cup \bigcup_{i\in[l]} V(F_i), E(F') \cup \bigcup_{i\in[l]} E(F_i))$ of all forbidden subgraphs. It has at most $l + l^2$ vertices and may not be connected. But if we assume that $\forall i \in [l] : V(F') \cap V(F_i) \neq \emptyset$, F is connected and its maximal number of vertices becomes $l^2 = 2^{2^{k-1}}$. Furthermore, for any vertex $v \in V(F)$, $F - v$ either contains F' or some F_i and thus $\mathrm{td}(F) \geq k+1$.

On the other hand, if $F' \cap F_i = \emptyset$ for some $i \in [l]$, we only need those two subgraphs to ensure the containment of either F' or F_i in $F - v$ for all $v \in V(F)$ and F being the graph induced by the union of the two vertex sets. Finally, we modify F by adding a path between the two subgraphs if needed (existent in G), but the length is still at most $2l + 2^k - 3 \leq 2^{2^{k-2}}$. □

7.2.2. Bounded treewidth and Courcelle's theorem

Unfortunately, a detailed discussion of Courcelle's theorem [Cou90] would be to long and distracting. We therefore only state the result and refer to e.g. [FG06, section 11.4]. Instead of the original theorem we use a generalization as proved in [FFG02], which helps us to reduce the runtime, when we like to find roots of tree-depth decompositions. We start by defining the problem in a way strictly tailored to our needs.

Definition 7.6. A ***graph formula*** shall be a first order formula over the signature (E), i.e. all its atomic formulas either have the form $x_1 = x_2$ or $E(x_1, x_2)$. A G-assignment for a graph G is a function from the variables (of ϕ) to $V(G)$. If β is G-assignment with $\beta(x_1) = u$ and $\beta(x_2) = v$ the atomic formula $E(x_1, x_2)$ evaluates to true under the assignment β if and only if $\{u, v\} \in E(G)$. If $(x_1, \ldots, x_l)$ are free variables of ϕ, we say $G \models \phi(a_1, \ldots, a_l)$ if $(G, \beta) \models \phi$, where $\beta = \{(x_i, a_i) \mid i \in [l]\}$. The model relation $\models$ itself is defined inductively over the structure of first order formulas as usual.

We use this to define the following parameterized problem.

		l-FO Graph Evaluation $(\mathrm{tw} = k)$
Input	:	$G \in \mathit{Graphs}$, a graph formula ϕ with l free variables and $k \in \mathbb{N}$
Parameter	:	$k + \lvert\phi\rvert$
Output	:	If $\mathrm{tw}(G) \leq k$ the set $\phi(G) = \{(a_1, \ldots, a_l) \in V(G)^l \mid G \models \phi(a_1, \ldots, a_l)\}$

•

The following theorem is a special case of the generalization of Courcelle's theorem found in [FFG02]. Note that the full version there also covers monadic second order logic and arbitrary structures.

Theorem 7.7 ([FFG02, Theorem 4.12]). *For all $l \in \mathbb{N}$, l-FO Graph Evaluation $(\mathrm{tw} = k)$ is fixed-parameter tractable (in $f(k+\phi)|V(G)|^l$ for some f).* ☑

One building block of theorem 7.7 is the ability to compute a tree decomposition of minimal width.

		Tree Decomposition
Input	:	A graph G and $k \in \mathbb{N}$
Parameter	:	k
Output	:	A tree decomposition of width k if one exists, else `null`.

Theorem 7.8 ([Bod96]). *Tree Decomposition is fixed-parameter tractable (in time $\mathcal{O}(f(k)|V(G)|)$ for some f).* ☑

What remains, is the proof of the fact that the treewidth is bounded by the tree-depth.

Lemma 7.9 ([BGHK95]). *For all graphs G the inequality $\mathrm{tw}(G) < \mathrm{td}(G)$ holds.*

Proof. Let F be a tree depth decomposition of G with height $\mathrm{td}(G) - 1$ and $\{u, v\} \in E(G)$. Then there is w.l.o.g. a simple path $r, \ldots, u \ldots, v$ in F, where r is the root of a component of F. For each $v \in F$ let

$$B_v = \{r, \ldots, v \mid r, \ldots, v \text{ is a simple path and } r \text{ is the root of the component containing } v\}.$$

Define (B, T, s) such that $V(T) = V(F) \cup \{s\}$ such that $s \notin V(F)$, $E(T) = E(V) \cup \{\{s, r_i\} \mid r_i \text{ is root of a component of } F\}$ and $B_s = \emptyset$. Then for each edge $e = \{u, v\} \in E(G)$ either $e \subseteq B_v$ or $e \subseteq B_u$ and for all vertices $v \in V(G)$, we have $B^{-1}(v) = V(\mathrm{subt}(F, v))$. Hence (B, T, s) is a tree decomposition of G having width $\mathrm{td}(G) - 1$ and thus $\mathrm{tw}(G) < \mathrm{td}(G)$. Alternatively, we may also just take the leaves of this decomposition, order them lexicographically by the path that leads to them in F (order of $V(G)$ does not matter) and join consecutive bags by an edge. We obtain a path decomposition with the same width (see chapter 8, equation (8.3.6)). □

Finally, we define the following parameterized problem and show it is in FFPT.

		Tree-Depth Decomposition
Input	:	A graph G and $k \in \mathbb{N}$
Parameter	:	k
Output	:	A tree-depth decomposition of width k if one exists, else `null`.

Theorem 7.10 ([NO12, Theorem 17.3]). *Tree-Depth Decomposition is fixed-parameter tractable.*

Proof sketch. If G is not connected, we handle each component separately, so assume G is connected. By lemma 7.9 and theorem 7.8, we may first try to compute a tree decomposition (B, T, r) of the input graph G and output `null` if $\mathrm{tw}(G) > k$. Using theorem 7.5 we further compute a set of forbidden induced subgraphs $\mathcal{F}_{k-1}$ for the class $\mathcal{T}_{k-1}$ of graphs with tree-depth at most $k - 1$. This set is used to generate a formula ϕ such that

$$G \models \phi(v) \Leftrightarrow G - v \in \mathcal{F}_{k-1}\text{-free}.$$

Theorem 7.7 then enables us to compute $\phi(G)$. If $\phi(G)$ is empty, there cannot be tree-depth decomposition of height $k-1$, because $\mathrm{td}(G) > k$ and we return `null`. In any other case we choose some member of $v \in \phi(G)$ as the (or ***a*** in case of multiple components) root of our tree-depth decomposition and recurse with $k-1$ if $k > 0$. Otherwise we add the isolated vertices of $G - v$ as leaves to our decomposition. All calls to other algorithms require only FPT-time. □

7.3. Application to Canonical Labeling$(\mathrm{td} = k)$

We already saw in many places of this work, that once we have a polynomially bounded set of tree-like structures for every graph, we can apply some modified tree isomorphism algorithm to construct an isomorphism algorithm for arbitrary graphs. Not surprisingly, [BDK12] also follows this scheme. Hence the first step is to obtain a bound on the set of tree-depth decompositions with height $\mathrm{td}(G) - 1$ for a given graph G. The kind of bound we obtain is different from chapter 6, where there was only one decomposition for a given root bag. This time, the number of decompositions for a given root remains unbounded (by $\mathrm{td}(G)$), but the set of roots of decomposition with minimal height itself is bounded. This bound follows directly from theorem 7.5.

Corollary 7.11 ([BDK12, end of section 3.1])**.** *Let G be a connected graph such that $\mathrm{td}(G) = k$. Then*

$$|\{r \mid F = (V(F), E(F), r) \text{ is a tree-depth decomposition of } G \text{ of height } k-1\}| \leq 2^{2^{k-2}}.$$

Proof. Apply theorem 7.5 for $k-1$, hence G has a subgraph F with at most $2^{2^{k-2}}$ vertices and $\mathrm{td}(F) = k$. We argue as in the proof of lemma 7.3 and observe that for any possible root r of a tree-depth decomposition $\mathrm{td}(G - r) = k - 1$. But if we remove a vertex $v \notin V(F)$, $G - v$ still contains F as a subgraph and thus $\mathrm{td}(G - v) = k$. Hence all possible roots belong to $V(F)$ and the proposition directly follows. □

Note that there is a second (or rather the main) proof for the boundedness of the set of possible roots in [BDK12], which uses yet another characterization of tree-depth via cops-and-robbers games, since the authors of [BDK12] were not aware of [DGT12] at first. We also remark, that the most important point of this corollary is, that the bound is a ***computable*** function of the tree-depth. A third possibility to proof the boundedness of the set of possible roots is to use [NO12, Lemma 6.13 and Exercise 6.6]. The lemma 6.6 there states that the class of graphs with tree-depth at most k is well-quasi-ordered by the subgraph isomorphism relation and thus definable via a finite set of forbidden (induced) subgraphs. Exercise 6.6. sketches, how to construct a first order formula to test for tree-depth $\leq k$ for a fixed k. Thus using this formula with theorem 7.7 and algorithm 4.2 (minimal forbidden subgraph) would even allow us to compute $V(F)$ (and thus $|V(F)|$) for a minimal forbidden subgraph F ***in a given input graph*** G. However, the definition of FPT requires a computable function of k alone and while [NO12, Lemma 6.13] implies that $|V(F)|$ has an upper bound solely depending on k, it does not show how to compute it.

A second remark concerns the unboundedness of the total number of tree-depth decompositions with height $\mathrm{td}(G) - 1$. For $v \in V(G)$, $G - v$ may have a number m of components $C_1, \ldots, C_m$ not bounded by $\mathrm{td}(G)$. Nevertheless, it is easy to see that we can choose roots for

tree-depth decompositions of each $G[C_i]$ independently, if we want to construct a canonical labeling for G. It only matters that the choice of each root $r_i \in C_i$ is in some sense canonical and that there is an order among the components that is not influenced by that choice.

7.3.1. An isomorphism order for subdecompositions

We now define an order among the components as demanded above, or rather an order among ***subdecompositions***. A subdecomposition is tree-depth decomposition of a subgraph augmented by the path from its root to the root of the global decomposition. The order of subdecompositions and thus components in a recursive algorithm will also depend on the previous choice of root vertices, specifically on the adjacency of the new root to all previous roots.

Definition 7.12 ([BDK12])**.** Let G and H be a graphs, $S \subseteq V(G), T \subseteq V(H)$ such that $G[S]$ and $H[T]$ are connected, $(p_1, \dots, p_l) \in V(G \setminus S)^l$, $(q_1, \dots, q_m) \in V(H \setminus T)^m$, F a tree-depth decomposition of $G[S]$ with root s and F' one of $H[T]$ with root t. We define the order $\lhd$ on subdecompositions such that

$$(G, S, (p_1, \dots, p_l), F) \lhd (H, T, (q_1, \dots, q_m), F') \quad \text{if}$$

1. $l = m$ and $\operatorname{adj}(G[\{p_1, \dots, p_l\}]) = \operatorname{adj}(H[\{q_1, \dots, q_l\}])$ (sorted according to index) and
2. $|S| < |T|$ or $|S| = |T|$ and
3. $c < d$, where c is the number of components in $G[S] - s$ and d this number in $H[T] - t$, or $c = d$ and
4. $(e_1, \dots, e_c) < (e'_1, \dots, e'_c)$, where $e_i = \mathbf{I}_{E(G)}(\{p_i, s\})$ and e'_i analogous for t or $(e_1, \dots, e_c) = (e'_1, \dots, e'_c)$ and
5. $((G, C_1, p_s, F_1), \dots, (G, C_c, p_s, F_c)) \lhd ((H, D_1, q_t, F'_1), \dots, (H, D_c, q_t, F'_c))$ lexicographically, where C_i / D_i are the components of $G[S] - s$ and $H[T] - t$ respectively, $p_s = (p_1, \dots, p_l, s)$, $q_t = q_1, \dots, q_l, t$, $F_i = F[C_i]$ (root of F_i is the topmost vertex of C_i in F), F'_i is defined analogously and the components itself are sorted according to $\trianglelefteq$ (s.b.). If the tuples for some index i are incomparable (s.b.), then the same holds for the entire pair.

