Space-Efficient Parameterized Algorithms on Graphs of Low Shrubdepth

Benjamin Bergougnoux
Institute of Informatics, University of Warsaw, Poland

Vera Chekan
Humboldt-Universität zu Berlin, Germany

Robert Ganian
Algorithms and Complexity Group, TU Wien, 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, 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

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.18

Related Version *Full Version*: https://arxiv.org/abs/2307.01285 [5]

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. [9, 12] 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. [29], 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 [19]; 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 [35] 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, 35] for further details.

However, starting with the work of Fürer and Yu [20], a long line of advances [25, 31, 32, 35] 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 [20, 25, 32, 35], 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 $\mathcal{O}(d \log n)$ [35]. 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.

Cliquewidth and Shrubdepth. In this work, we are interested in the parameter *cliquewidth* and its low-depth counterpart: *shrubdepth*. While treewidth applies only to sparse graphs, cliquewidth is a notion of tree-likeness suited for dense graphs as well. The decompositions underlying cliquewidth are called *clique expressions* [8]. 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 cliquewidth 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 cliquewidth. Furthermore, while several problems involving edge selection or edge counting, such as Hamiltonicity or Max Cut, remain W[1]-hard under the cliquewidth parameterization [16, 17], standard dynamic programming still allows us to solve them in XP time. In this sense, clique-width 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 [18]. Nevertheless, since the cliquewidth of a graph is at least as large as its linear cliquewidth, which in turn is as large as its pathwidth, the lower bounds of Allender et al. [2] and of Pilipczuk and Wrochna [35] carry over to the cliquewidth setting. Hence, reducing the space complexity to polynomial requires a sacrifice in the time complexity.

Shrubdepth, introduced by Ganian et al. [23], is a variant of cliquewidth 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 [21], and has become an important concept in the logic-based theory of well-structured dense graphs [13, 14, 22, 23, 33, 34].

Since shrubdepth is a bounded-depth analogue of cliquewidth 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 [20, 25, 31, 32, 35], 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 [19]. 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 [35].

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 Transveral and q-Coloring for a fixed constant q (details in the full version).

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 [17]. 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 on 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 [27] 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 [26], 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* $d \leqslant \delta(k)$ *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 [35] 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 [35], 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) [7, 28], 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.*

In this work some technical details have been omitted due to space constraints. We refer to the full version of the paper for all proofs [5].

2 Preliminaries

For a positive integer k, we denote by $[k] = \{1, \ldots, k\}$ and $[k]_0 = [k] \cup \{0\}$. For a function $f \colon A \to B$ and elements a, b (not necessarily from $A \cup B$), the function $f[a \mapsto b] \colon A \cup \{a\} \to B \cup \{b\}$ 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 [11]. The full proofs of our results also require the use of algebraic tools – notably the cover product and the fast subset convolution machinery of Björklund et al. [6].

We use the same computational model as Pilipczuk and Wrochna [35], 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 $\lambda\colon V(G) \to [k]$, and a family of renaming functions $\mathcal{R} = \{\rho_{ab}\}_{ab \in E(T)}$ with $\rho_{ab}\colon [k] \to [k]$ 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 $\lambda_a : V_a \to [k]$ 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.

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* [23] or *rank-depth* [10], that are suited for the treatment of single graphs. We remark that in the original definition proposed by Ganian et al. [23], relabeling is not allowed; however, using either definition yields the same notion of shrubdepth. Moreover, throughout this work we abstract away from the computation of the tree-models themselves and assume that a (d,k)-tree-model of the considered graph is provided on input.

We note that a fixed-parameter algorithm for computing tree-models has been proposed by Gajarský and Kreutzer [21] (in the sense of Ganian et al. [23]). 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 their algorithm [21] shows that it can be implemented with 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. We leave as an open question the computation of a (d,k)-tree model, for a given graph G, running in time $f(d,k) \cdot n^{\mathcal{O}(1)}$ and using space $(d+k)^{\mathcal{O}(1)} \cdot \log n$.

3 Space-Efficient Algorithms on Tree-Models

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 $\mathsf{poly}(d,k) \log n$ 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 $\mathsf{poly}(d,k,n)$ space and then show how a result by Pilipczuk and Wrochna [35] can be applied to decrease the space complexity to $\mathsf{poly}(d,k) \log n$.

