Computing Shrub-Depth Decompositions

Jakub Gajarský
Technical University Berlin, Germany
jakub.gajarsky@tu-berlin.de

Stephan Kreutzer
Technical University Berlin, Germany
stephan.kreutzer@tu-berlin.de

Abstract

Shrub-depth is a width measure of graphs which, roughly speaking, corresponds to the smallest depth of a tree into which a graph can be encoded. It can be thought of as a low-depth variant of clique-width (or rank-width), similarly as treedepth is a low-depth variant of treewidth. We present an fpt algorithm for computing decompositions of graphs of bounded shrub-depth. To the best of our knowledge, this is the first algorithm which computes the decomposition directly, without use of rank-width decompositions and FO or MSO logic.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Combinatorial algorithms

Keywords and phrases shrub-depth, tree-model, decomposition, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.56

Funding The research is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).

Acknowledgements We want to thank Sang-il Oum for suggesting that our techniques may be used to obtain the results of Section 6.

1 Introduction

Among the numerous width parameters used in graph theory and algorithmics, treewidth and its dense counterparts clique-width and rank width are arguably the most prominent and extensively studied. In recent years, more restrictive parameters, which could be collectively called depth parameters, are attracting increasing attention. The best-known of these is treedepth [10], which can be seen as a low-depth variant of treewidth. Inspired by the usefulness of treedepth, the authors of [8] defined the notion of shrub-depth, which can be seen as a low-depth variant of clique-width, analogously to the relation between treedepth and treewidth.

Since shrub-depth is a more restrictive notion than clique-width, it is natural to ask what algorithmic advantages it offers over clique-width, if any. The question whether some problems which are parameterized intractable on graphs of bounded clique-width are fixed-parameter tractable on graphs of bounded shrub-depth was addressed in [7], where it was shown that the Hamiltonian path and the chromatic number problem remain hard on graph classes of bounded shrub-depth. On the other hand, bounded shrub-depth offers certain quantitative advantages over clique-width. For instance, a well-known result of Courcelle, Makowski and Rotics [2] states that every MSO definable property φ of graphs can be solved in time $f(\varphi) \cdot |V(G)|$ on any class of graphs of bounded clique-width. The price for the generality of this result is that the function f is non-elementary, i.e. it grows like a tower of exponentials whose height depends on the formula. But as shown in [5], on any graph class shrub-depth d the function f is only d-fold exponential, i.e. its height does not depend on the formula.

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 56; pp. 56:1–56:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Besides being interesting in their own right, shrub-depth has important consequences for much more general graph classes. To explain this, we will first describe the analogous situation in the case of sparse graphs and treedepth. Nešetřil and Ossona de Mendez introduced [11] the notions of bounded expansion and nowhere denseness as a general approach to studying sparse graphs in a unified setting. Graphs from classes of bounded expansion and nowhere dense graph classes can be decomposed[1] into several overlapping graphs each of small treedepth. Such a decomposition has successfully been used for solving many problems on graph classes of bounded expansion and nowhere dense graph classes efficiently. In some cases, the problems can be reduced to solving them on the graphs of small treedepth obtained by this decomposition, and in more complicated cases one can repeatedly perform computations on graphs of small treedepth from the decomposition and then combine the results.

Recently a dense counterpart of the notion of bounded expansion has been studied under the name *structurally bounded expansion*, and the results of [6] indicate that shrub-depth could play a role analogous to treedepth in the sparse case: it was shown that every graph from a class of graphs of structurally bounded expansion can be decomposed into a bounded number of subgraphs of small shrub-depth and this decomposition can then be used for algorithmic purposes. Given this promising application of shrub-depth in the theory of dense but structurally simple graphs, it is very likely that a better structural understanding of shrub-depth will be of increasing relevance in the near future.

All width and depth measures mentioned so far are defined by an associated concept of decomposition. For shrub-depth this decomposition is called a *tree-model*. Unlike other width measures, tree-models are defined in terms of two parameters, commonly denoted by d and m. To use any of the mentioned depth or width measures algorithmically, one usually needs to be able to compute the corresponding types of decompositions for an input graph G. In most cases, the problem of finding an optimal decomposition of an input graph is NP-hard, but often for fixed values of the relevant width parameters one can, in polynomial time, either find a decomposition with the prescribed value or correctly decide that no such decomposition exists. This is the case for treewidth [1], rank-width [9] and treedepth [12]. Algorithms like this are called *parameterized algorithms* and are studied in the framework of parameterized complexity theory. We refer to [3] for an indepth introduction to parameterized complexity and only briefly recall the concepts from parameterized complexity needed below. A *parameterized problem* P is essentially a classical problem but in addition to the normal input instance w we are given an integer k, the so-called *parameter*. The problem P is called *fixed-parameter tractable*, or in the complexity class FPT, if there is a computable function f and a constant c such that the problem can be solved by an algorithm whose running time on input (w, k) is bounded by $f(k) \cdot |w|^c$. The class FPT can be seen as the parameterized equivalent to the classical complexity class P as abstraction of efficiently solvable problems.

A much weaker requirement on the running time is imposed by the parameterized complexity class XP. The problem P is in XP if there is a computable function f such that the problem can be solved by an algorithm whose running time is bounded by $|G|^{f(k)}$. Thus, a problem is in XP if it can be solved in polynomial time for every fixed value of the parameter k.

Our contribution. We provide combinatorial and conceptually simple algorithms for computing tree-models with given parameters d and m of input graphs G. To obtain our algorithms, in Section 3 and 4, we introduce a new concept of k-modules and prove several

[1] The precise meaning of this is rather technical and is not important for our purpose, so we omit the precise definition.

properties relating k-modules and tree-models of graphs. The structural results we obtain provide a new and very different insight into the structure of graphs of low shrub-depth which we believe will be of further interest. These results provide the basis for our algorithms for computing tree-models, which we present in Section 5. Our main algorithmic result is an fpt-algorithm for computing tree-models with prescribed parameters d and m in a graph G, provided G has such a model. Finally, in Section 6, we present another application of our results to forbidden induced subgraphs of graph classes of bounded shrub-depth.

Previous work. To the best of our knowledge, no papers explicitly address the problem of computing tree-models with given parameters d and m of a given input graph. However, there exists a folklore fpt algorithm for computing an optimal SC-decomposition of a given graph, where SC-decomposition is a notion closely related to tree-model. This algorithm requires computing a rank decomposition of the input graph G first and then uses a powerful algorithmic metatheorem for MSO logic to obtain the SC-decomposition. Similarly, the results and techniques of [5] can likely be adjusted to compute tree-models, but also in this case one would have to rely on computing a rank decomposition first and using a logical metatheorem.

2 Preliminaries

For $n \in \mathbb{N}$ we denote the set $\{1, \dots, n\}$ by $[n]$. For sets A, B we denote by $A \Delta B$ their symmetric difference $(A \setminus B) \cup (B \setminus A)$.

Graphs. All graphs in this paper are finite, undirected and simple. We use standard graph theoretic notation, see e.g. [4]. Let G be a graph. If $X \subseteq V(G)$ we denote by $G[X]$ the subgraph of G induced by X and by $G - X$ the subgraph induced by $V \setminus X$. If $X = \{v\}$ is a singleton set, we simply write $G - v$ for $G - \{v\}$. We denote by $N_G(v)$ the *neighbourhood* of $v \in V(G)$ in G. If G is understood we omit the index and just write $N(v)$. Let G, H be graphs and let $\lambda : V(G) \to [m]$ and $\lambda' : V(H) \to [m]$ be labelling functions. A *label preserving isomorphism* between (G, λ) and (H, λ') is a bijective function $f : V(G) \to V(H)$ such that $\{u, v\} \in E(G)$ if, and only if, $\{f(u), f(v)\} \in E(H)$ and $\lambda'(f(v)) = \lambda(v)$ for all $u, v \in V(G)$.