If the first point of $\lhd$ is not fulfilled, two subdecompositions are incomparable. We let $\trianglelefteq$ denote a relation such that $(G, S, (p_1, \dots, p_l), F) \trianglelefteq (H, T, (q_1, \dots, q_m), F')$ if the subdecompositions are comparable, but $(H, T, (q_1, \dots, q_m), F') \not\trianglelefteq (G, S, (p_1, \dots, p_l), F)$. •

It is not hard to see that this is an isomorphism order à la Lindell [Lin92], so we will only sketch the proof of the following theorem.

Lemma 7.13 ([BDK12])**.** *Let G_1 and G_2 be graphs, then $G_1 \cong G_2$ if and only if there are tree-depth decompositions F_1 and F_2 such that $(G_1, V(G_1), \emptyset, F_1) \trianglelefteq (G_2, V(G_2), \emptyset, F_2)$ and $(G_2, V(G_2), \emptyset, F_2) \trianglelefteq (G_1, V(G_1), \emptyset, F_1)$.*

Proof sketch. Let G_1 and G_2 be two isomorphic graphs, then the existence of F_1 and F_2 such that the propositions in the lemma are fulfilled can be easily proved by induction on $\operatorname{td}(G)$. Assume that $(G_1, V(G_1), \emptyset, F_1) \trianglelefteq (G_2, V(G_2), \emptyset, F_2)$ and $(G_2, V(G_2), \emptyset, F_2) \trianglelefteq (G_1, V(G_1), \emptyset, F_1)$. Then the third condition ensures $F_1 \cong F_2$ and by the first condition this further guarantees $G[V(P_1)] \cong G[V(P_2)]$ for each pair of root-leaf paths P_1 in F_1 and P_2 in F_2. Hence $G_1 \cong G_2$ because all edges connect vertices on such a path by the definition of tree-depth decompositions. □

This order at hand we now define algorithm 7.1. Note that the computation of $\trianglelefteq$ is spread among the algorithm and used to sort subdecompositions as well to select minimal roots.

7.3.2. Canonical labeling algorithm

Algorithm 7.1. Canonical labeling via tree-depth decompositions [BDK12]: ***canonTDepthDec***

Input : Connected graph G, $S \subseteq V(G)$, $(p_1, \dots, p_l) \in V(G \setminus S)^l$ and $k \in \mathbb{N}$
Output : Canonical labeling and tree-depth decomposition of $(G, S, p_1, \dots, p_l)$

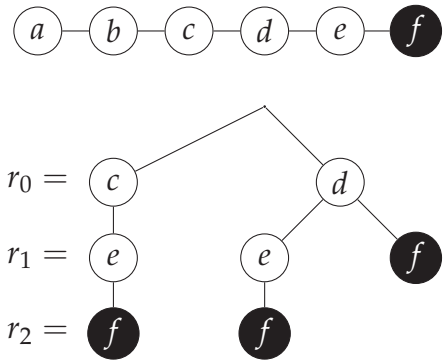
```
1  if (td(G[S])=1)    //S is singleton set S = {v_S}
2  |   return bijection φ : S → [|S|] and F←(S, ∅, v_S)
3  if td(G) > k
4  |   return null
5  if td(G) < k
6  |   k ← td(G)
7  R ← {v ∈ S | td(G[S] − v) ← k − 1}
8  R ← {v ∈ R | ∀ u ∈ R : adjStr(G, S, p_1, ..., p_l, v) ≤ adjStr(G, S, p_1, ..., p_l, u) }
9  for r ∈ R
10 |   (C_{r,1}, ..., C_{r,m}) ← components of G[S] − r
11 |   for i ∈ [m]
12 |   |   (φ_{r,i}, F_{r,i}) ← canonTDepthDec(G, C_{r,i}, (p_1, ..., p_l, r), k − 1)
13 |   end
14 |   sort the pairs (C_{r,i}, φ_{r,i}, F_{r,i}) according to ⊴
15 |   D_r ← ( (C_{r,1}, F_{r,1}, φ_{r,1}), ... , (C_{r,m}, F_{r,m}, φ_{r,m}) )
16 end
17 R ← {v ∈ R | ∀ u ∈ R : D_v ⊴ D_u}
18 choose r ∈ R;    m ← number of components of G[S] − r
19 φ ← φ_{r,0} ∪ ⋃_{i∈[m]} (φ_{r,i} + Σ_{j∈[0,i−1]} |φ_{r,j}|) where φ_{r,0} = {(r, 1)}
20 F ← (S, ⋃_{i∈[m]} ({{r, r(F_{r,i})}} ∪ E(F_{r,i}), r)
21 return (φ, F)

22 subprocedure adjStr(G, S, (p_1, ..., p_l), r)
23 |   m ← number of connected components in G[S] − r
24 |   return (m, e_1, ..., e_l), where e_i = I_{E(G)}({p_i, r})
25 end
```

Theorem 7.14 ([BDK12]). *Algorithm 7.1, called as* ***canonTDepthDec***$(G, V(G), \emptyset, k)$*, computes a canonical tree-depth decomposition and a corresponding canonical labeling for a connected input graph G if* td$(G) = k$ *in time* $\mathcal{O}(f(k)|V(G)|^3 \log |V(G)|)$*. Hence* Canonical Labeling(td $= k$) *is fixed-parameter tractable.*

Proof. The correctness follows nearly directly from the fact that $\trianglelefteq$ is an isomorphism order and we prove it by induction on the height in the recursive call tree. The trivial return for td$(G) = 1$ is obviously correct, so assume that td$(G) = k > 1$ after line 6. The second assignment to R already tests condition 3 and 4, while the conditions 1 and 2 are trivially fulfilled

Figure 7.2. Recursion tree for a tree-depth decomposition seen from a single vertex: we only consider the component including f on each level, so whether we choose a or b first is irrelevant.



is we compare different roots. By the inductive hypothesis we know that the subdecomposition tuples for each root are canonical and canonically sorted by the way they are computed in the for-loop. So the last assignment to R also chooses canonically among the roots.

To proof the runtime, we investigate the recursion tree. A leaf corresponds to a single vertex v of G. But this time we cannot argue by a simple partition argument as this is not a one-to-one correspondence. Nevertheless, if we look at one call in level i (from the root $i = 0$) of the recursion tree, then by corollary 7.11 and line 6 there are at most $g(i) = 2^{2^{k-2-i}}$ candidates for a root vertex and v is only in at most one component for each of them. Hence a vertex v corresponds to at most $h(k) = \prod_{i\in[0,k-1]} g(i)$ leaves, of which there are thus at most $|V(G)|h(k)$, where k is the argument k of the topmost call (see also figure 7.2). Once more, the inner vertices of the recursion tree all have outgoing degree greater than 1 and thus there are at most $\mathcal{O}(|V(G)|h(k))$ calls to ***canonTDepthDec***.

We proceed with the analysis of each recursive call. The first two assignments to R each take time $\mathcal{O}(g(k)|S|)$ (using theorem 7.7 as in the proof of theorem 7.10). The for-loop is dominated by the sort in line 14 which makes $\mathcal{O}(m \log m)$ comparisons of time complexity $\mathcal{O}(|S|^2)$ (we can use m, because it is equal for all $r \in R$). Line 14 also contains a sort, this time with $g(k) \log g(k) m$ comparisons. The assembly of the canonical labeling and the canonical tree-depth decomposition takes time $\mathcal{O}(|S|)$. Thus, if we set $g'(k) = g(k) \log g(k)$ and use $m \leq |S|$, the entire runtime of a single call is bounded by $\mathcal{O}(g'(k)|S|^3 \log |S|)$. But, observe that such a call is shared among $|S|$ leaves of the recursion tree and thus the amortized runtime for a single leaf of a single call is only $\mathcal{O}(g'(k)|S|^2 \log |S|)$. If we now define $f(k) = \prod_{i\in[0,k-1]} g'(i)^2$ and multiply the amortized runtime with the number of calls, the global runtime follows. □

Remark 7.15. The authors of [BDK12] further define a notion of generalized tree-depth by assigning a depth of 1 to a larger class of graphs and then recursively going on as in lemma 7.3. However, instead of a bounded set of forbidden subgraphs, the class of graphs with bounded generalized tree-depth has only a bounded set of forbidden ***minors*** (a minor of a graph is obtained by deleting vertices and/or edges and/or contracting connected subgraphs). Because of this, the Robertson-Seymour theorem then guarantees that an $f(k)\mathcal{O}(|V(G)|^c)$-time membership test exists for the graphs of bounded generalized tree-depth, but this is not enough to fulfill our uniform definition of FPT. **!**

Part III.

Overview, Conclusion and Outlook

…pick the fruits …

8. Relations among parameters

After we treated Graph Isomorphism and Canonical Labeling parameterized by different parameters, we will turn our attention to the parameters to sketch a complete picture. As we have seen in corollary 1.17, if a parameter κ is covered by some other parameter λ and the parameterized problem (A, λ) is fixed-parameter tractable, then (A, κ) is fixed-parameter tractable, too. The sections of this chapter are devoted to the treewidth (which covers many parameters discussed in this work), prime parameters (as defined in chapter 5) and a graph illustrating the cover relation. We start with an extension of definition 1.12, which defines further symbols for pairs of parameters depending on their relationship w.r.t. $\preceq$.

Definition 8.1. For two graph parameters κ and λ we write $\kappa \parallel \lambda$ if neither $\kappa \preceq \lambda$ nor $\lambda \preceq \kappa$, $\kappa \prec \lambda$ if $\kappa \preceq \lambda$ and $\lambda \not\preceq \kappa$ and $\kappa \approx \lambda$ if $\kappa \preceq \lambda$ and $\lambda \preceq \kappa$. •

8.1. Treewidth and related parameters

Definition 8.2 (cstw: [YBFT99])**.** The strong pathwidth $\mathrm{spw}(G)$, the connected strong treewidth $\mathrm{cstw}(G)$ and the connected strong pathwidth $\mathrm{cspw}(G)$ of a graph G are the minimal width over all strong tree decompositions (B, T, r) of G such that T is a path (spw) or $\forall t \in V(T)$: $G[B_t]$ is connected (cstw) or both conditions are met simultaneously (cspw). The pathwidth $\mathrm{pw}(G)$ is the minimal width over all tree-decompositions whose underlying tree is a path. For disconnected graphs G, we set $\mathrm{cstw}(G) = \mathrm{cspw}(G) = |V(G)|$. •

We remark that the notion of connected strong path/tree width in the previous definition requires all bags to be connected, as opposed to e.g. the c-connected path distance width (definition 6.7), where only the root bag has to be connected.