In order to simplify forthcoming definitions/statements, let a be an internal node of T with $b_1, \ldots, b_t$ 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 $\mathsf{IS}(a, S) = \sum_{p \in \mathbb{N}} q(a, S, p) \cdot x^p$. 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 $\sum_{S \subseteq [k]} \mathsf{IS}(r, S)$. Therefore, the problem boils down to the computation of $\mathsf{IS}(r, S)$ 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 $\mathsf{IS}(a, S)$ makes some requests to the coefficients $q(b_i, S_i, p_i)$ of $\mathsf{IS}(b_i, S_i)$ 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 (rsp. 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 $\alpha \colon S \to \{1_=, 2_\geqslant\}$. 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 $\beta \colon S \to \{1_=, 1_\geqslant\}$ where $1_\geqslant$ 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 $\beta(i) = 1_\geqslant$. 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 $\alpha \colon S \to \{1_=, 2_\geqslant\}$ 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 $\alpha \colon S \to \{1_=, 2_\geqslant\}$ be fixed. Let $s_1, \ldots, s_{|\alpha^{-1}(2_\geqslant)|}$ be an arbitrary linear ordering of $\alpha^{-1}(2_\geqslant)$. To compute the number of independent sets that match our choice of α, we proceed by iterating over $c \in \{0, \ldots, |\alpha^{-1}(2_\geqslant)|\}$, and we count independent sets where the labels in $\{s_1, \ldots, s_c\}$ 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 $\gamma \colon \{s_1, \ldots, s_c\} \to \{1_=, 1_\geqslant\}$, 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 $i \in \{s_1, \ldots, s_c\}$ 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 $i \in \{s_1, \ldots, s_c\}$ with $\gamma(i) = 1_\geqslant$, there exists at least one child b_j of a such that $I_a(i) \cap V_{b_j} \neq \emptyset$;

- for every label $i \in S \setminus \{s_1, \ldots, s_c\}$ 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 $i \in S \setminus \{s_1, \ldots, s_c\}$ with $\alpha(i) = 2_\geqslant$, 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 $I_a(i) \cap V_{b_{j_2}} \neq \emptyset$.

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

▶ **Observation 3.1.** *We have* $q(a, S, p) = \sum_{\alpha \in \{1_=, 2_\geqslant\}^S, \gamma \in \{1_=, 1_\geqslant\}^\emptyset} q(a, S, \alpha, 0, \gamma, p)$ *for every* $S \subseteq \lambda_a(V_a)$ *and integer* p. *Also, for every* $\alpha \in \{1_=, 2_\geqslant\}^S$, *every* $c \in \{0, \ldots, |\alpha^{-1}(2_\geqslant)| - 1\}$ *and every* $\gamma \colon \{s_1, \ldots, s_c\} \to \{1_=, 1_\geqslant\}$, *we have* $q(a, S, \alpha, c, \gamma, p) = q(a, S, \alpha, c+1, \gamma[s_{c+1} \mapsto 1_\geqslant], p) - q(a, S, \alpha, c+1, \gamma[s_{c+1} \mapsto 1_=], p)$.