Trees. By a tree in this paper we mean a rooted connected acyclic graph. Let T be a tree with root r. The *ancestors* of t in T are the vertices on the unique path from r to t in T other than t itself. The *parent* of t is the ancestor of t adjacent to t. The root r itself does not have a parent. For $t \in V(T) \setminus \{r\}$ we define the *subtree* T_t *of* T *rooted at* t as the component of $T - e$ containing t, where e is the edge incident to t and its parent. For $t = r$ we set $T_r = T$. The *children* of t are the neighbours of t other than the parent. The *descendants* of t are the vertices in $V(T_t) \setminus \{t\}$.

A *leaf* of T is a node of degree 1 which is not the root. We denote the set of leaves of T by *leaves*(T). Nodes $s, t \in V(T)$ are *comparable* (in T) if $t \in V(T_s)$ or $s \in V(T_t)$. Otherwise they are *incomparable*. The *height* of t is the maximal length of a path from t to a leaf of T_t. Given a set $X \subseteq V(T)$ we define the *least common ancestor* of X, denoted by *lca*(X), as the node $t \in V(T)$ of minimal height such that $X \subseteq V(T_t)$. We also write *lca*$(u_1, \dots, u_t)$ for *lca*$(\{u_1, \dots, u_t\})$. The *distance* between two nodes $s, t \in V(T)$, denoted by $dist_T(s, t)$, is the length of the unique path between s and t in T.

Shrub-depth. Shrub-depth was defined by Ganian et al. in [8]. It is defined using the following notion of *tree-model*.

▶ **Definition 2.1.** *Let d and m be non-negative integers and let G be a graph. A* tree-model *of G is a triple (T, S, λ), where T is a tree, $S \subseteq [m]^2 \times [d]$ is a relation, and $\lambda : leaves(T) \to [m]$ is a function, such that*

- **i.** *the length of each root-to-leaf path is exactly d,*
- **ii.** *the set $leaves(T)$ of leaves is exactly $V(G)$,*
- **iii.** *$(i, j, d) \in S$ if, and only, if $(j, i, d) \in S$ (symmetry in the colours), and*
- **iv.** *for any two vertices $u, v \in V(G)$, if $\lambda(v) = i$, $\lambda(v) = j$ and $dist_T(u, v) = 2l$, then $\{u, v\} \in E(G)$ if, and only if, $(i, j, l) \in S$.*

Note that the leaves $leaves(T)$ of T are the vertices of G. Thus, if $v \in V(G)$, then $v \in leaves(T)$ and therefore $\lambda(t)$ is defined. The number d in the above definition is referred to as the depth of the tree-model. We will often speak of a (d, m)-tree-model instead of a "tree-model of depth d with m colours".

Note that every graph G has a tree-model of depth 1 with $|V(G)|$ colours (each vertex gets its own colour and the relation S is essentially $E(G)$). Thus it does not make sense to ask about the smallest depth of a tree-model of a graph G. This is the reason why the notion of *shrub-depth* is defined only for classes of graphs.

▶ **Definition 2.2.** *The* shrub-depth *of a graph class $\mathcal{C}$ is the smallest $d > 0$ for which there exists an $m > 0$ such that every graph $G \in \mathcal{C}$ has a (d, m)-tree-model.*

We remark that even though the shrub-depth of a graph class $\mathcal{C}$ is defined as one number (d in the above definition), to each class $\mathcal{C}$ of graphs of bounded shrub-depth there actually correspond two numbers – d and m from the above definition. Thus, in what follows, we usually work directly with (classes of) graphs which have a (d, m)-tree-model for some fixed d and m.

▶ **Definition 2.3.** *Let G be a graph and $G' \subseteq G$ be an induced subgraph of G. Let $d, m > 0$. A (d, m)-tree-model (T, S, λ) of G* extends *a (d, m)-tree-model (T', S', λ') of G' if $T' \subseteq T$, $S = S'$ and $\lambda(v) = \lambda'(v)$ for all $v \in V(G')$.*

We frequently use the following result from [8] which follows immediately from the definition of shrub-depth: if (T, S, λ) is a (d, m)-tree-model of G, then we can obtain a (d, m)-tree-model (T', S', λ') of G' as follows: the tree T' is the minimal subtree of T containing the root of T and all leaves form $V(G')$. Similarly, λ' is the restriction of λ to $V(G')$ and $S' = S$. In particular, the tree-structure of T is preserved in the reduced tree-model T'.

▶ **Proposition 2.4** ([8]). *Let $d, m > 0$. If G has a (d, m)-tree-model and G' is an induced subgraph of G, then G' also has a (d, m)-tree-model.*

3 Twin tuples, k-modules and outline of our approach

In this section we introduce the concepts of *twin tuples* and *(strict) k-modules* which will be pivotal in the rest of the paper and briefly outline the key idea behind our algorithm for finding a (d, m)-tree-model of an input graph G.

3.1 Twin tuples and (strict) k-modules

▶ **Definition 3.1.** *Let $k \in \mathbb{N}$ and let G be a graph. Two disjoint tuples $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_k) \in V(G)^k$ are* twin tuples *if*

1. *the function $f(a_i) = b_i$, $1 \leq i \leq k$, is an isomorphism between $G[\{a_1, \ldots, a_k\}]$ and $G[\{b_1, \ldots, b_k\}]$,*
2. *$\{a_i, b_j\} \in E(G)$ if, and only if, $\{a_j, b_i\} \in E(G)$, for all $1 \leq i < j \leq k$, and*
3. *$N(a_i) \setminus \{a_1, \ldots, a_k, b_1, \ldots, b_k\} = N(b_i) \setminus \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$ for all $1 \leq i \leq k$.*

For $k = 1$ we simply call a_1 and b_1 twins*. A set M of pairwise disjoint k-tuples of vertices of G is a* structured k-module *if all tuples in M are pairwise twin tuples.*

The next definition introduces a different characterisation of k-modules which we will use frequently in the sequel. The equivalence between the two definitions is easily seen (and stated formally in the lemma thereafter).

▶ **Definition 3.2.** *Let $k \in \mathbb{N}$ and let α, β be symmetric relations on $[k]^2$. Let G be a graph. A set $M \subseteq V(G)^k$ of pairwise disjoint k-tuples is an* (α, β)–module *if*

1. *for every $(a_1, \ldots, a_k) \in M$, $\{a_i, a_j\} \in E(G)$ if, and only, if $(i, j) \in \alpha$,*
2. *for all distinct tuples $a_1, \ldots, a_k, b_1, \ldots, b_k \in M$, $\{a_i, b_j\} \in E(G)$ if, and only, if $(i, j) \in \beta$, and*
3. *$N(a_i) \setminus S = N(b_i) \setminus S$ for all $1 \leq i \leq k$ and all $(a_1, \ldots, a_k), (b_1, \ldots, b_k) \in M$, where $S := \bigcup\{a_i : 1 \leq i \leq k, (a_1, \ldots, a_k) \in M\}$.*

*We call k-tuples $\bar{a}, \bar{b} \in V(G)^k$ (α, β)-*twins*, if $\{\bar{a}, \bar{b}\}$ is an (α, β)-module in G.*

▶ **Lemma 3.3.** *A set M of pairwise distinct k-tuples is a structured k-module if, and only if, there are symmetric relations α, β on $[k]^2$ such that M is an (α, β)-module.*

Thus, in a structured k-module M, the relation α determines the adjacency within each tuple $\bar{a} \in M$, or the isomorphism type of the subgraphs of G induced by the tuples in the module and the relation β fixes the adjacency between different tuples from M. Furthermore, any two vertices at the same position within their respective tuples have the same adjacency to the vertices outside the module. The notion of k-module is illustrated on Figure 1.

Given α, β as above, we say that the structured k-module M is *determined* or *induced* by α, β. Also, if $\bar{a}$ and $\bar{b}$ are twin tuples such that the adjacency within $\bar{a}$ and $\bar{b}$ and between $\bar{a}$ and $\bar{b}$ is determined by relations α and β, we say that $\bar{a}$ and $\bar{b}$ are (α, β)-twin tuples. The following simple fact will be used often in the sequel.