Lemma 8.3. *For all graphs G*

$$\mathrm{stw}(G) \leq \mathrm{tdw}(G) \text{ and } \mathrm{spw}(G) \leq \mathrm{pdw}(G) \tag{8.3.1}$$
$$\mathrm{tw}(G) \leq 2\,\mathrm{stw}(G) - 1 \text{ and } \mathrm{pw}(G) \leq 2\,\mathrm{spw}(G) - 1 \tag{8.3.2}$$
$$\mathrm{tw}(G) \leq \mathrm{fvs}(G) + 1 \tag{8.3.3}$$
$$\mathrm{td}(G) \leq \mathrm{vc}(G) + 1 \text{ and } \mathrm{stw}(G) \leq \mathrm{vc}(G) \tag{8.3.4}$$
$$\max \deg(G) \leq 3\,\mathrm{spw}(G) - 1 \tag{8.3.5}$$
$$\mathrm{pw}(G) < \mathrm{td}(G) \tag{8.3.6}$$

Proof. The first inequality holds by definition of the tree distance decomposition and a path distance decomposition can be turned into a strong path decomposition by making a bag out of each of its levels. To turn a strong tree decomposition into a tree decomposition add to each bag the vertices of its parental bag (and mind that tw has the additional -1), for a path decomposition do the same with (e.g.) the left neighbor bags. Let M be a feedback vertex set

of a graph G. $G \setminus M$ has a trivial tree decomposition (B, T, r) with width 1, hence (B', T, r) with $B'_t = B_t \cup M$ is a tree decomposition having width $|M| + 1$. By a similar argument, we turn a minimal vertex cover into a path starting from the root in a tree-depth decomposition and add all remaining vertices as leaves to construct a tree-depth decomposition of height $\mathrm{vc}(G)$. Furthermore a vertex cover can also serve as the root bag of a strong tree decomposition. Since all other vertices form an independent set, each of them is contained in a singleton leaf bag. The proof of lemma 6.3 easily generalizes to strong path decompositions. Finally, take the path decomposition as constructed in lemma 7.9 to prove $\mathrm{pw}(G) < \mathrm{td}(G)$. □

We now turn or attention to pairs of parameters that do not cover each other.

Lemma 8.4.

$$\forall \kappa \in \{\mathrm{pw}, \mathrm{td}, \mathrm{stw}, \mathrm{tw}\} : \kappa \parallel \max \deg \tag{8.4.1}$$

$$\forall \kappa \in \{\mathrm{pw}, \mathrm{td}, \mathrm{cspw}, \mathrm{stw}\} : \kappa \parallel \mathrm{fvs} \tag{8.4.2}$$

$$\forall \kappa \in \{\mathrm{rtdw}, \mathrm{cstw}, \mathrm{stw}\} : \kappa \parallel \mathrm{pw} \tag{8.4.3}$$

$$\mathrm{cspw} \not\leq \mathrm{td} \textit{ and } \mathrm{td} \not\leq \mathrm{stw} \tag{8.4.4}$$

Proof. (8.4.1): It is well known ([RS91]) that the treewidth is unbounded on grids

$$G_{l \times m} = ([l] \times [m], \{\{(i,j),(i',j')\} \mid |i - i'| \leq 1, |j - j'| \leq 1\}) .$$

On the other hand the class of stars has unbounded degree.

(8.4.2): The pathwidth is unbounded on trees: e.g. join three trees with pathwidth k via a single edge to a new vertex to obtain a tree with pathwidth $k + 1$ (see [RS83] for details). Further the strong treewidth is unbounded on wheels (a cycle with an additional vertex connected to all vertices on the cycle), because the central vertex has to be in the same or a neighboring bag as any other vertex in the wheel. But since there is a cycle connecting each neighbor of the central vertex, two adjacent neighbors may not be in different subtrees and thus there are all contained it at most two bags. On the other hand a path of disjoint cycles C_4 connected by single edges has connected strong pathwidth 4 and unbounded feedback vertex number. To compare tree-depth and feedback vertex number, we use that the tree-depth is unbounded on paths (see lemma 7.4), while a star whose vertices are replaced with cycles C_4 has tree-depth 4 and unbounded feedback vertex number.

(8.4.3): Wheels also have bounded pathwidth (the central vertex and one additional vertex can be placed in all bags). On the other hand trees of course have bounded connected strong treewidth and rooted tree distance width.

(8.4.4): The tree-depth is unbounded on paths, while the closure of a rooted tree of height 2 whose non-leaves have degree k has tree-depth 3 and strong treewidth $k - 1$. This is because in a strong tree-decomposition all leaves of the rooted tree have to be in bags of distance at most one to the bag of the root (which is w.l.o.g. the root bag) and since k leaves are connected via their parents, they may not be in different subtrees of the strong tree-decomposition, seen from the root bag. □

We now consider the relations among different distance widths and compared with the strong widths.

Figure 8.1. The pincushion graph H_2 (see proof for equation (8.5.1)): pinheads are colored black.

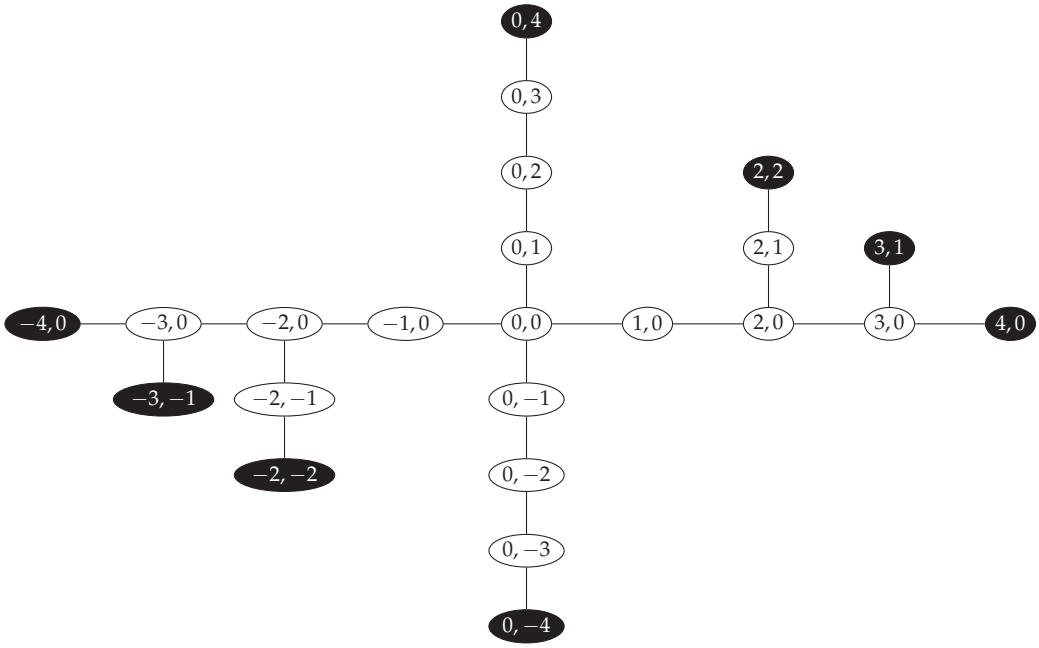
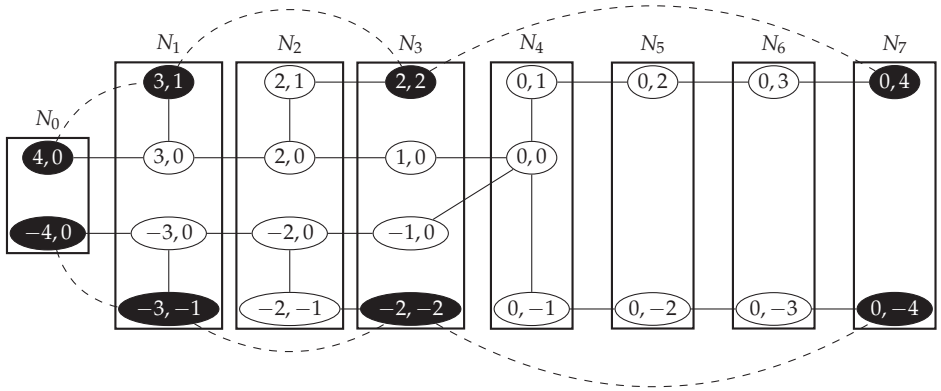


Figure 8.2. A path distance decomposition of the pincushion H_2: the dashed lines symbolize a path (with additional vertices) as constructed in the proof of equation (8.5.5).



Lemma 8.5.

$$\text{pdw} \prec \text{spw} \qquad \text{[YBFT99]} \qquad (8.5.1)$$

$$\text{rpdw} \prec \text{pdw} \qquad \text{[YBFT99]} \qquad (8.5.2)$$

$$\text{cspw} \prec \text{rpdw} \approx \text{cpdw} \approx \text{1-clpdw}(G) \qquad \text{[YBFT99; Ota12]} \qquad (8.5.3)$$

$$\text{rpdw} \not\preceq \text{cstw} \not\preceq \text{tdw} \qquad \text{[YBFT99]} \qquad (8.5.4)$$

$$\text{2-cpdw} \parallel \text{rtdw} \qquad \text{[Ota12]} \qquad (8.5.5)$$

$$\text{spw} \not\preceq \text{tdw} \qquad (8.5.6)$$

Figure 8.3. The book graph B_5 (see proof of equation (8.5.4))

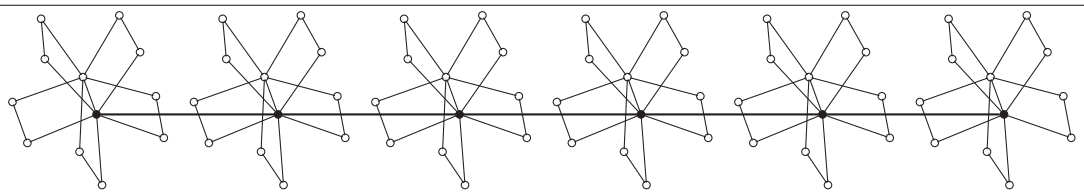
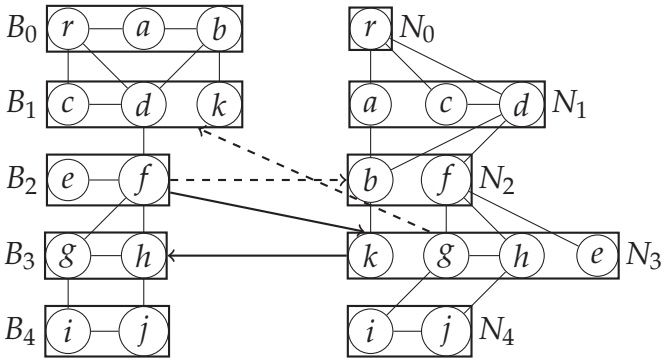


Figure 8.4. A connected strong path decomposition and a rooted path distance decomposition: as constructed in the proof of equation (8.5.3). The dashed arrows depict $b_{\min}(j)$ and $n_{\min}(i)$, the others $b_{\max}(j)$ and $n_{\max}(i)$ for $i = 2$ and $j = 3$.



Proof. (8.5.1): A path distance decomposition N defines the strong path decomposition $(B, [0, |N| - 1], 0)$ with $B_i = N_i$. For the other direction let a ***pincushion*** H_k (called ribbon in [YBFT99]) of size k be the minimal graph (w.r.t. $\subseteq$ for edges and vertices) such that

$$-2^k \in V(H_k) \wedge \forall i \in [-2^k + 1, 2^k] : ((i,0) \in V(H_k) \wedge \{(i-1,0),(i,0)\} \in E(H_k))$$

$$\text{and } \forall i \in [0,k], s \in \{-1,1\}, j \in [2^i] :$$

$$((s(2^k - 2^i), sj) \in V(H_k) \wedge \{(s(2^k - 2^i), sj), (s(2^k - 2^i), s(j-1))\} \in E(H_k)) .$$

See figure 8.1 for the pincushion H_2. Further let G_k be the graph consisting of $2k+2$ pincushions H_k (glued together at $(2^k, 0)$ and $(-2^k, 0)$ respectively). Observe that G_k has $\text{spw}(G_k) \leq 3$ by tilting each pin at position $G_k[\{(i,l) \mid l = j\}]$ (for some j) to the left if it is has no negative second component and to the right otherwise. Assume that N_0 is a root set of a minimal path distance decomposition N for G_k. Then there are two consecutive pincushions such that N_0 has no vertex in any of them. We now look w.l.o.g. on the right part of the left pincushion and assume that a shortest path to each (i,j) with positive i goes over $(0,0)$ (if not this is true for the left part of the right pincushion). But there are $k+2$ pinheads ((i,j) such that $i + j = 2^k$) and these pinheads all have distance $i + j$ to $(0,0)$ and are thus in the same distance level of N.