It remains then to show how to compute the value $q(a, S, \alpha, |\alpha^{-1}(2_\geqslant)|, \gamma, p)$ for every $\alpha \in \{1_=, 2_\geqslant\}^S$, every $\gamma \in \{1_=, 1_\geqslant\}^{\alpha^{-1}(2_\geqslant)}$, and every integer p. It is worth mentioning that if $\beta \colon S \to \{1_=, 1_\geqslant\}$ is such that $\beta^{-1}(1_=) = \alpha^{-1}(1_=) \cup \gamma^{-1}(1_=)$ and $\beta^{-1}(1_\geqslant) = \alpha^{-1}(2_\geqslant) \setminus \gamma^{-1}(1_=)$, then $q(a, S, \alpha, |\alpha^{-1}(1_\geqslant)|, \gamma, p)$ 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 $i \in \beta^{-1}(1_\geqslant)$, 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 $q(a, S, \alpha, |\alpha^{-1}(1_\geqslant)|, \gamma, p)$ and we define the polynomial $\mathsf{TIS}(a, S, \beta) \in \mathbb{Z}[x]$ (where "T" stands for "transformed") as $\mathsf{TIS}(a, S, \beta) = \sum_{p \in \mathbb{N}} q(a, S, \beta, p) \cdot x^p$. Recall that because we are computing $\mathsf{IS}(a, S)$ and $\mathsf{TIS}(a, S, \beta)$ in a top-down manner, some queries for $\mathsf{IS}(b_i, S_i)$ will be made during the computation. Before continuing in the computation of $\mathsf{TIS}(a, S, \beta)$, let us first explain how to request the polynomials $\mathsf{IS}(b_j, S_j)$ from each child b_j of a. If a is not the root, let a^* be its parent in T, and we use $\mathsf{PIS}(a, S)$ (where "P" stands for "parent") to denote the polynomial $\mathsf{PIS}(a, S) = \sum_{p \in \mathbb{N}_0} q^\rho(a, S, p) x^p$ where $q^\rho(a, S, p) = \sum_{D \subseteq \lambda_a(V_a) \colon \rho_{a^* a}(D) = S} q(a, D, p)$ 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 $\mathsf{PIS}(a, S) = \sum_{D \subseteq \lambda_a(V_a) \colon \rho_{a^* a}(D) = S} \mathsf{IS}(a, D)$.

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 $\mathsf{IS}(a, S)$ from the polynomials $\mathsf{TIS}(a, S, \beta)$, but will also motivate the forthcoming definitions. Recall that we have fixed $S \subseteq \lambda_a(V_a)$ and $\beta \colon S \to \{1_=, 1_\geqslant\}$, and in $\mathsf{IS}(a, S)$ and $\mathsf{TIS}(a, S, \alpha)$ 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* $\mathsf{IS}(a, S)$, *there exists a unique* $j \in [t]$ *such that* $I_a(i_1) \cup I_a(i_2) \subseteq V_{b_j}$.

Recall that for every label $i \in \alpha^{-1}(2_\geqslant)$, 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 $\alpha(i_1) = 2_\geqslant$, 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 $\mathsf{TIS}(a, S, \beta)$ made all have the property that for each $i_1 \in S$ the following holds: if $\beta(i_1) = 1_\geqslant$, 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 $\mathsf{TIS}(a, S, \beta)$

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 $\mathsf{TIS}(a, S, \beta)$, 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 $\mathsf{TIS}(a, S, \beta)$, 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 $\mathsf{TIS}(a, S, \beta)$ by requesting some $\mathsf{PIS}(b_j, S_j)$. Let I be some independent set counted in $\mathsf{TIS}(a, S, \beta)$. 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 $\beta^{-1}(1_\geqslant)$ 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 $\mathsf{cc}(F^{a,\beta})$ 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 $U^{a,\beta} = \beta^{-1}(1_\geqslant) \cup \mathsf{cc}(F^{a,\beta})$. For every $j \in [t]$, we define a mapping $f_j^{a,\beta} \colon 2^{U^{a,\beta}} \to \mathbb{Z}[x, z]$ (i.e., to polynomials over x and z) as follows: $f_j^{a,\beta}(X) = \mathsf{PIS}(b_j, \mathsf{flat}^{a,\beta}(X)) z^{|X \cap \mathsf{cc}(F^{a,\beta})|}$ where $\mathsf{flat}^{a,\beta} \colon 2^{U^{a,\beta}} \to 2^S$ intuitively performs a union over all the present labels – formally: $\mathsf{flat}^{a,\beta}(W) = (W \cap \beta^{-1}(1_\geqslant)) \cup \bigcup_{w \in W \cap \mathsf{cc}(F^{a,\beta})} w$. 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 $g_j^{a,\beta} \colon 2^S \to \mathbb{Z}[x, z]$ as follows:

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

Observe that the function $\mathsf{flat}^{a,\beta}$ 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".

