

Space-Efficient Parameterized Algorithms on Graphs of Low Shrubdepth

Benjamin Bergounoux ✉

Institute of Informatics, University of Warsaw, Poland

Vera Chekan ✉ 

Humboldt-Universität zu Berlin, Germany

Robert Ganian ✉ 


Algorithms and Complexity Group, TU Wien, Vienna, Austria

Mamadou Moustapha Kanté ✉ 

Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, CNRS, Clermont-Ferrand, France

Matthias Mnich ✉ 

Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany

Sang-il Oum ✉ 

Discrete Mathematics Group, Institute for Basic Science (IBS), Daejeon, Korea
Department of Mathematical Sciences, KAIST, Daejeon, Korea

Michał Pilipczuk ✉

Institute of Informatics, University of Warsaw, Poland

Erik Jan van Leeuwen ✉ 

Dept. Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

Dynamic programming on various graph decompositions is one of the most fundamental techniques used in parameterized complexity. Unfortunately, even if we consider concepts as simple as path or tree decompositions, such dynamic programming uses space that is exponential in the decomposition's width, and there are good reasons to believe that this is necessary. However, it has been shown that in graphs of low treedepth it is possible to design algorithms which achieve polynomial space complexity without requiring worse time complexity than their counterparts working on tree decompositions of bounded width. Here, *treedepth* is a graph parameter that, intuitively speaking, takes into account both the depth and the width of a tree decomposition of the graph, rather than the width alone.

Motivated by the above, we consider graphs that admit clique expressions with bounded depth and label count, or equivalently, graphs of low shrubdepth. Here, shrubdepth is a bounded-depth analogue of cliquewidth, in the same way as treedepth is a bounded-depth analogue of treewidth. We show that also in this setting, bounding the depth of the decomposition is a deciding factor for improving the space complexity. More precisely, we prove that on n -vertex graphs equipped with a tree-model (a decomposition notion underlying shrubdepth) of depth d and using k labels,

- INDEPENDENT SET can be solved in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ using $\mathcal{O}(dk^2 \log n)$ space;
- MAX CUT can be solved in time $n^{\mathcal{O}(dk)}$ using $\mathcal{O}(dk \log n)$ space; and
- DOMINATING SET can be solved in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ using $n^{\mathcal{O}(1)}$ space via a randomized algorithm.

We also establish a lower bound, conditional on a certain assumption about the complexity of LONGEST COMMON SUBSEQUENCE, which shows that at least in the case of INDEPENDENT SET the exponent of the parametric factor in the time complexity has to grow with d if one wishes to keep the space complexity polynomial.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, shrubdepth, space complexity, algebraic methods

Funding *Vera Chekan*: Supported by the DFG Research Training Group 2434 “Facets of Complexity.”
Robert Ganian: Project No. Y1329 of the Austrian Science Fund (FWF), WWTF Project ICT22-029.
Mamadou Moustapha Kanté: Supported by the French National Research Agency (ANR-18-CE40-0025-01 and ANR-20-CE48-0002).

Sang-il Oum: Supported by the Institute for Basic Science (IBS-R029-C1).

Michał Pilipczuk: This work is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 948057).

Acknowledgements This work was initiated at the *Graph Decompositions: Small Width, Big Challenges* workshop held at the Lorentz Center in Leiden, The Netherlands, in 2022.



1 Introduction

Treewidth and Treedepth. Dynamic programming on graph decompositions is a fundamental method in the design of parameterized algorithms. Among various decomposition notions, *tree decompositions*, which underly the parameter *treewidth*, are perhaps the most widely used; see e.g. [14, 18] for an introduction. A tree decomposition of a graph G of width k provides a way to “sweep” G while keeping track of at most $k + 1$ “interface vertices” at a time. This can be used for dynamic programming: during the sweep, the algorithm maintains a set of representative partial solutions within the part already swept, one for each possible behavior of a partial solution on the interface vertices. Thus, the width of the decomposition is the key factor influencing the number of partial solutions that need to be stored.

In a vast majority of applications, this number of different partial solutions depends (at least) exponentially on the width k of the decomposition, which often leads to time complexity of the form $f(k) \cdot n^{\mathcal{O}(1)}$ for an exponential function f . This should not be surprising, as most problems where this technique is used are NP-hard. Unfortunately, the space complexity—which often appears to be the true bottleneck in practice—is also exponential. There is a simple tradeoff trick, first observed by Lokshtanov et al. [38], which can often be used to reduce the space complexity to polynomial at the cost of increasing the time complexity. For instance, INDEPENDENT SET can be solved in $2^k \cdot n^{\mathcal{O}(1)}$ time and using $2^k \cdot n^{\mathcal{O}(1)}$ space on an n -vertex graph equipped with a width- k tree decomposition via dynamic programming [26]; combining this algorithm with a simple recursive Divide&Conquer scheme yields an algorithm with running time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ and space complexity $n^{\mathcal{O}(1)}$.

Allender et al. [2] and then Pilipczuk and Wrochna [45] studied the question whether the loss on the time complexity is necessary if one wants to achieve polynomial space complexity in the context of dynamic programming on tree decompositions. While the formal formulation of their results is somewhat technical and complicated, the take-away message is the following: there are good complexity-theoretical reasons to believe that even in the simpler setting of path decompositions, one cannot achieve algorithms with polynomial space complexity whose running times asymptotically match the running times of their exponential-space counterparts. We refer to the works [2, 45] for further details.

However, starting with the work of Fürer and Yu [27], a long line of advances [33, 40, 41, 45] showed that bounding the *depth*, rather than the width, of a decomposition leads to the possibility of designing algorithms that are both time- and space-efficient. To this end, we consider the *treedepth* of a graph G , which is the least possible depth of an *elimination forest*: a forest F on the vertex set of G such that every two vertices adjacent in G are in the ancestor/descendant relation in F . An elimination forest of depth d can be regarded as a tree

decomposition of depth d , and thus treedepth is the bounded-depth analogue of treewidth. As shown in [27, 33, 41, 45], for many classic problems, including 3-COLORING, INDEPENDENT SET, DOMINATING SET, and HAMILTONICITY, it is possible to design algorithms with running time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$ and polynomial space complexity, assuming the graph is supplied with an elimination forest of depth d . In certain cases, the space complexity can even be as low as $\mathcal{O}(d + \log n)$ or $\mathcal{O}(d \log n)$ [45]. Typically, the main idea is to reformulate the classic bottom-up dynamic programming approach so that it can be replaced by a simple top-down recursion. This reformulation is by no means easy—it often involves a highly non-trivial use of algebraic transforms or other tools of algebraic flavor, such as inclusion-exclusion branching.

Cliqewidth and Shrubdepth. In this work, we are interested in the parameter *cliqewidth* and its low-depth counterpart: *shrubdepth*. While treewidth applies only to sparse graphs, cliqewidth is a notion of tree-likeness suited for dense graphs as well. The decompositions underlying cliqewidth are called *clique expressions* [13]. A clique expression is a term operating over *k-labelled graphs*—graphs where every vertex is assigned one of k labels—and the allowed operations are: (i) apply any renaming function to the labels; (ii) make a complete bipartite graph between two given labels; and (iii) take the disjoint union of two k -labelled graphs. Then the cliqewidth of G is the least number of labels using which (some labelling of) G can be constructed. Similarly to treewidth, dynamic programming over clique expressions can be used to solve a wide range of problems, in particular all problems expressible in MSO_1 logic, in FPT time when parameterized by cliqewidth. Furthermore, while several problems involving edge selection or edge counting, such as HAMILTONICITY or MAX CUT, remain $W[1]$ -hard under the cliqewidth parameterization [23, 24], standard dynamic programming still allows us to solve them in XP time. In this sense, cliqewidth can be seen as the “least restrictive” general-purpose graph parameter which allows for efficient dynamic programming algorithms where the decompositions can also be computed efficiently [25]. Nevertheless, since the cliqewidth of a graph is at least as large as its linear cliqewidth, which in turn is as large as its pathwidth, the lower bounds of Allender et al. [2] and of Pilipczuk and Wrochna [45] carry over to the cliqewidth setting. Hence, reducing the space complexity to polynomial requires a sacrifice in the time complexity.

Shrubdepth, introduced by Ganian et al. [30], is a variant of cliqewidth where we stipulate the decomposition to have bounded depth. This necessitates altering the set of operations used in clique expressions in order to allow taking disjoint unions of multiple graphs as a single operation. In this context, we call the decompositions used for shrubdepth (d, k) -tree-models, where d stands for the depth and k for the number of labels used; a formal definition is provided in Section 2. Shrubdepth appears to be a notion of depth that is sound from the model-theoretic perspective, is FPT-time computable [28], and has become an important concept in the logic-based theory of well-structured dense graphs [19, 20, 29, 30, 42, 43].

Since shrubdepth is a bounded-depth analogue of cliqewidth in the same way as treedepth is a bounded-depth analogue of treewidth, it is natural to ask whether for graphs from classes of bounded shrubdepth, or more concretely, for graphs admitting (d, k) -tree-models where both d and k are considered parameters, one can design space-efficient FPT algorithms. Exploring this question is the topic of this work.

Our contribution. We consider three example problems: INDEPENDENT SET, MAX CUT, and DOMINATING SET. For each of them we show that on graphs supplied with (d, k) -tree-models where $d = \mathcal{O}(1)$, one can design space-efficient fixed-parameter algorithms whose running times asymptotically match the running times of their exponential-space counterparts working on general clique expressions. While we focus on the three problems mentioned above for concreteness, we in fact provide a more general algebraic framework, inspired by the work

on the treedepth parameterization [27, 33, 40, 41, 45], that can be applied to a wider range of problems. Once the depth d is not considered a constant, the running times of our algorithms increase with d . To mitigate this concern, we give a conditional lower bound showing that this is likely to be necessary if one wishes to keep the space complexity polynomial.

Recall that standard dynamic programming solves the INDEPENDENT SET problem in time $2^k \cdot n^{\mathcal{O}(1)}$ and space $2^k \cdot n^{\mathcal{O}(1)}$ on a graph constructed by a clique expression of width k [26]. Our first contribution is to show that on graphs with (d, k) -tree-models, the space complexity can be reduced to as low as $\mathcal{O}(dk^2 \cdot \log n)$ at the cost of allowing time complexity $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$. In fact, we tackle the more general problem of computing the independent set polynomial.

► **Theorem 1.1.** *There is an algorithm which takes as input an n -vertex graph G along with a (d, k) -tree model of G , runs in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and uses at most $\mathcal{O}(dk^2 \log n)$ space, and computes the independent set polynomial of G .*

The idea of the proof of Theorem 1.1 is to reorganize the computation of the standard bottom-up dynamic programming by applying the zeta-transform to the computed tables. This allows a radical simplification of the way a dynamic programming table for a node is computed from the tables of its children, so that the whole dynamic programming can be replaced by top-down recursion. Applying just this yields an algorithm with space polynomial in n . We reduce space to $\mathcal{O}(dk^2 \log n)$ by computing the result modulo several small primes, and using space-efficient Chinese remaindering. This is inspired by the algorithm for DOMINATING SET on graphs of small treedepth of Pilipczuk and Wrochna [45].

In fact, the technique used to prove Theorem 1.1 is much more general and can be used to tackle all coloring-like problems of local character. We formalize those under a single umbrella by solving the problem of counting List H -homomorphisms (for an arbitrary but fixed pattern graph H), for which we provide an algorithm with the same complexity guarantees as those of Theorem 1.1. The concrete problems captured by this framework include, e.g., ODD CYCLE TRANSVERSAL and q -COLORING for a fixed constant q ; details are provided in Section 3.2.

Next, we turn our attention to the MAX CUT problem. This problem is W[1]-hard when parameterized by cliquewidth, but it admits a simple $n^{\mathcal{O}(k)}$ -time algorithm on n -vertex graphs provided with clique expressions of width k [24]. Our second contribution is a space-efficient counterpart of this result for graphs equipped with bounded-depth tree-models.

► **Theorem 1.2.** *There is an algorithm which takes as input an n -vertex graph G along with a (d, k) -tree model of G , runs in time $n^{\mathcal{O}(dk)}$ and uses at most $\mathcal{O}(dk \log n)$ space, and solves the MAX CUT problem in G .*

Upon closer inspection, the standard dynamic programming for MAX CUT on clique expressions solves a SUBSET SUM-like problem whenever aggregating the dynamic programming tables of children to compute the table of their parent. We apply the approach of Kane [36] that was used to solve UNARY SUBSET SUM in logarithmic space: we encode the aforementioned SUBSET SUM-like problem as computing the product of polynomials, and use Chinese remaindering to compute this product in a space-efficient way.

Finally, we consider the DOMINATING SET problem, for which we prove the following.

► **Theorem 1.3.** *There is a randomized algorithm which takes as input an n -vertex graph G along with a (d, k) -tree model of G , runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and uses at most $\mathcal{O}(dk^2 \log n + n \log n)$ space, and reports the minimum size of a dominating set in G that is correct with probability at least $1/2$.*

Note that the algorithm of Theorem 1.3 is randomized and uses much more space than our previous algorithms: more than $n \log n$. The reason for this is that we use the inclusion-exclusion approach proposed very recently by Hegerfeld and Kratsch [34], which is able to count dominating sets only modulo 2. Consequently, while the parity of the number of dominating sets of certain size can be computed in space $\mathcal{O}(dk^2 \log n)$, to determine the existence of such dominating sets we use the Isolation Lemma and count the parity of the number of dominating sets of all possible weights. This introduces randomization and necessitates sampling—and storing—a weight function. At this point we do not know how to remove neither the randomization nor the super-linear space complexity in Theorem 1.3; we believe this is an excellent open problem.

Note that in all the algorithms presented above, the running times contain a factor d in the exponent compared to the standard (exponential-space) dynamic programming on clique expressions. The following conditional lower bound shows that some additional dependency on the depth is indeed necessary; the relevant precise definitions are provided in Section 4.

► **Theorem 1.4.** *Suppose LONGEST COMMON SUBSEQUENCE cannot be solved in time $M^{f(r)}$ and space $f(r) \cdot M^{\mathcal{O}(1)}$ for any computable function f , even if the length t of the sought subsequence is bounded by $\delta(N)$ for any unbounded computable function δ ; here r is the number of strings on input, N is the common length of each string, and M is the total bitsize of the instance. Then for every unbounded computable function δ , there is no algorithm that solves the INDEPENDENT SET problem in graphs supplied with (d, k) -tree-models satisfying $d \leq \delta(k)$ that would run in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and simultaneously use $n^{\mathcal{O}(1)}$ space.*

The possibility of achieving time- and space-efficient algorithms for LONGEST COMMON SUBSEQUENCE was also the base of conjectures formulated by Pilipczuk and Wrochna [45] for their lower bounds against time- and space-efficient algorithms on graphs of bounded pathwidth. The supposition made in Theorem 1.4 is a refined version of those conjectures that takes also the length of the sought subsequence into account. The reduction underlying Theorem 1.4 is loosely inspired by the constructions of [45], but requires new ideas due to the different setting of tree-models of low depth.

Finally, given that the above results point to a fundamental role of shrubdepth in terms of space complexity, it is natural to ask whether shrubdepth can also be used to obtain meaningful tractability results with respect to the “usual” notion of fixed-parameter tractability. We conclude our exposition by highlighting two examples of problems which are NP-hard on graphs of bounded cliquewidth (and even of bounded pathwidth) [12, 37], and yet which admit fixed-parameter algorithms when parameterized by the shrubdepth.

► **Theorem 1.5.** *METRIC DIMENSION and FIREFIGHTER can be solved in fixed-parameter time on graphs supplied with (d, k) -tree-models, where d and k are considered the parameters.*

2 Preliminaries

For a positive integer k , we denote by $[k] = \{1, \dots, k\}$ and $[k]_0 = [k] \cup \{0\}$. For a function $f: A \rightarrow B$ and elements a, b (not necessarily from $A \cup B$), the function $f[a \mapsto b]: A \cup \{a\} \rightarrow B \cup \{b\}$ is given by $f[a \mapsto b](x) = f(x)$ for $x \neq a$ and $f[a \mapsto b](a) = b$. We use standard graph terminology [17].

We use the same computational model as Pilipczuk and Wrochna [45], namely the RAM model where each operation takes time polynomially proportional to the number of bits of the input, and the space is measured in terms of bits. We say that an algorithm A runs

in time $t(n)$ and space $s(n)$ if, for every input of size n , the number of operations of A is bounded by $t(n)$ and the auxiliary used space of A has size bounded by $s(n)$ bits.

Shrubdepth. We first introduce the decomposition notion for shrubdepth: *tree-models*.

► **Definition 2.1.** For $d, k \in \mathbb{N}$, a (d, k) -tree-model $(T, \mathcal{M}, \mathcal{R}, \lambda)$ of a graph G is a rooted tree T of depth d together with a family of symmetric Boolean $k \times k$ -matrices $\mathcal{M} = \{M_a\}_{a \in V(T)}$, a labeling function $\lambda: V(G) \rightarrow [k]$, and a family of renaming functions $\mathcal{R} = \{\rho_{ab}\}_{ab \in E(T)}$ with $\rho_{ab}: [k] \rightarrow [k]$ for all $ab \in E(T)$ such that:

- The leaves of T are identified with vertices of G . For each node a of T , we denote by $V_a \subseteq V(G)$ the leaves of T that are descendants of a , and with $G_a = G[V_a]$ we denote the subgraph induced by these vertices.
- With each node a of T we associate a labeling function $\lambda_a: V_a \rightarrow [k]$ defined as follows. If a is a leaf, then $\lambda_a(a) = \lambda(a)$. If a is a non-leaf node, then for every child b of a and every vertex $v \in V_b$, we have $\lambda_a(v) = \rho_{ab}(\lambda_b(v))$.
- For every pair of vertices (u, v) of G , let a denote their least common ancestor in T . Then we have $uv \in E(G)$ if and only if $M_a[\lambda_a(u), \lambda_a(v)] = 1$.

We introduce some notation. If $(T, \mathcal{M}, \mathcal{R}, \lambda)$ is a (d, k) -tree model of a graph G , then for every node a of T and every $i \in [k]$, let $V_a(i) = \lambda_a^{-1}(i)$ be the set of vertices labeled i at a . Given a subset X of V_a and $i \in [k]$, let $X_a(i) = X \cap V_a(i)$ be the vertices of X labeled i at a .

A (d, k) -tree-model can be understood as a term of depth d that constructs a k -labelled graph from single-vertex graphs by means of the following operations: renaming of the labels, and joining several labelled graphs while introducing edges between vertices originating from different parts based on their labels. This makes tree-models much closer to the *NLC-decompositions* which underly the parameter *NLC-width* than to clique expressions. *NLC-width* is a graph parameter introduced by Wanke [46] that can be seen as an alternative, functionally equivalent variant of cliquewidth.

We say that a class \mathcal{C} of graphs has *shrubdepth* d if there exists $k \in \mathbb{N}$ such that every graph in \mathcal{C} admits a (d, k) -tree-model. Thus, shrubdepth is a parameter of a graph class, rather than of a single graph; though there are functionally equivalent notions, such as *SC-depth* [30] or *rank-depth* [16], that are suited for the treatment of single graphs.

We remark that in the original definition proposed by Ganian et al. [30], there is no renaming of the labels: for every vertex $u \in V(G)$, $\lambda_a(u)$ is always the same label $\lambda(u)$ for all relevant nodes a . This boils down to all the renaming functions ρ_{ab} equal to the identify function on $[k]$. Clearly, a (d, k) -tree-model in the sense of Ganian et al. is also a (d, k) -tree-model in our sense, while a (d, k) -tree-model in our sense can be easily turned into a (d, k^{d+1}) -model in the sense of Ganian et al. by setting $\lambda_a(u)$ to be the $d+1$ tuple of consisting of labels $\lambda_a(u)$, for a ranging over the ancestors of u in T . Thus, using either definition yields the same notion of shrubdepth for graph classes. We choose to use the definition with renaming, as it provides more flexibility in the construction of tree-models that can result in a smaller number of labels and, consequently, better running times. It is also closer to the original definitions of clique expressions or *NLC-decompositions*.

Within this work we will always assume that a (d, k) -tree-model of the considered graph is provided on input. Thus, we abstract away the complexity of computing tree-models, but let us briefly discuss this problem. Gajarský and Kreutzer [28] gave an algorithm that given a graph G and parameters d and k , computes a (d, k) -tree-model of G (in the sense of Ganian et al. [30]), if there exists one, in time $f(d, k) \cdot n^{\mathcal{O}(1)}$ for a computable function f . The approach of Gajarský and Kreutzer is essentially kernelization: they iteratively “peel off”

isomorphic parts of the graph until the problem is reduced to a kernel of size bounded only in terms of d and k . This kernel is then treated by any brute-force method. Consequently, a straightforward inspection of the algorithm of [28] shows that it can be implemented so that it uses polynomial space; but not space of the form $(d+k)^{\mathcal{O}(1)} \cdot \log n$, due to the necessity of storing all the intermediate graphs in the kernelization process.