(8.5.2): rpdw $\preceq$ pdw holds by definition, so we just show pdw $\not\preceq$ rpdw. To do this, we reuse our pincushion H_k from the first part of the proof. A path distance decomposition N of width 4 can be obtained by choosing $N_0 = \{(2^k, 0), (-2^k, 0)\}$ and tilting the pins as in the first part of this proof (see figure 8.2). On the other hand assume that there is some path distance decomposition N of H_k with $N_0 = \{(i,j)\}$. W.l.o.g. assume that $i \leq 0$. Then we are left with the same scenario as above and have $2k+2$ vertices (i', j') such that $i' + j' = 2^k$. Thus $\text{rpdw}(H_k) > k$.

(8.5.3): rpdw $\not\preceq$ cspw holds because cycles have unbounded cspw (see proof for (8.5.4)). We

now show that if the width of a connected strong path decomposition $(B, [0, l], 0)$ of a graph G is k, the width of a rooted path distance decomposition N with $\{r\} = N_0 \subseteq B_0$ (w.l.o.g. $B_0 \neq \emptyset$) is at most k^2. Let $n_{\max}(i) = \max\{j \mid N_j \cap B_i \neq \emptyset\}$, $b_{\max}(j) = \max\{i \mid N_j \cap B_i \neq \emptyset\}$ and $n_{\min}$, $b_{\min}$ be defined analogously (see figure 8.4). We then have

$$\forall i \in [0, l] : n_{\max}(i) - n_{\min}(i) \leq k - 1 \tag{1}$$

$$\forall i \in [0, l], h < i : n_{\min}(i) - n_{\min}(h) \geq i - h \tag{2}$$

The first inequality holds because $G[B_i]$ is connected, the second one holds because a vertex $v \in B_i$ whose distance to r is minimal in B_i needs a predecessor on its shortest path from r in an adjacent bag, i.e. B_{i-1}. The first inequality now yields

$$\forall j \in [0, |N| - 1] : j \leq n_{\max}(b_{\min}(j)) \leq n_{\min}(b_{\min}(j)) + k - 1\,,$$

while the second one yields

$$\forall j \in [0, |N| - 1] : n_{\min}(b_{\min}(j)) + k \leq n_{\min}(b_{\min}(j) + k)\,.$$

But we further know that for all $j \in [0, |N| - 1]$ the inequality $n_{\min}(b_{\{} \max\}(j)) \leq j$ holds. Thus $b_{\min}(j) + k \leq b_{\max}(j)$ would bring us the contradiction $j < j$ and we conclude $b_{\max}(j) - b_{\min}(j) < k$ and hence for all $j \in [0, |N| - 1]$, we have

$$|N_j| = \left|\bigcup\{B_i \mid B_i \cap N_j \neq \emptyset\}\right| < k^2\,.$$

rpdw $\preceq$ cpdw holds by definition, so we just show the converse. Let N be a path distance decomposition of width k such that N_0 is connected. Take for some $r \in N_0$ the path distance decomposition N' with $N'_0 = \{r\}$. Now any vertex v in N'_i cannot be in N_j for $j > i$, because r is an element of N_0. But it cannot be in N_j for $j < i - (k - 1)$ either, because if it is at such a distance j to some $r' \in N_0$ its distance to r may not exceed $j + k$ since N_0 is connected and has size k. Thus the width of N' is at most k^2. Exactly the same argument also works for the 1-clpdw, since we do not the connectedness, but the bounded distance.

(8.5.4): Cycles have rooted path distance width 2. Assume a bag of connected strong tree decomposition contains some subpath of a cycle. Then the remaining subpath has to be in the same subtree and both ends have to be at distance one to the first bag. Thus the cycle is split among two bags. To show cstw $\not\preceq$ tdw, we look at ***books***. A k-book shall be a graph

$$B = (\{a, b\} \cup \{c, d\} \times [k], \quad \{\{a, b\}\} \cup \bigcup_{i \in [k]} \{\{a, (c, i)\}, \{(c, i), (d, i)\}, \{(d, i), b\}\}).$$

The ***book graph*** B_k consists of $k + 1$ copies of k-books connected by a path (see figure 8.3). We refer to those vertices which are part of this path by b_i, $i \in [k + 1]$. Now assume that B_k has a tree distance decomposition of width at most k. Then its root bag does not contain any vertex from at least one of the books. The corresponding b_i is the closest vertex to the root bag among the vertices in its book. But because b_i has $k + 1$ neighbors in its book who are connected via the corresponding a_i, there has to be a bag of width $k = 1$ which contradicts the assumption. A connected strong tree decomposition of a single book is e.g. a star with $\{a, b\}$ as central bag and the others bags being $\{(c, i), (d, i)\}$ for $i \in [k]$.

(8.5.5): rtdw $\not\preceq$ 2-cpdw follows from rtdw $\not\preceq$ pw. For the other direction look again at the pincushion H_k and its decomposition with $N_0 = \{(2^k, 0), (-2^k, 0)\}$ (figure 8.2). We add two additional vertices v_i^- and v_i^+ in each level N_i that does ***not*** contain a pinhead and for those which do contain one define v_i^- to be the pinhead with a negative component and v_i^+ to be the other pinhead in N_i. Further for $i \in [0, |N| - 2]$ we add the edges $\{v_i^+, v_{i+1}^+\}$ and $\{v_i^-, v_{i+1}^-\}$ to H_k and name the new graph H'_k and the new partition N'. Clearly N' is a path distance

decomposition of H'_k and with $|N_0| = 2$ and width at most $|N| + 2 \leq 6$. The new paths (dashed in figure 8.2) do not change the distance of the nonnegative pinheads to $(0,0)$ and since they are now connected by a path (whose vertices have greater distance to $(0,0)$) they have to be in the same bag of a tree distance decomposition whose root bag has no vertex with strictly positive component. This holds likewise for the other pinheads and thus we can apply the argument we used for pdw $\not\leq$ rpdw.

(8.5.6): We combine the arguments for equation (8.5.5) and (8.5.1), i.e. we take the graph G_k consisting of $2k+2$ pincushions and add the path connecting the pinheads to every pincushion as for (8.5.5). Again we have to "explore" at least one pincushion from $(0,0)$ (as for (8.5.1)), but now the pinheads lie on a path and have to go into the same bag of a tree distance decomposition. □

Remark 8.6. If we compare the best known runtimes for Canonical Labeling parameterized by 1-clpdw ($\mathcal{O}(2^{2^{(3k-1)^k}}(k+1)!^2|V(G)|^2)$), cpdw ($\mathcal{O}((2k)!(k+1)!|V(G)|^2)$) and rpdw ($\mathcal{O}((k+1)!^2|V(G)|^2)$) and keep in mind that when one of the parameters is k the others are at most k^2, we see that the specialized algorithm for the connected path distance with is faster than that for the rooted path distance width with k replaced by k^2. **!**

8.2. Parameters vs. prime parameters

Even a superficial look at prime parameters suggests that they are at most as large as their corresponding original parameter for many "natural" parameters, i.e. those, which capture hard instances for graph problems well. As the prime parameters are defined via a maximum over prime graphs in the modular decomposition of a graph, we have to look at the relation between those prime graphs and the original graph. There are two operations needed to transform a graph G into one of the prime graphs in its modular decomposition: induced subgraphs (for some module M) and quotient graphs. But it is easy to see that the quotient graph is also isomorphic to an induced subgraph of G if we represent each module with exactly one of its vertices. So we are left with considering induced subgraphs and thus the "natural" class of parameters turns out to be the class of ***hereditary*** parameters. Below, we will give a formal definition and state the above observation as a lemma.

Definition 8.7. Let κ be a graph parameter. We say that κ is ***hereditary*** if for all graphs G and all $V' \subseteq V(G)$ the inequality $\kappa(G[V']) \leq \kappa(G)$ holds. •

Lemma 8.8. *Let κ be an invariant hereditary graph parameter. Then $'\kappa(G) \leq \kappa(G)$ holds for all graphs G.*

Proof. Let P be a prime graph appearing in $\mathrm{MD}(G)$ such that $\kappa(P)$ is maximized. Then either $\kappa(P) = 0$ or $P = \mathrm{quot}(G[M])$ for some module M of G. Furthermore there is a bijection $\phi : V(P) \to V(G[M])$ such that $\forall N \in V(P) : \phi(N) \in N$ and $P \cong_\phi P_\phi$, where $V(P_\phi) = \mathrm{rng}(\phi)$ and $E(P_\phi) = \{\phi(e) \mid e \in E(P)\}$. Hence P_ϕ is a subgraph of $G[M]$ and since κ is hereditary and an invariant, we have

$$'\kappa(G) \leq \kappa(P) = \kappa(P_\phi) \leq \kappa(G[M]) \leq \kappa(G)\,.$$

□

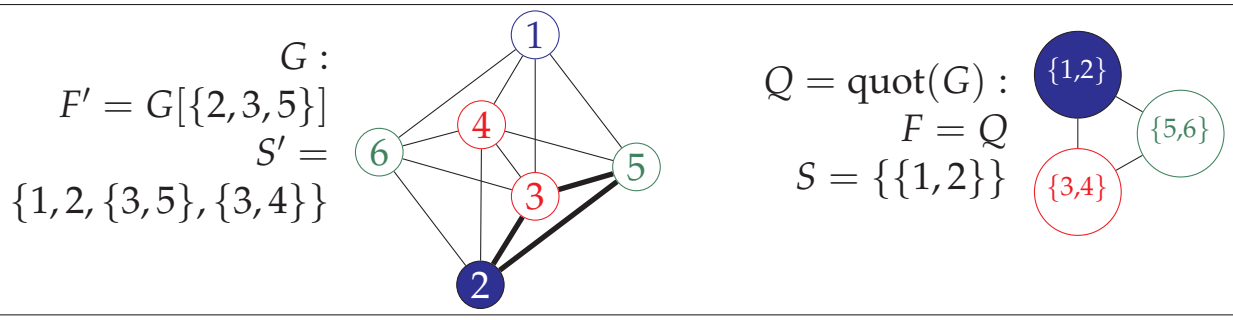
Note that this proof also works for quotients modulo any other modular decomposition, not only the quasi-maximal modular decomposition that is implied by our definition of quot(G) for a graph G.

