Monotone Bounded-Depth Complexity of Homomorphism Polynomials

C.S. Bhargav ✉ 🏠 🆔
Indian Institute of Technology, Kanpur, India

Shiteng Chen ✉ 🆔
Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

Radu Curticapean ✉ 🏠 🆔
University of Regensburg, Germany
IT University of Copenhagen, Denmark

Prateek Dwivedi ✉ 🏠 🆔
IT University of Copenhagen, Denmark

—— Abstract ——

For every fixed graph H, it is known that homomorphism counts from H and colorful H-subgraph counts can be determined in $O(n^{t+1})$ time on n-vertex input graphs G, where t is the treewidth of H. On the other hand, a running time of $n^{o(t/\log t)}$ would refute the exponential-time hypothesis. Komarath, Pandey, and Rahul (Algorithmica, 2023) studied algebraic variants of these counting problems, i.e., homomorphism and subgraph *polynomials* for fixed graphs H. These polynomials are weighted sums over the objects counted above, where each object is weighted by the product of variables corresponding to edges contained in the object. As shown by Komarath *et al.*, the *monotone* circuit complexity of the homomorphism polynomial for H is $\Theta(n^{\mathrm{tw}(H)+1})$.

In this paper, we characterize the power of monotone *bounded-depth* circuits for homomorphism and colorful subgraph polynomials. This leads us to discover a natural hierarchy of graph parameters $\mathrm{tw}_\Delta(H)$, for fixed $\Delta \in \mathbb{N}$, which capture the width of tree-decompositions for H when the underlying tree is required to have depth at most Δ. We prove that monotone circuits of product-depth Δ computing the homomorphism polynomial for H require size $\Theta(n^{\mathrm{tw}_\Delta(H^\dagger)+1})$, where $H^\dagger$ is the graph obtained from H by removing all degree-1 vertices. This allows us to derive an optimal depth hierarchy theorem for monotone bounded-depth circuits through graph-theoretic arguments.

1   Introduction

Counting and deciding the existence of patterns in graphs plays an important role in computer science. In theoretical computer science, pattern counting was among the first problems to be investigated in Valiant's seminal paper on the class #P of counting problems [42], which showed #P-hardness for the permanent of zero-one matrices, a problem that can equivalently be viewed as counting perfect matchings in bipartite graphs.

Counting small patterns

In many applications, the pattern is smaller in comparison to the target graph. Curticapean and Marx [16] modeled this setting by classifying the complexity of counting subgraphs from fixed pattern classes: Given any fixed class of graphs $\mathcal{H}$, they defined a problem #Sub($\mathcal{H}$) that asks, given a graph $H \in \mathcal{H}$ and a general graph G, to count the H-subgraphs in G. The parameter is $|V(H)|$. This problem is known to be polynomial-time solvable when the graphs in $\mathcal{H}$ do not contain arbitrarily large matchings – if they do contain arbitrarily large matchings, then #Sub($\mathcal{H}$) with parameter $|V(H)|$ is complete for the parameterized complexity class #W[1], i.e., the analogue of #P in parameterized complexity.

An analogously defined problem #Hom($\mathcal{H}$) of counting *homomorphisms* with patterns drawn from $\mathcal{H}$ was also classified by Dalmau and Jonsson [17]. Here, the tractability criterion is a constant bound on the *treewidth* of graphs in $\mathcal{H}$, a measure of the "tree-likeness" of H: The problem #Hom($\mathcal{H}$) is polynomial-time solvable when all graphs in $\mathcal{H}$ admit a constant upper bound on their treewidth, and the problem is #W[1]-hard otherwise with respect to the parameter $|V(H)|$. Here, a homomorphism from H to G is a function $h : V(H) \to V(G)$ such that $uv \in E(H)$ implies $h(u)h(v) \in E(G)$. Homomorphism counts from small patterns find direct applications in database theory, where they capture answer counts to so-called conjunctive queries [12]. It was also shown that #Hom($\mathcal{H}$) captures the complexity of other pattern counting problems, including that of counting subgraphs [14]: In a nutshell, (i) many pattern counting problems can be expressed as unique linear combinations of homomorphism counts from graphs H, and (ii) in many models of computation, such linear combinations turn out to be precisely as hard as their hardest terms. This motivates understanding the complexity of these individual terms, i.e., of homomorphism counts.

Lower bounds under ETH

Following the classification of #Sub($\mathcal{H}$) and #Hom($\mathcal{H}$), almost-tight quantitative bounds were obtained under the exponential-time hypothesis [23, 29]: For any graph H, there is an $O(n^{\mathrm{tw}(H)+1})$ time algorithm for counting homomorphisms from H into n-vertex target graphs, and assuming the exponential-time hypothesis, Marx ruled out $n^{o(\mathrm{tw}(H)/\log \mathrm{tw}(H))}$ time algorithms [33], recently revisited in [25, 15]. Through connections between homomorphism counts and other pattern counts [14], these bounds translate directly to other counting problems. For example, there is an $O(n^{\mathrm{vc}(H)})$ time algorithm for counting H-subgraphs in

an n-vertex graph G, where $\mathrm{vc}(H)$ is the vertex-cover number of H. As a consequence of the lower bound on counting homomorphisms, the exponential-time hypothesis rules out $n^{o(\mathrm{vc}(H)/\log \mathrm{vc}(H))}$ time algorithms [14] for counting H-subgraphs.

Thus, some slack remains between the known upper and conditional lower bounds on the exponents of pattern counting problems, and it would be desirable to settle the log-factor in the exponent of the running time. Moreover, one might also dare to ask for the precise exponent for concrete finite graphs H, such as K_3 (which amounts to triangle counting) or K_4 or C_6. Finally, let us stress that the lower bounds on the exponent are conditioned on the exponential-time hypothesis, an assumption that is *a priori* stronger than $\mathsf{P} \neq \mathsf{NP}$.

From counting problems to polynomials

Valiant's seminal papers [41, 43] studied the problem of counting perfect matchings from the perspective of both counting and algebraic complexity. Following the work of Komarath, Pandey, and Rahul [27], we consider an algebraic version of pattern counting problems.

For undirected graphs H and $n \in \mathbb{N}$, we consider the homomorphism polynomial $\mathsf{Hom}_{H,n}$ on variables $x_{i,j}$ for $i, j \in [n]$ and its set-multilinear version, the colorful subgraph polynomial $\mathsf{ColSub}_{H,n}$ on variables $x_{i,j}^{(e)}$ for $i, j \in [n]$ and $e \in E(H)$. The latter can often be handled more easily in proofs, while complexity results can be transferred between these two polynomials.

▶ Remark. We deviate slightly from the notation used by Komarath, Pandey, and Rahul [27], who defined a polynomial $\mathsf{ColIso}_{\mathcal{H},n}$ with variable indices that differ from ours. Our polynomial $\mathsf{ColSub}_{H,n}$ and their polynomial $\mathsf{ColIso}_{\mathcal{H},n}$ are obtained from each other by renaming variables. We consider our notation more intuitive, as it can be obtained from the homomorphism polynomial more directly and makes the set-multilinearity explicit.

The polynomial $\mathsf{Hom}_{H,n}$ can be viewed as the weighted homomorphism count from H into a complete n-vertex graph with generic indeterminates as edge-weights. Similarly, $\mathsf{ColSub}_{H,n}$ can be viewed as counting the color-preserving homomorphisms from a colorful graph H into a complete graph with indeterminate edge-weights and n vertices per color class. Formally, these multivariate polynomials are defined as

$$\mathsf{Hom}_{H,n} = \sum_{f:V(H)\to[n]} \prod_{uv\in E(H)} x_{f(u),f(v)},$$
$$\mathsf{ColSub}_{H,n} = \sum_{f:V(H)\to[n]} \prod_{uv\in E(H)} x_{f(u),f(v)}^{(uv)}.$$

The complexity of multivariate polynomials is commonly studied via algebraic circuits, first formalized by Valiant [41]. The efficiency of a circuit is usually quantified by its *size* (number of edges and gates) and its *depth* (number of layers). The size captures the total number of operations the circuit performs, and the *depth* roughly corresponds to *parallelism*. For convenience, we will instead consider the *product-depth*, which is the number of "multiplication layers" in the circuit. Refer to Section 2 for formal definitions. These efficiency measures define natural algebraic circuit complexity classes, for instance, VP – polynomials of polynomially bounded degree computable by circuits of size $\mathrm{poly}(n)$, and VNP – polynomials which can be expressed as a hypercube sum of VP circuits.