Cover products and transforms. We now recall the algebraic tools we are going to use. Let U be a finite set and R be a ring. Let $g_1, \dots, g_t: 2^U \rightarrow R$ be set functions, for some integer t . For every $i \in [t]$, the *zeta-transform* $\xi g_i: 2^U \rightarrow R$ of g_i is defined by

$$(\xi g_i)(Y) = \sum_{X \subseteq Y} g_i(X),$$

and similarly, the *Möbius-transform* $\mu g_i: 2^U \rightarrow R$ of g_i is given by

$$(\mu g_i)(Y) = \sum_{X \subseteq Y} (-1)^{|Y \setminus X|} g_i(X).$$

The *cover product* $g_1 *_c g_2 *_c \dots *_c g_t: 2^U \rightarrow R$ of g_1, \dots, g_t is defined by

$$(g_1 *_c g_2 *_c \dots *_c g_t)(Y) = \sum_{\substack{X_1, \dots, X_t \subseteq 2^{[k]} \\ X_1 \cup \dots \cup X_t = Y}} g_1(X_1) \cdot g_2(X_2) \cdot \dots \cdot g_t(X_t).$$

We emphasize that unlike another well-known concept of subset convolution, here the sets X_1, \dots, X_t are not required to be pairwise disjoint. The following result of Björklund et al. [6] will be relevant for us:

► **Lemma 2.2** ([6]). *Let U be a finite set, R be a ring, and $g_1, \dots, g_t: 2^U \rightarrow R$ be set functions for a positive integer t . Then for every $X \in 2^U$, it holds that*

$$(\xi(g_1 *_c g_2 *_c \dots *_c g_t))(X) = (\xi g_1)(X) \cdot (\xi g_2)(X) \cdot \dots \cdot (\xi g_t)(X).$$

Also for every $i \in [t]$, we have $\mu(\xi(g_i)) = g_i$.

3 Space-Efficient Algorithms on Tree-Models

3.1 Independent Set

In this section, we provide a fixed-parameter algorithm computing the independent set polynomial of a graph in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and using $\text{poly}(d, k) \log n$ space, when given a (d, k) -tree model. In particular, given a (d, k) -tree model $(T, \mathcal{M}, \mathcal{R}, \lambda)$ of an n -vertex graph G , our algorithm will allow to compute the number of independent sets of size p for each $p \in [n]$. For simplicity of representation, we start by describing an algorithm that uses $\text{poly}(d, k, n)$ space and then show how a result by Pilipczuk and Wrochna [45] can be applied to decrease the space complexity to $\text{poly}(d, k) \log n$.

In order to simplify forthcoming definitions/statements, let a be an internal node of T with b_1, \dots, b_t as children. For $S \subseteq [k]$, we denote by $q(a, S, p)$ the number of independent sets I of size p of G_a such that $S = \{i \in [k] : I_a(i) \neq \emptyset\}$. Let us define the polynomial

$$\text{IS}(a, S) = \sum_{p \in \mathbb{N}} q(a, S, p) \cdot x^p.$$

For the root r of T , the number of independent sets of G of size p is then given by

$$\sum_{S \subseteq [k]} q(r, S, p).$$

and the independent set polynomial of G is

$$\sum_{S \subseteq [k]} \text{IS}(r, S).$$

Therefore, the problem boils down to the computation of $\text{IS}(r, S)$ and its coefficients $q(r, S, p)$. A usual way to obtain a polynomial or logarithmic space algorithm is a top-down traversal of a rooted tree-like representation of the input—in our case, this will be the tree model. In this top-down traversal, the computation of coefficients $q(a, S, p)$ of $\text{IS}(a, S)$ makes some requests to the coefficients $q(b_i, S_i, p_i)$ of $\text{IS}(b_i, S_i)$ for each $i \in [t]$, for some integer p_i , and some set S_i of labels of G_{b_i} so that $\sum_{i \in [t]} p_i = p$ and $\bigcup_{i \in [t]} \rho_{ab_i}(S_i) = S$. Since there are exponentially many (in t) possible partitions of p into t integers and t can be $\Theta(n)$, we must avoid running over all such integer partitions, and this will be done by the fast computation of a certain subset cover.

We will later show that if some independent set of G_a contains vertices of labels i and j with $M_a[i, j] = 1$, then all these vertices come from the same child of a . In particular, the vertices of label i (resp. j) cannot come from multiple children of a . To implement this observation, after fixing a set S of labels, for each label class in S we “guess” (i.e., branch on) whether it will come from a single child of a or from many. Such a guess is denoted by $\alpha: S \rightarrow \{1_-, 2_\geq\}$. So, the assignment α will allow us to control the absence of edges in the sought-after independent set. For a fixed α , naively branching over all possibilities of assigning the labels of S to the children of a with respect to α would take time exponential in t , which could be as large as $\Theta(n)$. We will use inclusion-exclusion branching to speed-up the computations while retaining the space complexity. In some sense, we will first allow less restricted assignments of labels to the children of a , and then filter out the ones that result in non-independent sets using the construction of a certain auxiliary graph. The former will be implemented by using “less restricted” guesses $\beta: S \rightarrow \{1_-, 1_\geq\}$ where 1_\geq reflects that vertices of the corresponding label come from at least one child of a . Note that if the vertices of some label i come from exactly one child of a , then such an independent set satisfies both $\beta(i) = 1_-$ and $\beta(i) = 1_\geq$. Although it might seem counterintuitive, this type of guesses will enable a fast computation of a certain subset cover. After that, we will be able to compute the number of independent sets satisfying guesses of type $\alpha: S \rightarrow \{1_-, 2_\geq\}$ by observing that independent sets where some label i occurs in at least two children of a can be obtained by counting those where label i occurs in at least one child and subtracting those where this label occurs in exactly one child.

We now proceed to a formalization of the above. Let $S \subseteq \lambda_a(V_a)$ and $\alpha: S \rightarrow \{1_-, 2_\geq\}$ be fixed. Let $s_1, \dots, s_{|\alpha^{-1}(2_\geq)|}$ be an arbitrary linear ordering of $\alpha^{-1}(2_\geq)$. To compute the number of independent sets that match our choice of α , we proceed by iterating over $c \in \{0, \dots, |\alpha^{-1}(2_\geq)|\}$, and we count independent sets where the labels in $\{s_1, \dots, s_c\}$ occur exactly once, and the number of such sets where the labels occur at least once. Later, we will obtain the desired number of independent sets via carefully subtracting these two values. In particular, let $\gamma: \{s_1, \dots, s_c\} \rightarrow \{1_-, 1_\geq\}$, and we denote by $q(a, S, \alpha, c, \gamma, p)$ the number of independent sets I of size p of G_a such that

- for every label $i \notin S$, we have $I_a(i) = \emptyset$;
- for every label $i \in \{s_1, \dots, s_c\}$ with $\gamma(i) = 1_-$, there exists a unique child b_j of a such that $I_a(i) \cap V_{b_j} \neq \emptyset$;

- for every label $i \in \{s_1, \dots, s_c\}$ with $\gamma(i) = 1_{\geq}$, there exists at least one child b_j of a such that $I_a(i) \cap V_{b_j} \neq \emptyset$;
- for every label $i \in S \setminus \{s_1, \dots, s_c\}$ with $\alpha(i) = 1_{=}$, there exists a unique child b_j of a such that $I_a(i) \cap V_{b_j} \neq \emptyset$;
- and for every label $i \in S \setminus \{s_1, \dots, s_c\}$ with $\alpha(i) = 2_{\geq}$, there exist at least two children b_{j_1} and b_{j_2} of a such that $I_a(i) \cap V_{b_{j_1}} \neq \emptyset$ and $I_a(i) \cap V_{b_{j_2}} \neq \emptyset$.

Then for $c \in [\alpha^{-1}(2_{\geq})]_0$ we define the polynomial $T(a, S, \alpha, c, \gamma) \in \mathbb{Z}[x]$ as

$$T(a, S, \alpha, c, \gamma) = \sum_{p \in \mathbb{N}_0} q(a, S, \alpha, c, \gamma, p) x^p.$$

We now proceed with some observations that directly follow from the definitions.

► **Observation 3.1.** *For every $S \subseteq \lambda_a(V_a)$ and integer p , we have*

$$q(a, S, p) = \sum_{\substack{\alpha \in \{1_{=}, 2_{\geq}\}^S, \\ \gamma \in \{1_{=}, 1_{\geq}\}^{\emptyset}}} q(a, S, \alpha, 0, \gamma, p)$$

and hence,

$$\text{IS}(a, S) = \sum_{\substack{\alpha \in \{1_{=}, 2_{\geq}\}^S, \\ \gamma \in \{1_{=}, 1_{\geq}\}^{\emptyset}}} T(a, S, \alpha, 0, \gamma) \quad (1)$$

Moreover, for every $\alpha \in \{1_{=}, 2_{\geq}\}^S$, every $c \in \{0, \dots, |\alpha^{-1}(2_{\geq})| - 1\}$ and every $\gamma: \{s_1, \dots, s_c\} \rightarrow \{1_{=}, 1_{\geq}\}$, we have

$$q(a, S, \alpha, c, \gamma, p) = q(a, S, \alpha, c + 1, \gamma[s_{c+1} \mapsto 1_{\geq}], p) - q(a, S, \alpha, c + 1, \gamma[s_{c+1} \mapsto 1_{=}], p).$$

and hence

$$T(a, S, \alpha, c, \gamma) = T(a, S, \alpha, c + 1, \gamma[s_{c+1} \mapsto 1_{\geq}]) - T(a, S, \alpha, c + 1, \gamma[s_{c+1} \mapsto 1_{=}]). \quad (2)$$

It remains then to show how to compute, for every $\alpha \in \{1_{=}, 2_{\geq}\}^S$ and every $\gamma \in \{1_{=}, 1_{\geq}\}^{\alpha^{-1}(2_{\geq})}$, the polynomial $T(a, S, \alpha, |\alpha^{-1}(2_{\geq})|, \gamma)$. It is worth mentioning that if $\beta: S \rightarrow \{1_{=}, 1_{\geq}\}$ is such that $\beta^{-1}(1_{=}) = \alpha^{-1}(1_{=}) \cup \gamma^{-1}(1_{=})$ and $\beta^{-1}(1_{\geq}) = \alpha^{-1}(2_{\geq}) \setminus \gamma^{-1}(1_{=})$, then $q(a, S, \alpha, |\alpha^{-1}(1_{\geq})|, \gamma, p)$ is exactly the number of independent sets I of size p of G_a satisfying the following:

1. For every $i \in [k] \setminus S$, we have $I_a(i) = \emptyset$.
2. For every $i \in \beta^{-1}(1_{=})$, there exists a unique index $j \in [t]$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$.
3. For every $i \in \beta^{-1}(1_{\geq})$, there exists a (not necessarily unique) index $j \in [t]$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$.

We will therefore write $q(a, S, \beta, p)$ instead of $q(a, S, \alpha, |\alpha^{-1}(1_{\geq})|, \gamma, p)$ and we define the polynomial $\text{TIS}(a, S, \beta) \in \mathbb{Z}[x]$ (where ‘‘T’’ stands for ‘‘transformed’’) as $\text{TIS}(a, S, \beta) = \sum_{p \in \mathbb{N}} q(a, S, \beta, p) \cdot x^p$. Recall that because we are computing $\text{IS}(a, S)$ and $\text{TIS}(a, S, \beta)$ in a top-down manner, some queries for $\text{IS}(b_i, S_i)$ will be made during the computation. Before continuing in the computation of $\text{TIS}(a, S, \beta)$, let us first explain how to request the polynomials $\text{IS}(b_j, S_j)$ from each child b_j of a . If a is not the root, let a^* be its parent in

T , and we use $\text{PIS}(a, S)$ (where ‘‘P’’ stands for ‘‘parent’’) to denote the polynomial

$$\text{PIS}(a, S) = \sum_{p \in \mathbb{N}_0} q^p(a, S, p) \cdot x^p$$

where

$$q^p(a, S, p) = \sum_{\substack{D \subseteq \lambda_a(V_a): \\ \rho_{a^*a}(D) = S}} q(a, D, p)$$

is the number of independent sets of G_a of size p that contain a vertex with label $i \in [k]$ (i.e., $I_{a^*}(i) \neq \emptyset$) if and only if $i \in S$ holds, **where the labels are treated with respect to λ_{a^*}** . Then it holds that

$$\text{PIS}(a, S) = \sum_{\substack{D \subseteq \lambda_a(V_a): \\ \rho_{a^*a}(D) = S}} \text{IS}(a, D) . \quad (3)$$

As our next step, we make some observations that will not only allow to restrict the β 's we will need in computing the polynomial $\text{IS}(a, S)$ from the polynomials $\text{TIS}(a, S, \beta)$, but will also motivate the forthcoming definitions. Recall that we have fixed $S \subseteq \lambda_a(V_a)$ and $\beta: S \rightarrow \{1_-, 1_\geq\}$, and in $\text{IS}(a, S)$ and $\text{TIS}(a, S, \alpha)$ we are only counting independent sets I such that $I_a(i) \neq \emptyset$ if and only if $i \in S$.

► **Observation 3.2.** *If there exist $i_1, i_2 \in S$ such that $M_a[i_1, i_2] = 1$, then for any independent set I counted in $\text{IS}(a, S)$, there exists a unique $j \in [t]$ such that $I_a(i_1) \cup I_a(i_2) \subseteq V_{b_j}$.*

Proof. Both $I_a(i_1)$ and $I_a(i_2)$ are non-empty. So if there are at least two distinct j_1 and j_2 in $[t]$ such that $I_1 := I_a(i_1) \cap V_{b_{j_1}}$ and $I_2 := I_a(i_2) \cap V_{b_{j_2}}$ are non-empty, then $M_a[i_1, i_2] = 1$ implies that there is a complete bipartite graph between I_1 and I_2 . Hence the graph induced on I would contain an edge, which is a contradiction. ◀

Recall that for every label $i \in \alpha^{-1}(2_\geq)$, each independent set I contributing to the value $q(a, S, \alpha, 0, \gamma, p)$ has the property that there are distinct children b_{j_1} and b_{j_2} such that $I_a(i) \cap V_{b_{j_1}}$ and $I_a(i) \cap V_{b_{j_2}}$ are both non-empty. Then by Observation 3.2 for every $i_1 \in S$ it holds that if $\alpha(i_1) = 2_\geq$, then $M_a[i_1, i_2] = 0$ for all $i_2 \in S$. So if α does not satisfy this, the request $T(a, S, \alpha, 0, \gamma)$ can be directly answered with 0. Otherwise, since we use Observation 3.1 for recursive requests, the requests $\text{TIS}(a, S, \beta)$ made all have the property that for each $i_1 \in S$ the following holds: if $\beta(i_1) = 1_\geq$, then $M_a[i_1, i_2] = 0$ for all $i_2 \in S$. We call such β 's *conflict-free* and we restrict ourselves to only conflict-free β 's. In other words, we may assume that if $i_1, i_2 \in S$ and $M_a[i_1, i_2] = 1$, then we have $\beta(i_1) = \beta(i_2) = 1_-$. Observation 3.2 implies that for such i_1 and i_2 , each independent set I counted in $\text{TIS}(a, S, \beta)$ is such that $I_a(i_1) \cup I_a(i_2) \subseteq V_{b_j}$ for some child b_j of a . Now, to capture this observation, we define an auxiliary graph $F^{a, \beta}$ as follows. The vertex set of $F^{a, \beta}$ is $\beta^{-1}(1_-)$ and there is an edge between vertices $i_1 \neq i_2$ if and only if $M_a[i_1, i_2] = 1$. Thus, by the above observation, if we consider a connected component C of $F^{a, \beta}$, then in each independent set I counted in $\text{TIS}(a, S, \beta)$, all the vertices of I with labels from C come from a single child of a .

► **Observation 3.3.** *Let C be a connected component of $F^{a, \beta}$. For every independent set I counted in $\text{TIS}(a, S, \beta)$, there exists a unique $j \in [t]$ such that $\bigcup_{i \in C} I_a(i) \subseteq V_{b_j}$.*

We proceed with some intuition on how we compute $\text{TIS}(a, S, \beta)$ by requesting some $\text{PIS}(b_j, S_j)$. Let I be some independent set counted in $\text{TIS}(a, S, \beta)$. This set contains vertices

with labels from the set S , and the assignment β determines whether there is exactly one or at least one child from which the vertices of a certain label come from. Moreover, by Observation 3.3, for two labels i_1, i_2 from the same connected component of $F^{a,\beta}$, the vertices with labels i_1 and i_2 in I come from the same child of a . Hence, to count such independent sets, we have to consider all ways to assign labels from S to subsets of children of a such that the above properties are satisfied—namely, each connected component of $F^{a,\beta}$ is assigned to exactly one child while every label from $\beta^{-1}(1_{\geq})$ is assigned to at least one child. Since the number of such assignments can be exponential in n , we employ the fast computation of a certain subset cover.

We now formalize this step. Let $\text{cc}(F^{a,\beta})$ we denote the set of connected components of $F^{a,\beta}$. The universe $U^{a,\beta}$ (i.e., the set of objects we assign to the children of a) is defined as $U^{a,\beta} = \beta^{-1}(1_{\geq}) \cup \text{cc}(F^{a,\beta})$. For every $j \in [t]$, we define a mapping $f_j^{a,\beta} : 2^{U^{a,\beta}} \rightarrow \mathbb{Z}[x, z]$ (i.e., to polynomials over x and z) as follows: $f_j^{a,\beta}(X) = \text{PIS}(b_j, \text{flat}^{a,\beta}(X)) z^{|\text{cc}(F^{a,\beta})|}$ where $\text{flat}^{a,\beta} : 2^{U^{a,\beta}} \rightarrow 2^S$ intuitively performs a union over all the present labels—formally:

$$\text{flat}^{a,\beta}(W) = (W \cap \beta^{-1}(1_{\geq})) \cup \bigcup_{w \in W \cap \text{cc}(F^{a,\beta})} w.$$

So if we fix the set X of labels coming from the child b_j , then the (unique) coefficient in $f_j^{a,\beta}(X)$ reflects the number of independent sets of G_{b_j} using exactly these labels (with respect to λ_a). The exponent of the formal variable z is intended to store the number of connected components of $F^{a,\beta}$ assigned to b_j . This will later allow us to exclude from the computation those assignments of labels from S to children of a where the elements of some connected component of $F^{a,\beta}$ are assigned to multiple children of a . For every $j \in [t]$, we define a similar function $g_j^{a,\beta} : 2^S \rightarrow \mathbb{Z}[x, z]$ as follows:

$$g_j^{a,\beta}(Y) = \begin{cases} f_j^{a,\beta}(X) & \text{if } \text{flat}^{a,\beta}(X) = Y \text{ for some } X \in 2^{U^{a,\beta}}, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that the function $\text{flat}^{a,\beta}$ is injective and hence $g_j^{a,\beta}$ is well-defined. The mapping $g_j^{a,\beta}$ filters out those assignments where some connected component of $F^{a,\beta}$ is “split”. For simplicity of notation, when a and β are clear from the context, we omit the superscript a, β .