Corollary 8.9. *Let G be a graph. Then* $'\max\deg(G) \leq \max\deg(G)$, $'\mathrm{tw}(G) \leq \mathrm{tw}(G)$, $'\mathrm{td}(G) \leq \mathrm{td}(G)$ *and* $'\mathcal{F}$*-free*-mn$(G) \leq \mathcal{F}$*-free*-mn(G) *for any graph class $\mathcal{F}$.*

Proof. It is obvious that max deg is hereditary, the same holds for $\mathcal{F}$-free-mn as $\mathcal{F}$-free is defined via induced subgraphs. For the treewidth observe, that we simply may delete unused vertices from a tree distance decomposition, i.e. for any subgraph G' and any tree decomposition (B, T, r) of G we set $B'_i = B_i \cap V(G')$ and use (B', T, r) as a tree decomposition of G'. As the tree depth decomposition is defined via a subgraph, the tree-depth is clearly hereditary. □

Example 8.10. For $\mathcal{F}$-free-mn(G) we have a closer look on how modification sets and forbidden graphs are related when we consider a graph G and its quotient graph Q. W.l.o.g. let $F \in \mathcal{F}$ be a forbidden subgraph in Q (see figure 8.5, where F is cycle). Then there is a corresponding subgraph $F' = (\phi(V(F)), \phi(E(F)))$ in G where ϕ is an isomorphism like in the proof of lemma 8.8. Let S' be a $\mathcal{F}$-free modification set and s' a modification for F'. If s' is a vertex of G then we set $s = \phi^{-1}(s')$, if $s' = \{u, v\}$ such that $\forall M \in V(Q) : \{u, v\} \not\subseteq M$ we set $s = \phi^{-1}(s') = \{\phi^{-1}(u), \phi^{-1}(v)\}$. (Non-)edges within modules are ignored, as they are not present in the quotient graph and there preimage would be a vertex of Q. Observe that for a minimal set of changes $\{s'_1, \ldots, s'_l\} \subseteq S'$ that alters F' such that it is no longer in $\mathcal{F}$, $\phi^{-1}(\{s'_1, \ldots, s'_l\})$ does the same for F. Furthermore the correspondence ϕ^{-1} does not depend on F and thus $S = \phi^{-1}(\{s' \in S' \mid \nexists M \in V(Q) : s' \subseteq M\})$ is a $\mathcal{F}$-free modification set of Q and $|S| \leq |S'|$. e.g.

Figure 8.5. Forbidden subgraph, modification set and quotient graph (example 8.10)



For the color multiplicity we first have to define what we mean by the coloring of the quotient graph. A natural solution is to choose the multiset of colors present in each module. This directly gives rise to the following lemma.

Lemma 8.11. *Let (G, c) be a colored graph and the color of a module $M \in V(\mathrm{quot}(G))$ be defined as $c(M) = \{\!\{c(v) \mid v \in M\}\!\}$. Then* $'\mathrm{cm}(G, c) \leq \mathrm{cm}(G, c)$.

Proof. As always we only consider the quotient graph case. Let $Q = \mathrm{quot}(G)$ and C_i be a color class $c^{-1}(i)$ of G. Then at most $|C_i|$ modules $M \in V(Q)$ may contain a vertex $v \in C_i$ and thus $C'_i = \{M \in V(Q) \mid C_i \cap M \neq \emptyset\}$ is a union of color classes of (Q, c) for each i, it has with size $|C'_i| \leq |C_i|$ and any color class is a subset of some C'_i. □

Lampis [Lam12] introduces a parameter called the ***neighborhood diversity*** which is defined via a partition into modules and is able to show that this new parameter covers the vertex cover number. Luckily, the main idea of his proof works for prime parameters as well.

Lemma 8.12. *Let κ be a graph parameter. Then* $\mathrm{vc} \preceq {}'\kappa$.

Proof. Let S be a vertex cover of a graph G. Then $G \setminus S$ has no edges and its vertices cannot distinguish each other, thus we can partition $V(G) \setminus S$ into at most $2^{|S|}$ modules of G. Combined with the trivial partition of S into singleton sets, we obtain a partition of $V(G)$ into at most $2^{|S|} + |S|$ modules. Because vc is hereditary, any graph $G[M]$ (M is a module of G) has at most $2^{\mathrm{vc}(G)} + \mathrm{vc}(G)$ modules and thus any prime graph in $\mathrm{MD}(G)$ has at most this number of vertices. Furthermore we implicitly assume κ to be invariant and hence ${}'\kappa(G)$ is bounded from above by

$$\max\{\kappa(H) \mid H \in \mathcal{G}, |V(H)| \in [0, 2^{\mathrm{vc}(G)} + \mathrm{vc}(G)]\}\,.$$

□

8.2.1. Prime distance widths

The distance widths are not hereditary because the removal of a vertex or edge can result in a collapse of multiple distance levels.

Example 8.13. Look again at the pincushion graph H_k as defined in the proof of equation (8.5.1) and as depicted in figure 8.2. If we add a new vertex v that is connected to both $(-2^k, 0)$ and $(2^k, 0)$, the rooted path distance width becomes 4, whereas it was $k + 2$ without this modification. Nearly the same holds for the rooted tree distance width (6 vs. $k + 2$), if we add the additional path connecting all pinheads as in the proof of equation (8.5.5). e.g.

Nevertheless, we will show in the following lemmas that the prime distance widths cover their ordinary counterparts. This come from the fact that modules and quotient graphs are deeply connected with distances. In fact e.g. the algorithm from [TCHP08] partitions the vertex set according to a rooted path distance decomposition. We will only handle connected modules, because all prime graphs in $\mathrm{MD}(G)$ are quotient graphs of connected subgraphs induced by modules (see definition 5.11).

Lemma 8.14. *Let N be a path distance decomposition of a graph G having width k and M be a module of G such that $G[M]$ is connected. Then $G[M]$ has a path distance decomposition N' of width at most $3k$ such that $|N'_0| \leq |N_0|$.*

Proof. If no nonempty module $M' \subseteq V(G) \setminus M$ is adjacent to M, then the distances in $G[M]$ are the same as those in G for all pairs of vertices $u, v \in M$. Hence we assume there is such an adjacent module M'. Thus any pair of vertices of M is connected via path of length 2 in G and M is spread over at most three bags of N. Since no path distance decomposition of $G[M]$ can have width more than $|M|$, the claim follows. □

Lemma 8.15. *Let N be a path distance decomposition of a graph G having width k. Then* $\mathrm{quot}(G)$ *has a path distance decomposition N' such that for all $i \in [0, |N'| - 1]$ the inequality $|N'_i| \leq |N_i|$ holds.*

Proof. We set $N'_0 = \{M \in V(\text{quot}(G)) \mid M \cap N_0 \neq \emptyset\}$ and $N'_i = \{M \in V(\text{quot}(G)) \mid M \cap N_0 \neq \emptyset\} \setminus N'_0$ for all $i \in [0, |N|]$. Since the distance between vertices of different modules does not depend on distances within each module, N' is clearly a path distance decomposition of $\text{quot}(G)$ (after truncation if necessary). □

For (minimal) tree distance decompositions, we have to handle some additional cases, but apart from that the situation is similar.

Lemma 8.16. *Let (B, T, r) be a minimal tree distance decomposition of a graph G having width k and let M be a module of G such that $G[M]$ is connected, then $G[M]$ has a tree distance decomposition (C, U, s) of width at most k such that $|C_s| \leq |B_r|$.*

Proof. If M is not adjacent to any other module, we argue as in lemma 8.14, so assume that is is adjacent to at least one other module. If $B_r \cap M = \emptyset$ then M is contained in one distance level and thus each component of $G[M]$ is contained in one bag of (B, T, r). Thus we may assume that there is a set C_s with $C_s = M \cap B_r \neq \emptyset$. We now look at the minimal tree distance decomposition (C, U, s) of $G[M]$ which is uniquely defined by its root bag C_s. Again we are left with two cases for the root bag B_r.

If B_r contains a vertex $v \in V(G) \setminus M$ that is adjacent to all vertices in M then all vertices in $M \setminus C_s$ have to be in the first level of (B, T, r) and thus every connected component of $G[M] \setminus C_s$ is contained in a single bag of (B, T, r). Hence the width of (B, T, r) is at least the width of (C, U, s).

If B_r contains no such vertex the vertices in M with distance one to B_r are exactly those with distance one to C_s. Furthermore, since $B_r \cap M \neq \emptyset$, there has to be a vertex v with distance one to B_r that is adjacent to all vertices in M. Hence all bags C_u for a child u of s in U are subsets of a single bag B_t for some child t of r in T and obviously $v \in B_t$. Now we argue for the third level of (B, T, r) in a similar way as we did for the second level in the previous paragraph. □

Intuitively, an additional adjacent vertex from another module just flattens down the decomposition tree to a single level, so it cannot make bags smaller. Observe that the pathological example 8.13 works by connecting only ***two*** old vertices by a new one, whereas we would have to connect the new vertex to ***all*** old vertices if they shall form a module that does not include the new vertex.

Lemma 8.17. *Let (B, T, r) be a minimal tree distance decomposition of a graph G. Then $\text{quot}(G)$ has a tree distance decomposition (C, U, s) of such that there is an injective function $\phi : V(T) \to V(U)$ such that for all $t \in V(T)$ we have $|C_{\phi(t)}| \leq |B_t|$.*

Proof. Use lemma 8.15 and observe that whenever two modules M, M' of G are adjacent in $\text{quot}(G)$, then all of there vertices are and thus the levels of the minimal tree distance decomposition (C, U, s) (C_s defined as N'_0 in lemma 8.15) are split at least as fine as the levels of (B, T, r). □

We collect all those findings in the following theorem.

Theorem 8.18. *Let κ be in $\{\text{rpdw}, c\text{-cpdw}, \text{pdw}, \text{rtdw}, \text{tdw}\}$. Then $\kappa \preceq {}'\kappa$.*

Proof. For all parameters but the c-connected path distance width, this directly follows from the lemmas above. Furthermore the proof of equation (8.5.3) that we can replace root sets with c component with those of size c and the new width is quadratic in the old width. If we use c-rpdw for the minimal width over all root sets with at most c vertices, we thus have

$$c\text{-cpdw} \preceq c\text{-rpdw} \preceq {}'c\text{-rpdw} \preceq {}'c\text{-cpdw} .$$

□

8.2.2. Some incomparability lemmas

To complete our picture of prime parameters we show how they relate to the maximal parameters w.r.t $\preceq$ considered in this work.

Lemma 8.19. *Let κ be a parameter in $\{\max\deg, \mathrm{tw}, \mathrm{cm}, \mathrm{em}\}$ and ${}'\lambda$ be any prime parameter. Then ${}'\lambda \not\preceq \kappa$.*

Proof. All the non-prime parameters in question are unbounded on (colored) cographs and ${}'\lambda(G) = 0$ for cographs G by definition. □

This of course also holds for all the parameters covered by the treewidth.

Lemma 8.20. *The prime parameter ${}'\kappa$ of a graph parameter $\kappa \in \{\max\deg, \mathrm{tw}\}$ is unbounded.*

Proof. For $\max\deg$ take a star G with arbitrary maximal degree ≥ 2 and add an edge (with a new leaf) to all of its leaves. Then each pair of vertices lies on a common induced path of length at least 4, which enables us to apply lemma 5.6 and conclude that the inclusion of a pair in a module forces the inclusion of the entire path. But all paths intersect and thus G is prime.

With a similar argument as above it is easy to see that grids $G_{l\times m}$ with $l + m > 4$ are prime (see proof of equation (8.4.1)). □

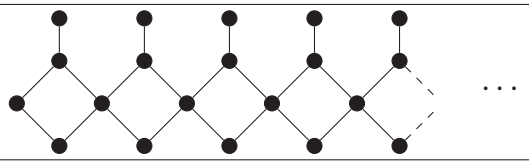
We again remark that this carries over to all the parameters covered by treewidth, e.g. fvs, td, stw and pw.

Lemma 8.21. $\mathrm{tw} \not\preceq {}'\kappa$ *for* $\kappa \in \{\mathrm{fvs}, \mathrm{pw}, \mathrm{stw}\}$.