▶ **Lemma 3.4.** *Let $\bar{a}, \bar{b}, \bar{c} \in V(G)^k$ be tuples such that $\bar{a}$ is an (α, β)-twin tuple of $\bar{b}$ and $\bar{b}$ is an (α, β)-twin tuple of $\bar{c}$. Then $\bar{a}$ is an (α, β)-twin tuple of $\bar{c}$. In other words, for any fixed α and β the relation of being (α, β)-twin tuples is transitive.*

The next lemma captures the intuition behind k-modules and their connection to tree-models.

▶ **Lemma 3.5.** *Let (T, S, λ) be a tree-model of a graph G and let $u \in V(T)$ be a node with $L \geq 2$ children $\{1, \ldots, L\}$ such that for any pair i, j of its children there exists a label preserving isomorphism ι_{ij} between T_i and T_j. Then G contains a k-module with L tuples, where $k = |\mathit{leaves}(T_1)|$.*

Proof. Fix an ordering $\leq_1$ on $\mathit{leaves}(T_1)$. Then, for every $j > 1$, we define an order $\leq_j$ on $\mathit{leaves}(T_j)$ as follows: $u \leq_j v$, for $u, v \in \mathit{leaves}(T_1)$, if, and only if, $\iota_{j1}(u) \leq_1 \iota_{j1}(v)$. Each pair $(\mathit{leaves}(T_i), \leq_i)$ can be thought of as a k-tuple, and it is easy to see that $M := \{(\mathit{leaves}(T_1), \leq_1), \ldots, (\mathit{leaves}(T_L), \leq_L)\}$ is a k-module with L tuples as claimed in the statement of the lemma. ◀

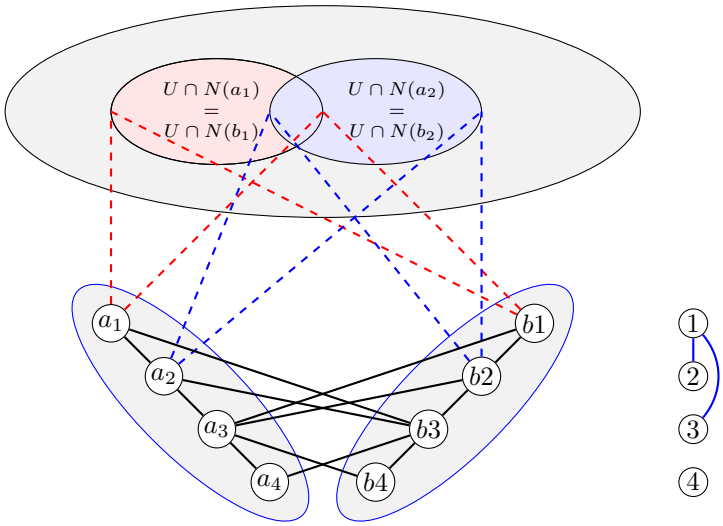

Figure 1 An example of a 4-module M with two tuples (a_1, a_2, a_3, a_4) and (b_1, b_2, b_3, b_4). The set U is $V(G) \setminus \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$. Relations α and β are defined as follows: $\alpha = \{(1,2),(2,1),(2,3),(3,2),(3,4),(4,3)\}$, $\beta = \{(1,3),(3,1),(2,3),(3,2),(3,4),(4,3)\}$. The conflict graph of M (Definition 3.7) is on the right. Note that $M' = \{(a_1, a_2, a_3)(b_1, b_2, b_3)\}$ is also a module, and since its conflict graph (only on vertices $1, 2, 3$) is connected, it is an example of a strict module (Definition 3.8).

3.2 Outline of our approach

We now present an outline of our approach for computing tree-models. By Lemma 3.5, if a tree-model of a graph G contains several isomorphic subtrees with common parent, then these subtrees induce a k-module in G. If the converse statement

> $(*)$ if $\bar{a}$, $\bar{b}$, $\bar{c}, \ldots$, form a module M of G, then in every tree-model T of G the vertices of tuples $\bar{a}, \bar{b}, \bar{c}, \ldots$ are the leaves of distinct but isomorphic subtrees $T^a, T^b, T^c, \ldots$ with a common parent u

was also true, then this could be used to compute tree-models as follows: compute a module $M = \{\bar{a}, \bar{b}, \bar{c}\}$ in the input graph G and remove $\bar{c}$ from G to obtain G' with module $M' = \{\bar{a}, \bar{b}\}$. Then it would be enough to find *any* (d, m)-tree-model T' of G' which then can easily be extended to a tree-model T of G – by $(*)$ the tree-model T' contains different isomorphic subtrees T^a and T^b (representing $\bar{a}$ and $\bar{b}$) with a common parent u, and so we can create a copy T_c of T_a and add it as a child of u to create a tree-model T of G. This essentially means that by finding a module M in G we can, by deleting a tuple from M, reduce the problem of computing a tree-model with parameters d and m of G to a problem of finding a tree-model T' with the same parameters but for a smaller graph G'.

We will essentially follow this idea but use a weaker statement than $(*)$ instead. It turns out that even less than having isomorphic subtrees is enough to make the above idea work – it is enough to have a tree-model T' of G' in which two tuples $\bar{a}$ and $\bar{b}$ are in a "good position" with respect to each other, as shown in the following lemma, proved below, which is the basis of our approach.

▶ **Lemma 3.6.** *Let G be a graph and let $\{(a_1, \ldots, a_k), (b_1, \ldots, b_k), (c_1, \ldots, c_k)\}$ be a k-module in G. Let $G' = G - \{c_1, \ldots, c_k\}$ and let (T', S', λ') be a (d, m)-tree-model of G', for some $d, m > 0$, st.*

1. *$lca(\{a_1, \dots, a_k\})$ and $lca(\{b_1, \dots, b_k\})$ are incomparable in T' and*
2. *for all $i \in [k]$ the labels $\lambda'(a_i)$ and $\lambda'(b_i)$ are the same in T'.*

Then (T', S', λ') can be extended to a (d, m)-tree-model (T, S, λ) of G.

Unfortunately, the statement $(*)$ does not hold and neither does the following weaker statement $(**)$ which would still be strong enough for our purpose: if M is a k-module in a graph G, then in any tree model T of G there are at least two tuples $(a_1, \dots, a_k), (b_1, \dots, b_k) \in M$ which satisfy the requirements of Lemma 3.6 with respect to T. In general, if $\bar{a}$ and $\bar{b}$ are twin tuples of graph G, the vertices $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ can be placed almost arbitrarily "badly" in a tree-model T of G. In order to be able prove a variant of $(**)$, we will have to restrict ourselves to a more structured notion of module, which we call a *strict* module. Strict modules are modules in which the vertices in each tuple are forced to "stick together" – we want to avoid the situation when it is possible to exchange a_i for b_i between two tuples $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ without violating α or β. More generally, we want it to be impossible for any non-empty subset I of $[k]$ to exchange $\{a_i\}_{i \in I}$ for $\{b_i\}_{i \in I}$ between two tuples $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ without violating α and β. This will be accomplished using the notions of *conflict* and *conflict graph*.

▶ **Definition 3.7.** *Let M be a k-module of a graph G induced by relations α, β. We say that positions $i, j \in [k]$ with $i \neq j$ are in conflict if $(i, j) \in \alpha$ but $(i, j) \notin \beta$ or vice versa. The* conflict graph *$\mathcal{C}(M)$ of M is the graph with vertex set $[k]$ and an edge between i and j if, and only if, the positions i and j are in conflict.*

▶ **Definition 3.8.** *An (α, β)-module M is a* strict k-module *if its conflict graph is connected.*

The definition of strict k-modules will allow us to prove Lemma 4.1, which can be seen as a variant of $(**)$. Informally it says that if G is a graph with a large strict k-module M, then in any (d, m)-tree-model T of G there are two tuples in a "good" mutual position in T (i.e. in the position required in Lemma 3.6). With Lemma 4.1 at hand, it remains to bound the value k in terms of d and m and to show that sufficiently large strict k-modules always exist in graphs which have (d, m)-tree-models (Corollary 4.6). Finally, we need to design algorithms for computing large strict k-modules in an input graph G (Section 5).