▶ **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 $b_1, \ldots, b_t$ be the children of a. For every $S \subseteq \lambda_a(V_a)$, and every conflict-free $\beta \colon S \to \{1_=, 1_\geqslant\}$, it holds that*

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

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 $P\langle z^{|\mathsf{cc}(F^{a,\beta})|} \rangle \in \mathbb{Z}[x]$ is defined as $P\langle z^{|\mathsf{cc}(F^{a,\beta})|} \rangle = \sum_{u_1 \in \mathbb{N}_0} q_{u_1, |\mathsf{cc}(F^{a,\beta})|} x^{u_1}$.

Now we can apply a result by Björklund et al. [6] to accelerate the computation of $\mathsf{TIS}(a,S,\beta)$: It holds $\mathsf{TIS}(a,S,\beta) = \left(\sum_{Y\subseteq S}(-1)^{|S\setminus Y|}\prod_{j=1}^{t}\sum_{Z\subseteq Y} g_j(Z)\right)\langle z^{|\mathsf{cc}(F)|}\rangle$. 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 [35] based on the Chinese remainder theorem:

▶ **Theorem 3.5** ([35]). *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 $0\leqslant q_i\leqslant 2^{n'}$ for $i=0,\ldots,n'$. Suppose that given a prime number $p\leqslant 2n'+2$ and $s\in\mathbb{F}_p$, the value $P(s) \pmod p$ can be computed in time T and space S. Then given $k\in\{0,...,n'\}$, the value q_k can be computed in time $\mathcal{O}(T\cdot\mathsf{poly}(n'))$ and space $\mathcal{O}(S+\log n')$.*

With this, we can finally prove Theorem 1.1.

Counting List-Homomorphisms. We now explain how to apply the techniques from the above 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 $\omega\colon V(G)\to\mathbb{N}$, a list function $L\colon V(G)\to 2^{V(H)}$, 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

$$\Big|\big\{\varphi\colon V(G)\to V(H)\ \big|\ \forall v\in V(G)\colon\varphi(v)\in L(v),\forall uv\in E(G)\colon\varphi(u)\varphi(v)\in E(H),\ |\varphi^{-1}(R)|=C,\text{ and }\omega(\varphi^{-1}(R))=W\big\}\Big|\ .$$

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.
- 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. We prove the following result.

▶ **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 $\mathcal{O}(k^2d(\log n+\log W^*))$, and solves the R-Weighted #-List-H-Homomorphism in G, where W^* denotes the maximum weight in ω.*

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 $\mathsf{sig}_c(X)$ – as the vector $(|X_c(1)|, |X_c(2)|, \ldots, |X_c(k)|)$. We let $\mathcal{S}(c)$ be the set of c-signatures of all the subsets of V_c, *i.e.*, $\mathcal{S}(c) := \{\mathsf{sig}_c(X) : X \subseteq V_c\}$. Observe that $|\mathcal{S}(c)| \in n^{\mathcal{O}(k)}$ holds. Also, for the children $b_1, \ldots, b_t$ of a, we define $\mathcal{S}(ab_1, \ldots, ab_t)$ as the set of all tuples $(s^1, \ldots, s^t)$ 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 $s = (s_1, \ldots, s_k) \in \mathcal{S}(ab)$ 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 $\mathsf{sig}_{ab}(X) = \rho_{ab}(\mathsf{sig}_b(X))$. 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 $b_1, \ldots, b_t$. Now, we explain how to compute $f_a(s)$ by making at most $n^{\mathcal{O}(k)}$ calls to the functions $f_{ab_1}, \ldots, f_{ab_t}$. The first step is to express $f_a(s)$ in terms of $f_{ab_1}, \ldots, f_{ab_t}$. 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 $E(V_{b_1}, \ldots, V_{b_t})$ the set of edges of $G[V_a]$ whose endpoints lie in different V_{b_i}'s, i.e. $E(G[V_{b_1}, \ldots, V_{b_t}]) \setminus (E(G[V_{b_1}] \cup \cdots \cup E(G[V_{b_t}]))$. Given $X \subseteq V_a$, we denote by $E_a(X)$ the intersection of $E(X, V_a \setminus X)$ and $E(V_{b_1}, \ldots, V_{b_t})$. 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 $c \in \{a, ab_1, \ldots, ab_t\}$, $s \in \mathcal{S}(c)$ and $i, j \in [k]$, we define