Proof. Let $\mathcal{C}$ be the class of graphs, obtained by connecting cycles C_4 at opposite vertices and connecting an additional vertex to one of the non-junctional vertices of each C_4 (see figure 8.6). Obviously any feedback vertex set of a graph $G \in \mathcal{C}$ has size at least $|V(G)|/8$ and while the treewidth is 2. By lemma 5.6 a graph $G \in \mathcal{C}$ is obviously prime.

By lemma 5.6 we can obtain a class of trees with unbounded prime pathwidth as follows. As in the proof of equation (8.4.2) we construct trees with pathwidth $k + 1$ by combining three trees of pathwidth k. Finally we replace the leaves of each tree by a path of length 2. This ensures that each pair of vertices lies on an induced path of length at least 3. Wheels with at least five vertices are already prime, thus the prime strong treewidth does not cover the treewidth. □

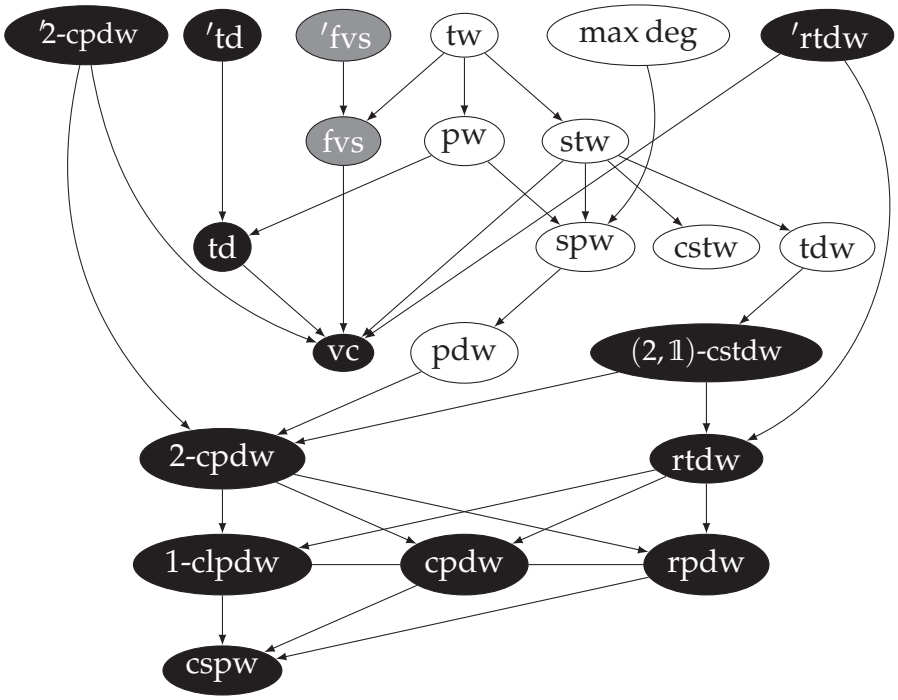
Figure 8.6. A class of graphs with bounded tw, but unbounded $'$fvs



8.3. Graph of the cover relation

The following graph represents a collection of the results from all lemmas in this chapter. To keep it clear, only a few prime parameters are depicted. For the same reason we did not include further parameters as bandwidth or cutwidth (see e.g. the overview graph in [YBFT99]).

Figure 8.7. Relations among parameters: This graph is similar to a Hasse diagram (but $\preceq$ is not antisymmetric). Adjacent parameters on the the same level cover each other, any other downwards edge from κ to λ means $\lambda \preceq \kappa$. A black colored parameter κ means that Canonical Labeling is fixed-parameter tractable w.r.t. κ, whereas gray means that Graph Isomorphism is fixed parameter tractable w.r.t. κ.



9. Overview of results

In this chapter we briefly hint towards other results showing the inclusion of parametrized Graph Isomorphism in either XP or FPT and state a table of known upper bounds.

Eigenvalue multiplicity Using similar group-theoretic techniques as in [Bab79; FHL80], [BGM82] showed that Graph Isomorphism(em) $\in$ XP. The key idea is to see the vertices $v \in V(G)$ of a graph G as the unit vectors of $\mathbb{R}^{|V(G)|}$, to see $\phi \in \mathrm{Aut}(G)$ as a permutation matrix and to use the fact that a projection Matrix P_W of an eigenspace W of the adjacency matrix of G commutes with each $\phi \in \mathrm{Aut}(G)$. Iteratively the decomposition of $\mathbb{R}^{|V(G)|}$ into eigenspaces is transformed into a coloring (using the lengths of projections). Linear algebra techniques provide a supergroup for the restriction of $\mathrm{Aut}(G)$ to each new color class of size at most $|V(G)|^{\mathrm{em}(G)}$. At last the tower-of-groups approach is applied as in section 3.3.

The paper of Evdokimov and Ponomarenko [EP99] which proved that Canonical Labeling(em) $\in$ FPT has little in common with the earlier approach. Nevertheless they use a technique (so called ***canonical labeling cosets***) introduced by Babai and Luks in [BL83]. But this technique is not applied to G itself, but to the ***cellular algebra*** of G. This algebra is an alternative interpretation of the output of the 2-dim. Weisfeiler-Lehman algorithm and the eigenvalue multiplicity relates to decompositions of this algebra using representation theory.

Treewidth A ***k-tree*** is a graph that can be constructed in the following way: start with a k-clique $G_0 = K_k$ and in each step choose a new vertex $v \notin V(G)$ and a subset $K \subseteq V(G)$ such that $G[K]$ is a k-clique and define

$$G_{i+1} = (V(G_i) \cup \{v\}, E(G_i) \cup \{\{v, w\} \mid w \in K\}) .$$

Subgraphs of k-trees are called ***partial k-trees*** and are exactly the graphs with treewidth at most k. In [Bod90] Bodlaender showed that Graph Isomorphism(tw $= k$) $\in$ XP. His algorithm constructs a special decomposition into k-separators S ($S \in \binom{V(G)}{k}$ such that $G \setminus S$ is not connected) with additional information for one input graph G_1. For the other graph G_2 the algorithm tries to construct a compatible decomposition testing all pairs (S, C), where C is a component of $G \setminus S$ and S a k-separator, against all such pairs of the decomposition of G_1. The .5 in the exponent (see table 9.1) comes from a reduction of a certain subproblem of the test for each such pair to the bipartite matching problem.

9.1. Table of all results

Let $V = V(G_1)$ for an instance (G_1, G_2, k) (or $((G_1, c_1), (G_2, c_2), k)$ resp.) then the following upper bounds for (Colored) Graph Isomorphism/ Canonical Labeling parametrized by κ are known:

Table 9.1. Known upper bounds for parametrized Graph Isomorphism and Canonical Labeling

		Known upper bound			
Parameter	κ	Graph Isomorphism		Canonical Labeling	
Color multiplicity	cm	$\mathcal{O}((k!)^3k^2\lvert V\rvert^6)$	[Bab79; FHL80]	identical	[KL81]
Eigenvalue multiplicity	em	FPT	[EP99]	identical	[EP99]
Feedback vertex number	fvs	$\mathcal{O}((2k+4k\log k)^k k\lvert V\rvert^2)$	[KS10]	XP	
$\mathcal{F}$-free modification number	$\mathcal{F}$-free-mn	FPT [a]	[KS10]	XP [a]	
Rooted path distance width	rpdw	$\mathcal{O}((k+1)!^2\lvert V\rvert^2)$	[YBFT99]	identical	cor. 6.6
Rooted tree distance width	rtdw	$\mathcal{O}((k+1)!^2\lvert V\rvert^2)$	cor. 6.23	identical	cor. 6.23
c-connected path distance width	c-cpdw	$\mathcal{O}((2k)!(k+1)!\lvert V\rvert^{c+1})$	[Ota12]	identical	cor. 6.11
c-con. d-sep. tree distance width	(c,d)-cstdw	FPT	thm. 6.26	identical	thm. 6.26
Tree-depth	td	$\mathcal{O}(f(k)\lvert V\rvert^3\log\lvert V\rvert)$	[BDK12]	identical	[BDK12]
Prime parameter	$'\kappa$	FPT [b]	cor. 5.20	FPT [c]	cor. 5.24
Treewidth	tw	$\mathcal{O}(\lvert V\rvert^{k+4.5})$	[Bod90]	XP	e.g. [Wag11] [d]
Maximum degree	max deg	$\lvert V\rvert^{\mathcal{O}(k)}$	[BL83]	identical	[BL83]
Genus	g	$\lvert V\rvert^{\mathcal{O}(k)}$	[Mil80; FM80]	identical	[Mil80; FM80]

[a] if $\lvert\mathcal{F}\rvert \in \mathbb{N}$ and $\mathcal{F}$-free Graph Isomorphism $\in$ P
[b] if Colored Graph Isomorphism is fixed-parameter tractable w.r.t. κ
[c] if Colored Canonical Labeling is fixed-parameter tractable w.r.t. κ
[d] in AC^1 for fixed k

10. Outlook and Conclusion

We discussed all but one (eigenvalue multiplicity [EP99]) parameters for which the fixed-parameter tractability of the graph isomorphism problem is currently settled. Furthermore, we enriched this collection by the definition of additional width parameters inspired by the work of Otachi [Ota12] and the treatment of modular decompositions which lead us to the concept of prime parameters. For all but one (modification numbers) class of parameters discussed in this work, we contrasted algorithms for Graph Isomorphism and Canonical Labeling and in most cases described the transformation from a Graph Isomorphism to a Canonical Labeling algorithm, the exception being the tree-depth where the first FPT algorithm [BDK12] for Graph Isomorphism already worked via canonical labelings. To aggregate all the things we learned in chapters 3 to 7, we collated the established relations among the considered parameters and added missing ones (mostly for the parameters introduced in this thesis) in chapter 8. Figure 8.7 graphically represents this and section 9.1 gives an overview of the fixed-parameter tractability results for Graph Isomorphism and Canonical Labeling. Based on this we separately draw our conclusions and provide an outlook for the following different issues.

Gap between different parameterized problems In sections 2.2 and 2.3, we saw that the considered polynomial time equivalent problems remain equivalent under fpt Turing reductions if parametrized by the same parameter, provided that the colored isomorphism/canonical labeling problem is fpt Turing reducible to the uncolored counterpart. This restrictions is not a huge obstacle, because the standard polynomial time reductions work for many natural parameters, after an adequate modification, if necessary. Nevertheless, we asked (questions 2.11 and 2.18) for a general solution, at least for FFPT parameters.

Of much greater interest is the gap between Graph Isomorphism and Graph Canonization. As usual, instead of Graph Canonization, we considered Canonical Labeling, which is equivalent under fpt Turing reductions (lemma 2.20, subject to Colored Graph Canonization $\leq^{\text{fpt-T}}$ Graph Canonization). For most of the parameters we considered, there is no gap between Graph Isomorphism and Canonical Labeling. When Babai introduced his tower of groups[Bab79] for graphs of bounded color multiplicity, he explicitly asked for a similar canonization procedure, but this question was answered soon [KL81] (see also [Luk10, 11–26]). For those parameters who offer a tree like description or decomposition (tree-depth, distance widths and prime parameters), directly constructing a canonical labeling algorithm using the ideas of the linear time algorithm for trees in [AHU74] combined with an isomorphism order on subtrees similar to Lindell's [Lin92] is a natural approach and hence there is no gap for these parameters. Nevertheless, Yamazaki, Bodlaender, Fluiter and Thilikos [YBFT99] went a different way when they introduced distance decompositions and it was not until [DTW12] that an isomorphism order was used for tree distance decompositions. This order heavily inspired our approach which closes the gap between path distance decompositions and tree distance decompositions as well as the gap between Graph Isomorphism and Canonical La-

BELING for graphs of bounded distance widths.