Valiant established that the permanent polynomial is complete for the class VNP, while its sibling, the determinant polynomial, is complete for a seemingly smaller class, VBP – the class of polynomially bounded algebraic branching programs (see Definition 6). In a recent line of work [11, 18, 31], homomorphism polynomials have been used to obtain natural polynomials which are complete for several well-studied algebraic circuit classes.

Monotone circuits

Monotone circuits over a field like $\mathbb{Z}$ are circuits that do not use negative constants, and hence computations performed by them cannot feature cancellations (Definition 7). Several important techniques for proving upper bounds on the complexity of polynomials (e.g., dynamic programming) directly yield monotone circuits. Compared to general computational models, lower bounds for monotone computation are much better understood, and many exponential lower bounds [36, 43, 24, 34, 19, 9] and strong algebraic complexity class separations [38, 22, 46, 39] are known. As a striking example, monotone variants of the algebraic complexity classes VP and VNP are proven to be different [39, 46]. In contrast, Hrubeš [21] showed that strong-enough monotone lower bounds of a special kind (called ϵ-*sensitive*) imply unconditional lower bounds for general arithmetic circuits!

In a fascinating recent work, Komarath, Pandey and Rahul [27] studied the monotone arithmetic circuit complexity of the polynomials $\mathsf{Hom}_{H,n}$ and discovered that this complexity is completely determined by the treewidth of the pattern graph H. More precisely, they show that the smallest monotone circuit computing $\mathsf{Hom}_{H,n}$ is of size $\Theta(n^{\mathrm{tw}(H)+1})$. Similarly, they show that algebraic branching programs for $\mathsf{Hom}_{H,n}$ are of size $\Theta(n^{\mathrm{pw}(H)+1})$, where $\mathrm{pw}(H)$ is the *pathwidth* of H, a linear version of treewidth. Moreover, they also consider the monotone formula complexity of $\mathsf{Hom}_{H,n}$ and show that it is $\Theta(n^{\mathrm{td}(H)+1})$, where $\mathrm{td}(H)$ is the *treedepth* of H, the minimum height of a tree on vertex set $V(H)$ that contains all edges of H in its tree-order. These results together show that, when considering homomorphism polynomials for fixed patterns H, the power of monotone computation is precisely characterized by graph-theoretic quantities of H: For natural and well-studied monotone computational models, the precise exponent c_H in the complexity $\Theta(n^{c_H})$ is the value of a natural and well-studied graph parameter of H.

Our results: Monotone bounded-depth models

In this paper, we investigate whether the correspondence between monotone circuit complexity and graph parameters can also be established for another restriction of monotone circuits, namely *bounded-depth* monotone circuits: Are there natural graph parameters that dictate the bounded-depth monotone complexity of $\mathsf{Hom}_{H,n}$?

Bounded-depth circuits are of central importance in algebraic complexity due to the phenomenon of *depth reduction*. In a sequence of works [44, 2, 26, 40], it was shown that any algebraic circuit of size s computing a polynomial of degree d can also be simulated by a product-depth Δ circuit of size $s^{O(d^{1/\Delta})}$. If the circuit was monotone to begin with, the resulting bounded-depth circuit is also monotone. Depth reduction also implies that strong enough lower bounds for bounded-depth circuits will lead to general circuit lower bounds – an exceptionally hard open question. A lot of work has gone into proving strong bounded-depth circuit lower bounds (see [35] for a survey). Recently, following the breakthrough result of Limaye, Srinivasan, and Tavenas [28], superpolynomial lower bounds have been shown for bounded-depth circuits (also see [7, 4]).

Studying the monotone bounded-depth complexity of $\mathsf{Hom}_{H,n}$ naturally leads us to define bounded-depth versions of treewidth, the Δ-*treewidth* $\mathrm{tw}_\Delta(H)$ for any fixed $\Delta \in \mathbb{N}$. These graph parameters ask to minimize the maximum bag size over all tree-decompositions of H, however, with the twist that only tree-decompositions with an underlying tree of height at most Δ are admissible. Their values interpolate between $|V(H)| - 1$ (when only height 1 is allowed) and $\mathrm{tw}(H)$ (when no height restrictions are imposed), and they are connected to the vertex-cover number in the special case $\Delta = 2$ (see Section 3). Bounded-depth variants

of treewidth implicitly appear in balancing techniques for tree-decompositions [10, 8], and the Δ-treewidth of paths also appears implicitly in divide-and-conquer schemes for iterated matrix multiplication in a bounded-depth setting [28]. A recent work of Adler and Fluck [1] studied a notion that bounds the width and depth simultaneously, which they call bounded depth treewidth. Our notion of $\mathrm{tw}_\Delta(H)$ only bounds the height of the tree decomposition to Δ. In particular, for a fixed Δ, there is always a tree decomposition of height Δ for any graph H, but with a possibly large treewidth.

We show that the Δ-treewidth of graphs completely characterizes the complexity of $\mathsf{Hom}_{H,n}$ for monotone circuits of product-depth at most Δ. For technical reasons described later, it is, however, not the Δ-treewidth of H itself that governs the complexity, but rather the Δ-treewidth of the graph $H^\dagger$ obtained by removing all vertices of degree at most 1. We call this the *pruned Δ-treewidth* ptw_Δ of a graph H. Our main result is as follows:

▶ **Theorem 1.** *Let $\Delta \geq 1$ and n be natural numbers, and let H be a fixed graph with pruned treewidth $\mathrm{ptw}_\Delta(H) \geq 1$. Then the polynomials $\mathsf{Hom}_{H,n}$ and $\mathsf{ColSub}_{H,n}$ have monotone circuits of size $O(n^{\mathrm{ptw}_\Delta(H)+1})$ and product-depth Δ. Moreover, any monotone circuit of product-depth Δ has size $\Omega(n^{\mathrm{ptw}_\Delta(H)+1})$.*

Note that any fixed pattern graph H on k vertices gives a homomorphism polynomial on n^k monomials, which has a trivial poly(n) sized monotone circuit of depth two. We stress that we wish to determine the precise *exponent* in this polynomial size: In general, k could be much larger than the pruned Δ-treewidth of H.

We also study the related computational model of algebraic branching programs (ABPs). An ABP is a directed acyclic graph with a source vertex s and a sink t, with edges between vertices labeled by variables or constants. The *size* of the ABP is the total number of vertices in the graph, and the length of the longest path from s to t is its length. We refer the reader to Section 2 for a formal definition of an ABP and its computation.

For a graph H, we define a new graph parameter called Δ-*pathwidth* $\mathrm{pw}_\Delta(H)$, which asks to minimize the bag size over all *path decompositions* of H where the underlying path is of length Δ. We defer the formal definition of these graph parameters to Section 3. Similar to the case of circuits, we show that the Δ-pathwidth of the pruned graph $H^\dagger$ (obtained by removing degree 1 vertices from H), which we call the *pruned Δ-pathwidth* ppw_Δ of H characterizes the monotone ABP of length Δ computing $\mathsf{Hom}_{H,n}$.

▶ **Theorem 2.** *Let $\Delta \geq 1$ and n be natural numbers and let H be a fixed graph with pruned Δ-pathwidth $\mathrm{ppw}_\Delta(H) \geq 1$ such that $\Delta \geq |E(H)|$. Then the polynomials $\mathsf{Hom}_{H,n}$ and $\mathsf{ColSub}_{H,n}$ can be computed by monotone algebraic branching programs of size $O(n^{\mathrm{ppw}_\Delta(H)+1})$ and length Δ. Moreover, any such program of length Δ has size $\Omega(n^{\mathrm{ppw}_\Delta(H)+1})$.*

For a length-Δ ABP to compute the polynomial $\mathsf{Hom}_{H,n}$ (or $\mathsf{ColSub}_{H,n}$), its length needs to be at least the degree of the polynomial (which is $|E(H)|$). Otherwise, we cannot even compute a single monomial. We note that the above theorem also implies a bound on the ABP *width*.