We close this section by proving Lemma 3.6 and a corollary, which will be used in Section 5.

Proof of Lemma 3.6. Let v_a, v_b be the least common ancestors of $\{a_1, \dots, a_k\}$ and $\{b_1, \dots, b_k\}$, resp., and let $v = lca(v_a, v_b)$. Let T_a be the minimal subtree of T_v containing v_a and $\{a_1, \dots, a_k\}$. Thus, T_a has exactly k leaves $a_1, \dots, a_k$. Let T_d be an isomorphic copy of T_a. Let $d_1, \dots, d_k$ be the leaves of T_d such that d_i is the copy of a_i, for all $1 \leq i \leq k$.

Let T be the tree obtained from $T' \cup T_d$ by identifying the root of T_d with v, i.e. T is the tree $T' \cup S_1 \cup \ldots \cup S_l$ plus the edges $\{v, s_i\}$, for $1 \leq i \leq l$, where $S_1, \dots, S_l$ are the subtrees of T_d rooted at the children $s_1, \dots, s_l$ of the root of T_d. We set $S = S'$ and define a labelling function λ on the leaves of T by setting $\lambda(t) = \lambda'(t)$, if $t \notin \{d_1, \dots, d_k\}$ and $\lambda(d_i) = \lambda'(a_i)$.

Then (T, S, λ) is a tree-model of the same height as (T', S', λ') using the same set of labels.

It remains to verify that the graph G^T defined by T is isomorphic to G. By construction, $V(G^T) = V(G) \setminus \{c_1, \dots, c_k\} \cup \{d_1, \dots, d_k\}$. Let $\pi : V(G^T) \to V(G)$ be the function with $\pi(u) = u$ for all $u \in V(G) \setminus \{d_1, \dots, d_k\}$ and $\pi(d_i) = c_i$, for $1 \leq i \leq k$. We claim that π is an isomorphism between G^T and G.

As π is bijective by construction, it suffices to show that $\{u, w\} \in E(G^T)$ if, and only if, $\{\pi(u), \pi(w)\} \in E(G)$. If $u, w \in V(G) \setminus \{c_1, \dots, c_k\}$ there is nothing to show as the adjacency of all vertices within $G' = G - \{c_1, \dots, c_k\}$ remains unchanged by attaching $S_1, \dots, S_l$ to T'.

Now suppose $u = d_i$ and $w = d_j$, for some $1 \leq i \neq j \leq k$. Then $\lambda(d_i) = \lambda(a_i)$ and $\lambda(d_j) = \lambda(a_j)$ and the distance $2l$ between d_i and d_j in T is the same as the distance between a_i and a_j. Thus

$$\{d_i, d_j\} \in E(G^T) \Leftrightarrow (\lambda(d_i), \lambda(d_j), l) \in S \Leftrightarrow \{a_i, a_j\} \in E(G^T) \Leftrightarrow \{a_i, a_j\} \in E(G),$$

as argued above. As $\{\{a_1, \dots, a_k\}, \{c_1, \dots, c_k\}\}$ is a k-module in G, $\{c_i, c_j\} \in E(G)$ if, and only if, $\{a_i, a_j\} \in E(G)$. Thus, the restriction of π to $\{d_1, \dots, d_k\}$ is an isomorphism between the subgraphs $G[\{c_1, \dots, c_k\}]$ and $G^T[\{d_1, \dots, d_k\}]$.

The last case to consider is when $u = d_i$, for some $1 \leq i \leq k$, and $w \notin \{d_1, \dots, d_k\}$. Suppose first that $w \notin V(T_v)$, where $v = lca(v_a, v_b)$. Then the distance between w and a_i in T is the same as between w and d_i and therefore

$$\{d_i, w\} \in E(G^T) \Leftrightarrow \{a_i, w\} \in E(G^T) \Leftrightarrow \{a_i, w\} \in E(G) \Leftrightarrow \{c_i, w\} \in E(G).$$

Finally, suppose $w \in V(T_v)$. In this case either the path between w and u_i contains v and therefore the distance between w and u_i is the same as the distance between w and a_i, or the path between w and u_i does not contain v and the distance between w and u_i is the same as the distance between w and b_i. As the adjacency between w, a_i, w, b_i and w, c_i is the same in G, this implies that $\{w, d_i\} \in E(G)$ if, and only if, $\{w, c_i\} \in E(G)$. ◀

The next result follows easily by induction on $|M \setminus \{\bar{a}, \bar{b}\}|$ using Lemma 3.6.

▶ **Corollary 3.9.** *Let $d \geq 0$ and $k, m \geq 1$. Let G be a graph and let M be a k-module in G. Let $\bar{a}, \bar{b} \in M$ and let $G' = G - \bigcup\{\bar{c} : \bar{c} \in M \setminus \{\bar{a}, \bar{b}\}\}$. If there is a (d, m)-tree-model (T', S', λ') of G' such that $lca(\{a_1, \dots, a_k\})$ and $lca(\{b_1, \dots, b_k\})$ are incomparable in T' and $\lambda'(a_i) = \lambda'(b_i)$ for all $i \in [k]$, then G has a (d, m)-tree-model (T, S, λ) which extends (T', S', λ').*

4 Strict modules in graphs of low shrub-depth

In this section we prove several results about strict k-modules and their relation to tree-models. We start with Lemma 4.1 which states that if G contains a sufficiently large strict k-module M, then in every tree-model T of G there are two tuples of M in a mutual position which allows us to apply Lemma 3.6 and Corollary 3.9. We then establish Lemma 4.4, which is an analogue of Lemma 3.5 and which establishes the connection between tuples in strict k-modules and groups of isomorphic subtrees of *unsplittable* tree-models. Finally, we prove the technical Lemma 4.5 and Corollary 4.6 which establish that for each d and m there exist bounded values of k such that sufficiently large strict modules exist in large enough graphs which have (d, m)-tree-models.

▶ **Lemma 4.1.** *There exists a function $L : \mathbb{N}^3 \to \mathbb{N}$ such that for all $d, m, k > 0$ the following holds: if G is a graph and M is a strict k-module in G of size $|M| \geq L(m, d, k)$, then in any (d, m)-tree-model (T, S, λ) of G there are at least two tuples $(a_1, \dots, a_k)$ and $(b_1, \dots, b_k)$ in M such that*

1. *$lca_T(\{a_1, \dots, a_k\})$ and $lca_T(\{b_1, \dots, b_k\})$ are incomparable and*
2. *$\lambda(a_i) = \lambda(b_i)$ for all $i \in [k]$.*