The situation is different for modification numbers and this may be due to the way the algorithms of Kratsch and Schweitzer [KS10] work. While all other isomorphism algorithms considered in this thesis either do symmetric things in both graphs or even combine both graphs (color multiplicity), their algorithms look for a forbidden subgraph in one graph and for a modification set in the other graph. Furthermore, the number of modification sets is for most classes not bounded by a function of the parameter or a polynomial of the number of vertices and edges in the graph. We already outlined the importance of such a bound in chapters 6 and 7. Hence it seems interesting to further study the following two questions.

Question 10.1. Let $\mathcal{F}$ be a finite set of graphs, such that Graph Isomorphism is in P for graphs in $\mathcal{F}$-free. Is Canonical Labeling($\mathcal{F}$-free-mn $= k$) fixed-parameter tractable? **?**

Question 10.2. Is Canonical Labeling(fvs $= k$) fixed-parameter tractable? **?**

Modular decompositions and prime parameters In chapter 5 we have seen that modular decompositions reduce the general isomorphism/canonization problems to those on prime graphs. Using this observation we defined prime parameters which turned out to be generalizations of their ordinary counterparts for many parameters in section 8.2. Especially, we could show this for many distance widths despite the fact that they are not hereditary. Nevertheless, the overall conclusion is mixed. On the positive side, prime parameters integrate well in the framework of graph parameters and they do not overestimate the hardness of instances that are actually easy. Unfortunately, on the other hand, the prime parameters do not cover any additional parameters considered here that are not already covered by their ordinary counterpart. We complete the discussion of modular decomposition with a hint towards split decompositions [Cun82]. Split decompositions can be seen as a generalization of modular decompositions and it is not unreasonable to expect that our methods carry over to them.

Distance widths beyond treewidth There is little to add as a conclusion regarding distance widths, however we outline further research directions. Most importantly, the general problem, whether Canonical Labeling parameterized by path or tree distance width is fixed-parameter tractable, is still open. It might be further interesting to study distance colorings based on the minimal distance to c sets instead of just one (= path distance decomposition) and whether faster algorithms than the tower-of-groups approach exist for $c = 2, 3, \ldots$. Note that the size of the color classes would be no longer covered by the treewidth, since grids have metric dimension 2, that is, the distances to e.g. $(1, 1)$ and $(1, m)$ are unique in the grid $G_{l \times m}$ [MT84].

Impact of parameterized complexity on Graph Isomorphism The conclusion regarding the fruitfulness of the interplay of parameterized complexity and Graph Isomorphism depends on the viewpoint. From the algorithmic or practical stance the notion of fixed-parameter tractability has motivated several results. The entire concept of distance widths was introduced [YBFT99] to examine the fixed-parameter tractability of Graph Isomorphism w.r.t. to parameters similar to the treewidth. Moreover all foreign vital lemmas in [KS10] come from research in parameterized complexity [Cai96; CFLLV08; RSS06], so it seems unlikely that

the findings in [KS10] would have been made and stated without the explicit notion of fixed-parameter tractability. To a lesser extent this also holds for the tree-depth [BDK12], especially if we keep in mind that a large part of this article discusses why the number of choices for the roots in each level is bounded. On the other hand the tree-depth itself has a variety of applications and definitions and the alternative way to proof the above mentioned bound on the number of root candidates comes from [DGT12], which is a purely combinatorial article.

The evaluation changes if we take the perspective of (parameterized) complexity theory. While the positive results for parameters like color multiplicity and the absence of such results for parameters like treewidth or genus after two decades of research may hint towards different complexities of their parameterized graph isomorphism problems, there is yet no hardness result, that could underpin this impression. Not to mention that the situation seemed similar for the eigenvalue multiplicity until the findings in [EP99]. As $\mathsf{W}[1]$-hardness of a parameterized problem implies that it is not in P unless $\mathsf{W}[1] = \mathsf{FPT}$ Kratsch and Schweitzer [KS10] ask for a weaker notion of hardness, i.e. hardness for a class C, $\mathsf{FPT} \subseteq \mathsf{C} \subseteq \mathsf{W}[1]$. A parameterized analogon of the classical class GI (see [KST93]) could fulfill this role, however there is no obvious candidate for a class para-GI, such that e.g. Graph Isomorphism$(\mathrm{tw} = k)$ would be hard for this class.

Note that, by contrast, it is easy to define a class like

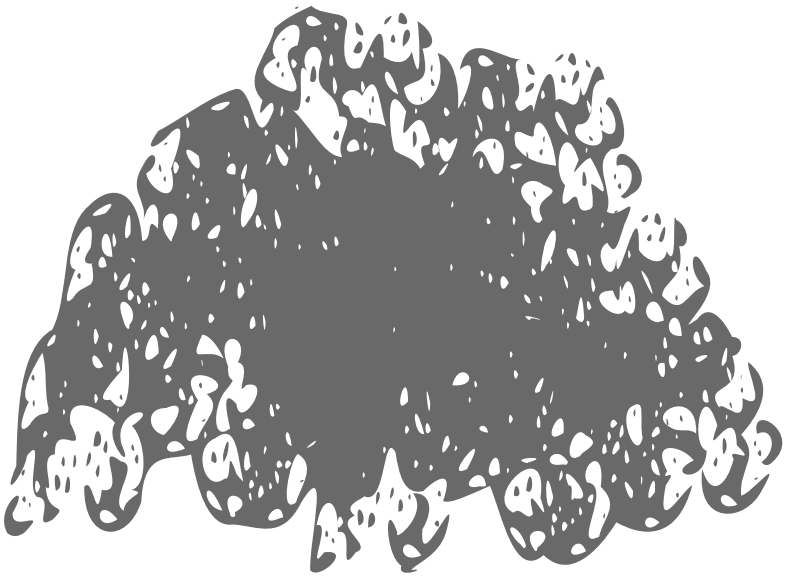
$$\mathsf{C} = \{(L, \kappa) \mid (L, \kappa) \leq^{\text{fpt-T}} \text{Graph Isomorphism}(\text{one})\}$$

to rephrase classical results in the language of parameterized complexity. Since, for instance, Graph Isomorphism for bipartite graphs is GI-complete [BC79], we could say that Graph Isomorphism$(\chi = k)$ is C-hard, where χ is the chromatic number, because Graph Isomorphism$(\text{one}) \leq^{\text{fpt-T}}$ Graph Isomorphism$(\chi = k)$. However, this is only helpful if one does not like to mix notion from classical and parameterized complexity and gives no additional knowledge.

All in all we conclude, that the parameterized complexity of Graph Isomorphism remains mainly uncharted and that the picture in the parameterized world is even more diverse than in the classical world, thus the need for further research is unabated.

Part IV.

Appendix



. . . and gather up the leaves.

List of Figures

List of Algorithms

Bibliography

[AHU74] A. Aho, J. Hopcroft and J. Ullman. ***The Design and Analysis of Computer Algorithms***. Addison-Wesley, 1974 (cit. on pp. 31, 32, 115, 120).

[Bab79] L. Babai. ***Monte Carlo Algorithms in Graph Isomorphism Testing***. Tech. rep. Université de Montréal, 1979 (cit. on pp. 37, 42, 43, 46, 113–115).

[BC79] K. S. Booth and C. J. Colbourn. ***Problems polynomially equivalent to graph isomorphism***. Tech. rep. TR 77-04. Computer Science Department, Univiversity of Waterloo, 1979 (cit. on pp. 26, 117).

[BDK12] A. Bouland, A. Dawar and E. Kopczyński. ***On tractable parameterizations of graph isomorphism***. Parameterized and Exact Computation. Springer, 2012, pp. 218–230 (cit. on pp. 92, 97–100, 114, 115, 117, 120).

[BGHK95] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson and T. Kloks. ***Approximating treewidth, pathwidth, frontsize, and shortest elimination tree***. Journal of Algorithms 18.2 (1995), pp. 238–255 (cit. on p. 96).

[BGM82] L. Babai, D. Y. Grigoryev and D. M. Mount. ***Isomorphism of graphs with bounded eigenvalue multiplicity***. Proceedings of the fourteenth annual ACM symposium on Theory of computing. STOC '82. San Francisco, California, United States: ACM, 1982, pp. 310–324 (cit. on p. 113).

[BK79] L. Babai and L. Kučera. ***Canonical labelling of graphs in linear average time***. Proceedings of the 20th Annual Symposium on Foundations of Computer Science. Washington, DC, USA: IEEE, 1979, pp. 39–46 (cit. on p. 29).

[BL83] L. Babai and E. M. Luks. ***Canonical labeling of graphs***. Proceedings of the fifteenth annual ACM symposium on Theory of computing. STOC '83. New York, NY, USA: ACM, 1983, pp. 171–183 (cit. on pp. 46, 113, 114).

[Bod+98] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller and Z. Tuza. ***Rankings of graphs***. SIAM Journal on Discrete Mathematics 11.1 (1998), pp. 168–181 (cit. on pp. 92, 93).

[Bod90] H. L. Bodlaender. ***Polynomial algorithms for graph isomorphism and chromatic index on partial$k$-trees***. Journal of Algorithms 11.4 (1990), pp. 631–643 (cit. on pp. 20, 113, 114).

[Bod96] H. L. Bodlaender. ***A linear-time algorithm for finding tree-decompositions of small treewidth***. SIAM Journal on computing 25.6 (1996), pp. 1305–1317 (cit. on p. 96).

[Cai96] L. Cai. ***Fixed-parameter tractability of graph modification problems for hereditary properties***. Information Processing Letters 58.4 (1996), pp. 171–176 (cit. on pp. 50, 51, 116, 120).

[CFI92] J. yi Cai, M. Fürer and N. Immerman. ***An optimal lower bound on the number of variables for graph identifications.*** Combinatorica 12.4 (1992), pp. 389–410 (cit. on p. 29).

[CFLLV08] J. Chen, F. V. Fomin, Y. Liu, S. Lu and Y. Villanger. ***Improved algorithms for feedback vertex set problems***. Journal of Computer and System Sciences 74.7 (2008), pp. 1188–1198 (cit. on pp. 55, 56, 116).

[CHM81] M. Chein, M. Habib and M.-C. Maurer. ***Partitive hypergraphs***. Discrete mathematics 37.1 (1981), pp. 35–50 (cit. on p. 67).

[Cou90] B. Courcelle. ***Graph Rewriting: An Algebraic and Logic Approach.*** Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics. Ed. by J. van Leeuwen. MIT Press, 1990, pp. 193–242 (cit. on pp. 93, 95).

[Cou96] B. Courcelle. ***The definition in monadic second-order logic of modular decompositions of ordered graphs***. Graph Grammars and Their Application to Computer Science. Vol. 1073. Lecture Notes in Computer Science. Springer, 1996, pp. 487–501 (cit. on p. 70).

[CPS85] D. Corneil, Y. Perl and L. Stewart. ***A Linear Recognition Algorithm for Cographs***. SIAM Journal on Computing 14.4 (1985), pp. 926–934 (cit. on p. 63).

[CTW08] Y. Chen, M. Thurley and M. Weyer. ***Understanding the Complexity of Induced Subgraph Isomorphisms***. Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I. ICALP '08. Reykjavik, Iceland: Springer-Verlag, 2008, pp. 587–596 (cit. on pp. 16, 19).

[Cun82] W. H. Cunningham. ***Decomposition of directed graphs***. SIAM Journal on Algebraic Discrete Methods 3.2 (1982), pp. 214–228 (cit. on p. 116).