Crucially for our algorithm, we claim that the following holds:

$$\text{TIS}(a, S, \beta) = \left(\sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S}} g_1(X_1) g_2(X_2) \dots g_t(X_t) \right) \langle z^{|\text{cc}(F)|} \rangle$$

where for a polynomial $P = \sum_{u_1, u_2 \in \mathbb{N}_0} q_{u_1, u_2} x^{u_1} z^{u_2} \in \mathbb{Z}[x, z]$ the polynomial $P \langle z^{|\text{cc}(F)|} \rangle \in \mathbb{Z}[x]$ is defined as $P \langle z^{|\text{cc}(F)|} \rangle = \sum_{u_1 \in \mathbb{N}_0} q_{u_1, |\text{cc}(F)|} x^{u_1}$. In simple words, the $\langle z^{|\text{cc}(F)|} \rangle$ operator first removes all terms where the degree of z is not equal to $|\text{cc}(F)|$ and then “forgets” about z . Before we provide a formal proof, let us sketch the idea behind it. On the left side of the equality, we have the polynomial keeping track of the independent sets of G_a that “respect” β . First, for every label $i \in S$, some vertex of this label must occur in at least one child of a : this is handled by considering all covers $X_1 \cup \dots \cup X_t = S$ where for every $j \in [t]$, the set X_j represents the labels assigned to the child b_j . Next, if some X_j “splits” a connected component, i.e., takes only a proper non-empty subset of this component, then such an assignment would not yield an independent set by Observation 3.3 and the function g_j ensures that the corresponding cover contributes zero to the result. Hence, for every cover

$X_1 \cup \dots \cup X_t = S$ with a non-zero contribution to the sum, every connected component of F is completely contained in at least one X_j . In particular, this implies that for every non-zero term on the right side, the degree of the formal variable z in this term is at least $z^{|\text{cc}(F)|}$. On the other hand, if some connected component of F is contained in several sets X_j , then the degree of the corresponding monomial is strictly larger than the total number of connected components and such covers X_1, \dots, X_t are excluded from the consideration by applying $\langle z^{|\text{cc}(F)|} \rangle$. We formalize this intuition below:

► **Lemma 3.4.** *Let $(T, \mathcal{M}, \mathcal{R}, \lambda)$ be a (d, k) -tree model of an n -vertex graph G . Let a be a non-leaf node of T and let b_1, \dots, b_t be the children of a . For every $S \subseteq \lambda_a(V_a)$, and every conflict-free $\beta: S \rightarrow \{1=, 1\geq\}$, it holds that*

$$\text{TIS}(a, S, \beta) = \left(\sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S}} \left(\prod_{j=1}^t g_j^{a, \beta}(X_j) \right) \right) \langle z^{|\text{cc}(F^{a, \beta})|} \rangle.$$

Proof. First, we bring the right-hand side of the equality into a more suitable form.

$$\begin{aligned} \sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S}} \prod_{j=1}^t g_j(X_j) &= \\ \sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S, \\ \forall j \in [t] \exists W_j \subseteq U: \text{flat}(W_j) = X_j}} \prod_{j=1}^t \text{PIS}(b_j, X_j) z^{|\text{cc}(F)|} &= \\ \sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S, \\ \forall j \in [t] \exists W_j \subseteq U: \text{flat}(W_j) = X_j}} \left(\prod_{j=1}^t \left(\sum_{p_j \in \mathbb{N}_0} q^\rho(b_j, X_j, p_j) x^{p_j} \right) z^{|\text{cc}(F)|} \right) &= \\ \sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S, \\ \forall j \in [t] \exists W_j \subseteq U: \text{flat}(W_j) = X_j \\ p_1, \dots, p_t \in \mathbb{N}_0}} \prod_{j=1}^t q^\rho(b_j, X_j, p_j) x^{p_j} z^{|\text{cc}(F)|} &= \\ \sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S, \\ \forall j \in [t] \exists W_j \subseteq U: \text{flat}(W_j) = X_j \\ p_1, \dots, p_t \in \mathbb{N}_0}} \left(\prod_{j=1}^t q^\rho(b_j, X_j, p_j) \right) x^{\sum_{j=1}^t p_j} z^{\sum_{j=1}^t |\text{cc}(F)|} &= \end{aligned}$$

We recall that flat is injective so the sum above is well-defined. So we have to prove that

$$\text{TIS}(a, S, \beta) = \left(\sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S, \\ \forall j \in [t] \exists W_j \subseteq U: \text{flat}(W_j) = X_j, \\ p_1, \dots, p_t \in \mathbb{N}_0}} \left(\prod_{j=1}^t q^\rho(b_j, X_j, p_j) \right) x^{\sum_{j=1}^t p_j} z^{\sum_{j=1}^t |\text{cc}(F)|} \right) \langle z^{|\text{cc}(F)|} \rangle,$$

i.e.,

$$\text{TIS}(a, S, \beta) = \sum_{\substack{X_1, \dots, X_t \subseteq 2^{[k]}: \\ X_1 \cup \dots \cup X_t = S, \\ \forall j \in [t] \exists W_j : \text{flat}(W_j) = X_j, \\ \sum_{j \in [t]} |W_j \cap \text{cc}(F)| = |\text{cc}(F)|, \\ p_1, \dots, p_t \in \mathbb{N}_0}} \left(\prod_{j=1}^t q^\rho(b_j, X_j, p_j) \right) x^{\sum_{j=1}^t p_j} \quad (4)$$

To prove that these two polynomials are equal, we show that for every power $p \in \mathbb{N}_0$ of x , the coefficients at x^p in both polynomials are equal. So let us fix an arbitrary integer p .

For one direction, let I be an independent set counted in the coefficient $q(a, S, \beta, p)$ at the term x^p on the left-hand side $\text{TIS}(a, S, \beta)$; in particular, we then have $|I| = p$. For every $j \in [t]$, let $I^j = I \cap V_{b_j}$, $p_j = |I^j|$, and $X_j = \{i \in [k] : I_a(i) \cap V_{b_j} \neq \emptyset\}$. Clearly, we have $p_1 + \dots + p_t = p$ and $X_1 \cup \dots \cup X_t = S$. Now consider some $j \in [t]$. The set I^j is an independent set of G_{b_j} that contains vertices with labels from exactly X_j (with respect to λ_a). So I^j is counted in $q^\rho(b_j, X_j, p_j)$. Let $A_j = X_j \cap \beta^{-1}(1_{=})$ and $B_j = X_j \cap \beta^{-1}(1_{\geq})$. Note that $A_j \cup B_j = X_j$, $A_1 \cup \dots \cup A_t = \beta^{-1}(1_{=})$, and $B_1 \cup \dots \cup B_t = \beta^{-1}(1_{\geq})$. Then by Observation 3.3, for every connected component C of F and every $j \in [t]$, we either have $C \subseteq X_j$ or $C \cap X_j = \emptyset$. Therefore, for every $j \in [t]$, we have $X_j = B_j \cup \bigcup_{C \in \text{cc}(F) : C \cap X_j \neq \emptyset} C$ and hence, $X_j = \text{flat}(W_j)$ where $W_j = B_j \cup \{C \in \text{cc}(F) : C \cap X_j \neq \emptyset\}$. Finally, by the definition of objects counted in $\text{TIS}(x, S, \beta)$, since the labels from $\beta^{-1}(1_{=})$ occur in exactly one child of a , it holds that $A_{j_1} \cap A_{j_2} = \emptyset$ for any $j_1 \neq j_2 \in [t]$. Together with $A_1 \cup \dots \cup A_t = \beta^{-1}(1_{=})$ this implies that for every connected component C of F , there exists exactly one index $j_C \in [t]$ with $C \subseteq A_{j_C}$, i.e., $C \in W_{j_C}$. So we obtain $\sum_{j \in [t]} |W_j \cap \text{cc}(F)| = |\text{cc}(F)|$. Altogether, the tuple (I^1, \dots, I^t) is

counted in the product $\prod_{j=1}^t q^\rho(b_j, X_j, p_j)$ and the properties shown above imply that this product contributes to the coefficient at the monomial x^p . Also note that the mapping of I to (I^1, \dots, I^t) is injective so we indeed obtain that the coefficient at x^p on the left-hand side of (4) is at most as large as one the right-hand side.

Now we show that the other inequality holds as well. Let $X_1, \dots, X_t \subseteq [k]$, $I^1, \dots, I^t \subseteq V$, $W_1, \dots, W_t \subseteq U$, and $p_1, \dots, p_t \in \mathbb{N}_0$ be such that the following properties hold:

- $p_1 + \dots + p_t = p$,
- $X_1 \cup \dots \cup X_t = S$,
- for every $j \in [t]$, it holds that $\text{flat}(W_j) = X_j$,
- $\sum_{j \in [t]} |W_j \cap \text{cc}(F)| = |\text{cc}(F)|$,
- and for every $j \in [t]$, the set I^j is an independent set of G_{b_j} of size p_j such that for every $i \in [k]$, $I_a^j(i) \neq \emptyset$ holds iff $i \in X_j$, i.e., I^j is counted in $q^\rho(b_j, X_j, p_j)$.

Let $I = I^1 \cup \dots \cup I^t$. Since for every $j \in [t]$, we have $I^j \subseteq V_{b_j}$, the sets I^1, \dots, I^t are pairwise disjoint and we have $|I| = p$. We also have $I \subseteq V_a$ and for every $i \in [k]$, we have $I_a(i) \neq \emptyset$ iff $i \in S$, i.e., I contains vertices with labels from exactly S with respect to λ_a . We claim that I is an independent set of G_a . Since I_1, \dots, I_t are independent sets of G_{b_1}, \dots, G_{b_t} , respectively, and G_{b_1}, \dots, G_{b_t} are induced subgraphs of G_a , it suffices to show that there are no edges between I_{j_1} and I_{j_2} for any $j_1 \neq j_2 \in [t]$. For this, suppose there is an edge $v_1 v_2$ of G_a with $v_1 \in I^{j_1}$ and $v_2 \in I^{j_2}$ for some $j_1 \neq j_2 \in [t]$. Also let $i_1 = \lambda_a(v_1)$ and $i_2 = \lambda_a(v_2)$. Since a is the lowest common ancestor of v_1 and v_2 , it holds that $M_a[i_1, i_2] = 1$. By the assumption of the lemma, the mapping β is conflict-free so we have $\beta(i_1) = \beta(i_2) = 1_{=}$. Then, the property $M_a[i_1, i_2] = 1$ implies that i_1 and i_2 belong to the same connected component, say C , of

F . Recall that we have $i_1 \in X_{j_1}$, $i_2 \in X_{j_2}$, $\text{flat}(W_{j_1}) = X_{j_1}$, and $\text{flat}(W_{j_2}) = X_{j_2}$. Hence, it holds that $C \in W_{j_1}$ and $C \in W_{j_2}$. On the other hand, let C' be an arbitrary connected component of F and let $i \in C'$ be some label. The property $X_1 \cup \dots \cup X_t = S$ implies that there exists an index $j_{C'}$ with $i \in X_{j_{C'}}$. Due to $\text{flat}(W_{j_{C'}}) = X_{j_{C'}}$, we then have $C' \in W_{j_{C'}}$. I.e., every connected component of F is contained in at least one of the sets W_1, \dots, W_t while C is contained in at least two such sets so we get $\sum_{j \in [t]} |W_j \cap \text{cc}(F)| > |\text{cc}(F)|$ – a contradiction.

Hence, the set I is indeed an independent set of G_a of size p such that it contains vertices with labels from exactly S with respect to λ_a . So it is counted in the coefficient $q^\rho(a, S, \beta, p)$ of the term x^p in $\text{TIS}(a, S, \beta)$. Finally, first note that (I^1, \dots, I^t) is uniquely mapped to the tuple $(X_1, \dots, X_t, p_1, \dots, p_t)$ so it is counted only once on the right-hand side. And second, the mapping of (I^1, \dots, I^t) to I is injective (since V_{b_1}, \dots, V_{b_t} are pairwise disjoint). Therefore, the coefficient at the term x^p on the right-hand side of (4) is at most as large as on the left-hand side. Altogether, we conclude that the two polynomials in (4) are equal, as desired. \blacktriangleleft

The above lemma implies that:

$$\begin{aligned}
\text{TIS}(a, S, \beta) &= \\
&\sum_{\substack{X_1, \dots, X_t \subseteq [k]: \\ X_1 \cup \dots \cup X_t = S}} \left(\prod_{j=1}^t g_j(X_j) \right) \langle z^{|\text{cc}(F)|} \rangle = \\
&(g_1 *_{\mathcal{C}} g_2 *_{\mathcal{C}} \dots *_{\mathcal{C}} g_t)(S) \langle z^{|\text{cc}(F)|} \rangle \stackrel{\text{Lemma 2.2}}{=} \\
&((\mu(\xi(g_1 *_{\mathcal{C}} g_2 *_{\mathcal{C}} \dots *_{\mathcal{C}} g_t)))(S)) \langle z^{|\text{cc}(F)|} \rangle = \\
&\left(\sum_{Y \subseteq S} (-1)^{|S \setminus Y|} (\xi(g_1 *_{\mathcal{C}} g_2 *_{\mathcal{C}} \dots *_{\mathcal{C}} g_t))(Y) \right) \langle z^{|\text{cc}(F)|} \rangle \stackrel{\text{Lemma 2.2}}{=} \\
&\left(\sum_{Y \subseteq S} (-1)^{|S \setminus Y|} (\xi_{g_1})(Y) (\xi_{g_2})(Y) \dots (\xi_{g_t})(Y) \right) \langle z^{|\text{cc}(F)|} \rangle = \\
&\left(\sum_{Y \subseteq S} (-1)^{|S \setminus Y|} \prod_{j=1}^t (\xi_{g_j})(Y) \right) \langle z^{|\text{cc}(F)|} \rangle = \\
&\left(\sum_{Y \subseteq S} (-1)^{|S \setminus Y|} \prod_{j=1}^t \sum_{Z \subseteq Y} g_j(Z) \right) \langle z^{|\text{cc}(F)|} \rangle \tag{5}
\end{aligned}$$

We now have the equalities required for our algorithm to solve INDEPENDENT SET parameterized by shrubdepth. By using these equalities directly, we would obtain an algorithm running in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2n^2)$. However, the latter can be substantially improved by using a result of Pilipczuk and Wrochna [45] based on the Chinese remainder theorem:

► **Theorem 3.5** ([45]). *Let $P(x) = \sum_{i=0}^{n'} q_i x^i$ be a polynomial in one variable x of degree at most n' with integer coefficients satisfying $0 \leq q_i \leq 2^{n'}$ for $i = 0, \dots, n'$. Suppose that given a prime number $p \leq 2n' + 2$ and $s \in \mathbb{F}_p$, the value $P(s) \pmod{p}$ can be computed in time T and space S . Then given $k \in \{0, \dots, n'\}$, the value q_k can be computed in time $\mathcal{O}(T \cdot \text{poly}(n'))$ and space $\mathcal{O}(S + \log n')$.*

With this, we can finally prove:

► **Theorem 1.1.** *There is an algorithm which takes as input an n -vertex graph G along with a (d, k) -tree model of G , runs in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and uses at most $\mathcal{O}(dk^2 \log n)$ space, and computes the independent set polynomial of G .*

Proof. The independent set polynomial of the graph $G = G_r$ is exactly $\sum_{S \subseteq [k]} \text{IS}(r, S)$ where r is the root of T . Let us denote this polynomial P . To apply Theorem 3.5, we use $n' := n$. Let $p \leq 2n + 2$ be a prime number. The bound on p implies that any number from \mathbb{F}_p can be encoded using $\mathcal{O}(\log n)$ bits, this will bound the space complexity. There are at most 2^n independent sets of G so every coefficient of P lies between 0 and 2^n , and therefore the prerequisites stated in the first sentence of Theorem 3.5 are satisfied. Let $s \in \mathbb{F}_p$. We will now show that the value $P(s) \bmod p$ can be evaluated in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2 \log n)$. At that point, the result will follow by Theorem 3.5.

Since we are interested in the evaluation of P at s modulo p , instead of querying and storing all coefficients, as a result of the recursion, we return the evaluation of a certain polynomial (e.g., $\text{TIS}(a, S, \beta)$) at s modulo p . For this, the formal variable x is always substituted by s and then arithmetic operations in \mathbb{F}_p are carried out. In the following, when computing a sum (resp. product) of certain values, these values are computed recursively one after another and we store the counter (e.g., current subset $S \subseteq [k]$) as well as the current value of the sum (resp. product). Our algorithm relies on the equalities provided above and we now provide more details to achieve the desired time and space complexity. Let us denote $\text{AIS}(a, S, \alpha) := T_0(a, S, \alpha, 0, \emptyset)$ for simplicity.

First, if a is a leaf of T , then $\text{IS}(a, S)$ can be computed directly via

$$\text{IS}(a, S) = \begin{cases} 1 & \text{if } S = \emptyset \\ x & \text{if } S = \{\lambda_a(a)\} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

and this is our base case. Otherwise, the queries are answered recursively and five types of queries occur, namely $\text{IS}(a, S)$, $\text{AIS}(a, S, \beta)$, $T(a, S, \alpha, c, \gamma)$, $\text{TIS}(a, S, \beta)$, and $\text{PIS}(a, S)$. Let a be an inner node with children b_1, \dots, b_t . To answer a query $T(a, S, \alpha, c, \gamma)$ for $c < |\alpha^{-1}(2_{\geq})|$, we recurse via (1). If $c = |\alpha^{-1}(2_{\geq})|$, then we first construct $\beta: S \rightarrow \{1_{=}, 1_{\geq}\}$ given by $\beta^{-1}(1_{=}) = \alpha^{-1}(1_{=}) \cup \gamma^{-1}(1_{=})$ and $\beta^{-1}(1_{\geq}) = \alpha^{-1}(2_{\geq}) \setminus \gamma^{-1}(1_{=})$ and then query $\text{TIS}(a, S, \beta)$. Then, to answer a query $\text{TIS}(a, S, \beta)$, we recurse via (5). And finally, to answer a query $\text{PIS}(a, S)$, we recurse using (3).

Each of the above recurrences is given by a combination of sums and products of the results of recursive calls and these values are from \mathbb{F}_p . To keep the space complexity of the algorithm bounded, for such recursion, the result is computed “from inside to outside” by keeping track of the current sums (resp. products) as well as the next value to be queried. For example, for (5), we iterate through all $Y \subseteq S$, store the current value of the outer sum (modulo p), then for fixed Y , we iterate over $j \in [t]$ and store j and the current value of the product (modulo p), and then for fixed Y and j , iterate through $Z \subseteq Y$ and store the current value of Z and the current inner sum. After the complete iteration over Z (resp. j) we update the current value of the product (resp. outer sum) and move on to the next j (resp. Y).

Now we analyze the time and space complexity of the algorithm. We start with the running time. For this, we analyze how often every query is answered. Namely, for all relevant values of a, S, α, β, c , and γ , for each query $\text{IS}(a, S)$, $\text{AIS}(a, S, \alpha)$, $T(a, S, \alpha, c, \gamma)$, $\text{TIS}(a, S, \beta)$, resp. $\text{PIS}(a, S)$, we use $Q(\text{IS}(a, S))$, $Q(\text{AIS}(a, S, \alpha))$, $Q(T(a, S, \alpha, c, \gamma))$, $Q(\text{TIS}(a, S, \beta))$, $Q(\text{PIS}(a, S))$,

respectively, to denote the number of times the query is answered by the algorithm and we call this value the *multiplicity* of the query. Then, for $h \in [d]_0$, we define the value $q_{\text{IS}}(h)$ to be the maximum multiplicity of a query $\text{IS}(a, S)$ over all nodes a at height h in T and all reasonable S . Similarly, we define the values $Q_{\text{AIS}}(h)$, $Q_T(h)$, $Q_{\text{TIS}}(h)$, and $Q_{\text{PIS}}(h)$ where we maximize over all nodes a at height h and all reasonable values of S , α , β , γ and c . We now upper bound these values.

Let b be a node at height h for some $h \in [d]_0$. If $b = r$, then a query $\text{PIS}(b, S)$ is not asked at all. Otherwise, let a be the parent of b , and let j be such that $b = b_j$ is the j -th child of a . Then $\text{PIS}(b, S)$ can be asked when answering some query of the form $\text{TIS}(a, D, \beta)$ to compute some value $\xi g_j^{a, \beta}(Y)$ such that $S \subseteq Y \subseteq D$. Therefore, for fixed D and β , the value $\text{IS}(b, S)$ is queried at most 2^k times, so we obtain $Q(\text{PIS}(b, S)) \leq \sum_{S \subseteq D \subseteq [k], \beta: D \rightarrow \{1=, 1\geq\}} Q(\text{TIS}(a, D, \beta))$ and hence,

$$Q_{\text{PIS}}(h) \begin{cases} = 0 & \text{if } h = 0 \\ \leq 2^{3k} Q_{\text{TIS}}(h-1) & \text{otherwise} \end{cases}.$$

Next, we consider a query of form $\text{TIS}(b, S, \beta)$. Observe that for every $\alpha: S \rightarrow \{1=, 2\geq\}$ when recursing via (1) to answer $T(a, S, \alpha, 0, \emptyset)$, we *branch* on the values $1=$ and $1\geq$ for $s_1, \dots, s_{|\alpha^{-1}(2\geq)|}$ one after another. Thus, after $|\alpha^{-1}(2\geq)|$ steps every branch results in its own $\gamma: s_1, \dots, s_{|\alpha^{-1}(2\geq)|} \rightarrow \{1=, 1\geq\}$, and hence, in its own $\beta: S \rightarrow \{1=, 1\geq\}$. Therefore, if we fix α , then every $\text{TIS}(a, S, \beta)$ is queried at most once when answering $T(a, S, \alpha, 0, \emptyset)$. Hence, we have

$$Q(\text{TIS}(b, S, \beta)) \leq \sum_{\alpha: S \rightarrow \{1=, 1\geq\}} Q(\text{AIS}(b, S, \alpha))$$

and therefore, $Q_{\text{TIS}}(h) \leq 2^k Q_{\text{AIS}}(h)$. Every query $T(b, S, \alpha, c, \gamma)$ is also asked at most once while answering a query of $T(b, S, \alpha, 0, \emptyset)$, i.e., $Q(T(b, S, \alpha, c, \gamma)) \leq Q(\text{AIS}(b, S, \alpha))$ and $Q_T(h) \leq Q_{\text{AIS}}(h)$.

Further, for each fixed α , a query $\text{AIS}(b, S, \alpha)$ is asked exactly once for every query of $\text{IS}(b, S)$, i.e., $Q(\text{AIS}(b, S, \alpha)) \leq Q(\text{IS}(b, S))$ and $Q_{\text{AIS}}(h) \leq Q_{\text{IS}}(h)$. Finally, a query of the form $\text{IS}(b, S)$ is queried at most once for every query of the form $\text{PIS}(b, D)$, so we have $Q(\text{IS}(b, S)) \leq \sum_{D \subseteq [k]} Q(\text{PIS}(b, D))$ and $Q_{\text{IS}}(h) \leq 2^k Q_{\text{PIS}}(h)$.