Proof. Let T be a (d,m)-tree-model of G. Fix a linear order $<_\rho$ of the leaves of T such that for any three leaves u,v,w of T the following holds: if $u <_\rho v <_\rho w$ then v is a descendant of $lca_T(u,w)$ or $v = lca_T(u,v)$. It is easy to see that such order exists – for example if we run the DFS algorithm from the root T, then the order in which the leaves of T are visited by the algorithm has this property. For any tuple $\bar{a} = (a_1,\dots,a_k)$ let $T^{\bar{a}}$ denote the smallest subtree of T which contains $a_1,\dots,a_k$ and $lca_T(a_1,\dots,a_k)$. We say that two tuples $\bar{a},\bar{b}$ are (T,ρ)-similar if $T^{\bar{a}}$ and $T^{\bar{b}}$ are isomorphic, where we require that the isomorphism maps a_i to b_i and respects the colors of leaves and also the order $<_\rho$. It is easy to see that the (T,ρ)-similarity relation is an equivalence with finitely many classes; let $\gamma(m,d,k)$ denote the number of equivalence classes. We set $L(m,d,k) := (d+1)\gamma(m,d,k)$. Assume now that a strict k-module M of G has size at least $L(m,d,k)$. Then there are at least $d+1$ tuples in M which are (T,ρ)-similar; let us denote this set of tuples by S. We claim that there exists a pair of tuples $\bar{a} = (a_1,\dots,a_k)$ and $\bar{b} = (b_1,\dots,b_k)$ in S such that $lca_T(\{a_1,\dots,a_k\})$ and $lca_T(\{b_1,\dots,b_k\})$ are incomparable, in which case we are done. Assume for contradiction that there is no such pair. In this case there have to be two tuples $\bar{a} = (a_1,\dots,a_k)$ and $\bar{b} = (b_1,\dots,b_k)$ in S such that $lca_T(a_1,\dots,a_k) = lca_T(b_1,\dots,b_k)$, because the least common ancestors of all tuples in S are comparable and $|S| \geq d+1$. In the remainder of the proof we show that there exist i,j with $1 \leq i,j \leq k$ such that i,j is a conflict pair in M but the adjacency between a_i,a_j,b_i,b_j in G does not lead to a conflict, which is a contradiction. Set $v := lca_T(a_1,\dots,a_k) = lca_T(b_1,\dots,b_k)$. We take as the pair i,j any conflicting pair of M such that $lca_T(a_i,a_k) = v$. To see that such pair exists, partition $\{a_1,\dots,a_k\}$ into sets $A^1,A^2,\dots$ according to the following rule: two vertices of $\{a_1,\dots,a_k\}$ are in the same set if their least common ancestor is not v (note that in this case the least common ancestor is a descendant of v as $v = lca_T(a_1,\dots,a_k)$). It is easily seen that this is an equivalence. Since the conflict graph of M is connected, there have to be vertices $a_i \in A^1$ and $a_j \in A^2$ such that i,j is a conflict pair. Since they are in different sets, it has to hold that $lca_T(a_i,a_k) = v$. Without loss of generality we may assume that $i = 1$ and $j = 2$.

We now examine the adjacency of a_1,a_2,b_1,b_2 in G. To disprove that $(1,2)$ is a conflict pair in M, it is enough to show that (i) the adjacency between a_1 and a_2 is the same as the adjacency between a_1 and b_2 or (ii) the adjacency between b_1 and b_2 is the same as the adjacency between b_1 and a_2 Without loss of generality assume that $a_1 <_\rho a_2$ (which also means that $b_1 <_\rho b_2$ because $\bar{a}$ and $\bar{b}$ are isomorphic) and that $a_1 < b_1$ (otherwise we just swap $\bar{a}$ and $\bar{b}$). There are two cases to consider. First, if $a_2 <_\rho b_1$, then we have $a_1 <_\rho a_2 <_\rho b_1 <_\rho b_2$. In this case, since $lca_T(a_1,a_2) = v$ we also have to have $lca_T(a_1,b_2) = v$ (this follows from the definition of $<_\rho$), which means that the distance between a_1 and a_2 is the same as the distance between a_1 and b_2. Since $\lambda(a_2) = \lambda(b_2)$ (again, because $\bar{a}$ and $\bar{b}$ are isomorphic), we have $\{a_1,a_2\} \in E(G) \Leftrightarrow \{a_1,b_2\} \in E(G)$ and we have shown (i) above. The second case to consider is when $b_1 <_\rho a_2$. Then we have either $a_1 <_\rho b_1 <_\rho a_2 <_\rho b_2$ or $a_1 <_\rho b_1 <_\rho b_2 <_\rho a_2$. In the first situation we argue as in the previous case, and in the second situation we know that since $lca_T(b_1,b_2) = v$ it also has to hold $lca_T(b_1,a_2) = v$, and we get that $\{b_1,b_2\} \in E(G) \Leftrightarrow \{b_1,a_2\} \in E(G)$, which is the situation (ii) above. ◀

Let T be a tree and A a subset of leaves of T. We define T^A to be the smallest subtree of T which contains the root of T and all vertices from A.

▶ **Definition 4.2.** *A tree-model T is* splittable *if there exists a node u such that the leaves of the tree T_u can be partitioned into two sets A and B such that if we remove T_u from T and replace it by attaching T_u^A and T_u^B to the parent of u, then the resulting tree-model defines the same graph as T. If T is not splittable, it is* unsplittable.

▶ **Lemma 4.3.** *Every graph which has a (d, m)-tree-model has an unsplittable (d, m)-tree-model.*

Proof. Let T be a (d, m)-tree-model of G. If T is splittable, we keep splitting it (as in the Definition 4.2) as long as possible. Each splitting increases the number of internal nodes in T, and since there are at most $|V(G)| \cdot (d-1) + 1$ internal nodes in any tree-model of depth d of G, the process has to stop. ◀

The next lemma captures the connection between unsplittable tree-models and strict modules.

▶ **Lemma 4.4.** *Let (T, S, λ) be an unsplittable tree-model of a graph G and let u be a node of T with $L \geq 2$ children $\{1, \ldots, L\}$ such that for any two of its children i and j there exists a label preserving isomorphism ι_{ij} between T_i and T_j. Then G contains a strict k-module with L tuples, where k is the number of leaves in T_1.*

Proof. Let W_i denote the set of leaves of T_i. Fix any ordering on W_1 and for every $j > 1$ use ι_{ij} to define the corresponding ordering on W_j, i.e. define $\leq_j$ on W_j by setting $u \leq_j v$ if, and only if, $\iota_{j1}(u) \leq_1 \iota_{j1}(v)$. Each pair $(W_i, \leq_i)$ can be thought of as an ordered k-tuple, and we claim that $M := \{(W_1, \leq_1), \ldots, (W_L, \leq_L)\}$ is a strict k-module with L tuples claimed in the statement of the lemma.

The fact that M is a k-module in G with L tuples is clear from its definition, and so it remains to argue that M is in fact strict. For the sake of contradiction assume that M is not strict, which means that its conflict graph $\mathcal{C}(M)$ on the vertex set $[k]$ is not connected. Let C be a connected component of $\mathcal{C}(M)$ on $p < k$ vertices, and without loss of generality assume that $V(C) = [p]$. We will prove that T is splittable, which is the contradiction with our assumption on T. To simplify the notation in the rest of the proof, we will from now on denote the tuples of M by the usual $\bar{a}, \bar{b}, \ldots$ instead of $(W_1, \leq_1), (W_2, \leq_2), \ldots$. Let $\bar{a}$ be a tuple of M. Let $A := \{a_1, \ldots, a_p\}$ be the set of the first p vertices in $\bar{a}$ and let $A' = \{a_1, \ldots, a_k\} \setminus A$. We remove T_1 from T and replace it by attaching to u the trees T_u^A and $T_u^{A'}$ to obtain a new tree-model (T', S, λ) (the relation S and labeling function λ are the same as in T). We claim that (T', S, λ) is a tree-model of G. Since the transformation of T into T' does not change any labels, the only change in the adjacency defined by T' compared to T can come from a change of distance between two leaves. The only situation when $dist_T(v, w) \neq dist_{T'}(v, w)$ is when $v \in A$ and $w \in A'$ or vice versa – in this case it holds that $dist_{T'}(v, w) = 2l$, where l is the height of u in T (and also in T'). We now argue that in this case the adjacency between u and u remains unchanged, i.e. $\{v, w\} \in E(G^T) \Leftrightarrow \{v, w\} \in E(G^{T'})$. Since $v \in A$ and $w \in A'$ we have that $v = a_i$ for some $i \leq p$ and $w = a_j$ for some $j > p$. Assume that $\{a_i, a_j\} \in E(G^T)$. We need to show that $\{a_i, a_j\} \in E(G^{T'})$, which is the case exactly when $(\lambda(a_i), \lambda(a_j), 2l) \in S$. Let α and β be the relations of M. Since a_i and a_j are in the same tuple of M, it holds that $\{i, j\} \in \alpha$. Since i and j are in different connected components of the conflict graph $\mathcal{C}(M)$, it also holds that $\{i, j\} \in \beta$. Let $\bar{b} := (b_1, \ldots, b_k)$ be a tuple of M different from $\bar{a}$. Since $\{i, j\} \in \beta$, there is an edge between a_i and b_j in G. Because the distance between a_i and b_j in T is $2l$, this means that $(\lambda(a_i), \lambda(b_j), 2l) \in S$. Because M is a module, $\lambda(a_j) = \lambda(b_j)$, and so $(\lambda(a_i), \lambda(a_j), 2l) \in S$, which means that $\{a_i, a_j\} \in E(G^{T'})$ as desired. The other direction is proved analogously. ◀

▶ **Lemma 4.5.** *For every $d, m \geq 1$ and every sequence $0 < L_1 \leq L_2 \leq \ldots$ there exist K and N such that every graph which has a (d, m)-tree-model and has more than N vertices contains, for some $k \leq K$, a strict k-module with more than L_k tuples.*

Proof. For a tree-model T we say that two nodes u, v of T are T-isomorphic if they have the same parent and there is a label preserving isomorphism between T_u and T_v.