For a fixed pattern graph, it was shown in [27] that $\mathsf{Hom}_{H,n}$ and $\mathsf{ColSub}_{H,n}$ have the same "monotone complexity". We observe that the reduction holds even in the bounded-depth (bounded-length) case (Lemma 21), so we prove our results only for $\mathsf{ColSub}_{H,n}$. The upper bounds of Theorem 1 and Theorem 2 are shown in Section 4 and Section 5, respectively.

Our results: Monotone depth hierarchy

Finally, by turning our attention to pattern graphs of non-constant size, we can prove a depth hierarchy theorem for monotone circuits: Using the tight characterization from the above theorem and the properties of pruned Δ-treewidth, we are able to obtain the following.

▶ **Theorem 3.** *For any natural numbers n and $\Delta \geq 1$, there exists a pattern graph H_Δ of size $\Theta(n)$ such that $\mathsf{ColSub}_{H_\Delta,n}$ can be computed by a monotone circuit of size $\mathrm{poly}(n)$ and product-depth $\Delta + 1$, but every monotone circuit of product-depth Δ computing the polynomial needs size $n^{\Omega(n^{1/\Delta})}$.*

By general depth-reduction results, any monotone circuit of size $\mathrm{poly}(n)$ computing a polynomial of degree n can be flattened to a monotone circuit of product-depth Δ and size $n^{O(n^{1/\Delta})}$, showing that the above theorem is optimal.

We also note that a similar *near-optimal* statement with a lower bound of $\exp(n^{\Omega(1/\Delta)})$ can be obtained from earlier results provided the product-depth $\Delta = o(\log n / \log \log n)$ is small (see [13]). Our results improve upon this in two ways: Firstly, our $\Omega(\cdot)$ appears in the first rather than second exponent, thus yielding a stronger and optimal lower bound. Secondly, our results hold for any product-depth Δ.

2   Preliminaries

For a natural number $n \in \mathbb{N}$, we will use $[n]$ to refer to the set $\{1, \ldots, n\}$. We begin with some algebraic complexity and graph-theoretic preliminaries. Readers comfortable with these notions can safely skip this section.

2.1   Algebraic complexity theory

Algebraic circuits are analogous to Boolean circuits, where logical operators are replaced with underlying field operations. In this paper, we fix our field to be the rationals $\mathbb{Q}$.[1] To study the complexity of polynomials, Valiant formulated the algebraic complexity theory [41]. Here, we will only give relevant definitions, but we encourage interested readers to refer to surveys for a more comprehensive overview of the area [30, 35, 37].

▶ **Definition 4** (Algebraic Circuits). *An algebraic circuit C is a layered directed acyclic graph with a unique root node, called the output gate. Each leaf node of C is called an input gate and is labeled by one of the variables $x_1, \ldots, x_n$ or a field constant. Gates are either labeled $+$ or $\times$, and on every path from the output gate to some input gate, these gate types alternate. The (product) depth of C is the number of (multiplication) layers in the circuit, while its size is the number of vertices of the underlying graph. The circuit computes a polynomial: a $+(\times)$ gate computes the sum (product) of the polynomials computed by its children.*

▶ **Definition 5** (Skew Circuits). *An algebraic circuit is called skew if, for every multiplication gate, at most one of its children is an internal (non-input) gate.*

▶ **Definition 6** (Algebraic Branching Programs). *An Algebraic Branching Program (ABP) is a directed acyclic graph with edges labeled by a variable or a field constant. It has a designated source node s (of in-degree 0) and a sink node t (of out-degree 0). A path from s to t computes the product of all edge labels along the path. The polynomial computed by the ABP is the sum of the terms computed along all the paths from s to t. If all the constants in the ABP are non-negative, the ABP is monotone. The size of an ABP is the total number of vertices in the graph, and the length of the longest path from s to t is the length of the ABP.*

---

[1] Our results hold for any field by making appropriate changes to the definition of monotone computation so that cancellations are avoided.

Monotone computation, which is free of cancellations, can be simulated by algebraic circuits (branching programs) by restricting the choice of field constants.

▶ **Definition 7** (Monotone Circuits and ABPs). *A monotone circuit (ABP) is an algebraic circuit (ABP) where all the field constants are non-negative. The circuit (ABP) computes a monotone polynomial, where the coefficients of all the monomials are non-negative.*

▶ Remark 8. It is not hard to show that skew circuits and ABPs are essentially the same model, up to constant factors (e.g., see the discussion in [30]). In particular, an ABP of size s and length Δ can be converted to a skew circuit of size $O(s)$ and product-depth Δ. If the original ABP is monotone, the skew circuit is monotone as well.

Finally, we define *parse trees* to analyze the computation of each monomial in a circuit. The notion has appeared in several previous results [3, 24, 27, 32, 45].

▶ **Definition 9** (Parse Trees). *A parse tree $\mathcal{T}$ of an algebraic circuit C is obtained as follows:*
- *We include the root gate of C in $\mathcal{T}$.*
- *For every $+$ gate in $\mathcal{T}$, we arbitrarily include* any one *of its children in $\mathcal{T}$.*
- *For every $\times$ gate in $\mathcal{T}$, we include* all *of its children in $\mathcal{T}$.*
We call a parse tree reduced *if we ignore every $+$ gate, and its parent and (only) child are directly connected by an edge.*

▶ Remark 10. It is easy to see that every (reduced) parse tree is associated with a monomial of the polynomial computed by the circuit. For a (reduced) parse tree $\mathcal{T}$, let val($\mathcal{T}$) be its output. Then the polynomial computed by the circuit C is $\sum_{\mathcal{T}} \text{val}(\mathcal{T})$, where the sum is over all the parse trees of C. From here on, whenever we use the term parse tree, we mean the reduced parse tree.

2.2 Graph theory

In the following, let H be a graph. A vertex cover of H is a subset $C \subseteq V(H)$ such that for every edge $e \in E(H)$, some vertex $v \in C$ is an endpoint of e. The vertex-cover number of H is the minimum size of a vertex-cover in H.

▶ **Definition 11** (Tree-decomposition and treewidth). *A tree-decomposition of H is a tree T whose vertices are annotated with bags $\{X_t\}_{t \in V(T)}$, subject to the following conditions:*
1. *Every vertex v of H is in at least one bag.*
2. *For every edge uv in H there is a bag in T that contains both u and v.*
3. *For any vertex v of H, the subgraph of T induced by the bags containing v is a subtree.*
The width *of a tree decomposition T is $\max_{t \in V(T)} |X_t| - 1$. The* treewidth *of the graph H, denoted $\text{tw}(H)$, is the minimum width over all tree decompositions of H.*

▶ **Definition 12** (Path-decomposition and pathwidth). *A path-decomposition of a graph H is a tree-decomposition where the underlying tree is a path. The* width *of a path decomposition P is $\max_{t \in V(P)} |X_t| - 1$. The* pathwidth *of the graph H, denoted $\text{pw}(H)$, is the minimum width over all path decompositions of H.*

We usually consider trees with a designated root vertex. The height of such a tree is the number of vertices on a longest root-to-leaf path. We assume trees in tree-decompositions to be rooted in a way that minimizes their height.

3   Bounded versions of treewidth and pathwidth

In this section, we describe Δ-treewidth, our bounded-depth version of treewidth, and Δ-pathwidth, a bounded-length version of pathwidth. We relate these to other known graph parameters. Moreover, we show that full d-ary trees of depth Δ, while having Δ-treewidth 1, have $(\Delta - 1)$-treewidth $d - 1$. This behavior around the threshold Δ will ultimately allow us to conclude Theorem 3.