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

It is not hard to check that, for every subset $X \subseteq V_a$ with a-signature s, $\#\mathsf{pairs}_a(s, i, j)$ is the size of $\mathsf{pairs}_a(X, i, j)$ 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 $\mathsf{pairs}_a(X, i, j)$ whose endpoints belong to different sets among $V_{b_1}, \ldots, V_{b_t}$. Moreover, given a child b of a, the number of pairs in $\mathsf{pairs}_a(X, i, j)$ whose both endpoints belong to V_b is exactly $\#\mathsf{pairs}_{ab}(\mathsf{sig}_{ab}(X), i, j)$. Thus when $M_a[i, j] = 1$, we have

$$|E_a(X, i, j)| = \#\mathsf{pairs}_a(\mathsf{sig}_a(X), i, j) - \sum_{i \in [t]} \#\mathsf{pairs}_{ab_i}(\mathsf{sig}_{ab_i}(X), i, j) \ . \tag{1}$$

We capture the size of $E_a(X)$ with the following notion. For every $c \in \{a, ab_1, \ldots, ab_t\}$, $s \in \mathcal{S}(c)$ and $(k \times k)$-matrix M, we define $m_c(s, M) := \sum_{\substack{i,j \in [k], i \leqslant j \\ M[i,j]=1}} \#\mathsf{pairs}_c(s, i, j)$. Note that $|E_a(X)| = \sum_{i,j \in [k] :\, i \leqslant j, M_a[i,j]=1} |E_a(X, i, j)|$. Hence, by Equation 1, we deduce that $|E_a(X)| = m_a(\mathsf{sig}_a(X), M_a) - \sum_{i \in [t]} m_{ab_i}(\mathsf{sig}_{ab_i}(X), M_a)$. Since $E(X, V_a \setminus X)$ is the disjoint union of $E_a(X)$ and the sets $E(X \cap V_{b_1}, V_{b_1} \setminus X), \ldots, E(X \cap V_{b_t}, V_{b_t} \setminus X)$, we deduce:

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

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

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

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

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

To compute $f_a(s)$ we use a twist of Kane's algorithm [27] for solving the k-DIMENSIONAL UNARY SUBSET SUM in Logspace.

Dominating Set. We now prove 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 [35] 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*".

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

▶ **Theorem 3.10** (Isolation lemma, [30])**.** *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 $1, \ldots, 2n$ uniformly and independently at random. Storing ω takes $\mathcal{O}(n \log n)$ space. The remainder of the algorithm uses only $\mathcal{O}(dk^2 \log n)$ space.

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 **TA** and **AF**. Further, let $R \coloneqq \{\mathbf{T}\}$. Following our approach for H-homomorphisms, for every set $S \subseteq \mathbf{States}$ with $\mathbf{States} \coloneqq \{(\mathbf{T}, 1), (\mathbf{F}, 1), \ldots, (\mathbf{T}, k), (\mathbf{F}, k)\}$, 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 $W^* \leqslant 2n$) 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 $\widehat{I} \cap V(i) \neq \emptyset$.

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 [26] 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.

We can apply the same transformation to obtain from $a_{S,c,w}$'s the number of dominating sets of size c and weight w modulo 2. Isolation lemma implies that with probability at least $1/2$ for some w this number if non-zero if a dominating set of size c exists.

▶ 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 $(N, t, \Sigma, s_1, \ldots, s_r)$ where N and t are positive integers, Σ is an alphabet and $s_1, \ldots, s_r$ 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 $\mathcal{O}(N^r)$. 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. [15], 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. 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 [3]. This motivated Pilipczuk and Wrochna to formulate the following conjecture [35].

▶ **Conjecture 4.1** ([35]). *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 $t \leqslant \delta(N)$.*

Let $(N, t, \Sigma, s_1, \ldots, s_r)$ be an instance of LCS. We assume, without loss of generality, that N is a power of 2. We provide a reduction from $(N, t, \Sigma, s_1, \ldots, s_r)$ to an equivalent instance of Independent Set consisting of a graph G with $(r + t + N)^{\mathcal{O}(1)}$ vertices which admits a (d, k)-tree-model where $d = \mathcal{O}(\log t)$ and $k = \mathcal{O}(r \log N)$. This implies Theorem 1.4 since for every unbounded and computable function δ there exists an unbounded and computable function δ' such that if $t \leqslant \delta'(N)$, then $d \leqslant \delta(k)$ for all sufficiently large $N, r \in \mathbb{N}$.