We will prove by induction on d the following statement, which implies the lemma by means of Lemma 4.4. For every d and m there exist numbers $K(d,m)$ and $N(d,m)$ such that the following holds: In every (d,m)-tree-model with at least $N(d,m)$ leaves there is a node which, for some $k \leq K(d,m)$, has more than L_k pairwise T-isomorphic children each of which has k leaves.

For $d = 1$ we set $k(1,m) := 1$ and $N(1,m) := mL_1$. Let G be a graph on more than $N(1,m)$ vertices and let T be its tree-model of height 1. Then T has more than $N(1,m)$ leaves and for at least one label there are more than L_1 leaves having this label, which means that they form a set of more than L_1 pairwise T-isomorphic children of the root.

Assume now that $d > 1$ and the statement holds for $d-1$. Set $K(d,m) := N(d-1,m)$ and $N(d,m) := N(d-1,m) \cdot L_{K(d,m)} \cdot \gamma(d-1,m)$, where $\gamma(d-1,m)$ is the number of non-isomorphic $(d-1,m)$-tree-models with at most $N(d-1,m)$ leaves, and where it is understood that the isomorphisms are label preserving. Let T be a (d,m)-tree-model with more than $N(d,m)$ leaves. We distinguish two cases:

1. There is a child u of the root of T such that T_u has more than $N(d-1,m)$ leaves. Then by the inductive assumption there is a node in T_u which, for some $k \leq K(d-1,m)$, has more than L_k pairwise T-isomorphic children each of which has k leaves. Since $K(d-1,m) < K(d,m)$ we are done.
2. For every child u of the root r of T it holds that T_u has at most $N(d-1,m)$ leaves. In this case r has more than $\frac{N(d,m)}{N(d-1,m)} = L_{K(d,m)} \cdot \gamma(d-1,m)$ children, each of which corresponds to a subtree T_u of T with at most $N(d-1,m)$ leaves. We group the subtrees of T determined by the children of r into groups $C_1, \ldots, C_{\gamma(d-1,m)}$ according to their labeled isomorphism type. Because there are more than $L_{K(d,m)} \cdot \gamma(d-1,m)$ of these trees, at least one group C_i has more than $L_{K(d,m)}$ trees in it. All these trees are pairwise isomorphic and have at most $N(d-1,m) = K(d,m)$ leaves; let us denote this number of leaves by k. Since $k \leq K(d,m)$, we have $L_k \leq L_{K(d,m)}$ and therefore C_i has more than L_k trees, as desired. ◀

▶ **Corollary 4.6.** *For every $d, m \geq 1$ there exist K and N such that every graph which has a (d,m)-tree-model and has more than N vertices contains, for some $k \leq K$, a strict k-module with more than $L(m,d,k)$ tuples, where $L(m,d,k)$ is the function from Lemma 4.1.*

Proof. For every k set L_k to be the $L(m,d,k)$ from Lemma 4.1 (where it is easily seen that $L(m,d,k-1) \leq L(m,d,k)$ for each k) and apply Lemma 4.5. ◀

5 Algorithms

In this section we use the results from the previous section to obtain two algorithms for computing tree-models of graphs.

The results obtained in the previous section suggest the following strategy to compute, given a graph G and $d, m > 0$ as input, a (d,m)-tree-model of G, provided such a tree-model of G exists.

The main algorithmic strategy. Let G be a graph and $d, m > 0$ be integers.

Step 1. Given d, m, let K and N be the numbers stated in Corollary 4.6.

Step 2. As long as $|G| > N$, repeat the following steps.

- **a.** find, for some $k < K$, a strict k-module M in G of size $|M| > L(m,d,k)$
- **b.** choose a set $M^* \subseteq M$ of order exactly $L(m,d,k)$ and delete all elements of all tuples in $M \setminus M^*$. Remember the sets M and M^* for each such step.

Step 3. Let G' be the remaining graph of order $|G'| \leq N$. Compute a (d, m)-tree-model (T', S', λ') of G' by brute force.

Step 4. In reverse order of their creation, for each module M and set $M^* \subseteq M$ constructed in the iterations of Step 2

a. find tuples $\bar{a}, \bar{b}$ in T' satisfying the requirements of Corollary 3.9.

b. Extend (T', S', λ') by adding the vertices in $M \setminus M^*$ as described in Corollary 3.9.

The correctness of this approach follows from the results in Section 4. By Corollary 4.6, given d and m, the numbers K and N used in Step 1 depending only on d and m but not on G exist such that if $|G| > N$, then G contains a strict k-module M of size $> L = L(m, d, k)$, for some $k < K$. Here and below $L(m, d, k)$ is the function defined in Lemma 4.1.

In Step 2 we iteratively reduce the size of G until its size is bounded by a function of the parameters d and m. This creates a sequence $G = G_0 \supset_i G_1 \supset_i \ldots \supset_r = G'$ of graphs, where r is the number of iterations in Step 2. Notice that in each iteration, when we remove $M \setminus M^*$ from G_i to obtain G_{i+1}, we keep in G_{i+1} enough tuples of M to be able to construct a (d, m)-tree-model T^i of G_i from a (d, m)-tree-model T^{i+1} of G_{i+1}. To see this, notice that M^* (which was not removed and is a strict k-module of G_{i+1}) has $L(m, d, k)$ tuples, and so by of Lemma 4.1 there are tuples $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ in M^* such that the vertices $\{a_1, \ldots, a_k, b_1, \ldots, b_k\}$ are placed in T^{i+1} in accordance with the assumptions of Corollary 3.9, the application of which allows us to put all tuples from $M \setminus M^*$ into T^{i+1} to obtain T^i.

After completing Step 2 we are left with an induced subgraph G' of G of size $|G'| \leq N$. In Step 3 we compute a (d, m)-tree-model (T', S', λ') of G'. As N only depends on the parameters d and m we can compute T' by brute-force. If no such (d, m)-tree-model of G' exists, then G does not have a (d, m)-tree-model as G' is an induced subgraph of G and the existance of tree-models is preserved by taking induced subgraphs.

Otherwise, if we find a (d, m)-tree-model (T', S', λ') of G' we extend it to a (d, m)-tree-model of the input graph G in Step 4. For this, we iterate again over all modules M and subsets M^* constructed in the iterations of Step 2 and apply Corollary 3.9 to extend T' so that it contains the vertices in $M \setminus M^*$.

This proves the general correctness of the algorithmic approach described above. In the remainder of this section we show how the various steps in the algorithm above can be implemented to eventually yield an fpt-algorithm for comptuting tree-models.

As a first step towards this goal we present a simple XP-algorithm implementing the approach described above. The methods we use for this algorithm for the Steps 3 and 4 but not for Step 2 are already good enough for an fpt-algorithm. What remains to be done is to improve Step 2.

As a second step, we present an improved XP-algorithm with a better algorithm for Step 2. Finally we show how this new strategy for Step 2 can be implemented in a way to yield an fpt-algorithm as required.

The XP algorithms. The first algorithmic step we prove is the following lemma, which is an easy consequence of Lemma 3.6 and 4.1. The lemma essentially states that once we have found a tree-model T' for the reduced graph G' obtained from G by removing some tuples of a k-module, the tree-model of G can be computed efficiently from T.