3.1   Connections to other graph parameters

First, recall the definition of Δ-treewidth from the introduction. We stress that a tree with a single node has height 1 according to our definition.

▶ **Definition 13.** *For fixed $\Delta \in \mathbb{N}$, the Δ-treewidth of a graph H, denoted by $\mathrm{tw}_\Delta(H)$, is the minimum width over all tree decompositions of H with underlying tree T of height at most Δ.*

While depth-restricted tree-decompositions did arise before in the literature [10], their depth was not fixed to concrete *constants* in these contexts, but rather to, say, $O(\log |V(H)|)$. In particular, differences between Δ-treewidth and $(\Delta - 1)$-treewidth were not considered.

Let us connect the Δ-treewidth of graphs to other graph parameters:

- The 1-treewidth of a graph H is merely its number of vertices $|V(H)|$, as the requirement on the height forces the tree-decomposition to consist of a single bag. On the other extreme, the $|V(H)|$-treewidth of H equals the treewidth of H.
- The 2-treewidth is already more curious: For any vertex-cover C, a tree-decomposition of height 2 for H can be obtained by placing C into a root bag X_r that is connected to bags X_t for $t \in V(H) \setminus C$, where X_t contains t and its neighbors, all of which are in C. This shows that the 2-treewidth of H is at most the vertex-cover number $\mathrm{vc}(H)$ of H. In fact, the 2-treewidth of H equals the so-called *vertex-integrity* of H (minus 1). This graph parameter is defined as $\min_{S \subseteq V(H)}(|S| + \max_C |V(C)|)$, where C ranges over all connected components in the graph $H - S$, see [5, 20].
- By balancing tree-decompositions [10], a universal constant c can be identified such that, for all graphs H on k vertices, the $c \log k$-treewidth of H is bounded by $4\mathrm{tw}(H) + 3$. That is, at the cost of increasing width by a constant multiplicative factor, tree-decompositions can be assumed to be of logarithmic height.

Our upper bound proof will show that the vertices of degree 1 can be removed safely from H without changing the bounded-depth complexity of $\mathsf{Hom}_{H,n}$. This holds essentially because such vertices and their incident edges can be assumed to be present in the leaves of a tree-decomposition; these leaves then do not contribute to the product-depth of the constructed circuit. This naturally leads to the notion of *pruned* Δ-treewidth.

▶ **Definition 14.** *Given a graph H, the pruning of H is the graph $H^\dagger$ defined by removing every vertex of degree 1 The* pruned Δ-*treewidth of a graph H, denoted by $\mathrm{ptw}_\Delta(H)$, is the Δ-treewidth of $H^\dagger$.*

We also define analogous *bounded-length* versions of pathwidth.

▶ **Definition 15.** *For fixed $\Delta \in \mathbb{N}$, the Δ-pathwidth of a graph H, denoted by $\mathrm{pw}_\Delta(H)$, is the minimum width over all path decompositions of H with underlying path P of length at most Δ.*

▶ **Definition 16.** *The* pruned Δ-*pathwidth of a graph H, denoted by $\mathrm{ppw}_\Delta(H)$, is the Δ-pathwidth of the pruning $H^\dagger$.*

3.2 Full d-ary trees

We conclude this section by exhibiting a pattern H whose Δ-treewidth shows a strong phase transition that we can exploit in our depth hierarchy theorem: Its Δ-treewidth is low, but even its $(\Delta - 1)$-treewidth is high. As it turns out, H can be chosen to be the full d-ary tree.

▶ **Theorem 17.** *Let Δ, d be positive integers and let T_Δ be the full d-ary tree of height Δ. Then $\mathrm{tw}_\Delta(T_\Delta) = 1$ whereas $\mathrm{tw}_{\Delta-1}(T_\Delta) \geq d - 1$.*

In order to prove the theorem, we first prove the following useful lemma for inductively bounding the Δ-treewidth of a given graph.

▶ **Lemma 18.** *For any integers d and Δ, if a graph G contains at least d disjoint connected subgraphs $G_1, G_2, \ldots, G_d$, and the $(\Delta - 1)$-treewidth of each of them is at least $d - 1$, then the Δ-treewidth of G is at least $d - 1$.*

Proof. Suppose T is a rooted tree-decomposition of graph G, and R is the root bag of T. We are going to prove that either the height of T is larger than Δ or the width of T is at least $d - 1$. There are two cases to consider:
1. R contains at least one vertex from each of the subgraphs $G_1, G_2, \ldots, G_d$.
2. R does not contain any vertex from (at least) one of the subgraphs $G_1, G_2, \ldots, G_d$.

In the first case, the size of R is at least d, so the width of T is at least $d - 1$. In the second case, we can assume without loss of generality that R does not contain any vertex from G_1. Let $T_1, T_2, \ldots, T_k$ be the subtrees obtained by removing R from T. Since R does not contain any vertex of G_1, at least one of $T_1, T_2, \ldots, T_k$ must contain some vertex from G_1. Suppose that the subtree is T_1. For any vertex v contained in both T_1 and G_1, since v is not in the root bag R, it must be the case that v is not contained in any other $T_2, \ldots, T_k$ as well. Similarly, every neighbor u of v in G_1 is also contained in T_1 as u is not contained in R, and there must be a bag in T which contains both u and v. Proceeding this way, we get that T_1 contains the whole of G_1, and the vertices from G_1 appear in no other subtree.

Removing vertices not in G_1 from each bag of T_1, we obtain a new tree T_1'. We claim that T_1' is a tree-decomposition of G_1. Indeed, all vertices of G_1 are in T_1, and the bags in T_1 that contain a vertex v form a connected component since T was a tree decomposition of G. So the same holds for T_1'. Moreover, for every edge (u, v) in G_1, there is a bag in T_1 that contains both u and v, so the same holds for T_1'.

Since the $(\Delta - 1)$-treewidth of G_1 is at least $d - 1$, either the height of T_1' is larger than $\Delta - 1$ or the width of T_1' is at least $d - 1$. Since T_1' was formed by removing vertices from T_1, the same holds for T_1. Consequently, either the height of T is larger than Δ or the width of T is at least $d - 1$. In both cases, the lemma holds.   ◀

We are now ready to prove Theorem 17.

Proof of Theorem 17. We have $\mathrm{tw}_\Delta(T_\Delta) = 1$ for all Δ, since T_Δ is a tree of height Δ. For the lower bound, consider the base case $\Delta = 2$. The height-1 tree-decomposition of the height-2 full d-ary tree T_2 has only one bag, and this bag contains all the vertices from T_2. Hence, its treewidth is $d \geq d - 1$.

Assume by induction that the theorem holds for all $2 \leq \Delta \leq k$ for $k \in \mathbb{N}$. Then, for $\Delta = k + 1$, consider the height-$(k + 1)$ full d-ary tree T_{k+1}. Removing the root node of T_{k+1} yields d pairwise disjoint height-k full d-ary trees. By our inductive assumption, the $(k - 1)$-treewidth of each of these trees is at least $d - 1$. Then by Lemma 18, the k-treewidth of T_{k+1} is at least $d - 1$.   ◀

4   Upper bounds in Theorem 1 and Theorem 2

We prove the upper bound in Theorem 1. First, we require additional standard notation for tree-decompositions: We consider T to be rooted with a choice of root that minimizes its height. Given a tree-decomposition of H with underlying tree T and bags $\{X_t\}_{t \in V(T)}$, write $\gamma(t) := \bigcup_{s \geq t} X_s$ for the cone at t, where s ranges over all descendants of t in the tree T.

Our second definition is more technical and specific to the dynamic programming approach we use to compute homomorphism polynomials in a bottom-up manner: It allows us to track *where* in the tree-decomposition an edge contributes to a monomial of the final polynomial. We say that an *edge-representation* of H in T is a function rep : $E(H) \to V(T)$ that assigns to each edge of H a node in T such that $\{u, v\} \subseteq X_{\mathrm{rep}(uv)}$ for all $uv \in E(H)$. Note that each edge $uv \in E(H)$ is already entirely contained in *at least one* bag by the definition of a tree-decomposition; the function rep simply chooses one such bag for each edge.