By induction over h , we obtain that

$$Q_{\text{AIS}}(h), Q_T(h), Q_{\text{TIS}}(h), Q_{\text{IS}}(h), Q_{\text{PIS}}(h) \leq 2^{5hk}$$

and

$$Q_{\text{AIS}}(h), Q_T(h), Q_{\text{TIS}}(h), Q_{\text{IS}}(h), Q_{\text{PIS}}(h) \leq 2^{5dk} \in 2^{\mathcal{O}(kd)}$$

for every $h \in [d]_0$, i.e., any fixed query is asked $2^{\mathcal{O}(kd)}$ times.

There are $\mathcal{O}(nd)$ nodes in T and there are at most 2^k reasonable values of S ; for any S , there are at most 2^k choices for α , β , and γ ; and there are at most k reasonable values of b . Hence, there are at most

$$\mathcal{O}(nd)(2^{2k} + 2^{3k}k + 2^{2k} + 2^k + 2^k) \in 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$$

different forms of queries and so there are at most $2^{\mathcal{O}(kd)} 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ recursive calls.

Next, we bound the time spent on each query additionally to the recursive calls. For each query, this additional time is mostly determined by $\mathcal{O}(2^{2k}n)$ arithmetic operations. For a

query of the form $\text{TIS}(\cdot)$, arithmetic operations are carried out over polynomials in a formal variable z where the coefficients are from \mathbb{F}_p . It is crucial to observe that since in the end of the computation we apply the $\langle z^{|\text{cc}(F)|} \rangle$ operation and the auxiliary graph F has at most k connected components, we can safely discard coefficients at terms z^r for any $r > k$. Therefore, it suffices to keep track of at most k coefficients from \mathbb{F}_p . For the remaining queries, the arithmetic operations are carried out over \mathbb{F}_p . So in any case, there are at most k relevant values from \mathbb{F}_p to store as a partial sum resp. product and a single arithmetic operation can be therefore carried out in $n^{\mathcal{O}(1)}$ time. Further, when answering a query of the form $\text{TIS}(a, S, \beta)$ and computing a value of the form $g_j^{a,\beta}(X_j)$ for this, we can check whether for X_j there is W_j with $\text{flat}^{a,\beta}(W_j) = X_j$ as follows. First, we compute the connected components of $F^{a,\beta}$: we start with a partition of $\beta^{-1}(1_{\perp})$ into singletons and then iterate over all pairs of vertices i_1 and i_2 and if $M_a[i_1, i_2] = 1$, then we merge the sets containing i_1 and i_2 . As a result of this process, we obtain the set of connected components of $F^{a,\beta}$. Then for each connected component C , we check if $C \cap X_j \in \{\emptyset, C\}$ holds. If this does not hold for at least one connected component, then we conclude that $g_j^{a,\beta}(X_j) = 0$. Otherwise, the desired set W_j exists and we have $|W_j \cap \text{cc}(F^{a,\beta})| = r$ where r is the number of connected components C with $C \cap X_j = C$. This process then runs in time $\mathcal{O}(k^3)$ and space $\mathcal{O}(k)$. Although this can be accelerated, this step is not a bottleneck so this time and space complexity suffices for our purposes. Also, when answering a query of the form $\text{AIS}(a, S, \alpha)$, we need to check whether there exist labels $i_1, i_2 \in S$ with $\alpha(i_1) = 1_{\geq}$ and $M_a[i_1, i_2] = 1$: this can be done in time $\mathcal{O}(k^2)$ and space $\mathcal{O}(\log k)$ by considering all pairs $i_1, i_2 \in S$ and looking up these properties. So for any query, the time spent on this query apart from the recursive calls is bounded by $\mathcal{O}(2^{2k} n \cdot k^3 \cdot \log n) = 2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. And the total running time of the algorithm is bounded by $2^{\mathcal{O}(kd)} \cdot \text{poly}(n) \cdot 2^{\mathcal{O}(k)} \cdot \text{poly}(n) = 2^{\mathcal{O}(kd)} \cdot \text{poly}(n)$, i.e., the number of queries times the complexity of a single query.

Finally, we bound the space complexity. The space used by a single query is to store the partial sums and/or products modulo p as well as the counters that store the information about the next recursive call (e.g., current S). For any query other than $\text{TIS}(\cdot)$, the partial result is in \mathbb{F}_p . For a query of the form $\text{TIS}(\cdot)$, we are working with a polynomial in the formal variable z . Above we have argued why the coefficients at z^p for $p > r$ can be discarded. Therefore, it suffices to keep track of at most k coefficients from \mathbb{F}_p . Recall that $p \leq 2n + 2$ so any value from \mathbb{F}_p can be encoded with $\log n$ bits. When answering a query of the form $\text{TIS}(a, S, \beta)$, we also need to consider the connected components of $F^{a,\beta}$: as argued above, this can be accomplished in $\mathcal{O}(k)$ space. So the space complexity of a single query can be bounded by $\mathcal{O}(k \log n) + \mathcal{O}(k) + \log n = \mathcal{O}(k \log n)$. The depth of the recursion is bounded by $\mathcal{O}(kd)$: the depth of T is d and for each node, there are at most $k + 4$ recursive calls queried at this node (namely, $\text{PIS}(\cdot)$, $\text{IS}(\cdot)$, $\text{AIS} = T(\cdot, c = 0, \cdot)$, \dots , $T(\cdot, c = |\alpha^{-1}(2_{\geq})| \leq k, \cdot)$, $\text{TIS}(\cdot)$). Finally, during the algorithm we need to keep track of the node we are currently at. Therefore, the space complexity of the algorithm is $\mathcal{O}(kd)\mathcal{O}(k \log n) + \mathcal{O}(\log n) = \mathcal{O}(k^2 d \log n)$. ◀

3.2 Counting List-Homomorphisms

We now explain how to apply the techniques from Section 3.1 to a broader class of problems, namely all problems expressible as instantiations of the $\#$ -LIST- H -HOMOMORPHISM problem for a fixed pattern graph H (which we will introduce in a moment). In this way, we cover problems such as ODD CYCLE TRANSVERSAL and q -COLORING, for a fixed q . Furthermore, the techniques will be useful for solving DOMINATING SET later.

Let H be a fixed undirected graph (possibly with loops) and let $R \subseteq V(H)$ be a designated set of vertices. An instance of the R -WEIGHTED $\#$ -LIST- H -HOMOMORPHISM problem

consists of a graph G , a weight function $\omega: V(G) \rightarrow \mathbb{N}$, a list function $L: V(G) \rightarrow 2^{V(H)}$, a cardinality $C \in \mathbb{N}$ and a total weight $W \in \mathbb{N}$. The goal is to count the number of list H -homomorphisms of G such that exactly C vertices of G are mapped to R and their total weight in ω is W . More formally, we seek the value

$$\left| \left\{ \varphi: V(G) \rightarrow V(H) \mid \forall v \in V(G): \varphi(v) \in L(v), \forall uv \in E(G): \varphi(u)\varphi(v) \in E(H), \right. \right. \\ \left. \left. |\varphi^{-1}(R)| = C, \text{ and } \omega(\varphi^{-1}(R)) = W \right\} \right| .$$

We say that such φ has cardinality C and weight W . For the “standard” $\#$ -LIST H -HOMOMORPHISM problem we would use $R = V(H)$, $C = W = |V(G)|$, and unit weights. We also have the following special cases of the R -WEIGHTED $\#$ -LIST- H -HOMOMORPHISM problem. In all cases, we consider unit weights.

- To model INDEPENDENT SET, the pattern graph H consists of two vertices \mathbf{u} and \mathbf{v} and the edge set contains a loop at \mathbf{v} and the edge \mathbf{uv} . The set R consists of \mathbf{u} only. Then INDEPENDENT SET is equivalent to finding the largest C for which we have a positive number of solutions in the constructed instance of R -WEIGHTED $\#$ -LIST- H -HOMOMORPHISM.
- Similarly, to model ODD CYCLE TRANSVERSAL, the pattern graph H is a triangle on vertex set $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ with a loop added on \mathbf{u} . Again, we take $R = \{\mathbf{u}\}$.
- To model q -COLORING, we take H to be the loopless clique on q vertices, and $R = V(H)$. While in all the cases described above we only use unit weights, we need to work with any weight function in our application to DOMINATING SET.

► **Theorem 3.6.** *Fix a graph H (possibly with loops) and $R \subseteq V(H)$. There is an algorithm which takes as input an n -vertex graph G together with a weight function ω and a (d, k) -tree-model, runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)} \cdot (W^*)^{\mathcal{O}(1)}$ and uses space $\mathcal{O}(k^2 d(\log n + \log W^*))$, and solves the R -WEIGHTED $\#$ -LIST- H -HOMOMORPHISM in G , where W^* denotes the maximum weight in ω .*

Using the argumentation above, from Theorem 3.6 we can derive the following corollaries.

► **Corollary A.** *Fix a graph H (possibly with loops). Then given an n -vertex graph G together with a (d, k) -tree-model, $\#$ -LIST- H -HOMOMORPHISM in G can be solved in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2 \log n)$.*

► **Corollary B.** *Fix $q \in \mathbb{N}$. Then given an n -vertex graph G together with a (d, k) -tree-model, q -COLORING and ODD CYCLE TRANSVERSAL in G can be solved in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2 \log n)$.*

The remainder of this section is devoted to the proof of Theorem 3.6. We assume that the reader is familiar with the approach presented in Section 3.1, as we will build upon it.

Let now H and R be fixed and let $W^* = \max_{v \in V} \omega(v)$ be the maximum weight in ω . Now we show how to adapt our techniques from Section 3.1 to the R -WEIGHTED $\#$ -LIST- H -HOMOMORPHISM problem. We assume that the graph G is provided with a (d, k) -model $(T, \mathcal{M}, \mathcal{R}, \lambda)$. There are two main changes: first, we adapt the dynamic programming formulas and second, we show how to apply Theorem 3.5 to polynomials in two variables that will appear in the proof.

We start with dynamic programming. Let a be a node of T . For MAXIMUM INDEPENDENT SET, our guess S was the set of labels occurring in an independent set of the current subgraph G_a . Now, instead, we guess a subset S of **States** := $\{(\mathbf{h}, i) : \mathbf{h} \in V(H), i \in [k]\}$. For each label $i \in [k]$, the set S is intended to reflect to which vertices of H the set V_a^i is mapped

by a homomorphism. The set **States** has size $|V(H)| \cdot k$, i.e., $\mathcal{O}(k)$ for fixed H . So as in Section 3.1, there are still $2^{\mathcal{O}(k)}$ possibilities for S and this will be the reason for the running time of $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ as in that section. As before, we then employ guesses of the form $\alpha: S \rightarrow \{1_-, 2_\geq\}$ and $\beta: S \rightarrow \{1_\geq, 1_-\}$ to compute the polynomials reflecting the number of H -homomorphisms of certain cardinality via inclusion-exclusion. Further, we need to forbid that edges of G are mapped to non-edges of H . For this, the auxiliary graph $F^{a,\beta}$ again has vertex set $\beta^{-1}(1_-)$ but now there is an edge between two vertices (\mathbf{h}, i) and (\mathbf{h}', j) whenever $M_a[i, j] = 1$ and $\mathbf{h}\mathbf{h}'$ is not an edge of H . Then, if a homomorphism maps a vertex v_1 with label i to \mathbf{h} and a vertex v_2 with label j to \mathbf{h}' , our approach from Section 3.1 ensures that v_1 and v_2 come from the same child of a so that no edge between v_1 and v_2 is created at a .

In Section 3.1, all polynomials had only one variable x whose degree reflected the size of an independent set. Here, additionally to cardinality we are interested in the weight of vertices mapped to H . So instead of univariate polynomials from $\mathbb{Z}[x]$, we use polynomials in two variables x and y where the degree of y keeps track of the weight. The weights of partial solutions are initialized in the leaves of the tree-model, there we also take care of lists L : the polynomial for a guess S and a leaf v is given by $x \cdot y^{\omega(v)}$ provided $S = \{(\mathbf{h}, i)\}$ for some $\mathbf{h} \in L(v)$ and $i = \lambda(v)$, and otherwise this polynomial is the zero polynomial.

With this adaptations in hand, by a straightforward implementation of the recursion we can already obtain a $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ -time algorithm that uses only polynomial space and computes the polynomial $Q(x, y) = \sum_{p \in [n]_0, w \in [nW^*]_0} q_{p,w} x^p y^w$ where $q_{p,w}$ is the number of list H -homomorphisms of G of cardinality p and weight w . The answer to the problem is then the value $q_{C,W}$. To obtain logarithmic dependency on the graph size in space complexity, in Section 3.1 we relied on Theorem 3.5. However, Theorem 3.5 concerns univariate polynomials, while Q has two variables. We now explain how to model Q as a univariate polynomial $P \in \mathbb{Z}[t]$ in order to apply the theorem.

Let

$$P(t) = \sum_{j_1 \in [n]_0, j_2 \in [nW^*]_0} q_{j_1, j_2} t^{j_1(nW^*+1)+j_2}.$$

First, observe that j_1 and j_2 form a base $nW^* + 1$ representation of the degree of the corresponding monomial. So the coefficient standing by $t^{C(nW^*+1)+W}$ in P is exactly $q_{C,W}$, i.e., the value we seek. Further, it holds

$$P(t) = \sum_{j_1 \in [n]_0, j_2 \in [nW^*]_0} q_{j_1, j_2} (t^{nW^*+1})^{j_1} t^{j_2},$$

so evaluating P at some value $s \in \mathbb{Z}$ modulo a prime number p is equivalent to computing the value $Q(s^{nW^*+1}, s) \bmod p$.

It remains to choose suitable values to apply Theorem 3.5. The degree of P is bounded by $\mathcal{O}(n^2W^*)$. The number of H -homomorphisms of G , and hence each coefficient of P as well, is bounded by $|V(H)|^n$. Since $|V(H)|$ is a problem-specific constant, there is a value n' of magnitude $\mathcal{O}(n^2W^*)$ satisfying the prerequisites of Theorem 3.5. Then for a prime number $p \leq 2n' + 2$, any value from \mathbb{F}_p is $\mathcal{O}(\log n + \log W^*)$ bits long. Now to compute the value $Q(s^{nW^*+1}, s) \bmod p$ for some $s \in \mathbb{F}_p$, we proceed similarly to Section 3.1: during the recursion, instead of storing all coefficients of the polynomials, as a partial result we only store the current result of the evaluation at $x = s^{nW^*+1}$ and $y = s$ modulo p .

Let us now summarize the time and space complexity of this evaluation similarly to Section 3.1. The depth of T is d and per node of T , there are at most $4 + |\mathbf{States}|$ recursive calls where $|\mathbf{States}|$ reflects that the transformation from 1_\geq to 2_\geq is carried out for every

element of a guess $S \subseteq \mathbf{States}$ (recall the tables $T(\cdot, c = 0, \cdot), \dots, T(\cdot, c = |\alpha^{-1}(2_{\geq})|, \cdot)$ in Section 3.1). Due to $|\mathbf{States}| = k \cdot |V(H)|$, the recursion depth is then $\mathcal{O}(kd)$. The number of possible guesses S as well as reasonable α and β is bounded by $2^{\mathcal{O}(\mathbf{States})} = 2^{\mathcal{O}(k)}$. Also, for a node a and a reasonable β , the auxiliary graph $F^{a, \beta}$ has at most $|\mathbf{States}| = \mathcal{O}(k)$ vertices. Recall that in Section 3.1, at some point of the computation we work with a polynomial using a variable z . For this variable, only coefficients at monomials z^i for $i \leq |V(F^{a, \beta})|$ are relevant. Hence, for each query we need to keep only $\mathcal{O}(k)$ coefficients from \mathbb{F}_p and such a coefficient uses $\mathcal{O}(\log n + \log W^*)$ bits. The addition and multiplication of two such coefficients can be done in time $\mathcal{O}(\log n + \log W^*)$. These properties imply that following the argument from Section 3.1 we obtain the running time of $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)} \cdot \log W^*$ and space complexity of $\mathcal{O}(kd) \cdot k \cdot \mathcal{O}(\log n + \log W^*) = \mathcal{O}(k^2 d(\log n + \log W^*))$.

With that, Theorem 3.5 implies that the coefficients of P , and in particular the sought value $q_{C, W}$, can be reconstructed in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)} \cdot \log W^*$ and using $\mathcal{O}(k^2 d(\log n + \log W^*))$ space. This concludes the proof of Theorem 3.6.

We remark that the result of Theorem 3.6 can be combined with the Cut&Count technique of Cygan et al. [15] in order to incorporate also connectivity constraints to LIST H -HOMOMORPHISM and solve problems like CONNECTED VERTEX COVER and CONNECTED ODD CYCLE TRANSVERSAL. In essence Cut&Count provides a randomized reduction from LIST H -HOMOMORPHISM with connectivity constraints to #-LIST H' -HOMOMORPHISM for a new pattern graph H' with at most twice as many vertices as H . Since in the reduction only the parity of the number of solutions is preserved, in Cut&Count one typically uses the Isolation Lemma [35] to sample a weight function so that with high probability, there is exactly one (and thus, an odd number) solution of minimum possible weight; then counting the number of solutions mod 2 for all possible weights reveals the existence of a solution. Note here that the algorithm of Theorem 3.6 is already prepared to count weighted solutions. In our setting, the usage of Isolation Lemma necessitates allowing randomization and adds an $\mathcal{O}(n \log n)$ factor to the space complexity for storing the sampled weights. We leave the details to the reader.

3.3 Max-Cut

In the classical MAX CUT problem, we are given a graph G and the task is to output $\max_{X \subseteq V(G)} |E(X, V(G) \setminus X)|$. Towards solving the problem, let us fix a graph G and a (d, k) -tree model $(T, \mathcal{M}, \mathcal{R}, \lambda)$ of G . Recall that for every node a of T , $i \in [k]$ and $X \subseteq V_a$, we denote by $X_a(i)$ the set of vertices in X labeled i at a , i.e., $X \cap \lambda_a^{-1}(i)$. Given a child b of a , we let $V_{ab} = V_b$ and we denote by $V_{ab}(i)$ the set of vertices in V_b labeled i at a , i.e., $V_b \cap V_a(i)$. By $X_{ab}(i)$ we denote the set $X \cap V_{ab}(i)$. Given $c \in \{a, ab\}$, we define the c -signature of $X \subseteq V_c$ — denoted by $\text{sig}_c(X)$ — as the vector $(|X_c(1)|, |X_c(2)|, \dots, |X_c(k)|)$. We let $\mathcal{S}(c)$ be the set of c -signatures of all the subsets of V_c , i.e., $\mathcal{S}(c) := \{\text{sig}_c(X) : X \subseteq V_c\}$. Observe that $|\mathcal{S}(c)| \in n^{\mathcal{O}(k)}$ holds. Also, for the children b_1, \dots, b_t of a , we define $\mathcal{S}(ab_1, \dots, ab_t)$ as the set of all tuples (s^1, \dots, s^t) with $s^i \in \mathcal{S}(ab_i)$ for each $i \in [t]$. Given $s \in \mathcal{S}(c)$, we define $f_c(s)$ as the maximum of $|E(X, V_c \setminus X)|$ over all the subsets $X \subseteq V_c$ with c -signature s . To solve MAX CUT on G , it suffices to compute $\max_{s \in \mathcal{S}(r)} f_r(s)$ where r is the root of T .

Let b be a child of a . We start explaining how to compute $f_{ab}(s)$ by making at most $n^{\mathcal{O}(k)}$ calls to the function f_b . Given $s' \in \mathcal{S}(b)$, we define $\rho_{ab}(s')$ as the vector $s = (s_1, \dots, s_k) \in \mathcal{S}(ab)$ such that, for each $i \in [k]$, we have $s_i = \sum_{j \in \rho_{ab}^{-1}(i)} s'_j$. Observe that for every $X \subseteq V_b$, we have $\text{sig}_{ab}(X) = \rho_{ab}(\text{sig}_b(X))$. Consequently, for every $s \in \mathcal{S}(ab)$, $f_{ab}(s)$ is the maximum of $f_b(s')$ over the b -signatures $s' \in \mathcal{S}(b)$ such that $\rho_{ab}(s') = s$. It follows that we can

compute $f_{ab}(s)$ with at most $n^{\mathcal{O}(k)}$ calls to the function f_b .