▶ **Lemma 5.1.** *Let G be a graph which has a (d, m)-tree-model and let M be a strict k-module containing more than $L = L(m, d, k)$ tuples. Let $Q \subseteq M$ be such that $|M \setminus Q| = L$ and let G' be the graph obtained from G by deleting all vertices contained in tuples in Q. Then any (d, m)-tree-model T' of G' can be extended in linear time to a (d, m)-tree-model T of G.*

Proof. Let (T', S', λ') be a (d, m)-tree-model of G'. Since $M \setminus Q$ is a strict k-module in G' containing $L = L(m, d, k)$ tuples, Lemma 4.1 guarantees that there are tuples $\bar{a} := (a_1, \ldots, a_k)$, $\bar{b} := (b_1, \ldots, b_k) \in M \setminus Q$ in G such that $lca_{T'}(\{a_1, \ldots, a_k\})$ and $lca_{T'}(\{b_1, \ldots, b_k\})$ are incomparable in T' and $\lambda'(a_i) = \lambda'(b_i)$ for all $i \in [k]$. By Corollary 3.9, (T', S', λ') can be extended to a (d, m)-tree-model (T, S, λ) of G. It is straight forward to verify that the proof of 3.9 can be made algorithmic and can be implemented in linear time. Note that when we delete the tuples in Q, we will store with Q also the tuples $\bar{a}, \bar{b}$, as we need them to add the elements of Q to the tree-model T'. ◀

The previous lemma shows how Step 4 above can be implemented. Step 3 can be done by brute-force, so all that remains is to provide an algorithm for Step 2.

Towards this aim, note that since k and $L_k = L(m, d, k)$ depend only on d and m and not on $|V(G)|$, we can simply go over all subsets $X \subseteq V(G)$ of size $|X| = k(L_k + 1)$ in time $|V(G)|^{k(L_k+1)}$ and for each such X check whether it can be partitioned into a strict k-module. Using this as a sub-routine for Step 2 above to obtain our first XP-algorithm for computing tree-models.

However, this way of finding strict k-modules is highly inefficient, and in the remainder of this section we argue that the runtime can be improved to $|V(G)|^{3k+1}$ by using the following simple greedy procedure. For every set $X \subseteq V(G)$ of $2k$ vertices of G we 1) generate all partitions of X into two disjoint sets X_a, X_b of k vertices each and 2) for each of these we consider all possible ways to order the vertices in X_a and X_b so that we obtain ordered k-tuples $\bar{a}$ and $\bar{b}$ and then we 3) check whether $\bar{a}$ and $\bar{b}$ are k-twin tuples. For any pair $\bar{a}, \bar{b}$ of twin tuples obtained in this way we let $M := \{\bar{a}, \bar{b}\}$ and iterate over all k-tuples $\bar{c}$ of vertices from $V(G) \setminus V(M)$. For any such $\bar{c}$ we check whether $\bar{c}$ is a twin tuple of every tuple already contained in M. If so, we add $\bar{c}$ to M and repeat, extending M as long as possible.

Observe that this procedure is approximate in the following sense: it finds a strict k-module with more than L_k tuples provided that a strict k-module with more than kL_k tuples exists in G. As this procedure requires G to contain a strict k-module of size kL_k instead of L_k, whenever we apply Lemma 4.5 in the general algorithm above, we use m, d but with a different sequence $L_1 \leq L_2 \leq \ldots$ defined as $L_k = kL(m, d, k)$. This guarantees that Lemma 4.5 applied to m, d and this new sequence $L_1 \leq L_2 \leq \ldots$ yields suitable K and N that make the algorithm above work.

We show next that this revised procedure for Step 2a is correct in the sense that it indeed produces a strict k-module of size $> L(m, d, k)$ as required. Towards this aim, let Z be a strict k-module of G with the maximum number of tuples (in particular note that Z has more than $kL(m, d, k)$ tuples).

1. At least one initial guess of $\bar{a}$ and $\bar{b}$ yields a pair of twin tuples from Z, because we go over all sets of size $2k$, all possible ways to split them into k-tuples $\bar{a}$ and $\bar{b}$. The pair $\bar{a}, \bar{b}$ also determines relations α and β, which guarantees that these are the same for M and Z.
2. Even if every tuple $\bar{c}$ we find is suboptimal (i.e. $\bar{c}$ is not one of the tuples in Z but intersects several of these), it can intersect at most k tuples in Z. The remaining tuples in Z which do not contain any vertex contained in a tuple in M are twin tuples of every tuple in M and therefore also of $\bar{c}$, as the relation of being an (α, β)-twin tuple with respect to fixed α and β is transitive.

Item 2 implies that after the i-th iteration of adding a tuple $\bar{c}$ to M there are more than $kL_k - 2 - ki$ tuples in Z left which can still be added to M. Thus, at least $L_k - 1$ iterations will be performed and therefore M will have at least $L_k + 1$ tuples.

The FPT-algorithm. We are now ready to present the fpt-algorithm for computing (d, m)-tree-models. The reason why the previous algorithm is only an XP-algorithm is that it iterates over all possible sets of vertices of size $2k$ to find a pair of twin tuples $\bar{a}, \bar{b}$ and then later on again iterates over all sets of size k to find a suitable tuple $\bar{c}$.

In this section we prove that in order to find a pair of twin tuples $\bar{a}, \bar{b}$ in G it is enough to guess a pair u, v of vertices and then check in linear time whether they can be extened to an (α, β)-twin tuple. Similarly, given a strict k-module M of G, to find a twin k-tuple $\bar{c}$ of all tuples in M it is enough to guess one vertex u of $\bar{c}$ and the check in linear time whether it can be extended appropriately.

▶ **Lemma 5.2.** *Let $k \in \mathbb{N}$ and let α and β be relations on $[k]$ such that they determine a connected conflict graph. Let G be a graph and u, v be vertices of G. Then the number of twin tuples $(a_1, \ldots, a_k)$, $(b_1, \ldots, b_k)$ such that $u = a_1$ and $v = b_1$ is bounded by a function in k and there is an algorithm running in time $f(k) \cdot |V(G)|$ which, given G, u and v as input, computes all such pairs of twin tuples.*

Proof. We will prove that for every pair $\big((A, f_A), (B, f_B)\big)$ where A and B are disjoint subsets of $V(G)$ with $|A| = |B| = p \leq k$ and $f_A : A \to [k]$ and $f_B : B \to [k]$ are injective functions with the same image in $[k]$ we can generate in time $f(k) \cdot |V(G)|$ all pairs $\big((\bar{A}, f_{\bar{A}}), (\bar{B}, f_{\bar{B}})\big)$ such that:

- $\bar{A}$ and $\bar{B}$ are disjoint, $|\bar{A}| = |\bar{B}| = k$ and $A \subseteq \bar{A}$, $B \subseteq \bar{B}$
- $f_{\bar{A}} : \bar{A} \to [k]$ and $f_{\bar{B}} : \bar{B} \to [k]$ are injective
- $\bar{A}$ and $\bar{B}$ with the orderings induced by $f_{\bar{A}}$ and $f_{\bar{B}}$ in the obvious way are (α, β)-twin-tuples.

Moreover, the number of such pairs $(\bar{A}, f_{\bar{A}})$, $(\bar{B}, f_{\bar{B}})$ will be bounded in terms of k.

We prove this by induction on $j = k - p$. If $j = 0$, then there is nothing to generate and we only need to check whether (A, f_A) and (B, f_B) have the adjacency prescribed by α and β. Now assume that the statement holds for all integers less than j and let (A, f_A), (B, f_B) be an instance with $|A| = |B| = p$ where $j = k - p$. First, we check whether $G[A]$ and $G[B]$ and all edges between A and B in $G[A \cup B]$ are consistent with α and β (with respect to the ordering induced by f_A and f_B). If this is not the case, we can immediately say that (A, f_A), (B, f_B) cannot be extended.