Given an edge-representation rep, we define the rep-height of T (which will be the product-depth of the constructed circuit) as the maximum number of "active" nodes t on a root-to-leaf path in T, where we call a node t active iff

- there are distinct $e, e' \in E(H)$ with $\mathrm{rep}(e) = \mathrm{rep}(e') = t$, or
- there is at least one $e \in E(H)$ with $\mathrm{rep}(e) = t$ and t has a child, or
- t has at least two children.

In our dynamic programming approach that proceeds bottom-up on a tree-decomposition, only active nodes require multiplication gates; the rep-height will thus amount to the overall product-depth of the circuit.

▶ **Lemma 19.** *Let H be a graph with a tree-decomposition consisting of tree T and bags $\{X_t\}_{t \in V(T)}$, and let* rep *be an edge-representation of H in T. Then there are circuits for* $\mathsf{Hom}_{H,n}$ *and* $\mathsf{ColIso}_{H,n}$ *with product-depth equal to the* rep-*height of T and $O(|V(T)| \cdot n^w)$ gates for $\max_{t \in V(T)} |X_t| = w$.*

Proof. We describe the circuit for $\mathsf{Hom}_{H,n}$ and remark that the circuit for $\mathsf{ColSub}_{H,n}$ can be constructed analogously. Considering T to be rooted, and proceeding from the leaves of T to the root, we inductively compute polynomials $\mathsf{Restr}_{t,h}$ for nodes $t \in V(T)$ and functions $h : X_t \to [n]$. The polynomials are defined as

$$\mathsf{Restr}_{t,h} = \sum_{\substack{f : \gamma(t) \to [n] \\ f \text{ extends } h}} \prod_{\substack{uv \in E(H) \\ \mathrm{rep}(uv) \geq t}} x_{f(u), f(v)}.$$

Here, we write $s \geq t$ to denote that s is a descendant of t in T. Note that $\mathsf{Restr}_{t,h}$ is the restriction of $\mathsf{Hom}_{H,n}$ to homomorphisms f that extend a given homomorphism h for the bag at t, such that only those edges feature in the monomials that are represented in the cone $\gamma(t)$. Then $\mathsf{Hom}_{H,n}$ is the sum of $\mathsf{Restr}_{r,h}$ over all $h : X_r \to [n]$ at the root r of T.

We show how to compute the polynomials $\mathsf{Restr}_{p,h}$ for nodes $p \in V(T)$. Let $p \in V(T)$ be a node with children $N \subseteq V(T)$, possibly with $N = \emptyset$ if p is a leaf. Assume that $\mathsf{Restr}_{t,h'}$ is known for all $t \in N$ and functions $h' : X_t \to [n]$. Then we have

$$\mathsf{Restr}_{p,h} = \left(\prod_{\substack{uv \in E(H) \\ \mathrm{rep}(uv) = p}} x_{h(u), h(v)} \right) \cdot \prod_{t \in N} \sum_{\substack{h' : X_t \to [n] \\ \text{agreeing with } h \\ \text{on } X_t \cap X_p}} \mathsf{Restr}_{t,h'}. \tag{1}$$

From this construction of the circuit, the size bound claimed in the lemma is obvious. Let us investigate its product-depth: In the final circuit computing $\mathsf{Hom}_{H,n}$, every path from the

output gate to an input gate corresponds to a path in T from the root to a leaf. Analyzing (1), we see that every node t on this path contributes 1 to the product-depth iff t is active under the edge-representation rep. Indeed, a leaf p only contributes to the product-depth if two edges $e, e' \in E(H)$ are represented in its bag, i.e., $\text{rep}(e) = \text{rep}(e') = p$, as then there is a nontrivial product in the first product (over uv, shown in parentheses in (1)). A node p with one child only contributes if at least one edge is represented in its bag, as then the product between the parentheses and the remaining factor is nontrivial. A node p with at least two children always contributes to the product-depth.

To show that the circuit correctly computes $\text{Hom}_{H,n}$, we need to show that the recursive expression for $\text{Restr}_{p,h}$ in (1) is correct. Note that every edge is represented by rep in exactly one bag and thus appears precisely once in a monomial. Because X_p is a separator in H, any function $f : \gamma(p) \to [n]$ gives rise to $|N|$ functions $f_t : \gamma(t) \to [n]$ for $t \in N$ that all agree on their values for X_p (that is, on their values on $X_p \cap X_t$) and can otherwise be chosen independently. Conversely, any ensemble of such consistent functions can be merged to a function $h : \gamma(p) \to [n]$. The product over all $t \in N$ as in (1) thus yields $\text{Restr}_{p,h}$.   ◄

Finally, to prove the upper bound in Theorem 1, let $H^\dagger$ be the pruning of H. We obtain a tree-decomposition for $H^\dagger$ with underlying tree $T^\dagger$ of height Δ and width w witnessing that $\text{ptw}_\Delta(H) = w$. Note that $w \geq 1$ by the requirements of the theorem, so $T^\dagger$ is not empty. Then we construct a tree-decomposition for H with tree T and an edge-representation rep of H in T of rep-height Δ.

To this end, for each vertex $v \in V(H)$ of degree 1, with unique neighbor $u \in V(H)$, we add a node t' to $T^\dagger$ with bag $X_{t'} = \{v, u\}$. This node is inserted into $T^\dagger$ as follows:

- If u lies in $V(H^\dagger)$, choose some node $t \in T^\dagger$ with $u \in X_t$ and make t' its child.
- If u is not contained in $V(H^\dagger)$, then u and v are both of degree 1 and form a connected component consisting of a singleton edge. In this case, make t' a child of the root of $T^\dagger$.

Choose an arbitrary representation rep of H in the resulting tree-decomposition with tree T and observe that its rep-height is at most the height Δ of $T^\dagger$, even though the height of T may be $\Delta + 1$: The bags added for degree-1 vertices and their incident edges do not contribute towards the rep-height, as they are leaf nodes and represent single edges. The upper bound thus follows from Lemma 19.

▶ **Remark 20.** The construction from Lemma 19 also yields an ABP of length $|V(T)|$ and size $O(|V(T)| \cdot n^w)$ when given a path-decomposition T of H with maximum bag size w. To see this, note that the product over $t \in N$ in (1) involves only a single factor when T is a path-decomposition, so (1) overall amounts to a skew-multiplication of a single monomial with the recursively computed polynomial.

5   Lower bounds in Theorem 1 and Theorem 2

We adapt the lower bound proofs of [27] to prove the lower bounds in our theorems. Recall that proving the lower bound for $\text{ColSub}_{H,n}$ is enough, since we can use a circuit computing $\text{Hom}_{H,n}$ to obtain a circuit computing $\text{ColSub}_{H,n}$ without changing the depth of the circuit using [27, Lemma 8]. We summarize the results here for completeness.

▶ **Lemma 21.** *Let k, Δ be positive integers and H be a fixed pattern graph on k vertices.*

- *If there is a monotone circuit of product-depth Δ and size s for $\text{ColSub}_{H,n}$, then there is such a circuit of size $O(s)$ for $\text{Hom}_{H,n}$.*
- *If there is a monotone circuit of product-depth Δ and size s for $\text{Hom}_{H,n'}$, then there is such a circuit of size $O(s^{|E(H)|})$ for $\text{ColSub}_{H,n}$, where $n' = kn$.*

The results also hold if circuits are replaced by ABPs, and product-depth is replaced by the length of the ABP, provided the length is at least the degree of the polynomials.

Proof. Given a monotone circuit of product-depth Δ that computes $\mathsf{ColSub}_{H,n}$, we replace each variable $x^{(uv)}_{f(u),f(v)}$ with $x_{f(u),f(v)}$ if $f(u) \neq f(v)$ and 0 otherwise. The circuit now computes $\mathsf{Hom}_{H,n}$.