To outline the main idea of the reduction, let $s^\star$ be a potential common substring of $s_1, \ldots, s_r$ of length t. We use matchings to represent the binary encoding of the positions of the letters of $s^\star$ 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 $\mathsf{S}_p^q | I$ 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 $s^\star$ in s_p.

We need to guarantee that the selected positions in the gadgets $\mathsf{S}_p^1, \ldots, \mathsf{S}_p^t$ 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 $\mathsf{Inf}(p, q)$ 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 $\mathsf{S}_p^q, \mathsf{S}_p^{q+1}$, and their inferiority gadget, is the following: there exist $I < J \in [N]$ such that the independent set contains $\mathsf{S}_p^q | I \cup \mathsf{S}_p^{q+1} | J$. 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. We refer to the full version of this paper for the construction of these inferiority gadgets and the arguments proving the following observation.

▶ **Observation 4.3.** *Let $p \in [r]$ and $q \in [t-1]$. The independence number of $\mathsf{Inf}(p, q)$ 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 $\mathsf{Inf}(p, q)$ such that the union of S, $\mathsf{S}_p^q | I$ and $\mathsf{S}_p^{q+1} | J$ induces an independent set.*

Next, we need to ensure that the t positions chosen in $s_1, \ldots, s_r$ indeed correspond to a common subsequence, i.e., for every $q \in [t]$, the q-th chosen letter must be the same in every $s_1, \ldots, s_r$. 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* $\mathsf{Match}(p, q)$ as follows:

- For every pair $(I, J) \in \mathcal{M}_p$ and for each $p^\star \in \{p, p+1\}$, we create a copy $\mathsf{M}_{p^\star}^{p,q,I,J}$ of $\mathsf{S}_{p^\star}^q$ 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 $\mathsf{S}_{p^\star}^q$ and the $(1-x)$-endpoint of the ℓ-edge of $\mathsf{M}_{p^\star}^{p,q,I,J}$.
- 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 $\mathsf{M}_p^{p,q,I,J}$ that are not in $\mathsf{M}_p^{p,q,I,J} | I$ and (2) all the vertices from $\mathsf{M}_{p+1}^{p,q,I,J}$ that are not in $\mathsf{M}_{p+1}^{p,q,I,J} | J$.

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

▶ **Observation 4.4.** *Let $p \in [r-1]$ and $q \in [t]$. The independence number of $\mathsf{Match}(p,q)$ 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 $\mathsf{Match}(p,q)$ with $1 + 2|\mathcal{M}_p| \cdot \log N$ vertices such that the union of S, $\mathsf{S}^q_p|I$ and $\mathsf{S}^q_{p+1}|J$ is an independent set.*

This concludes the construction of the graph G. See Figure 1 below for an overview.

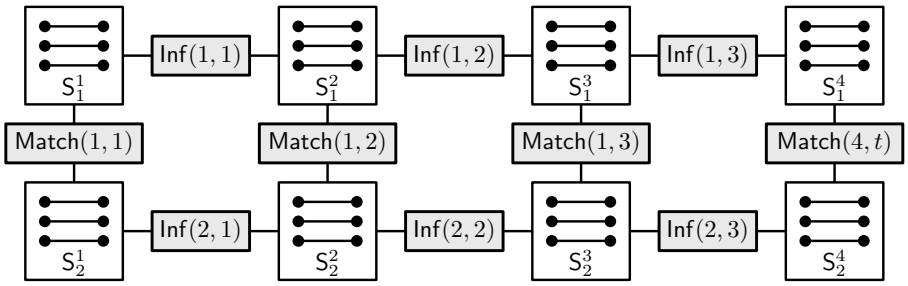

Figure 1 Overview of the graph G with $\log N = 3$, $r = 2$ and $t = 4$. There are some edges between two gadgets if and only if there are some edges between their vertices in G.