So we may assume that $G[A], G[B]$, and $G[A \cup B]$ are consistent with α and β. To simplify the notation, in the rest of this proof we will denote by a_i the element of A such that $f_A(a_i) = i$ and by b_i the element of B with $f_B(b_i) = i$. For all i in the image $im(f_A)$ of f_A (and thus also in the image $im(f_B)$ of f_B), let $S_i := (N(a_i) \Delta N(b_i)) \setminus (A \cup B)$. That is, S_i is the set of all vertices outside of $A \cup B$ on which a_i and b_i differ. Let $S := \bigcup_{i \in im(f_A)} S_i$. Since the conflict graph determined by α and β is connected, for any $j > 0$ and any pair (A, f_A), (B, f_B) which can be extended to an (α, β)-twin pair $(\bar{A}, f_{\bar{A}})$, $(\bar{B}, f_{\bar{B}})$ the set S will be non-empty. Moreover, all vertices in S have to be included in all extensions $(\bar{A}, f_{\bar{A}})$, $(\bar{B}, f_{\bar{B}})$ satisfying the properties above. For, otherwise there would be an i such that a_i and b_i are not twins in $V(G) \setminus \bar{A} \cup \bar{B}$, which contradicts the definition of twin-tuples. If S is larger than $2k - 2j$ we know that A and B cannot be extended as desired, because then we would have $|\bar{A}| + |\bar{B}| > 2k$, again a contradiction. If S has size at most $2k - 2j$, then we consider all partitions of S into sets S_A and S_B of equal size, set $A' := A \cup S_A$ and $B' := B \cup S_B$ and consider all injective functions $f_{A'} : A' \to [k]$, $f_{B'} : B' \to [k]$ which have the same image and which agree with f_A and f_B on A and B, respectively. Since $|A'| > |A|$ and $|B'| > |B|$, we have that $k - |A| < j$ and we can apply the induction hypothesis to $(A', f_{A'})$, $(B', f_{B'})$.

Clearly in the case when we use the induction hypothesis the size of the set S is bounded by k, and so is the number of its bipartitions into S_A and S_B and also the number of different functions $f_{A'}$ and $f_{B'}$. This completes the proof. ◀

▶ **Lemma 5.3.** *Let $k \in \mathbb{N}$ and let α and β be relations on $[k]$ such that they determine a connected conflict graph. Let G be a graph, M be a strict (α, β)-module in G and let $v \in V(G) \setminus V(M)$. Then in time $g(k) \cdot |V(G)|$ one can find a tuple $\bar{c} := (c_1, \ldots, c_k)$ in $V(G) \setminus V(M)$ with $c_1 = v$ such that $\bar{c}$ is an (α, β)-twin of all tuples in M, or determine that no such tuple exists.*

Proof. Let $\bar{a}$ be a tuple of M and let G' be obtained from G by deleting all tuples of M with the exception of $\bar{a}$. Note that if $\bar{c}$ exists in G, then it is an (α, β)-twin tuple of $\bar{a}$ in G'. We apply Lemma 5.2 to a_1 and v to obtain the set of all (α, β)-twin tuples which have a_1 and v on their first positions, resp. If any of these pairs of tuples contains $\bar{a}$, then the the second tuple of this pair can be taken as $\bar{c}$. The fact that $\bar{c}$ is also an (α, β)-twin tuple of all tuples in M in G follows from trasitivity of being (α, β)-twin tuple. ◀

We are now ready to prove the main algorithmic result of this section.

▶ **Theorem 5.4.** *Let G be a graph which contains a strict k-module Z with more than kL tuples. Then it is possible to find a strict k-module M with more than L tuples in time $h(k) \cdot |V(G)|^4$.*

Proof. The algorithm iterates over all pairs of symmetric relations α and β on $[k]^2$ which determine a connected conflict graph and for each such pair proceeds as follows. For every pair u, v of vertices in G it uses the algorithm from Lemma 5.2 to generate the set S of all pairs of (α, β)-twin tuples which have u and v on their first positions. Then, for each pair $\bar{a}, \bar{b} \in S$ we set $M := \{\bar{a}, \bar{b}\}$ and by repeated application of Lemma 5.3 we extend M by finding a tuple $\bar{c}$ which is an (α, β)-twin of every tuple in M and adding $\bar{c}$ to M for as long as possible.

We now argue that at least for one choice of α, β, u and v the algorithm produces a strict k-module with more than L elements. Set α and β to be the relations of Z and let u, v be vertices which are on the first position of tuples $\bar{a}, \bar{b}$ of Z. By applying the algorithm from Lemma 5.2 to α, β, u, v we find all pairs of (α, β)-twin tuples which have u and v on their first position. In particular we will find $\bar{a}$ and $\bar{b}$. We then set $M := \{\bar{a}, \bar{b}\}$ and try to extend M as much as possible using Lemma 5.3. We can argue in exactly the same way as in the previous section that the number of successful iterations of extending M by a tuple $\bar{c}$ will be at least $L - 1$. Thus, together with $\bar{a}$ and $\bar{b}$, the k-module M we find will have at least $L + 1$ tuples. ◀

Using the algorithm of Theorem 5.4 in Step 2 of the general algorithmic strategy outlined at the beginning of this section, we obtain our main algorithmic result.

▶ **Theorem 5.5.** *There is an algorithm which, given a graph G and numbers $m, d > 0$ as input, in time $f(d, m) \cdot |G|^c$, for a computable function f and a constant c both independent of G, d, and m, either computes a (d, m)-tree-model of G or correctly determines that no such module exists.*

6 Application for forbidden induced subgraphs

As an easy consequence of our results from Section 4 we obtain a simple proof of the following theorem, which was originally proven in [8].

▶ **Theorem 6.1.** *For every d, m there exists a finite set of graphs $\mathcal{F}_{d,m}$ such that a graph G has a (d, m)-tree-model if, and only if, G does not have an induced subgraph isomorphic to a member of $\mathcal{F}_{d,m}$.*

Compared to the proof of Theorem 6.1 given in [8], our proof given below has the advantage of providing explicit bounds on the size of graphs in $\mathcal{F}_{d,m}$ and therefore being constructive.

Proof. Fix d and m. Let $\mathcal{F}_{d,m}$ be the set of all graphs H such that H does not have a (d, m)-tree-model and every proper induced subgraph of H has a (d, m)-tree-model. It is easy to see that for every graph G it holds that G has a (d, m)-tree-model if, and only if, G does not have an induced subgraph isomorphic to a member of $\mathcal{F}_{d,m}$. We will show that there is a bound on the size of graphs in $\mathcal{F}_{d,m}$.

Let K and N be the numbers obtained from Corollary 4.6 applied to d and m. We claim that no graph in $\mathcal{F}_{d,m}$ has more than N vertices. Assume towards a contradiction that there is a graph H from $\mathcal{F}_{d,m}$ which has more than N vertices. By Corollary 4.6 there is a strict k-module M in H with more than $L_(d, m, k)$ tuples for some $k \leq K$. Let H' be the graph obtained from H by removing one tuple $\bar{c}$ from M. Then H' is a proper induced subgraph of H and has strict k-module M' with at least $L_(d, m, k)$ tuples. Since H' is a proper induced subgraph of H, it has (d, m)-tree-model T'. By Lemma 4.1 there are two tuples $\bar{a}$ and $\bar{b}$ in M' such that they are in different subtrees of T'. By Lemma 3.6 we can extend T' (by adding $\bar{c}$ into it) to a (d, m)-tree-model T of H, which is a contradiction with the assumption that H has no (d, m)-tree-model. Thus, no member of $\mathcal{F}_{d,m}$ has more than N vertices. ◀

References

1 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

2 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

3 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

4 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

5 Jakub Gajarský and Petr Hlinený. Kernelizing MSO properties of trees of fixed height, and some consequences. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:19)2015.

6 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetril, Patrice Ossona de Mendez, Michal Pilipczuk, Sebastian Siebertz, and Szymon Toru'nczyk. First-order interpretations of bounded expansion classes. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 126:1–126:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.126.

7 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

8 Robert Ganian, Petr Hlinený, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Logical Methods in Computer Science*, 15(1), 2019. URL: https://lmcs.episciences.org/5149, doi:10.23638/LMCS-15(1:7)2019.

9 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

10 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

11 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

12 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.