For the other direction, let C be the monotone circuit of product-depth Δ computing the Hom_H polynomial over the vertex set $[k] \times [n]$. Note that a homomorphism ϕ from H to the complete graph on $[k] \times [n]$ maps a vertex $u \in [k]$, to (v, p) where $v \in [k]$ and $p \in [n]$. We introduce auxiliary variables y_{uv} for each edge $uv \in E(H)$. For $u, v \in [k]$ and $p, q \in [n]$, we replace the variable $x_{(u,p),(v,q)}$ with $x^{(uv)}_{p,q} y_{uv}$ if $uv \in E(H)$ and 0 otherwise.

Let C' be the new circuit obtained after the replacement, and consider the partial derivative $D := \frac{\partial^{|E(H)|}}{\partial y_{e_1} \cdots \partial y_{e_{|E(H)|}}} C'$, with respect to all the edge variables of H. Note that every monomial in D contains at least one variable corresponding to each edge of H. Further, set $y_{uv} = 0$ in D for all $uv \in E(H)$. This ensures that every monomial in $D|_{y_{uv}=0}$ contains exactly one variable corresponding to every edge of H, i.e., it counts only the color-preserving homomorphisms. The coefficient of each monomial is $|aut(H)|$, the number of automorphisms of H, and dividing by this number gives us $\mathsf{ColSub}_{H,n}$.

We can compute D using the partial derivatives' sum and product rules applied to every gate in a bottom-up fashion. For a gate g, we maintain both g and $\partial_{y_e} g$. The partial derivative of a sum gate, $\partial_{y_e} \sum_i g_i = \sum_i \partial_{y_e} g_i$ is straightforward and does not increase the depth. For a product gate, the derivative $\partial_{y_e} \prod_i g_i = \sum_i \left(\partial_{y_e} g_i \prod_{j \neq i} g_j \right)$ increases the depth by one, but this can be absorbed in the sum layer above. Note that the product-depth does not change in both cases. A partial derivative with respect to a single variable increases the circuit size by a factor of s. Hence, the final circuit for D is of size $O(s^{|E(H)|})$, and has product-depth Δ, the same as C.

We also note that both constructions preserve monotonicity. Moreover, if the original circuit C was *skew* (i.e., an ABP), then so is the final circuit D. From Remark 8, we obtain the same results for ABPs as well. ◀
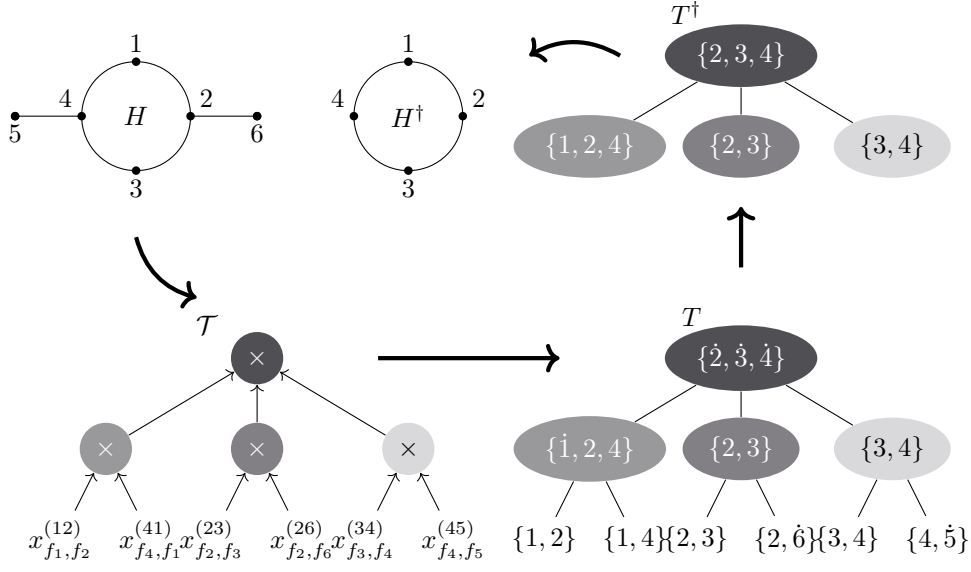
5.1   Tree decompositions from parse trees

Consider a pattern graph H on the vertex set $V(H) := [k]$. An alternative and more intuitive way to think about the n-th colored subgraph isomorphism polynomial $\mathsf{ColSub}_{H,n}$ is to consider the blown-up graph G, where each vertex $u \in [k]$ of H is replaced by a "cloud" of n vertices $C_u := \{(u, 1), \ldots, (u, n)\}$. Every edge $uv \in E(H)$ is replaced by a complete bipartite graph between C_u and C_v with an appropriate label for each of the n^2 edges; that is, an edge between (u, i) and (v, j) is labeled $x^{(uv)}_{i,j}$ where $u, v \in [k]$ and $i, j \in [n]$. The polynomial $\mathsf{ColSub}_{H,n}$ is now obtained by choosing a copy of H in G by picking a vertex from every cloud using a function $f : V(H) \to [n]$, and adding the monomial

$$m = \prod_{uv \in E(H)} x^{(uv)}_{f(u),f(v)}.$$

We say that the monomial m above is supported on a set $S \subseteq [k] \times [n]$ if every element of S looks like $(u, f(u))$ for $u \in [k]$. The polynomial $\mathsf{ColSub}_{H,n}$ is the sum over all such monomials m

$$\mathsf{ColSub}_{H,n} = \sum_{f:V(H) \to [n]} \prod_{uv \in E(H)} x^{(uv)}_{f(u),f(v)}.$$

▷ **Claim 22.** Let Δ be a natural number and $\mathcal{T}$ be a monotone parse tree of product-depth Δ computing a monomial m of $\mathsf{ColSub}_{H,n}$. Let $H^\dagger$ be the pruned graph obtained by removing all degree-1 vertices from H. We can extract from $\mathcal{T}$ a tree decomposition of $H^\dagger$ with underlying tree $T^\dagger$ of height Δ.

Figure 1 Extracting a tree decomposition of height 2 for $H^\dagger$ from a parse-tree of product-depth 2 for a monomial of $\mathsf{ColSub}_{H,n}$. We have for all $i \in [6]$, $f_i := f(i) \in [n]$.

Proof. Suppose that the monomial m is supported on vertices $(u, f(u))$ where $u \in [k]$ and $f : [k] \to [n]$ is a function. The parse tree $\mathcal{T}$ has height $\Delta + 1$. Note that since $\mathsf{ColSub}_{H,n}$ has $0/1$ coefficients, we can assume that a multiplication gate has only non-constant terms as its children. We build the tree decomposition bottom-up. We "mark" certain vertices in the bags created during this procedure. All such marks are dropped at the end (see Figure 1).

1. For an input gate $x^{(uv)}_{f(u),f(v)}$, we add the bag $\{u, v\}$ as a leaf in the tree decomposition. We *mark* all the vertices of degree 1. The rest are *unmarked*.

2. Let g be a multiplication gate. Suppose $X_1, \ldots, X_m$ are the bags corresponding to the children of g (that we have already constructed) and let $U_i \subseteq X_i$ be the *unmarked* elements of X_i. We then add the bag $X_g := \bigcup_{i \in [m]} U_i$ as the root of $X_1, \ldots, X_m$. If there are vertices $(u, f(u))$ such that the monomial computed at g includes all the edges incident on $(u, f(u))$ in the copy of H that f picked, we *mark* all such vertices u in the bag X_g.

3. Finally, after applying the procedure in the previous step to all the gates, we drop the bags (and edges) corresponding to input gates.

We claim that the tree decomposition just constructed, with underlying tree $T^\dagger$ and bags $\{X_u\}_{u \in V(T^\dagger)}$, is a tree decomposition of $H^\dagger$. *All* the edges of H were covered at the leaf bags (that we finally dropped), as they must be present in the monomial. Since only the degree-1 vertices in a leaf bag were *marked*, the parent bags of the leaves (which we include in our tree decomposition) have exactly the vertices of $H^\dagger$, and thus cover all its edges.