► **Observation 3.7.** *Given a node a of T with a child b and $s \in \mathcal{S}(ab)$, we can compute f_{ab} in space $\mathcal{O}(k \log(n))$ and time $n^{\mathcal{O}(k)}$ with $n^{\mathcal{O}(k)}$ oracle access to the function f_b .*

In order to simplify forthcoming statements, we fix a node a of T with children b_1, \dots, b_t . Now, we explain how to compute $f_a(s)$ by making at most $n^{\mathcal{O}(k)}$ calls to the functions $f_{ab_1}, \dots, f_{ab_t}$. The first step is to express $f_a(s)$ in terms of $f_{ab_1}, \dots, f_{ab_t}$. We first describe $|E(X, V_a \setminus X)|$ in terms of $|E(X \cap V_{b_i}, V_{b_i} \setminus X)|$. We denote by $E(V_{b_1}, \dots, V_{b_t})$ the set of edges of $G[V_a]$ whose endpoints lie in different V_{b_i} 's, i.e. $E(G[V_{b_1}, \dots, V_{b_t}]) \setminus (E(G[V_{b_1}] \cup \dots \cup E(G[V_{b_t}])))$. Given $X \subseteq V_a$, we denote by $E_a(X)$ the intersection of $E(X, V_a \setminus X)$ and $E(V_{b_1}, \dots, V_{b_t})$. In simple words, $E_a(X)$ is the set of all cut-edges (i.e., between X and $V_a \setminus X$) running between distinct children of a . For $i, j \in [k]$, we denote by $E_a(X, i, j)$ the subset of $E_a(X)$ consisting of the edges whose endpoints are labeled i and j . We capture the size of $E_a(X, i, j)$ with the following notion. For every $c \in \{a, ab_1, \dots, ab_t\}$, $s \in \mathcal{S}(c)$ and $i, j \in [k]$, we define

$$\#\text{pairs}_c(s, i, j) := \begin{cases} s_i \cdot (|V_c(j)| - s_j) + s_j \cdot (|V_c(i)| - s_i) & \text{if } i \neq j, \\ s_i \cdot (|V_c(i)| - s_i) & \text{otherwise.} \end{cases}$$

It is not hard to check that, for every subset $X \subseteq V_a$ with a -signature s , $\#\text{pairs}_a(s, i, j)$ is the size of $\text{pairs}_a(X, i, j)$ being the set of pairs of distinct vertices in V_a labeled i and j at a such that exactly one of them is in X . Observe that when $M_a[i, j] = 1$, then $|E_a(X, i, j)|$ is the number of pairs in $\text{pairs}_a(X, i, j)$ whose endpoints belong to different sets among V_{b_1}, \dots, V_{b_t} . Moreover, given a child b of a , the number of pairs in $\text{pairs}_a(X, i, j)$ whose both endpoints belong to V_b is exactly $\#\text{pairs}_{ab}(\text{sig}_{ab}(X), i, j)$. Thus when $M_a[i, j] = 1$, we have

$$|E_a(X, i, j)| = \#\text{pairs}_a(\text{sig}_a(X), i, j) - \sum_{i \in [t]} \#\text{pairs}_{ab_i}(\text{sig}_{ab_i}(X), i, j) . \quad (7)$$

We capture the size of $E_a(X)$ with the following notion. For every $c \in \{a, ab_1, \dots, ab_t\}$, $s \in \mathcal{S}(c)$ and $(k \times k)$ -matrix M , we define

$$m_c(s, M) := \sum_{\substack{i, j \in [k], i \leq j \\ M[i, j] = 1}} \#\text{pairs}_c(s, i, j).$$

Note that $|E_a(X)| = \sum_{i, j \in [k]: i \leq j, M_a[i, j] = 1} |E_a(X, i, j)|$. Hence, by Equation 7, we deduce that $|E_a(X)| = m_a(\text{sig}_a(X), M_a) - \sum_{i \in [t]} m_{ab_i}(\text{sig}_{ab_i}(X), M_a)$. Since $E(X, V_a \setminus X)$ is the disjoint union of $E_a(X)$ and the sets $E(X \cap V_{b_1}, V_{b_1} \setminus X), \dots, E(X \cap V_{b_t}, V_{b_t} \setminus X)$, we deduce:

► **Observation 3.8.** *For every $X \subseteq V_a$ we have*

$$|E(X, V_a \setminus X)| = m_a(\text{sig}_a(X), M_a) + \sum_{i=1}^t (|E(X \cap V_{b_i}, V_{b_i} \setminus X)| - m_{ab_i}(\text{sig}_{ab_i}(X), M_a)) .$$

We are ready to express $f_a(s)$ in terms of $f_{ab_1}, \dots, f_{ab_t}$ and $m_a, m_{ab_1}, \dots, m_{ab_t}$.

► **Lemma 3.9.** *For every $s \in \mathcal{S}(a)$, we have*

$$f_a(s) = m_a(s, M_a) + \max_{\substack{(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t) \\ s = s^1 + \dots + s^t}} \left(\sum_{i=1}^t (f_{ab_i}(s^i) - m_{ab_i}(s^i, M_a)) \right) .$$

Proof. Let $s \in \mathcal{S}(a)$. By Observation 3.8 we know that

$$f_a(s) = \max_{\substack{X \subseteq V_a \\ \text{sig}_a(X)=s}} |E(X, V_a \setminus X)| = m_a(s, M_a) + \max_{\substack{X \subseteq V_a \\ \text{sig}_a(X)=s}} \left(\sum_{i=1}^t |E(X_i, V_{b_i} \setminus X_i)| - m_{ab_i}(\text{sig}_{ab_i}(X_i), M_a) \right)$$

where X_i is a shorthand for $X \cap V_{b_i}$. Observe that for every $X \subseteq V_a$, we have $\text{sig}_a(X) = s$ iff $s = \sum_{i=1}^t \text{sig}_{ab_i}(X \cap V_{b_i})$. Since $f_{ab_i}(s^i)$ is the maximum $|E(X_i, V_{b_i} \setminus X_i)|$ over all $X_i \subseteq V_{b_i}$ with ab_i -signature s^i while $m_{ab_i}(\text{sig}_{ab_i}(X_i), M_a)$ only depends on s^i and not on the concrete choice of X_i , we conclude that $f_a(s)$ equals $m_a(s, M_a)$ plus

$$\max_{\substack{(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t) \\ s = s^1 + \dots + s^t}} \left(\sum_{i=1}^t f_{ab_i}(s^i) - m_{ab_i}(s^i, M_a) \right).$$

◀

To compute $f_a(s)$ we use a twist of Kane's algorithm [36] for solving the k -DIMENSIONAL UNARY SUBSET SUM in Logspace. The twist relies on using a polynomial, slightly different from the original work of Kane [36], defined in the following lemma.

Given a vector $s = (s_1, \dots, s_k) \in \mathbb{Z}^k$ and $B \in \mathbb{Z}$, we denote by $s|B$ the vector (s_1, \dots, s_k, B) . We denote by C the number $2n^2 + 1$ and, given a vector $s' \in \mathbb{Z}^{k+1}$, we denote by $C(s')$ the sum $\sum_{i \in [k+1]} C^{i-1} s'_i$.

► **Lemma C.** *Let $s \in \mathcal{S}(a)$ and $B \in [|E(G[V_a])|]$. Let $A(s, B)$ be the number of tuples $(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t)$ such that $s = s^1 + \dots + s^t$ and*

$$B - m_a(s, M_a) = \sum_{j=1}^t f_{ab_j}(s^j) - m_{ab_j}(s^j, M_a).$$

For every prime number $p > C^{k+1} + 1$, we have $-A(s, B) \equiv P_{a,s}(B, p) \pmod{p}$ where

$$P_{a,s}(B, p) := \sum_{x=1}^{p-1} x^{C(s|B - m_a(s, M_a))} \left(\prod_{j=1}^t \left(\sum_{s^j \in \mathcal{S}(ab_j)} x^{-C(s^j|f_{ab_j}(s^j) - m_{ab_j}(s^j, M_a))} \right) \right).$$

Proof. First, note that

$$x^{C(s|B - m_a(s, M_a))} \left(\prod_{j=1}^t \left(\sum_{s^j \in \mathcal{S}(ab_j)} x^{-C(s^j|f_{ab_j}(s^j) - m_{ab_j}(s^j, M_a))} \right) \right) = \sum_{s^1, \dots, s^t \in \mathcal{S}(b_1, \dots, b_t)} x^{\alpha(s^1, \dots, s^t)} \quad (8)$$

where

$$\alpha(s^1, \dots, s^t) = C(s|B - m_a(s, M_a)) - \sum_{j=1}^t (C(s^j|f_{ab_j}(s^j) - m_{ab_j}(s^j, M_a))).$$

As in [36], the idea of this proof is to change the order of summation, show that the terms where $\alpha(s^1, \dots, s^t) \neq 0$ cancel out, and prove that the sum of the terms where $\alpha(s^1, \dots, s^t) = 0$ is $-A(s, B)$. The latter is implied by the following claim.

► **Claim D.** For every $(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t)$, the absolute value of $\alpha(s^1, \dots, s^t)$ is at most C^{k+1} . Moreover, $\alpha(s^1, \dots, s^t) = 0$ iff $s = s^1 + \dots + s^t$ and $B - m_a(s, M_a) = \sum_{i=1}^t f_{ab_i}(s^i) - m_{ab_i}(s^i, M_a)$.

Proof. By definition of $C(\cdot)$, we have

$$\alpha(s^1, \dots, s^t) = \left(\sum_{i=1}^k C^{i-1} \left(s_i - \sum_{j=1}^t s_i^j \right) \right) + C^{k+1} \left(B - m_a(s, M_a) - \sum_{j=1}^t (f_{ab_j}(s^j) - m_{ab_j}(s^j, M_a)) \right).$$

I.e.,

$$\alpha(s^1, \dots, s^t) = \sum_{i=1}^{k+1} C^{i-1} e_i$$

with

$$e_i = \begin{cases} s_i - \sum_{j=1}^t s_i^j & \text{if } 1 \leq i \leq k \\ B - m_a(s, M_a) - \sum_{j=1}^t (f_{ab_j}(s^j) - m_{ab_j}(s^j, M_a)) & \text{if } i = k+1 \end{cases}.$$

We claim that the absolute value of each e_i is at most $C-1$. For every $i \in [k]$, by definition, s_i and $\sum_{j=1}^t s_i^j$ are at least 0 and at most $|V_a(i)| \leq n$. Hence, for each $i \in [k]$ the absolute value of e_i is at most $n < C$. Both B and $\sum_{j=1}^t f_{ab_j}(s^j)$ are upper bounded by $|E(G[V_a])| \leq n^2$. Moreover, from the definition of the functions $m_a, m_{ab_1}, \dots, m_{ab_t}$, we deduce that both $m_a(s, M_a)$ and $\sum_{j=1}^t m_{ab_j}(s^j, M_a)$ are upper bounded by $|V_a|^2 \leq n^2$. It follows that the absolute value of e_{k+1} is at most $2n^2 < C$. Thus, the absolute value of $\alpha(s^1, \dots, s^t)$ is at most $\sum_{i=1}^{k+1} C^{i-1} e_i \leq \sum_{i=1}^{k+1} C^{i-1} (C-1) = C^{k+1} - 1$.

It remains to prove that $\alpha(s^1, \dots, s^t) = 0$ iff $e_j = 0$ for every $j \in [k+1]$. One direction is trivial. For the other direction, observe that if $e_{k+1} \neq 0$, then the absolute value of $C^k e_{k+1}$ is at least C^k . But the absolute value of $\alpha(s^1, \dots, s^t) - C^k e_{k+1} = \sum_{i=1}^k C^{i-1} e_i$ is at most $\sum_{i=1}^k C^{i-1} (C-1) = C^k - 1$. Hence, if $e_{k+1} \neq 0$, then $\alpha(s^1, \dots, s^t) \neq 0$. By induction, it follows that $\alpha(s^1, \dots, s^t) = 0$ is equivalent to $e_i = 0$ for every $i \in [k+1]$. \triangleleft

By using Equation (8) on $P_{a,s}(B, p)$ and interchanging the sums, we deduce that

$$P_{a,s}(B, p) = \sum_{s^1, \dots, s^t \in \mathcal{S}(b_1, \dots, b_t)} \left(\sum_{x=1}^{p-1} x^{\alpha(s^1, \dots, s^t)} \right).$$

It was proven in the proof of Lemma 1 in [36] that

$$\sum_{x=1}^{p-1} x^\ell \pmod{p} = \begin{cases} -1 & \text{if } \ell \equiv 0 \pmod{p-1} \\ 0 & \text{otherwise.} \end{cases}$$

We infer from the above formula that

$$P_{a,s}(B, p) \pmod{p} = \sum_{\substack{s^1, \dots, s^t \in \mathcal{S}(ab_1, \dots, ab_t) \\ \alpha(s^1, \dots, s^t) \equiv 0 \pmod{p-1}}} -1 \pmod{p}.$$

Observe that, for every $(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t)$, we have $\alpha(s^1, \dots, s^t) \equiv 0 \pmod{p-1}$ iff $\alpha(s^1, \dots, s^t) = 0$ because $C^{k+1} < p-1$ and the absolute value of $\alpha(s^1, \dots, s^t)$ is at most C^{k+1} by Claim D. From the equivalence given by Claim D, we deduce that there are $A(s, B)$ tuples $(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t)$ such that $\alpha(s^1, \dots, s^t) = 0$, i.e.,

$$P_{a,s}(B, p) \pmod{p} = -A(s, B) \pmod{p}.$$

\blacktriangleleft

■ **Algorithm 1** Algorithm for computing $f_x(s)$.

Input: A internal node a of T and $s \in \mathcal{S}(a)$.
Output: $f_a(s)$
1 for $B = E(G[V_a]) $ to 0 do
2 $c := 0$
3 $p := \text{NextPrime}(C^{k+1})$
4 while $c \leq nk \log(n)$ do
5 if $P_{a,s}(B, p) \not\equiv 0 \pmod{p}$ then return B
6 $c := c + \lfloor \log(p) \rfloor$
7 $p := \text{NextPrime}(p)$

With this, we can prove Theorem 1.2 via Algorithm 1. As a subroutine, we use the function $\text{NextPrime}(p)$, which computes the smallest prime larger than p .

► **Lemma E.** *Let $s \in \mathcal{S}(a)$. Algorithm 1 computes $f_a(s)$ in space $\mathcal{O}(k \log(n))$ and time $n^{\mathcal{O}(k)}$ with $n^{\mathcal{O}(k)}$ oracle access to the functions $f_{ab_1}, \dots, f_{ab_t}$.*

Proof. The correctness of Algorithm 1 follows from the following claims. Let B be an integer between 0 and $|E(G[V_a])|$, and let $A(s, B)$ be the integer defined in Lemma C.

► **Claim F.** If the algorithm returns B , then $A(s, B) \neq 0$.

Proof. Suppose there exists a prime number $p > C^{k+1}$ such that $P_{a,s}(B, p) \not\equiv 0 \pmod{p}$. As $P_{a,s}(B, p) \equiv A(s, B) \pmod{p}$ by Lemma C, we have $A(s, B) \neq 0$ and thus there exists $(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t)$ such that $s = s^1 + \dots + s^t$ and $B - m_a(s, M_a) = \sum_{i=1}^t f_{ab_i}(s^i) - m_{ab_i}(s^i, M_a)$. From Observation 3.8, we deduce that there exists $X \subseteq V_x$ such that $\text{sig}(X) = s^1 + \dots + s^t = s$ and $|E(X, V_x \setminus X)| = B$. \triangleleft

► **Claim G.** If $P_{a,s}(B, p) \equiv 0 \pmod{p}$ for every value taken by the variable p , then $A(s, B) = 0$.

Proof. Let d be the product of the values taken by p . Then d is a product of distinct primes p such that $P_{a,s}(B, p) \equiv 0 \pmod{p}$. By Lemma C, we have $P_{a,s}(B, p) \equiv A(s, B) \pmod{p}$ for every prime $p > C^{k+1}$. Therefore, $A(s, B)$ is a multiple of d . Observe that $d > 2^c$ and $c > nk \log(n)$. Hence, we have $d > n^{nk}$. Since $A(s, B)$ corresponds to the number of tuples $(s^1, \dots, s^t) \in \mathcal{S}(ab_1, \dots, ab_t)$ that satisfy some properties, we have $A(s, B) \leq \prod_{i=1}^t |\mathcal{S}(ab_i)| \leq n^{nk}$. As d divides $A(s, B)$ and $d > A(s, B)$, we conclude that $A(s, B) = 0$. \triangleleft

From Claims F and G, we infer that Algorithm 1 returns B where B is the maximum between 0 and $|E(G[V_a])|$ such that $A(s, B) \neq 0$. By definition of $A(s, B)$ and Lemma 3.9, we conclude that $f_a(s) = B$.

Complexity. We adapt the arguments used in [36] to prove the complexity of our algorithm.

- First, the variable p is never more than $n^{\mathcal{O}(k)}$. Indeed, standard facts about prime numbers imply that there are $nk \log(n)$ prime numbers between C^{k+1} and $(C^{k+1} + nk \log(n))^{\mathcal{O}(1)} = n^{\mathcal{O}(k)}$. Each of these primes causes c to increase by at least 1. Thus, each value of p can be encoded with $\mathcal{O}(k \log(n))$ bits.
- Secondly, observe that we can compute $P_{a,s}(p, B) \pmod{p}$ in space $\mathcal{O}(k \log(n))$. Recall that

$$P_{a,s}(B, p) := \sum_{x=1}^{p-1} x^{C(s|B - m_a(s, M_a))} \left(\prod_{i=1}^t \left(\sum_{s^i \in \mathcal{S}(b_i)} x^{-C(s^i | f_{ab_i}(s^i) - m_{ab_i}(s^i, M_a))} \right) \right).$$

To compute $P_{a,s}(B,p)$, it is sufficient to keep track of the current value of x , the current running total (modulo p) and enough information to compute the next term, i.e. $x^{C(s|B-m_a(s,M_a))}$ or $x^{-C(s^i|f_{ab_i}(s^i)-m_{ab_i}(s^i,M_a))}$. For that, we need only the current values of i (at most $\log n$ bits) and s^i (at most $k \log n$ bits) and the current running total to compute $C(s|B-m_a(s,M_a))$ (or $C(s^i|f_{ab_i}(s^i)-m_{ab_i}(s^i,M_a))$ modulo p .

- Finally, primality testing of numbers between C^{k+1} and $n^{\mathcal{O}(k)}$ can be done in space $\mathcal{O}(k \log(n))$ via $n^{\mathcal{O}(k)}$ divisions, and thus each call to `NextPrime(\cdot)` can be computed in $n^{\mathcal{O}(k)}$ time and $\mathcal{O}(k \log(n))$ space. ◀

We are now ready to prove that one can solve MAX-CUT in time $n^{\mathcal{O}(dk)}$ using $\mathcal{O}(dk \log(n))$ space.

► **Theorem 1.2.** *There is an algorithm which takes as input an n -vertex graph G along with a (d,k) -tree model of G , runs in time $n^{\mathcal{O}(dk)}$ and uses at most $\mathcal{O}(dk \log n)$ space, and solves the MAX CUT problem in G .*

Proof. Given r the root of T , we solve MAX-CUT by computing $\max_{s \in \mathcal{S}(r)} f_r(s)$. For every internal node a of T with children b_1, \dots, b_t , we use Algorithm 1 to compute each call of f_a from calls to $f_{ab_1}, \dots, f_{ab_t}$. For every internal node a with child b , we use Observation 3.7 to compute each call of f_{ab} from calls to f_b . Finally, for every leaf ℓ of T , we simply have $f_\ell(s) = 0$ for every $s \in \mathcal{S}(\ell)$ because V_ℓ is a singleton.

First, we prove the running time. By Lemma E, for each node a with children b_1, \dots, b_t and $s \in \mathcal{S}(a)$, we compute $f_a(s)$ by calling at most $n^{\mathcal{O}(k)}$ times the functions $f_{ab_1}, \dots, f_{ab_t}$. By Observation 3.7, for each node b with parent a and $s \in \mathcal{S}(ab)$, we compute $f_{ab}(s)$ by calling at most $n^{\mathcal{O}(k)}$ times the function f_b . Consequently, we call each of these functions at most $n^{\mathcal{O}(dk)}$ times in total. Since T has $\mathcal{O}(n)$ nodes, we conclude that computing $\max_{s \in \mathcal{S}(r)} f_r(s)$ this way takes $n^{\mathcal{O}(dk)}$ time.

Finally, observe that the stack storing the calls to these functions is of size at most $\mathcal{O}(d)$. Our algorithm solves MAX CUT in space $\mathcal{O}(dk \log(n))$. ◀

3.4 Dominating Set

In this section we prove Theorem 1.3, which we recall for convenience.

► **Theorem 1.3.** *There is a randomized algorithm which takes as input an n -vertex graph G along with a (d,k) -tree model of G , runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and uses at most $\mathcal{O}(dk^2 \log n + n \log n)$ space, and reports the minimum size of a dominating set in G that is correct with probability at least $1/2$.*

The remainder of this section is devoted to the proof of Theorem 1.3. Note that DOMINATING SET cannot be directly stated in terms of H -homomorphisms for roughly the following reason. For H -homomorphisms, the constraints are *universal*: every neighbor of a vertex with a certain state must have one of allowed states. For DOMINATING SET, there is an *existential* constraint: a vertex in state “dominated” must have at least one neighbor in the dominating set. Also, the state of a vertex might change from “undominated” to “dominated” during the algorithm. The techniques we used for H -homomorphisms cannot capture such properties.