[Dam91] P. Damaschke. ***Induced subgraph isomorphism for cographs is NP-complete***. Proc. 16th Int. Worksh. Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science 484. Springer-Verlag, 1991, pp. 72–78 (cit. on p. 63).

[DF95a] R. Downey and M. Fellows. ***Fixed-parameter tractability and completeness I: Basic results***. SIAM J. Comput. 24.4 (1995), pp. 873–921 (cit. on p. 13).

[DF95b] R. Downey and M. Fellows. ***Fixed-parameter tractability and completeness II: On completeness for W [1]***. Theoretical Computer Science 141.1-2 (1995), pp. 109–131 (cit. on p. 13).

[DF95c] R. G. Downey and M. R. Fellows. ***Parameterized Computational Feasibility***. Ed. by P. Clote and J. Remmel. Birkhäuser, 1995, pp. 219–244 (cit. on p. 55).

[DF99] R. G. Downey and M. R. Fellows. ***Parameterized complexity***. Springer, 1999 (cit. on pp. 13, 19).

[DGT12] Z. Dvořák, A. C. Giannopoulou and D. M. Thilikos. ***Forbidden graphs for tree-depth***. European Journal of Combinatorics 33.5 (2012), pp. 969–979 (cit. on pp. 94, 97, 117).

[DTW12] B. Das, J. Torán and F. Wagner. ***Restricted space algorithms for isomorphism on bounded treewidth graphs***. Information and Computation 217 (2012), pp. 71–83 (cit. on pp. 85, 90, 115).

[EP62] P. Erdős and L. Pósa. ***On the maximal number of disjoint circuits of a graph***. Publicationes Mathematicae Debrecen 9 (1962), pp. 3–12 (cit. on p. 57).

[EP99] S. Evdokimov and I. Ponomarenko. ***Isomorphism of coloured graphs with slowly increasing multiplicity of Jordan blocks***. Combinatorica 19.3 (1999), pp. 321–333 (cit. on pp. 20, 113–115, 117).

[EST12] M. Elberfeld, C. Stockhusen and T. Tantau. ***On the space complexity of parameterized problems***. Parameterized and Exact Computation. IPEC 2012. Springer, 2012, pp. 206–217 (cit. on p. 21).

[FFG02] J. Flum, M. Frick and M. Grohe. ***Query evaluation via tree-decompositions***. Journal of the ACM 49.6 (2002), pp. 716–752 (cit. on pp. 95, 96).

[FG03] J. Flum and M. Grohe. ***Describing parameterized complexity classes***. Information and Computation 187.2 (2003), pp. 291–319 (cit. on p. 21).

[FG06] J. Flum and M. Grohe. ***Parameterized complexity theory***. Springer-Verlag New York Inc, 2006 (cit. on pp. 13, 14, 17, 19, 20, 95).

[FHL80] M. Furst, J. Hopcroft and E. Luks. ***Polynomial-time algorithms for permutation groups***. Foundations of Computer Science, 1980., 21st Annual Symposium on. 1980 (cit. on pp. 37–42, 113, 114, 120).

[FM80] I. S. Filotti and J. N. Mayer. ***A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus***. Proceedings of the twelfth annual ACM symposium on Theory of computing. STOC '80. Los Angeles, California, United States: ACM, 1980, pp. 236–243 (cit. on p. 114).

[Gal67] T. Gallai. ***Transitiv orientierbare Graphen***. Acta Mathematica Academiae Scientiarum Hungarica 18 (1967), pp. 25–66 (cit. on pp. 64, 65).

[Gur97] Y. Gurevich. ***From invariants to canonization***. Bulletin of the EATCS 63 (1997) (cit. on pp. 26, 29).

[Hof82] C. M. Hoffmann. ***Group-theoretic algorithms and graph isomorphism***. Vol. 136. Lecture Notes in Computer Science. Springer, 1982 (cit. on p. 25).

[HP10] M. Habib and C. Paul. ***A survey of the algorithmic aspects of modular decomposition***. Computer Science Review 4.1 (2010), pp. 41–59 (cit. on pp. 64, 67).

[Kar72] R. M. Karp. ***Reducibility Among Combinatorial Problems***. Complexity of Computer Computations. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103 (cit. on pp. 14, 55).

[KL81] P. Klingsberg and E. M. Luks. ***Succinct certificates for a class of graphs***. St. Joseph's University, preprint. 1981 (cit. on pp. 46, 114, 115).

[KMP77] D. E. Knuth, J. H. Morris and V. R. Pratt. ***Fast pattern matching in strings.*** SIAM Journal on Computing 6 (1977), pp. 323–350 (cit. on pp. 34, 39).

[KMS95] M. Katchalski, W. McCuaig and S. Seager. ***Ordered colourings***. Discrete Mathematics 142.1 (1995), pp. 141–154 (cit. on p. 92).

[KS10] S. Kratsch and P. Schweitzer. ***Isomorphism for graphs of bounded feedback vertex set number***. Algorithm Theory-SWAT 2010 (2010), pp. 81–92 (cit. on pp. 33, 52–54, 58–60, 114, 116, 117, 120).

[KST93] J. Köbler, U. Schöning and J. Torán. ***The Graph Isomorphism Problem: Its Structural Complexity***. Birkhäuser, 1993 (cit. on p. 117).

[Lam12] M. Lampis. ***Algorithmic meta-theorems for restrictions of treewidth***. Algorithmica 64.1 (2012), pp. 19–37 (cit. on p. 109).

[Lin92] S. Lindell. ***A logspace algorithm for tree canonization***. Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. ACM. 1992, pp. 400–404 (cit. on pp. 85, 92, 98, 115).

[Luk10] E. M. Luks. ***Permutation Groups in Parallel: Canonical Forms***. Ohio State University. Combinatorics, Groups, Algorithms, and Complexity: Conference in honor of Laci Babai's 60th birthday. 2010. URL: `http://www.babai60.org/slides/luks.pdf` (cit. on pp. 46, 115).

[Luk82] E. M. Luks. ***Isomorphism of graphs of bounded valence can be tested in polynomial time***. Journal of Computer and System Sciences 25 (1982), pp. 42–65 (cit. on p. 20).

[Mat79] R. Mathon. ***A note on the graph isomorphism counting problem***. Information Processing Letters 8.3 (1979), pp. 131–136 (cit. on p. 25).

[Mil79] G. L. Miller. ***Graph isomorphism, general remarks***. Journal of Computer and System Sciences 18.2 (1979), pp. 128–142 (cit. on pp. 5, 22).

[Mil80] G. Miller. ***Isomorphism testing for graphs of bounded genus***. Proceedings of the twelfth annual ACM symposium on Theory of computing. ACM. 1980, pp. 225–235 (cit. on p. 114).

[MT84] R. A. Melter and I. Tomescu. ***Metric bases in digital geometry***. Computer Vision, Graphics, and Image Processing 25.1 (1984), pp. 113–121 (cit. on p. 116).

[Nie06] R. Niedermeier. ***Invitation to Fixed-Parameter Algorithms***. Oxford Univerity Press, 2006 (cit. on pp. 13, 14, 19).

[NO06] J. Nešetřil and P. Ossona de Mendez. ***Tree-depth, subgraph coloring and homomorphism bounds***. European Journal of Combinatorics 27.6 (2006), pp. 1022–1041 (cit. on pp. 92–94).

[NO12] J. Nešetřil and P. Ossona de Mendez. ***Sparsity: Graphs, Structures, and Algorithms***. Algorithms and Combinatorics 28. Springer, 2012 (cit. on pp. 93, 94, 96, 97).

[Ota12] Y. Otachi. ***Isomorphism for Graphs of Bounded Connected-Path-Distance-Width***. Algorithms and Computation. Springer, 2012, pp. 455–464 (cit. on pp. 73, 74, 77–79, 104, 114, 115, 120).

[RS04] N. Robertson and P. D. Seymour. ***Graph minors. XX. Wagner's conjecture***. Journal of Combinatorial Theory, Series B 92.2 (2004), pp. 325–357 (cit. on p. 93).

[RS83] N. Robertson and P. D. Seymour. ***Graph minors. I. Excluding a forest***. Journal of Combinatorial Theory, Series B 35.1 (1983), pp. 39–61 (cit. on p. 103).

[RS84] N. Robertson and P. D. Seymour. ***Graph minors. III. Planar tree-width***. Journal of Combinatorial Theory, Series B 36.1 (1984), pp. 49–64 (cit. on p. 15).

[RS86] N. Robertson and P. D. Seymour. ***Graph minors. II. Algorithmic aspects of tree-width***. Journal of algorithms 7.3 (1986), pp. 309–322 (cit. on p. 15).

[RS91] N. Robertson and P. D. Seymour. ***Graph minors. X. Obstructions to tree-decomposition***. Journal of Combinatorial Theory, Series B 52.2 (1991), pp. 153–190 (cit. on p. 103).

[RSS06] V. Raman, S. Saurabh and C. R. Subramanian. ***Faster fixed parameter tractable algorithms for finding feedback vertex sets***. ACM Transactions on Algorithms 2.3 (2006), pp. 403–415 (cit. on pp. 56, 116).

[RSV04] B. Reed, K. Smith and A. Vetta. ***Finding odd cycle transversals***. Operations Research Letters 32.4 (2004), pp. 299–301 (cit. on p. 55).

[Sab61] G. Sabidussi. ***Graph derivatives***. Mathematische Zeitschrift 76.1 (1961), pp. 385–401 (cit. on p. 64).

[Sch88] U. Schöning. ***Graph isomorphism is in the low hierarchy***. Journal of Computer and System Sciences 37.3 (1988), pp. 312–323 (cit. on p. 3).

[See85] D. Seese. ***Tree-partite graphs and the complexity of algorithms***. Fundamentals of Computation Theory. Ed. by L. Budach. Springer. 1985, pp. 412–421 (cit. on p. 15).

[Spi83] J. Spinrad. ***Transitive orientation in $\mathcal{O}(n^2)$ time***. Proceedings of the fifteenth annual ACM symposium on Theory of computing. ACM. 1983, pp. 457–466 (cit. on p. 64).

[Sum73] D. P. Sumner. ***Graphs indecomposable with respect to the X-join***. Discrete Mathematics 6.3 (1973), pp. 281–298 (cit. on p. 62).

[TCHP08] M. Tedder, D. Corneil, M. Habib and C. Paul. ***Simpler linear-time modular decomposition via recursive factorizing permutations***. Automata, Languages and Programming. Springer, 2008, pp. 634–645 (cit. on pp. 67, 109).

[Wag11] F. Wagner. ***Graphs of Bounded Treewidth Can Be Canonized in*** AC^1. Computer Science--Theory and Applications: : 6th International Computer Science Symposium in Russia. Springer, 2011, pp. 209–222 (cit. on p. 114).

[Wei76] B. Weisfeiler, ed. ***On Construction and Identification of Graphs***. Lecture Notes in Mathematics. Springer, 1976 (cit. on p. 29).

[WL68] B. Weisfeiler and A. A. Lehman. ***Reduction of a graph to a canonical form and an algebra which appears in the process***. Russian. Nauchno-Technicheskaya Informatsia 2.9 (1968), pp. 12–16 (cit. on p. 29).

[YBFT99] K. Yamazaki, H. L. Bodlaender, B. de Fluiter and D. M. Thilikos. ***Isomorphism for graphs of bounded distance width***. Algorithmica 24.2 (1999), pp. 105–127 (cit. on pp. 17, 73–75, 80, 82–84, 102, 104, 105, 112, 114–116, 120).

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Diplomarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 25. September 2013 .....

Statement of authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Berlin, September 25, 2013 .....