We mark (forget) a vertex only after multiplying all its incident edges. Hence, the sub-graph induced by a vertex u (in $H^\dagger$) is *connected* in $T^\dagger$ and is, in fact, a subtree. As every multiplication gate of the parse tree has exactly one associated bag, the procedure does indeed result in a tree decomposition of $H^\dagger$ of height Δ.   ◁

5.2   Lower bounds for ColSub$_{H,n}$

▶ **Theorem 23.** *Let Δ be a natural number and H be a pattern graph. Any monotone circuit of* product-depth Δ *computing the polynomial* ColSub$_{H,n}$ *has size* $\Omega(n^{\mathrm{ptw}_\Delta(H)+1})$.

Proof. Let C be the monotone circuit computing ColSub$_{H,n}$, and let the pruned Δ-treewidth of H, $\mathrm{ptw}_\Delta(H) = t$. Consider a monomial m of ColSub$_{H,n}$ supported on vertices $(u, f(u))$ for $u \in [k]$ and $f : [k] \to [n]$. Let $\mathcal{T}$ be a parse tree of C associated with m. Now, Claim 22 gives a tree decomposition of $H^\dagger$ with tree $T^\dagger$ and bags $\{X_u\}_{u \in V(T^\dagger)}$. Consequently, there is a bag X of size at least $t + 1$ in the tree decomposition. Without loss of generality, we assume that $|X| = t + 1$. If it is greater, we will only obtain a better lower bound. We also assume that the vertices in the bag are $1, \ldots, t + 1$ (relabeling the vertices of H does not change the complexity of ColSub$_{H,n}$). Let the gate in $\mathcal{T}$ associated with X be g.

We show that only a "few" monomials can contain g in their parse tree. More precisely, we claim that any monomial m' (other than m) that contains g in its parse tree is supported on vertices $\{(u, f(u))\}_{u \in [t+1]}$. Suppose not. Let m' have a parse tree $\mathcal{T}'$ with gate g in it but vertex $(u, f'(u))$ for some $u \in [t + 1]$, with $f(u) \neq f'(u)$. Recall that we obtained the tree decomposition using the parse tree $\mathcal{T}$ of m. For a gate g in a parse tree, we denote by $\mathcal{T}_g$ the subtree rooted at g. Note that if two parse trees contain a multiplication gate g, all the children of g are the same in both parse trees. We now analyze two cases:

1. The vertex u is marked at the bag associated with g: There are at least two children g_1, g_2 of g in $\mathcal{T}$ that compute monomials with $(u, f(u))$ in them. This holds because there are no degree-1 vertices in the bags. If g_1 in $\mathcal{T}'$ contains the vertex $(u, f'(u))$, we replace $\mathcal{T}'_{g_2}$ with $\mathcal{T}_{g_2}$. Similarly, in the other case, when g_2 contains $(u, f'(u))$. If both g_1, g_2 do not contain $(u, f'(u))$ in $\mathcal{T}'$, we arbitrarily replace $\mathcal{T}'_{g_1}$ (say) with $\mathcal{T}_{g_1}$.
2. The vertex u is not marked at the bag associated with g: The vertex $(u, f(u))$ appears in $\mathcal{T}_g$ *as well as* outside $\mathcal{T}_g$. In $\mathcal{T}'$, if $(u, f'(u))$ appears in $\mathcal{T}'_g$, we replace $\mathcal{T}_g$ with $\mathcal{T}'_g$ in $\mathcal{T}$. Otherwise, we replace $\mathcal{T}'_g$ with $\mathcal{T}_g$ in $\mathcal{T}'$.

In all cases, we obtain a valid parse tree $\mathcal{T}''$ of C that produces a monomial supported on $(u, f(u))$ *and* $(u, f'(u))$. This leads to a contradiction, since the monomial produced by $\mathcal{T}''$ is spurious and cannot be cancelled because the circuit is monotone. Every monomial (parse tree) m has a gate g whose corresponding bag has at least $t + 1$ vertices. And any other monomial m' (parse tree) that contains this gate g must share at least $t + 1$ vertices in its support with m. Thus, the maximum number of monomials containing this gate g equals the number of colored isomorphisms that fix $t + 1$ vertices, which is n^{k-t-1}. Recall that there are n^k monomials in ColSub$_{H,n}$, and so we need at least n^{t+1} gates in the circuit.   ◀

The lower bound proof for algebraic branching programs is very similar.

▶ **Theorem 24.** *Let Δ be a positive integer and H be a pattern graph such that $\Delta \geq |E(H)|$. Any monotone ABP of* length Δ *computing the polynomial* ColSub$_{H,n}$ *has size* $\Omega(n^{\mathrm{ppw}_\Delta(H)+1})$.

Proof. As mentioned earlier in Remark 8, the size-s monotone ABP of length Δ computing ColSub$_{H,n}$ has an equivalent monotone skew-circuit C of size $O(s)$ and product-depth Δ. Consider a monomial m of ColSub$_{H,n}$ supported on vertices $(u, f(u))$ for $u \in [k]$ and $f : [k] \to [n]$. Let $\mathcal{T}$ be a parse tree of C associated with m.

We observe that the procedure described in the proof of Claim 22 extracts a length-Δ *path decomposition* of the pruned graph $H^\dagger$ instead: as the circuit is skew, all the multiplication gates in $\mathcal{T}$ have at most one non-leaf child. Since we finally dropped the bags corresponding to input gates in our procedure, the tree decomposition obtained is a path decomposition!

Taking the pruned Δ-pathwidth of H to be t, the same proof implies that the number of monomials containing gate g is n^{k-t-1}, thus implying a size lower bound of n^{t+1}.   ◀

6   Depth Hierarchy

Combining our previous results allows us to prove a depth-hierarchy theorem for bounded-depth monotone algebraic circuits.[2]

▶ **Theorem 25.**   *For all integers n and Δ, there exists a pattern graph H_Δ such that $\mathsf{ColSub}_{H_\Delta,n}$ can be computed by a monotone product-depth $(\Delta+1)$ circuit of size $O(n|H_\Delta|)$ but any product-depth Δ monotone circuit computing the polynomial needs size $n^{\Omega(|H|^{1/\Delta})}$.*

Proof. For an integer d, let $H_\Delta := T_{\Delta+2}$ be the full d-ary tree of height $\Delta+2$. Note that $d = \Theta(|H_\Delta|^{1/\Delta})$. The pruned $(\Delta+1)$-treewidth of H_Δ is equal to the $(\Delta+1)$-treewidth of the full d-ary tree of height $(\Delta+1)$. That is, $\mathrm{ptw}_{\Delta+1}(H_\Delta) = \mathrm{tw}_{\Delta+1}(T_{\Delta+1}) = 1$. So by Lemma 19, there exists a monotone circuit of product-depth $\Delta+1$ and size $O(n|H_\Delta|)$, which computes $\mathsf{ColSub}_{H_\Delta,n}$.

On the other hand, we have $\mathrm{ptw}_\Delta(H_\Delta) = \mathrm{tw}_\Delta(T_{\Delta+1}) \geq d-1$ by Theorem 17. Hence, by Theorem 23, every monotone circuit of product-depth Δ computing $\mathsf{ColSub}_{H_\Delta,n}$ necessarily has size at least $\Omega(n^{\mathrm{ptw}_\Delta(H_\Delta)+1}) = n^{\Omega(|H_\Delta|^{1/\Delta})}$.   ◀

If we consider the case when the pattern graph is of size $\Theta(n)$ in Theorem 25, we obtain the depth hierarchy result in Theorem 3. As an aside, we note that an analogous depth hierarchy cannot be obtained for the polynomials $\mathsf{Hom}_{H,n}$ using our methods, as the blow up in the size given by Lemma 21 is exponential, when $|H| = \Theta(n)$ is not a constant.

────── **References** ──────

1   Isolde Adler and Eva Fluck. Monotonicity of the cops and robber game for bounded depth treewidth. In *49th International Symposium on Mathematical Foundations of Computer Science*, volume 306 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 6, 18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/lipics.mfcs.2024.6`.