The problem occurs for other parameters as well. One approach that circumvents the issue is informally called *inclusion-exclusion branching*, and was used by Pilipczuk and Wrochna [45] in the context of DOMINATING SET on graphs of low treedepth. Their dynamic programming uses the states *Taken* (i.e., in a dominating set), *Allowed* (i.e., possibly

dominated), and *Forbidden* (i.e., not dominated). These states reflect that we are interested in vertex partitions into three groups such that there are no edges between *Taken* vertices and *Forbidden* vertices; these are constraints that can be modelled using H -homomorphisms for a three-vertex pattern graph H . Crucially, for a single vertex v , if we fix the states of the remaining vertices, the number of partitions in which v is dominated is given by the number of partitions where v is possibly dominated minus the number of partitions where it is not dominated, i.e., informally “*Dominated* = *Allowed* - *Forbidden*”. We will come back to this state transformation later to provide more details. We also remark that the transformed formulation of dynamic programming is exactly what one gets by applying the zeta-transform to the standard dynamic programming for DOMINATING SET.

For technical reasons explained later, our algorithm uses the classic Isolation Lemma:

► **Theorem 3.10** (Isolation lemma, [39]). *Let $\mathcal{F} \subseteq 2^{[n]}$ be a non-empty set family over the universe $[n]$. For each $i \in [n]$, choose a weight $\omega(i) \in [2n]$ uniformly and independently at random. Then with probability at least $1/2$ there exists a unique set of minimum weight in \mathcal{F} .*

Consequently, we pick a weight function ω that assigns every vertex a weight from $1, \dots, 2n$ uniformly and independently at random. Storing ω takes $\mathcal{O}(n \log n)$ space. By Theorem 3.10, with probability at least $1/2$ among dominating sets with the smallest possible cardinality there will be a unique one of minimum possible weight.

To implement the above idea, we let the graph H have vertex set $\{\mathbf{T}, \mathbf{A}, \mathbf{F}\}$ standing for *Taken*, *Allowed*, and *Forbidden*. This graph H has a loop at each vertex as well as the edges \mathbf{TA} and \mathbf{AF} . Further, let $R := \{\mathbf{T}\}$. Following our approach for H -homomorphisms, for every set $S \subseteq \mathbf{States}$ with $\mathbf{States} := \{(\mathbf{T}, 1), (\mathbf{F}, 1), \dots, (\mathbf{T}, k), (\mathbf{F}, k)\}$, every cardinality $c \in [n]_0$, and every weight $w \in [2n^2]_0$, in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2 \log n)$ (recall that here for the maximum weight W^* we have $W^* \leq 2n$) we can compute the value $a_{S,c,w}$ being the number of ordered partitions $(\widehat{T}, \widehat{F}, \widehat{A})$ of $V(G)$ satisfying the following properties:

1. there are no edges between \widehat{T} and \widehat{F} ;
2. $|\widehat{T}| = c$ and $\omega(\widehat{T}) = w$; and
3. for every $i \in [k]$ and $I \in \{T, F\}$, we have $(\mathbf{I}, i) \in S$ iff $\widehat{T} \cap V(i) \neq \emptyset$.

Note that we do not care whether vertices of some label i are mapped to A or not.

After that, we aim to obtain the number of dominating sets of cardinality c and weight w from values $a_{S,c,w}$. For this we need to transform the “states” *Allowed* and *Forbidden* into *Dominated*. Above we have explained how this transformation works if we know the state of a single vertex. However, now the set S only captures for every label i , which states occur on the vertices of label i . First, the vertices of this label might be mapped to different vertices of H . And even if we take the partitions where all vertices of label i are possibly dominated and subtract the partitions where all these vertices are not dominated, then we obtain the partitions where *at least one vertex* with label i is dominated. However, our goal is that *all vertices* of label i are dominated. So the *Dominated* = *Allowed* - *Forbidden* equality is not directly applicable here.

Recently, Hegerfeld and Kratsch [34] showed that when working with label sets, this equality is in some sense still true modulo 2. On a high level, they show that if we fix a part \widehat{T} of a partition satisfying the above properties, then any undominated vertex might be put to any of the sides \widehat{A} and \widehat{F} . Thus, if \widehat{T} is not a dominating set of G , then there is an even number of such partitions and they cancel out modulo 2.

Now we follow their ideas to formalize this approach and conclude the construction of the algorithm. For $i \in [k]$ and $S \subseteq \{(\mathbf{T}, 1), (\mathbf{F}, 1), \dots, (\mathbf{T}, i), (\mathbf{F}, i)\}$ we define the value $D_i^w(S)$ as the number of ordered partitions $(\widehat{T}, \widehat{F}, \widehat{X})$ of $V(G)$ with the following properties:

1. there are no edges between \widehat{T} and \widehat{F} ;
2. $|\widehat{T}| = c$ and $\omega(\widehat{T}) = w$;
3. for every $j \in [i]$ and $I \in \{T, F\}$, we have $(I, j) \in S$ iff $\widehat{T} \cap V(j) \neq \emptyset$; and
4. $(V(i+1) \cup \dots \cup V(k)) \setminus \widehat{T}$ is dominated by \widehat{T} .

The following observation is obvious.

► **Claim H.** For every $S \subseteq \mathbf{States}$, we have $D_k^{c,w}(S) = a_{S,c,w}$.

Next, we observe that it suffices to compute values $D_i^{c,w}(S)$ for $i = 0$ and $S = \emptyset$.

► **Claim I.** $D_0^{c,w}(\emptyset)$ is the number of dominating sets of size c and total weight w .

Proof. Consider a partition $(\widehat{T}, \widehat{F}, \widehat{X})$ counted in $D_0^{c,w}(\emptyset)$. Recall that $V(1) \cup \dots \cup V(k) = V(G)$. So the fourth property implies that $V(G) \setminus \widehat{T}$ is dominated by \widehat{T} , i.e., \widehat{T} is a dominating set of G . The first property then implies that \widehat{F} is empty and $\widehat{X} = V(G) \setminus \widehat{T}$. Finally, by definition of $D_0^{c,w}(\emptyset)$, we know that the size of \widehat{T} is c and its weight is w . On the other hand, every dominating set \widehat{T} of cardinality c and weight w defines a partition $(\widehat{T}, \emptyset, V(G) \setminus \widehat{T})$ counted in $D_0^{c,w}(\emptyset)$. ◀

Finally, we prove that modulo 2, $D_i^{c,w}(S)$ can be computed from $D_{i+1}^{c,w}(S)$.

► **Claim J.** For every $i \in [k-1]_0$ and every $S \subseteq \{(\mathbf{T}, 1), (\mathbf{F}, 1), \dots, (\mathbf{T}, i), (\mathbf{F}, i)\}$, it holds that

$$D_i^{c,w}(S) \equiv \sum_{B \subseteq \{(\mathbf{T}, i+1), (\mathbf{F}, i+1)\}} D_{i+1}^{c,w}(S \cup B) \pmod{2}.$$

Proof. We follow the proof idea of Hegerfeld and Kratsch. For $B \subseteq \{(\mathbf{T}, i+1), (\mathbf{F}, i+1)\}$, let $\mathcal{A}_{i+1}(S \cup B)$ be the set of partitions counted in $D_{i+1}^{c,w}(S \cup B)$ (see the definition above). Note that we have $\mathcal{A}_{i+1}(S \cup B_1) \cap \mathcal{A}_{i+1}(S \cup B_2) = \emptyset$ for any $B_1 \neq B_2 \subseteq \{(\mathbf{T}, i+1), (\mathbf{F}, i+1)\}$. So

$$\sum_{B \subseteq \{(\mathbf{T}, i+1), (\mathbf{F}, i+1)\}} D_{i+1}^{c,w}(S \cup B) = \left| \bigcup_{B \subseteq \{(\mathbf{T}, i+1), (\mathbf{F}, i+1)\}} \mathcal{A}_{i+1}(S \cup B) \right|.$$

Let \mathcal{L} be the set of partitions counted in $D_i^{c,w}(S)$ and let $\mathcal{R} = \bigcup_{B \subseteq \{(\mathbf{T}, i+1), (\mathbf{F}, i+1)\}} \mathcal{A}_{i+1}(S \cup B)$. The goal is to prove $|\mathcal{L}| \equiv |\mathcal{R}| \pmod{2}$.

By definition of these values we have $\mathcal{L} \subseteq \mathcal{R}$. We claim that the size of $\mathcal{R} \setminus \mathcal{L}$ is even. To see this, consider some fixed partition $(\widehat{T}, \widehat{F}, \widehat{X}) \in \mathcal{R} \setminus \mathcal{L}$. This is exactly the case if the following properties hold:

1. there are no edges between \widehat{T} and \widehat{F} ;
2. $|\widehat{T}| = c$ and $\omega(\widehat{T}) = w$;
3. for every $j \in [i]$ and $I \in \{T, F\}$, we have $(I, j) \in S$ iff $\widehat{T} \cap V(j) \neq \emptyset$; and
4. the set $(V(i+2) \cup \dots \cup V(k)) \setminus \widehat{T}$ is dominated by \widehat{T} while the set $(V(i+1) \cup \dots \cup V(k)) \setminus \widehat{T}$ is not dominated by \widehat{T} ,

Let $U = V(i+1) \setminus N[\widehat{T}]$. The last property implies that U is non-empty. Also let $X' = \widehat{X} \setminus V(i+1)$ and $F' = \widehat{F} \setminus V(i+1)$. Observe that $N(\widehat{T}) \cap V(i+1) \subseteq \widehat{X}$ due to the first property. We claim that if we fix the first set \widehat{T} of the partition as well as the partition of $V \setminus V(i+1)$ (by fixing X' and F'), then the extensions of (\widehat{T}, F', X') to a partition in $\mathcal{R} \setminus \mathcal{L}$ are exactly the partitions of form

$$(\widehat{T}, F' \cup (U \setminus U'), X' \cup (N(\widehat{T}) \cap V(i+1)) \cup U') \tag{9}$$

for $U' \subseteq U$. So informally speaking, if we fix \widehat{T}, X', F' , every vertex of U can be put to either \widehat{X} or \widehat{F} thus giving rise to an even number $2^{|U|}$ of such extensions.

Now we prove this claim following the idea of Hegerfeld and Kratsch. First, consider a partition of form (9) for an arbitrary $U' \subseteq U$. Since \widehat{T} is fixed and the partition on $V \setminus V(i+1)$ is fixed as well, the last three properties defining $\mathcal{R} \setminus \mathcal{L}$ trivially hold. Next, due to $F' \subseteq \widehat{F}$, there are no edges between \widehat{T} and F' . And since $U \setminus U' \subseteq U$ is not dominated by \widehat{T} , there are no edges between \widehat{T} and $U \setminus U'$ as well, so the first property holds too.

For the other direction, if we consider an extension $(\widehat{T}, \widehat{F}, \widehat{X}) \in \mathcal{R} \setminus \mathcal{L}$ of (\widehat{T}, F', X') , then by the first property we know that $\widehat{F} \cap V(i+1)$ has no edges to \widehat{T} and hence, it is a subset of U .

So, for any fixed (\widehat{T}, F', X') , either there is no extension to a partition from $\mathcal{R} \setminus \mathcal{L}$ at all or there are $2^{|U|}$ of them where U is a non-empty set. So the size of $\mathcal{R} \setminus \mathcal{L}$ is even and this concludes the proof. \blacktriangleleft

The application of Claim J for $i = 0, \dots, k-1$ implies

$$\begin{aligned} D_0^{c,w}(\emptyset) &\equiv \sum_{B_1 \subseteq \{(\mathbf{T},1), (\mathbf{F},1)\}} D_1^{c,w}(B_1) \equiv \\ &\sum_{B_1 \subseteq \{(\mathbf{T},1), (\mathbf{F},1)\}} \sum_{B_2 \subseteq \{(\mathbf{T},2), (\mathbf{F},2)\}} D_2^{c,w}(B_1 \cup B_2) \equiv \\ &\dots \\ &\sum_{B_1 \subseteq \{(\mathbf{T},1), (\mathbf{F},1)\}} \sum_{B_2 \subseteq \{(\mathbf{T},2), (\mathbf{F},2)\}} \dots \sum_{B_k \subseteq \{(\mathbf{T},k), (\mathbf{F},k)\}} D_k^{c,w}(B_1 \cup B_2 \dots \cup B_k) \equiv \\ &\sum_{S \subseteq \{(\mathbf{T},1), (\mathbf{F},1), \dots, (\mathbf{T},k), (\mathbf{F},k)\}} D_k^w(S) \pmod{2}. \end{aligned}$$

By Claim H, the parity of the number of dominating sets of size c and weight w can be expressed as

$$D_0^{c,w}(\emptyset) \equiv \sum_{S \subseteq \{(\mathbf{T},1), (\mathbf{F},1), \dots, (\mathbf{T},k), (\mathbf{F},k)\}} a_{S,c,w} \pmod{2}.$$

Recall that every $a_{S,c,w}$ can be computed in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2 \log n)$, hence this is also the case for their sum modulo 2. We compute the value $D_0^{c,w}(\emptyset)$ for all cardinalities $c \in [n]_0$ and all weights $w \in [2n^2]_0$ and output the smallest value c such that for some w the value $D_0^{c,w}(\emptyset)$ is non-zero (or it outputs n if no such value exists).

Now we argue the correctness of our algorithm. Let C denote the size of the smallest dominating set of G . First, this implies that for any $c < C$ and any $w \in [2n^2]_0$, the value $D_0^{c,w}(\emptyset)$ is zero. And second, Isolation Lemma (Theorem 3.10) implies that with probability at least $1/2$, the weight function ω isolates the family of dominating sets of G of size C , i.e., there exists a weight W such that there is exactly one dominating set of size C and weight W , and therefore $D_0^{c,w}(\emptyset) = 1$. In this case, the algorithm outputs C . So with probability at least $1/2$ our algorithm outputs the minimum size of a dominating set of G .

The iteration over all c and w increases the space complexity by an additive $\mathcal{O}(\log n)$ and it increases the running time by a factor of $\mathcal{O}(n^2)$. Recall that in the beginning, to sample the weight function we have used space $\mathcal{O}(n \log n)$. So all in all, the running time of the algorithm is $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and the space complexity is $\mathcal{O}(dk^2 \log n + n \log n)$. This concludes the proof of Theorem 1.3.

Note that in our algorithm, the only reason for super-logarithmic dependency on n in the space complexity is the need to sample and store a weight function in order to isolate a minimum-weight dominating set. We conjecture that this can be avoided and ask:

► **Question 3.11.** *Is there an algorithm for DOMINATING SET of n -vertex graphs provided with a (d, k) -tree-model that runs in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and uses $(d + k)^{\mathcal{O}(1)} \log n$ space?*

4 The Lower Bound

In this section, we prove Theorem 1.4. This lower bound is based on a reasonable conjecture on the complexity of the problem LONGEST COMMON SUBSEQUENCE (LCS).

An instance of LCS is a tuple $(N, t, \Sigma, s_1, \dots, s_r)$ where N and t are positive integers, Σ is an alphabet and s_1, \dots, s_r are r strings over Σ of length N . The goal is to decide whether there exists a string $s \in \Sigma^t$ of length t appearing as a subsequence in each s_i . There is a standard dynamic programming algorithm for LCS that has time and space complexity $\mathcal{O}(N^r)$. From the point of view of parameterized complexity, LCS is $W[p]$ -hard for every level p when parameterized by r [7]. It remains $W[1]$ -hard when the size of the alphabet is constant [44], and it is $W[1]$ -complete when parameterized by $r + t$ [32]. Abboud et al. [1] proved that the existence of an algorithm with running time $\mathcal{O}(N^{r-\varepsilon})$ for any $\varepsilon > 0$ would contradict the Strong Exponential-Time Hypothesis. As observed by Elberfeld et al. [21], LCS parameterized by r is complete for the class XNLP: parameterized problems solvable by a nondeterministic Turing machine using $f(k) \cdot n^{\mathcal{O}(1)}$ time and $f(k) \log n$ space, for a computable function f . See also [7, 8, 9, 10, 11] for further research on XNLP and related complexity classes. The only known progress on the space complexity is due to Barsky et al. with an algorithm running in $\mathcal{O}(N^{r-1})$ space [4]. This motivated Pilipczuk and Wrochna to formulate the following conjecture [45].

► **Conjecture 4.1** ([45]). *There is no algorithm that solves the LCS problem in time $M^{f(r)}$ and using $f(r)M^{\mathcal{O}(1)}$ space for any computable function f , where M is the total bitsize of the instance and r is the number of input strings.*

Note that in particular, the existence of an algorithm with time and space complexity as in Conjecture 4.1 implies the existence of such algorithms for all problems in the class XNLP.

Our lower bound is based on the following stronger variant of Conjecture 4.1, in which we additionally assume that the sought substring is short.

► **Conjecture 4.2.** *For any unbounded and computable function δ , Conjecture 4.1 holds even when $t \leq \delta(N)$.*

Thus, we may rephrase Theorem 1.4 as follows.

► **Theorem 4.3.** *Unless Conjecture 4.2 fails, for any unbounded and computable function δ , there is no algorithm that solves the INDEPENDENT SET problem in graphs supplied with (d, k) -tree-models satisfying $d \leq \delta(k)$ that would run in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and use $n^{\mathcal{O}(1)}$ space.*

The remainder of this section is devoted to the proof of Theorem 4.3. Not surprisingly, we provide a reduction from LCS to INDEPENDENT SET on graphs provided with suitable tree-models.

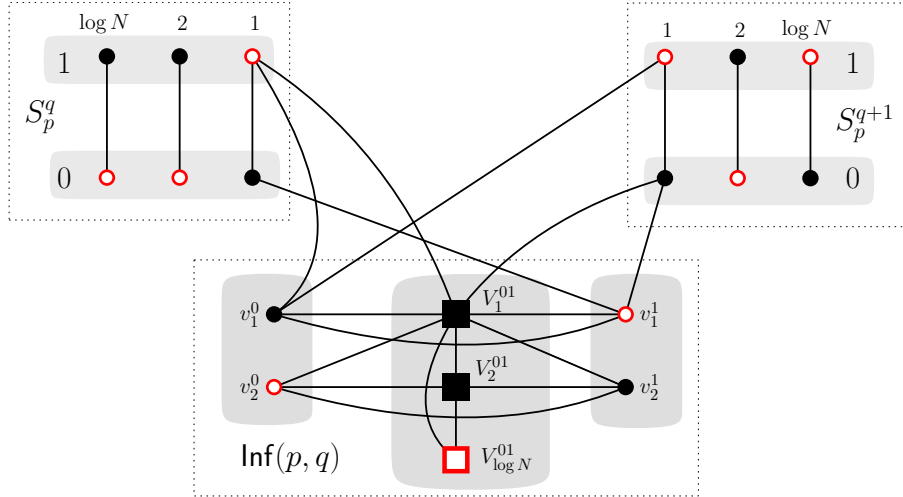
Let $(N, t, \Sigma, s_1, \dots, s_r)$ be an instance of LCS. For the sake of clarity, we assume without loss of generality that N is a power of 2. Indeed, we can always obtain an equivalent instance $(2^{\lceil \log N \rceil}, t + t', \Sigma', s'_1, \dots, s'_r)$ where $t' = 2^{\lceil \log N \rceil} - N$, Σ' is obtained from Σ by adding a new letter \spadesuit and each s'_i is obtained by adding t' times \spadesuit at the end of s_i .

For every $I \in [N]$, we denote the I -th letter of s_p by $s_p[I]$. In the following, we present our reduction from $(N, t, \Sigma, s_1, \dots, s_r)$ to an equivalent instance of INDEPENDENT SET consisting of a graph G with $(r + t + N)^{\mathcal{O}(1)}$ vertices and a (d, k) -tree-model where $d = \mathcal{O}(\log t)$ and $k = \mathcal{O}(r \log N)$. This implies Theorem 4.3 since for every unbounded and computable function δ there exists an unbounded and computable function δ' such that if $t \leq \delta'(N)$, then $d \leq \delta(k)$ (we explain this in more details at the end of this section).

In the intuitions along the construction, we denote by s^* a potential common substring of s_1, \dots, s_r of length t . The main idea is to use matchings to represent the binary encoding of the positions of the letters of s^* in each string.

For every string s_p and $q \in [t]$, we define the **selection gadget** S_p^q which contains, for every $i \in [\log N]$, an edge called the i -edge of S_p^q . One endpoint of this edge is called the 0 -endpoint and the other is called the 1 -endpoint; i.e., a selection gadget induces a matching on $\log N$ edges. This results in the following natural bijection between $[N]$ and the maximal independent sets of S_p^q . For every $I \in [N]$, we denote by $S_p^q|I$ the independent set that contains, for each $i \in [\log N]$, the x -endpoint of the i -edge of S_p^q where x is the value of the i -th bit of the binary representation of $I - 1$ (we consider the first bit to be the most significant one and the $\log N$ -th one the least significant). Then the vertices selected in S_p^q encode the position of the q -th letter of s^* in s_p .