We prove correctness of the reduction in the following lemma which follows mostly from Observations 4.3 and 4.4.

▶ **Lemma 4.5.** *There exists an integer* goal *such that G admits an independent set of size at least* goal *iff the strings $s_1, \ldots, s_r$ admit a common subsequence of length t.*

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

▶ **Lemma 4.6.** *We can compute in polynomial time a (d,k)-tree-model of G where $d = 2\log t + 4$ and $k = 14r\log N - 3$.*

Sketch of proof. 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^q_p with $p \in [r]$ and the matching gadgets $\mathsf{Match}(p,q)$ with $p \in [r-1]$.

For each $q \in [t]$, we prove that G^q admits a $(3,k)$-tree-model $(T^q, \mathcal{M}^q, \mathcal{R}^q, \lambda^q)$ where the tree T^q is constructed as follows. We create the root a^q of T^q and we attach all the vertices in the selection gadgets S^q_p with $p \in [r]$ as leaves adjacent to a^q. Then, for every $p \in [r-1]$, we create a node a^q_p adjacent to a^q and for every $(I,J) \in \mathcal{M}_p$, we create a node $a^q_{p,I,J}$ adjacent to a^q_p. For each $(I,J) \in \mathcal{M}_p$, we make $a^q_{p,I,J}$ adjacent to the vertex $v^q_{p,I,J}$ and all the vertices in $\mathsf{M}^{p,q,I,J}_p$ and $\mathsf{M}^{p,q,I,J}_{p+1}$. Note that all the vertices in $\mathsf{Match}(p,q)$ are the leaves of the subtree rooted at a^q_p, and the leaves of T^q are exactly the vertices in G^q. See Figure 2 for an illustration of T^q.

For every $q \in [t-1]$, we denote by $\mathsf{Inf}(q)$ the union of $\mathsf{Inf}(1,q), \ldots, \mathsf{Inf}(r,q)$. 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 $\mathsf{Inf}(q)$ over $q \in [x,y]$ such that $q+1 \in [x,y]$.

For every interval $[x,y]$, we prove by induction on $y-x$ that $G^{x,y}$ 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 also easy to see from our proof that this (d,k)-tree-model is computable in polynomial

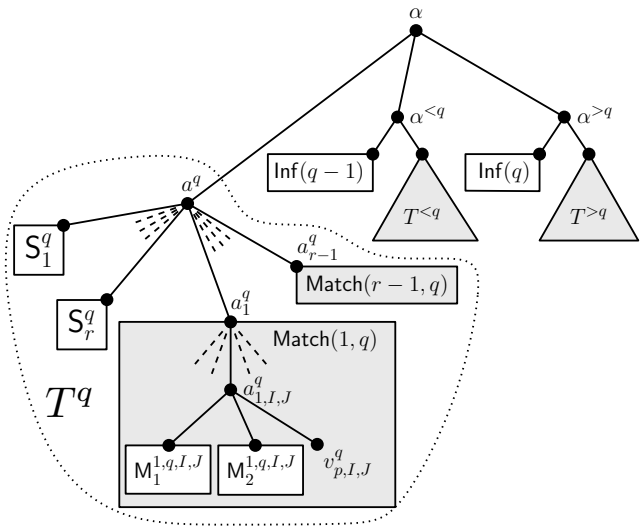

Figure 2 Illustration of the tree T and its subtree T^q for the tree-model constructed in Lemma 4.6. An edge between a white filled rectangle labeled X and a node a of the tree means that all the vertices in X are leaves adjacent to a.

time. For the base case of the induction, when $y = x$, we have $G^{x,y} = G^x$ and we have proved above that it admits a $(3, k)$-tree-model. When $x < y$, let $q = \lfloor (y - x)/2 \rfloor$. We use the induction hypothesis to obtain:

- A $(2\log(q - x) + 4, k)$-tree model $(T^{<q}, \mathcal{R}^{<q}, \mathcal{M}^{<q}, \lambda^{<q})$ for $G^{x,q-1}$.
- A $(2\log(y - q) + 4, k)$-tree-model $(T^{>q}, \mathcal{R}^{>q}, \mathcal{M}^{>q}, \lambda^{>q})$ for $G^{q+1,y}$.

Then, we construct a $(4 + 2\log(y - x + 1), k)$-tree-model $(T, \mathcal{R}, \mathcal{M}, \lambda)$ of $G^{x,y}$ from the 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. 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 $\mathsf{Inf}(q-1)$. Symmetrically, we make $\alpha^{>q}$ adjacent to the root of $T^{>q}$ and to all the vertices in $\mathsf{Inf}(q)$. See Figure 2 for an illustration of T. ◀

5 Fixed-Parameter Algorithms for Metric Dimension and Firefighting

Theorem 1.5 – and in particular the fixed-parameter tractability of Metric Dimension and Firefighter parameterized by shrub-depth – can be obtained by combining known results about these problems [4, 24] with a bound on the maximum length of induced paths in graph classes of bounded shrubdepth [23, Theorem 3.7]. These results contrast the NP-hardness of both problems on graphs of bounded pathwidth [7, 28].

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. doi:10.1109/FOCS.2015.14.

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. doi:10.4086/toc.2014.v010a012.

3 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. doi:10.1109/BIBE.2007.4375584.

4 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. doi:10.1016/j.jcss.2014.03.001.

5 Benjamin Bergougnoux, Vera Chekan, Robert Ganian, Mamadou Moustapha Kanté, Matthias Mnich, Sang il Oum, Michał Pilipczuk, and Erik Jan van Leeuwen. Space-efficient parameterized algorithms on graphs of low shrubdepth. *arXiv*, 2023. doi:10.48550/arXiv.2307.01285.

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 Janka Chlebíková and Morgan Chopin. The firefighter problem: further steps in understanding its complexity. *Theoret. Comput. Sci.*, 676:42–51, 2017. doi:10.1016/j.tcs.2017.03.004.

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

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

10 Matt DeVos, O-joung Kwon, and Sang-il Oum. Branch-depth: Generalizing tree-depth of graphs. *European J. Combin.*, 90:Article 103186, 2020.

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

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 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. doi:10.1109/LICS52264.2021.9470680.

14 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. doi:10.1145/3531130.3533349.

15 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. doi:10.1007/s00453-014-9944-y.

16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

17 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. doi:10.1137/130910932.

18 Fedor V. Fomin and Tuukka Korhonen. Fast FPT-approximation of branchwidth. In *Proc. STOC 2022*, pages 886–899, 2022.

19 Martin Fürer. Multi-clique-width. In *Proc. ITCS 2017*, volume 67 of *Leibniz Int. Proc. Inform.*, pages 14:1–14:13, 2017. doi:10.4230/LIPIcs.ITCS.2017.14.

20 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory Comput. Syst.*, 61(2):283–304, 2017. doi:10.1007/s00224-017-9751-3.

21 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. doi:10.4230/LIPIcs.STACS.2020.56.

22 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. doi:10.1145/3382093.

23 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. doi:10.23638/LMCS-15(1:7)2019.

24 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. doi:10.1016/j.tcs.2022.03.021.

25 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.

26 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *arXiv*, 2023. doi:10.48550/ARXIV.2302.03627.

27 Daniel M. Kane. Unary subset-sum is in logspace. *arXiv*, 2010. arXiv:1012.1336.

28 Shaohua Li and Marcin Pilipczuk. Hardness of metric dimension in graphs of constant treewidth. *Algorithmica*, 84(11):3110–3155, 2022. doi:10.1007/s00453-022-01005-y.

29 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. doi:10.1007/978-3-642-25870-1_24.

30 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

31 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. doi:10.4230/lipics.esa.2022.79.

32 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. doi:10.1007/978-3-030-60440-0_3.

33 Pierre Ohlmann, Michał Pilipczuk, Wojciech Przybyszewski, and Szymon Toruńczyk. Canonical decompositions in monadically stable and bounded shrubdepth graph classes. *arXiv*, 2023. doi:10.48550/arXiv.2303.01473.

34 Patrice Ossona de Mendez, Michał Pilipczuk, and Sebastian Siebertz. Transducing paths in graph classes with unbounded shrubdepth. *European J. Combin.*, page 103660, 2022. doi:10.1016/j.ejc.2022.103660.

35 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. doi:10.1145/3154856.