We need to guarantee that the selected positions in the gadgets S_p^1, \dots, S_p^t are coherent, namely, for every $q \in [t]$, the position selected in S_p^q is strictly smaller than the one selected in S_p^{q+1} . For this, we construct an **inferiority gadget** denoted by $\text{Inf}(p, q)$ for every string s_p and every $q \in [t - 1]$. The idea behind it is to ensure that the only possibility for an independent set to contain at least $3 \log N$ vertices from S_p^q, S_p^{q+1} , and their inferiority gadget, is the following: there exist $I < J \in [N]$ such that the independent set contains $S_p^q|I \cup S_p^{q+1}|J$. The maximum solution size in the constructed instance of INDEPENDENT SET—which is the sum of the independence number of each gadget—will guarantee that only such selections are possible.



■ **Figure 1** Example of an inferiority gadget with $\log N = 3$. The squares represent the sets of vertices V_i^{01} . For legibility, among the edges going out of the inferiority gadget, we only represent those incident to v_1^0, v_1^1 and V_1^{01} . The independent set consisting of the white filled vertices has $3 \log N$ vertices, the position selected for S_p^q is 5 and for S_p^{q+1} it is 6.

Figure 1 provides an example of the following construction. The vertex set of $\text{Inf}(p, q)$

consists of the following vertices: for each $i \in [\log N - 1]$, there are two vertices $v_i^{0,p,q}$ and $v_i^{1,p,q}$. Moreover, for each $i \in [\log N]$, there is a set $V_{i,p,q}^{01}$ of $\log N - i + 1$ vertices (we drop p, q from the notation when they are clear from the context). We now describe the edges incident to the inferiority gadget:

- For every $i \in [\log N - 1]$, v_i^0 and v_i^1 are adjacent and for each $x \in \{0, 1\}$, v_i^x is adjacent to the $(1 - x)$ -endpoints of the i -edges from S_p^q and S_p^{q+1} .
- For every $i \in [\log N]$, all the vertices in V_i^{01} are adjacent to (1) the 1-endpoint of the i -edge from S_p^q , (2) the 0-endpoint from the i -edge of S_p^{q+1} , (3) all the vertices v_j^0, v_j^1 for every $j \geq i$ and (4) all the vertices in V_j^{01} for every $j > i$.

On a high level, an inferiority gadget reflects that for values $I < J \in [N]$, if we go from high-order to low-order bits, then the binary encodings of I and J first contains the same bits and then there is an index, where I has a zero-bit and J has a one-bit. If such a difference first occurs at some position $\ell \in [\log N]$, then the corresponding independent set first takes $\ell - 1$ vertices of the form $v_{\ell'}^0$ or $v_{\ell'}^1$ (for $\ell' < \ell$) and then takes $\log N - (\ell - 1)$ vertices from V_ℓ^{01} – this results in $\log N$ vertices taken in the inferiority gadget. The following statement follows from

► **Observation 4.4.** *Let $p \in [r]$ and $q \in [t - 1]$. The independence number of $\text{Inf}(p, q)$ is $\log N$ and for every $I, J \in [N]$, we have $I < J$ iff there exists a set of $\log N$ vertices S from $\text{Inf}(p, q)$ such that the union of S , $S_p^q|I$ and $S_p^{q+1}|J$ induces an independent set.*

Next, we need to ensure that the t positions chosen in s_1, \dots, s_r indeed correspond to a common subsequence, i.e., for every $q \in [t]$, the q -th chosen letter must be the same in every s_1, \dots, s_r . For $p \in [r - 1]$, let \mathcal{M}_p denote the set of all ordered pairs $(I, J) \in [N]^2$ such that the I -th letter of s_p and the J -th of s_{p+1} are identical. For each $p \in [r - 1]$ and $q \in [t]$, we create the *matching gadget* $\text{Match}(p, q)$ as follows:

- For every pair $(I, J) \in \mathcal{M}_p$ and for each $p^* \in \{p, p + 1\}$, we create a copy $M_{p^*}^{p,q,I,J}$ of $S_{p^*}^q$ and for every $\ell \in [\log N]$ and $x \in \{0, 1\}$, we add an edge between the x -endpoint of the ℓ -edge of $S_{p^*}^q$ and the $(1 - x)$ -endpoint of the ℓ -edge of $M_{p^*}^{p,q,I,J}$.
- For every pair $(I, J) \in \mathcal{M}_p$, we add a new vertex $v_{p,I,J}^q$ adjacent to (1) all the vertices from $M_p^{p,q,I,J}$ that are not in $M_p^{p,q,I,J}|I$ and (2) all the vertices from $M_{p+1}^{p,q,I,J}$ that are not in $M_{p+1}^{p,q,I,J}|J$.

Finally, we turn $\{v_{p,I,J}^q : (I, J) \in \mathcal{M}_p\}$ into a clique. Observe that, for each $p^* \in \{p, p + 1\}$, an independent set S contains $(|\mathcal{M}_p| + 1) \log N$ vertices from $S_{p^*}^q$ and its copies $M_{p^*}^{p,q,I,J}$ if and only if there exists a value $I \in [N]$ such that S contains $S_{p^*}^q|I$ and $M_{p^*}^{p,q,I,J}|I$ for each copy. This leads to the following observation.

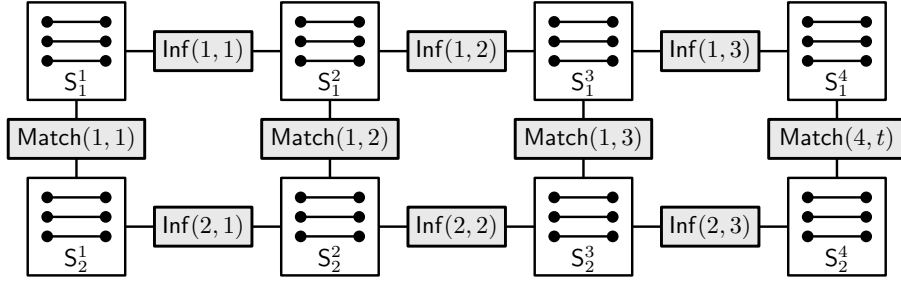
► **Observation 4.5.** *Let $p \in [r - 1]$ and $q \in [t]$. The independence number of $\text{Match}(p, q)$ is $1 + 2 \cdot |\mathcal{M}_p| \cdot \log N$ and for every $I, J \in [N]$, we have $(I, J) \in \mathcal{M}_p$ iff there exists an independent set S of $\text{Match}(p, q)$ with $1 + 2|\mathcal{M}_p| \cdot \log N$ vertices such that the union of S , $S_p^q|I$ and $S_{p+1}^q|J$ is an independent set.*

This concludes the construction of the graph G . See Figure 2 below for an overview.

We prove the correctness of the reduction in the following lemma.

► **Lemma 4.6.** *There exists an integer goal such that G admits an independent set of size at least goal iff the strings s_1, \dots, s_r admit a common subsequence of length t .*

Proof. Let $\text{goal} = (rt + r(t - 1)) \log N + \sum_{p \in [r-1]} t(1 + 2 \cdot |\mathcal{M}_p| \cdot \log N)$.



■ **Figure 2** Overview of the graph G with $\log N = 3$, $r = 2$ and $t = 4$. There are some edges between two gadgets if and only if there are some edges between their vertices in G .

(\Rightarrow) Assume that s_1, \dots, s_r admit a common subsequence s^* of length t . Then, for every string s_p , there exist $I_p^1, \dots, I_p^t \in [N]$ such that $I_p^1 < \dots < I_p^t$ and $s_p[I_p^q] = s^*[q]$ for every $q \in [t]$. We construct an independent set S as follows. For every selection gadget S_p^q , we add $S_p^q|I_p^q$ to S . Note that, at this point, S is an independent set because there is no edge between the selection gadgets S_p^q in G . For every inferiority gadget $\text{Inf}(p, q)$, since $I_p^q < I_{p+1}^{q+1}$, we can use Observation 4.4 and add a set of $\log N$ vertices from $\text{Inf}(p, q)$ to S . Note that S remains an independent set because the added vertices are not adjacent to $S_p^q|I_p^q$ and $S_{p+1}^q|I_{p+1}^{q+1}$ by Observation 4.4 and the only edges going out of $\text{Inf}(p, q)$ are incident to S_p^q and S_{p+1}^{q+1} . At this point, we have $(rt + r(t-1)) \log N$ vertices in S .

Observe that for every $p \in [r-1]$ and $q \in [t]$, we have $s_p[I_p^q] = s^*[q] = s_{p+1}[I_{p+1}^{q+1}]$. Thus, we have $(I_p^q, I_{p+1}^{q+1}) \in \mathcal{M}_p$ and by Observation 4.5, there exists an independent set $S_{p,q}$ of $\text{Match}(p, q)$ with $1 + 2|\mathcal{M}_p| \cdot \log N$ vertices such that the union of $S_{p,q}$, $S_p^q|I_p^q$ and $S_{p+1}^q|I_{p+1}^{q+1}$ is an independent set. We add $S_{p,q}$ to S and note that S remains an independent set since the only edges going out of $\text{Match}(p, q)$ are incident to S_p^q and S_{p+1}^q . As we do this for every $p \in [r-1]$ and $q \in [t]$, the union of the $S_{p,q}$'s contains $\sum_{p \in [r-1]} t(1 + 2|\mathcal{M}_p| \cdot \log N)$ vertices. We conclude that G admits an independent set of size **goal**.

(\Leftarrow) Assume G admits an independent set S of size at least **goal**. The independence number of each selection gadget S_p^q is $\log N$ and, by Observation 4.4, this is also the case for each inferiority gadget $\text{Inf}(p, q)$. Hence, S contains at most $(rt + r(t-1)) \log N$ vertices from selection and inferiority gadgets. By Observation 4.5, the independence number of each matching gadget $\text{Match}(p, q)$ is $1 + 2|\mathcal{M}_p| \cdot \log N$, thus S contains at most $\sum_{p \in [r-1]} t(1 + 2|\mathcal{M}_p| \cdot \log N)$ vertices from the matching gadgets. From the definition of **goal**, we obtain that S contains exactly $\log N$ vertices from each selection and inferiority gadget, and it contains exactly $1 + 2|\mathcal{M}_p| \cdot \log N$ vertices from each matching gadget. We make the following deductions:

- For each $p \in [r]$, there exist $I_p^1, \dots, I_p^t \in [N]$ such that S contains $S_p^q|I_p^q$ for every $q \in [t]$.
- For each $p \in [r]$ and $q \in [t-1]$, the independent set S contains the vertices in $S_p^q|I_p^q$ and $S_{p+1}^q|I_{p+1}^{q+1}$ as well as $\log N$ vertices from $\text{Inf}(p, q)$. Observation 4.4 implies that $I_p^q < I_{p+1}^{q+1}$. Thus, $s_p[I_p^1] \dots s_p[I_p^t]$ is a subsequence of s_p .
- For every $p \in [r-1]$ and $q \in [t]$, the independent set S contains $S_p^q|I_p^q$ and $S_{p+1}^q|I_{p+1}^{q+1}$ as well as $1 + 2|\mathcal{M}_p| \cdot \log N$ vertices from $\text{Match}(p, q)$. We deduce from Observation 4.5 that $(I_p^q, I_{p+1}^{q+1}) \in \mathcal{M}_p$ and consequently, $s_p[I_p^q] = s_{p+1}[I_{p+1}^{q+1}]$. Hence, for every $q \in [t]$, we have $s_1[I_1^q] = \dots = s_r[I_r^q]$.

We conclude that $s_1[I_1^1] \dots s_1[I_1^t]$ is a common subsequence of s_1, \dots, s_r . ◀

The next step is to construct a tree-model of G .

► **Lemma 4.7.** *We can compute in polynomial time a (d, k) -tree-model of G where $d = 2 \log t + 4$ and $k = 14r \log N - 3$.*

Proof. Let $L_S, L_M, L_{\min}, L_{\max}, L_{\text{Inf}}$ and $\{\ell_0\}$ be disjoint subsets of $[k]$ such that each set among $L_S, L_M, L_{\min}, L_{\max}, L_{\text{Inf}}$ has size $2r \log N$ and L_{Inf} has size $6r \log N - 4$. First, we prove that the union of the gadgets associated with a position $q \in [t]$ admits a simple tree-model. For every $q \in [t]$, we denote by G^q the union of the selection gadgets S_p^q with $p \in [r]$ and the matching gadgets $\text{Match}(p, q)$ with $p \in [r - 1]$.

▷ **Claim 4.8.** For every $q \in [t]$, we can construct in polynomial time a $(3, k)$ -tree-model $(T^q, \mathcal{M}^q, \mathcal{R}^q, \lambda^q)$ for G^q .

Proof. Let $q \in [t]$. We create the root a^q of T^q and we attach all the vertices in the selection gadgets S_p^q with $p \in [r]$ as leaves adjacent to a^q . Then, for every $p \in [r - 1]$, we create a node a_p^q adjacent to a^q and for every $(I, J) \in \mathcal{M}_p$, we create a node $a_{p,I,J}^q$ adjacent to a_p^q . For each $(I, J) \in \mathcal{M}_p$, we make $a_{p,I,J}^q$ adjacent to the vertex $v_{p,I,J}^q$ and all the vertices in $M_{p+1}^{p,q,I,J}, M_{p+1}^{p,q,I',J'}$. Note that all the vertices in $\text{Match}(p, q)$ are the leaves of the subtree rooted at a_p^q , and the leaves of T^q are exactly the vertices in G^q . See Figure 3 for an illustration of T^q .

We define λ^q as follows:

- λ^q maps each vertex in S_1^q, \dots, S_r^q to a unique label in L_S .
- For every $(p, i, x) \in [r] \times [\log N] \times \{0, 1\}$, λ^q maps all the x -endpoints of the i -edges from the different copies $M_p^{p',q,I,J}$ of S_p^q to a unique label in L_M .
- We have $\lambda^q(v_{p,I,J}^q) = \ell_0$ for every $p \in [r - 1]$ and $(I, J) \in \mathcal{M}_p$.

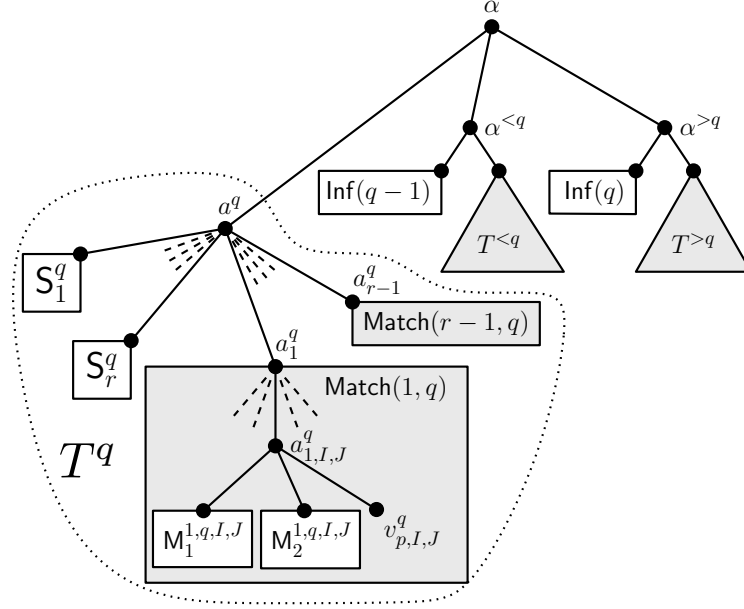
We define $\mathcal{R}^q = \{\rho_{ab} : ab \in E(T^q)\}$ such that ρ_{ab} is the identity function for every $ab \in E(T^q)$. It follows that $\lambda_a^q = \lambda^q$ for every node a of T^q . We finish the construction of the tree-model of G^q by proving that there exists a family of matrices \mathcal{M}^q such that $(T^q, \mathcal{M}^q, \mathcal{R}^q, \lambda^q)$ is a $(3, k)$ -tree-model of G^q . For doing so, we simply prove that the property $\varphi(a, \ell)$ holds for every label $\ell \in [k]$ and internal node a of T^q with children b_1, \dots, b_c , where $\varphi(a, \ell)$ is true if:

- For every $u \in V_{b_i}$ and $v \in V_{b_j}$ with $\lambda_a^q(u) = \lambda_a^q(v) = \ell$, we have $N(u) \cap (V_a \setminus (V_{b_i} \cup V_{b_j})) = N(v) \cap (V_a \setminus (V_{b_i} \cup V_{b_j}))$.

Observe that $\varphi(a, \ell)$ trivially holds when there is at most one vertex labeled ℓ in V_a . Consequently, $\varphi(a_{p,I,J}^q, \ell)$ is true for every node $a_{p,I,J}^q$ and $\ell \in [k]$. Moreover, $\varphi(a, \ell)$ is true for every internal node a and every $\ell \in L_S \cup L_{\min} \cup L_{\max} \cup L_{\text{Inf}}$. Recall that $\lambda_a^q = \lambda^q$ for every node a of T^q . For every pair (u, v) of distinct vertices in $V(G^q)$, if $\lambda^q(u) = \lambda^q(v)$ then either:

- There exists $(p, i, x) \in [r] \times [\log N] \times \{0, 1\}$ such that u and v are the x -endpoints of the i -edges in respectively $M_p^{p^*,q,I,J}$ and $M_{p'}^{p',q,I',J'}$ for some $p^*, p' \in \{p - 1, p\}$, $(I, J) \in \mathcal{M}_{p^*}$ and $(I', J') \in \mathcal{M}_{p'}$. Observe that the parent of u is $a_{p^*,I,J}^q$ and the neighbors of u in $V_{a_{p^*,I,J}^q}$ are all children of $a_{p^*,I,J}^q$. Indeed, the only neighbors of u in $V_{a_{p^*,I,J}^q}$ are the $(1-x)$ -endpoint of the i -edge of $M_p^{p^*,q,I,J}$ and potentially $v_{p^*,I,J}^q$. Symmetrically, v belongs to $V_{a_{p',I',J'}^q}$, its parent is $a_{p',I',J'}^q$ and the only neighbors of v in $V_{a_{p',I',J'}^q}$ are children of $a_{p',I',J'}^q$. Moreover, observe that u and v have both only one neighbor in $V_{a^q} \setminus V_{a_{p^*,I,J}^q}$ which is the $(1-x)$ -endpoint of the i -edge of S_p^q . We deduce that $\varphi(a^q, \ell)$ and $\varphi(a_{p^*,I,J}^q, \ell)$ and $\varphi(a_{p',I',J'}^q, \ell)$ are true for every $\ell \in L_M$.
- We have $u = v_{p,I,J}^q$ and $v = v_{p',I',J'}^q$. In that case, u is a child of $a_{p,I,J}^q$ and v a child of $a_{p',I',J'}^q$. The only neighbors of u and v that are not children of $a_{p,I,J}^q$ nor $a_{p',I',J'}^q$ are all the other vertices of label ℓ_0 . Thus, $\varphi(a, \ell_0)$ holds for every internal node a of T^q .

We conclude that $\varphi(a, \ell)$ holds for every internal node a of T^q and every $\ell \in [k]$. Hence, there exists a family of matrices \mathcal{M}^q such that $(T^q, \mathcal{R}^q, \mathcal{M}^q, \lambda^q)$ is a $(3, k)$ -tree model of G^q for every $q \in [t]$. \triangleleft



■ **Figure 3** Illustration of the tree T and its subtree T^q for the tree-model constructed in Lemma 4.7. An edge between a white filled rectangle labeled X and a node a of the tree means that all the vertices in X are leaves adjacent to a .

For every $q \in [t-1]$, we denote by $\text{Inf}(q)$ the union of $\text{Inf}(1, q), \dots, \text{Inf}(r, q)$. Moreover, for every interval $[x, y] \subseteq [t]$, we denote by $G^{x,y}$, the union of the graphs G^q over $q \in [x, y]$ and the inferiority gadgets in $\text{Inf}(q)$ over $q \in [x, y]$ such that $q+1 \in [x, y]$. We prove by induction that for every interval $[x, y] \subseteq [t]$, there exists a $(2 \log(y-x+1) + 4, k)$ -tree-model $(T, \mathcal{R}, \mathcal{M}, \lambda)$ of $G^{x,y}$ such that given the root α of T , the following properties are satisfied:

- (A) λ_α maps each vertex from S_1^x, \dots, S_r^x to a unique label in L_{\min} .
- (B) λ_α maps each vertex from S_1^y, \dots, S_r^y to a unique label in L_{\min} .

When $x = y$, we only require that exactly one property among (A) and (B) is satisfied (we can choose which one as it is symmetric). The induction is on $y - x$. The base case is when $x = y$, in which case $G^{x,y} = G^x$ and we simply modify the $(3, k)$ -tree-model $(T^x, \lambda^x, \mathcal{R}^x, \mathcal{M}^x)$ for G^x as follows. We add to T^x a new root α adjacent to the former root a^x . We add to \mathcal{R}^x the function $\rho_{\alpha a^x}$ that bijectively maps L_S to L_{\min} (or L_{\max} if we want to satisfy (B) rather than (A)) and every label not in L_S to ℓ_0 . Finally, we add M_α the zero $k \times k$ -matrix to \mathcal{M}^x . After these modifications, it is easy to see that $(T^x, \lambda^x, \mathcal{R}^x, \mathcal{M}^x)$ is a $(4, k)$ -tree model of $G^{x,y}$ that satisfies (A) or (B).

Now assume that $x < y$ and that $G^{x',y'}$ admits the desired tree-model for every $[x', y']$ strictly included in $[x, y]$. Let $q = \lfloor (y-x)/2 \rfloor$. By induction hypothesis, there exist:

- A $(2 \log(q-x) + 4, k)$ -tree model $(T^{<q}, \mathcal{R}^{<q}, \mathcal{M}^{<q}, \lambda^{<q})$ for $G^{x,q-1}$ with the desired properties (if $x = q-1$, we require (A) to be satisfied).
- A $(2 \log(y-q) + 4, k)$ -tree-model $(T^{>q}, \mathcal{R}^{>q}, \mathcal{M}^{>q}, \lambda^{>q})$ for $G^{q+1,y}$ with the desired properties (if $y = q+1$, we require (B) to be satisfied).

For the sake of legibility, we assume that x is different from q , which implies that $G^{x,q-1}$ is not empty graph (note that $G^{q+1,y}$ is not empty as $x < y$ and $q = \lfloor (y-x)/2 \rfloor$). We lose some generality with this assumption, but we can easily deal with the case $x = q$ with some simple modifications on the following construction (i.e. removing some nodes and changing some renaming functions).

In the following, we construct a $(4 + 2 \log(y-x+1), k)$ -tree-model $(T, \mathcal{R}, \mathcal{M}, \lambda)$ of $G^{x,y}$ from the above tree-models of $G^{x,q-1}, G^{q+1,y}$, but also the $(3, k)$ -tree-model $(T^q, \lambda^q, \mathcal{R}^q, \mathcal{M}^q)$ of G^q given by Claim 4.8. To obtain T , we create the root α of T and we make it adjacent to a^q the root of T^q and two new vertices: $\alpha^{<q}$ and $\alpha^{>q}$. We make $\alpha^{<q}$ adjacent to the root of $T^{<q}$ and to all the vertices in $\text{Inf}(q-1)$. Symmetrically, we make $\alpha^{>q}$ adjacent to the root of $T^{>q}$ and to all the vertices in $\text{Inf}(q)$. See Figure 3 for an illustration of T .

We define λ as follows:

$$\lambda(v) = \begin{cases} \lambda^{<q}(v) & \text{if } v \in V(G^{x,q-1}), \\ \lambda^{>q}(v) & \text{if } v \in V(G^{q+1,y}), \\ \lambda^q(v) & \text{if } v \in V(G^q), \\ \lambda'(v) & \text{otherwise (when } v \text{ belongs to } \text{Inf}(q-1) \text{ or } \text{Inf}(q)) \end{cases}$$

where λ' maps the vertices in $G^{x,y}$ from $\text{Inf}(q-1)$ and $\text{Inf}(q)$ to L_{Inf} such that for each label ℓ of L_{Inf} , there exists $q' \in \{q-1, q\}$ and $p \in [r]$ such that ℓ is associated with either: (1) $v_i^{x,p,q'}$ for some $i \in [\log N - 1]$ and $x \in \{0, 1\}$ or (2) all the vertices in $V_i^{01,p,q'}$ for some $i \in [\log N]$. Since $|L_{\text{Inf}}| = 6r \log(N) - 4$, we have enough labels for doing so. The family of renaming function \mathcal{R} is obtained from the union of $\mathcal{R}^{<q} \cup \mathcal{R}^{>q} \cup \mathcal{R}^q$ by adding for every edge e in T that is not in $T^q, T^{<q}$ or $T^{>q}$ a function ρ_e defined as follows:

- ρ_e is the identify function when e is an edge adjacent to a leaf from $\text{Inf}(q-1)$ or $\text{Inf}(q)$.
 - ρ_e maps every label in $L_{\min} \cup L_{\max}$ to itself and every other label to ℓ_0 , when e is the edge between $a^{\otimes q}$ and the root of $T^{\otimes q}$ for $\otimes \in \{<, >\}$.
 - ρ_e maps every label in L_S to itself and every other label to ℓ_0 when $e = \alpha a^q$.
 - ρ_e maps every label in $L_{\min} \cup L_{\text{Inf}}$ to itself and every other label to ℓ_0 , when $e = \alpha \alpha^{<q}$.
 - ρ_e maps every label in $L_{\max} \cup L_{\text{Inf}}$ to itself and every other label to ℓ_0 , when $e = \alpha \alpha^{>q}$.
- Observe that λ_α satisfies Properties (A) and (B). As $\lambda_{\alpha^{<q}}$ satisfies (A), this function maps every vertex from S_1^x, \dots, S_r^x to a unique label in L_{\min} . The above renaming functions guarantee that the only vertices mapped to a label in L_{\min} by λ_α are from $V_{\alpha^{<q}}$. We deduce that Property (A) holds and symmetrically, Property (B) holds too.

Now we prove that a family of matrices \mathcal{M} exists such that $(T, \mathcal{R}, \mathcal{M}, \lambda)$ is a tree model of $G^{x,y}$. As before, we prove that $\varphi(a, \ell)$ holds for every internal node a of T and every label $\ell \in [k]$. Since our construction is based on tree-models for $G^q, G^{x,q-1}$ and $G^{q+1,y}$, we only need to prove that $\varphi(a, \ell)$ holds for every $a \in \{\alpha^{<q}, \alpha^{>q}, \alpha\}$ and $\ell \in [k]$.

We first deal with $\alpha^{<q}$. Let us describe the labeling function $\lambda_{\alpha^{<q}}$. Remember that $(T^{<q}, \mathcal{R}^{<q}, \mathcal{M}^{<q}, \lambda^{<q})$ satisfies Properties (A) and (B), or just (A) when $x = q-1$. Moreover, the renaming function associated with the edge between $\alpha^{<q}$ and $T^{<q}$ preserves the labels in $L_{\min} \cup L_{\max}$ and maps the other labels to ℓ_0 . Thus, $\lambda_{\alpha^{<q}}$ assign every vertex in $S_1^x, S_1^{q-1}, \dots, S_r^x, S_r^{q-1}$ to a unique label in $L_{\min} \cup L_{\max}$. We deduce that for every a pair (u, v) of distinct vertices in $V_{\alpha^{<q}}$, if $\lambda_{\alpha^{<q}}$ assigns u and v to the same label $\ell \in [k]$, then either:

- $u, v \in V_i^{01,p,q-1}$ for some $p \in [r]$ and $i \in [\log N]$. In this case, u and v are false twins by construction of $\text{Inf}(p, q-1)$ —i.e. $N(u) = N(v)$ —and we deduce that $\varphi(\alpha^{<q}, \ell)$ holds.
- $\ell = \ell_0$ and u, v are in $V(G^{x,q-1})$ and not from $S_1^x, S_1^{q-1}, \dots, S_r^x, S_r^{q-1}$. Then, all the neighbors of u and v are in $G^{x,q-1}$ and thus $N(u) \setminus V(G^{x,q-1}) = N(v) \setminus V(G^{x,q-1})$. We deduce that $\varphi(\alpha^{<q}, \ell)$ holds in this case too.

We conclude that $\varphi(\alpha^{<q}, \ell)$ holds for every $\ell \in [k]$ and with symmetric arguments, we can prove that $\varphi(\alpha^{>q}, \ell)$ holds also for every $\ell \in [k]$.

For α , notice that for every $a \in \{a^q, \alpha^{<q}, \alpha^{>q}\}$, the vertices in V_a labeled ℓ_0 by λ_α have neighbors only in V_a , hence $\varphi(a, \ell_0)$ holds. Furthermore, every label ℓ in $L_{\min} \cup L_{\max} \cup L_S$ is mapped by λ_α to a unique vertex in V_α , so $\varphi(\alpha, \ell)$ holds. Finally, each label in L_{Inf} is mapped by λ_α to a unique vertex or to all the vertices in $V_i^{01,p,q'}$ for some $p \in [r], q' \in \{q-1, q\}$ and $i \in [\log N]$. Since the vertices in $V_i^{01,p,q'}$ are false twins, we deduce that $\varphi(\alpha, \ell)$ holds for every $\ell \in L_{\text{Inf}}$. We conclude that $\varphi(\alpha, \ell)$ holds for every $\ell \in [k]$ and thus there exists a family \mathcal{M} of matrices such that $(T, \mathcal{M}, \mathcal{R}, \lambda)$ is a tree-model of $G^{x,y}$.

It remains to prove that the depth of T is at most $d = 2 \log(y - x + 1) + 4$. By definition of q , both $q - x$ and $y - q$ are smaller than $(y - x + 1)/2$. Thus, $\log(q - x)$ and $\log(y - q)$ are smaller than $\log(y - x + 1) - 1$. Now observe that the depth of T is the maximum between (i) the depth of T^q plus 1 which is 4, (ii) the depth of $T^{<q}$ plus 2, and (iii) the depth of $T^{>q}$ plus 2. The depth of $T^{<q}$ is at most $2 \log(q - x) + 4$. Since $\log(q - x) \leq \log(y - x + 1) - 1$, the depth of $T^{<q}$ plus 2 is at most $2 \log(y - x + 1) + 4$. Symmetrically, the depth of $T^{>q}$ plus 2 is also upper bounded by $2 \log(y - x + 1) + 4$. It follows that the depth of T is at most $2 \log(y - x + 1) + 4$.

We conclude that for every interval $[x, y]$, $G^{x,y}$ admits a $(2 \log(y - x + 1) + 4, k)$ -tree-model. In particular, it implies that $G^{1,t} = G$ admits a (d, k) -tree-model. It is easy to see from our proof that this (d, k) -tree-model is computable in polynomial time. \blacktriangleleft

We are now ready to prove Theorem 4.3.

Proof of Theorem 4.3. Let δ be an unbounded and computable function. Assume towards a contradiction that there exists an algorithm \mathcal{A} that solves the INDEPENDENT SET problem in graphs supplied with (d, k) -tree-models satisfying $d \leq \delta(k)$ that runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and uses $n^{\mathcal{O}(1)}$ space.

Since δ is unbounded and computable and \log is monotone, there exists an unbounded and computable function δ' such that for all sufficiently large $N, r \in \mathbb{N}$, we have

$$2 \log(\delta'(N)) + 4 \leq \delta(14r \log N - 3).$$

Let $(N, t, \Sigma, s_1, \dots, s_r)$ be an instance of LCS such that $t \leq \delta'(N)$. Our reduction provides us with a graph G and an integer goal such that the following holds:

- G has $\mathcal{O}(rtN^2 \log N)$ vertices and thus it can be constructed in $M^{\mathcal{O}(1)}$ time where M is the total bitsize of $(N, t, \Sigma, s_1, \dots, s_r)$. Indeed, the selection gadgets are made of $2rt \log N$ vertices, the inferiority gadgets have exactly $r(t-1)(2 \log N + \log N(1 + \log N)/2)$ vertices and the matching gadgets consist of $\sum_{p \in [r-1]} t \cdot 2|\mathcal{M}_p| \cdot (1 + 2 \log N)$ vertices.
- By Lemma 4.6, G admits an independent set of size at least goal iff s_1, \dots, s_r admits a common subsequence of size t .
- By Lemma 4.7, we can construct in polynomial time a (d, k) -tree-model of G with $d = 2 \log t + 4$ and $k = 14r \log N - 3$.

Observe that we have

$$d = 2 \log t + 4 \leq 2 \log(\delta'(N)) + 4 \leq \delta(14r \log N - 3) = \delta(k).$$

Consequently, we can run \mathcal{A} to check whether G admits an independent set of size at least goal in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and space $n^{\mathcal{O}(1)}$. Since $k = 14r \log N - 3$ and $n = \mathcal{O}(rtN^2 \log N) \leq M^{\mathcal{O}(1)}$, it follows that we can solve $(N, t, \Sigma, s_1, \dots, s_r)$ in time $N^{\mathcal{O}(r)} \cdot M^{\mathcal{O}(1)} \leq M^{\mathcal{O}(r)}$ and space $M^{\mathcal{O}(1)}$. As this can be done for every instance $(N, t, \Sigma, s_1, \dots, s_r)$ where $t \leq \delta'(N)$, it contradicts Conjecture 4.2. \blacktriangleleft

5 Fixed-Parameter Algorithms for Metric Dimension and Firefighting

The FIREFIGHTER problem on a graph G is the following. At time 0, a vertex $r \in V(G)$ catches fire. Then at each time step $i \geq 1$, first a firefighter is permanently placed on a vertex that is not currently on fire. This vertex is now permanently *protected*. Then the fire spreads to all unprotected neighbors of all vertices currently on fire. This process ends in the time step when the fire no longer spreads to new vertices. All vertices that do not catch fire during this process (including the protected vertices) are called *saved*; the rest are called *burned*. The goal is to maximize the number of saved vertices.

Bazgan et al. [5] showed that the FIREFIGHTER problem is fixed-parameter tractable when parameterized by the treewidth of the input graph and the number k of vertices that may be protected during the process. In this result, one first writes an MSO_2 formula $\varphi(X)$ that expresses that a set of vertices X can be saved assuming that k vertices can be protected, and then applies the optimization variant of Courcelle's Theorem, due to Arnborg et al. [3], to find the largest vertex subset A for which $\varphi(A)$ is satisfied. By inspection of the formula, we can see that it does not quantify over edge sets, hence $\varphi(X)$ is in fact an MSO_1 formula. Then, by replacing the usage of the algorithm of Arnborg et al. with the algorithm of Courcelle et al. [13], we conclude that the FIREFIGHTER problem is fixed-parameter tractable when parameterized by the cliquewidth of the input graph and the number of vertices that may be protected.

We now recall that in graphs with a (d, k) -tree model, any induced path has length at most $\mathcal{O}(2^{k^{d+1}})$ (this follows from [30, Theorem 3.7]; the bound accounts for our slightly different definition of a tree model). This implies that the firefighting game has at most $\mathcal{O}(2^{k^{d+1}})$ time steps and the same amount of vertices can be protected. Hence, recalling that a graph with a (d, k) -tree model has bounded cliquewidth [30, Proposition 3.4], we immediately obtain the following result.

► **Theorem 5.1.** *The FIREFIGHTER problem is fixed-parameter tractable when parameterized by d and k on graphs provided with a (d, k) -tree-model.*

We observe that this is in contrast to the complexity of the FIREFIGHTER problem on graphs of bounded treewidth. The FIREFIGHTER problem is in fact already NP-hard on trees of maximum degree 3 (which are graphs of treewidth 1) [22] and trees of pathwidth 3 [12].

A similar situation arises for the METRIC DIMENSION problem. In METRIC DIMENSION, given a graph G , we are asked to find a smallest set $Z \subseteq V(G)$ such that for any pair $u, v \in V(G)$, there is a vertex $z \in Z$ such that the distance between u and z and the distance between v and z are distinct. Gima et al. [31] observed that METRIC DIMENSION is fixed-parameter tractable when parameterized by the cliquewidth and the diameter of the input. Since in graphs with a (d, k) -tree model, any induced path has length at most $\mathcal{O}(2^{k^{d+1}})$ (the bound accounts for our slightly different definition of a tree model), and any such graph has bounded cliquewidth [30], we immediately obtain the following.

► **Theorem 5.2.** *The METRIC DIMENSION problem is fixed-parameter tractable when parameterized by d and k on graphs provided with a (d, k) -tree-model.*

This is again in contrast to the complexity of the METRIC DIMENSION problem on graphs of bounded treewidth. The METRIC DIMENSION problem is in fact already NP-hard on graphs of pathwidth 24 [37].

References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proc. FOCS 2015*, pages 59–78, 2015.
- 2 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: Time–space tradeoffs. *Theory Comput.*, 10(12):297–339, 2014.
- 3 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.
- 4 Marina Barsky, Ulrike Stege, Alex Thomo, and Chris Upton. Shortest path approaches for the longest common subsequence of a set of strings. In *Proc. BIBE 2007*, pages 327–333, 2007.
- 5 Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R. Fellows, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. *J. Comput. System Sci.*, 80(7):1285–1297, 2014.
- 6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. STOC 2007*, pages 67–74, 2007.
- 7 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Harold T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoret. Comput. Sci.*, 147(1&2):31–54, 1995.
- 8 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Lars Jaffke, and Paloma T. Lima. Xnlp-completeness for parameterized problems on graphs with a linear structure. In *17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPICs*, pages 8:1–8:18. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2022.
- 9 Hans L. Bodlaender, Carla Groenland, Hugo Jacob, Marcin Pilipczuk, and Michał Pilipczuk. On the complexity of problems on tree-structured graphs. In *17th International Symposium on Parameterized and Exact Computation, IPEC 2022*, volume 249 of *LIPICs*, pages 6:1–6:17. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2022.
- 10 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 193–204. IEEE, 2021.
- 11 Hans L. Bodlaender, Carla Groenland, and Michał Pilipczuk. Parameterized complexity of binary CSP: Vertex cover, treedepth, and related parameters. *CoRR*, abs/2208.12543, 2022.
- 12 Janka Chlebíková and Morgan Chopin. The firefighter problem: further steps in understanding its complexity. *Theoret. Comput. Sci.*, 676:42–51, 2017.
- 13 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.
- 14 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022.
- 16 Matt DeVos, O-joung Kwon, and Sang-il Oum. Branch-depth: Generalizing tree-depth of graphs. *European J. Combin.*, 90:Article 103186, 2020.
- 17 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 4th edition, 2012.
- 18 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 19 Jan Dreier. Lacon- and shrub-decompositions: A new characterization of first-order transductions of bounded expansion classes. In *Proc. LICS 2021*, pages 1–13, 2021.
- 20 Jan Dreier, Jakub Gajarský, Sandra Kiefer, Michał Pilipczuk, and Szymon Toruńczyk. Treelike decompositions for transductions of sparse graphs. In *Proc. LICS 2022*, pages 31:1–31:14, 2022.

- 21 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015.
- 22 Stephen Finbow, Andrew D. King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discret. Math.*, 307(16):2094–2105, 2007.
- 23 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.
- 24 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014.
- 25 Fedor V. Fomin and Tuukka Korhonen. Fast FPT-approximation of branchwidth. In *Proc. STOC 2022*, pages 886–899, 2022.
- 26 Martin Fürer. Multi-clique-width. In *Proc. ITCS 2017*, volume 67 of *Leibniz Int. Proc. Inform.*, pages 14:1–14:13, 2017.
- 27 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory Comput. Syst.*, 61(2):283–304, 2017.
- 28 Jakub Gajarský and Stephan Kreutzer. Computing shrub-depth decompositions. In *Proc. STACS 2020*, volume 154 of *Leibniz Int. Proc. Inform.*, pages 56:1–56:17, 2020.
- 29 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Trans. Comput. Log.*, 21(4):Art. 29, 41, 2020.
- 30 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1):7:1–7:25, 2019.
- 31 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoret. Comput. Sci.*, 918:60–76, 2022.
- 32 Sylvain Guillemot. Parameterized complexity and approximability of the longest compatible sequence problem. *Discret. Optim.*, 8(1):50–60, 2011.
- 33 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *Proc. STACS 2020*, volume 154 of *Leibniz Int. Proc. Inform.*, 2020.
- 34 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. Technical report, 2023. <https://arxiv.org/abs/2302.03627>.
- 35 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- 36 Daniel M. Kane. Unary subset-sum is in logspace. Technical report, 2010. <https://arxiv.org/abs/1012.1336>.
- 37 Shaohua Li and Marcin Pilipczuk. Hardness of metric dimension in graphs of constant treewidth. *Algorithmica*, 84(11):3110–3155, 2022.
- 38 Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar k -path in subexponential time and polynomial space. In *Proc. WG 2011*, volume 6986 of *Lecture Notes Comput. Sci.*, pages 262–270, 2011.
- 39 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
- 40 Wojciech Nadara, Michał Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. In *Proc. ESA 2022*, volume 244 of *Leibniz Int. Proc. Inform.*, pages 79:1–79:14, 2022.
- 41 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In *Proc. WG 2020*, volume 12301 of *Lecture Notes Comput. Sci.*, pages 27–39, 2020.

- 42 Pierre Ohlmann, Michał Pilipczuk, Wojciech Przybyszewski, and Szymon Toruńczyk. Canonical decompositions in monadically stable and bounded shrubdepth graph classes. Technical report, 2023. <https://arxiv.org/abs/2303.01473>.
- 43 Patrice Ossona de Mendez, Michał Pilipczuk, and Sebastian Siebertz. Transducing paths in graph classes with unbounded shrubdepth. *European J. Combin.*, page 103660, 2022.
- 44 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003.
- 45 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018.
- 46 Egon Wanke. k -NLC graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2-3):251–266, 1994.