2   Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 67–75. IEEE Computer Society, 2008. `doi:10.1109/FOCS.2008.32`.

3   Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theoretical Computer Science*, 209(1-2):47–86, 1998. `doi:10.1016/S0304-3975(97)00227-2`.

4   Prashanth Amireddy, Ankit Garg, Neeraj Kayal, Chandan Saha, and Bhargav Thankey. Low-depth arithmetic circuit lower bounds: bypassing set-multilinearization. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/lipics.icalp.2023.12`.

5   C. A. Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs—a comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1:13–22, 1987. URL: `https://combinatorialpress.com/article/jcmcc/Volume%201/vol-001-paper%202.pdf`.

6   C. S. Bhargav, Shiteng Chen, Radu Curticapean, and Prateek Dwivedi. Monotone bounded-depth complexity of homomorphism polynomials, 2025. `doi:10.48550/arXiv.2505.22894`.

─────────

[2] A similar hierarchy can also be shown for monotone ABPs.

7   C. S. Bhargav, Sagnik Dutta, and Nitin Saxena. Improved lower bound, and proof barrier, for constant depth algebraic circuits. *ACM Transactions on Computation Theory*, 16(4):23:1–23:22, 2024. `doi:10.1145/3689957`.

8   Hans L. Bodlaender and Torben Hagerup. Tree decompositions of small diameter. In *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic, August 24-28, 1998, Proceedings*, volume 1450 of *Lecture Notes in Comput. Sci.*, pages 702–712. Springer, Berlin, 1998. `doi:10.1007/BFb0055821`.

9   Bruno Pasqualotto Cavalar, Mrinal Kumar, and Benjamin Rossman. Monotone circuit lower bounds from robust sunflowers. *Algorithmica*, 84(12):3655–3685, 2022. `doi:10.1007/s00453-022-01000-3`.

10   Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Optimal reachability and a space-time tradeoff for distance queries in constant-treewidth graphs. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.ESA.2016.28`.

11   Prasad Chaugule, Nutan Limaye, and Aditya Varre. Variants of homomorphism polynomials complete for algebraic complexity classes. *ACM Transactions on Computation Theory*, 13(4):21:1–21:26, 2021. `doi:10.1145/3470858`.

12   Hubie Chen and Stefan Mengel. Counting answers to existential positive queries: A complexity classification. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 315–326. ACM, 2016. `doi:10.1145/2902251.2902279`.

13   Suryajith Chillara, Christian Engels, Nutan Limaye, and Srikanth Srinivasan. A near-optimal depth-hierarchy theorem for small-depth multilinear circuits. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 934–945. IEEE Computer Society, 2018. `doi:10.1109/FOCS.2018.00092`.

14   Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. `doi:10.1145/3055399.3055502`.

15   Radu Curticapean, Simon Döring, Daniel Neuen, and Jiaheng Wang. Can you link up with treewidth? In *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*, volume 327 of *LIPIcs*, pages 28:1–28:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. `doi:10.4230/LIPIcs.STACS.2025.28`.

16   Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139. IEEE Computer Society, 2014. `doi:10.1109/FOCS.2014.22`.

17   Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004. `doi:10.1016/j.tcs.2004.08.008`.

18   Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. Homomorphism polynomials complete for VP. *Chicago Journal of Theoretical Computer Science*, 2016:Art. 3, 25, 2016. `doi:10.4086/cjtcs.2016.003`.

19   Sergey B. Gashkov and Igor' S. Sergeev. A method for deriving lower bounds for the complexity of monotone arithmetic circuits computing real polynomials. *Sbornik: Mathematics*, 203(10):1411, October 2012. Publisher: IOP Publishing. `doi:10.1070/SM2012v203n10ABEH004270`.

20   Tatsuya Gima, Tesshu Hanaka, Yasuaki Kobayashi, Ryota Murai, Hirotaka Ono, and Yota Otachi. Structural parameterizations of vertex integrity. *Theoretical Computer Science*, 1024:114954, 2025. `doi:10.1016/j.tcs.2024.114954`.

21 Pavel Hrubeš. On ϵ-sensitive monotone computations. *Computational Complexity*, 29(2):6, 2020. `doi:10.1007/s00037-020-00196-6`.

22 Pavel Hrubeš and Amir Yehudayoff. On isoperimetric profiles and computational complexity. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 89:1–89:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.ICALP.2016.89`.

23 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62:367–375, 2001. `doi:10.1006/jcss.2000.1727`.

24 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the Association for Computing Machinery*, 29(3):874–897, 1982. `doi:10.1145/322326.322341`.

25 C. S. Karthik, Dániel Marx, Marcin Pilipczuk, and Uéverton Souza. Conditional lower bounds for sparse parameterized 2-csp: A streamlined proof. In *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 383–395. SIAM, 2024. `doi:10.1137/1.9781611977936.35`.

26 Pascal Koiran. Arithmetic circuits: the chasm at depth four gets wider. *Theoretical Computer Science*, 448:56–65, 2012. `doi:10.1016/j.tcs.2012.03.041`.

27 Balagopal Komarath, Anurag Pandey, and Chengot Sankaramenon Rahul. Monotone arithmetic complexity of graph homomorphism polynomials. *Algorithmica. An International Journal in Computer Science*, 85(9):2554–2579, 2023. `doi:10.1007/s00453-023-01108-0`.

28 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 804–814. IEEE, 2021. `doi:10.1109/FOCS52979.2021.00083`.

29 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the European Association for Theoretical Computer Science. EATCS*, 105:41–72, 2011. URL: `https://www.ii.uib.no/~daniello/papers/surveyETH.pdf`.

30 Meena Mahajan. Algebraic complexity classes. In *Perspectives in computational complexity*, volume 26 of *Progr. Comput. Sci. Appl. Logic*, pages 51–75. Birkhäuser/Springer, Cham, 2014. `doi:10.1007/978-3-319-05446-9_4`.

31 Meena Mahajan and Nitin Saurabh. Some complete and intermediate polynomials in algebraic complexity theory. *Theory of Computing Systems*, 62(3):622–652, 2018. `doi:10.1007/s00224-016-9740-y`.

32 Guillaume Malod and Natacha Portier. Characterizing valiant's algebraic complexity classes. *Journal of Complexity*, 24(1):16–38, 2008. `doi:10.1016/j.jco.2006.09.006`.

33 Dániel Marx. Can you beat treewidth? *Theory of Computing. An Open Access Journal*, 6(1):85–112, 2010. `doi:10.4086/toc.2010.v006a005`.

34 Ran Raz and Amir Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *Journal of Computer and System Sciences*, 77(1):167–190, 2011. `doi:10.1016/j.jcss.2010.06.013`.

35 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. v9.0.3, 2021. URL: `https://github.com/dasarpmar/lowerbounds-survey/releases`.

36 C.-P. Schnorr. A lower bound on the number of additions in monotone computations. *Theoretical Computer Science*, 2(3):305–315, 1976. `doi:10.1016/0304-3975(76)90083-9`.

37 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. `doi:10.1561/0400000039`.

38 Marc Snir. On the size complexity of monotone formulas. In *Automata, languages and programming (Proc. Seventh Internat. Colloq., Noordwijkerhout, 1980)*, volume 85 of *Lecture Notes in Comput. Sci.*, pages 621–631. Springer, Berlin-New York, 1980. `doi:10.1007/3-540-10003-2_103`.

39   Srikanth Srinivasan. Strongly exponential separation between monotone VP and monotone VNP. *ACM Trans. Comput. Theory*, 12(4):23:1–23:12, 2020. `doi:10.1145/3417758`.

40   Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Information and Computation*, 240:2–11, 2015. `doi:10.1016/j.ic.2014.09.004`.

41   L. G. Valiant. Completeness classes in algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. `doi:10.1145/800135.804419`.

42   Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. `doi:10.1016/0304-3975(79)90044-6`.

43   Leslie G. Valiant. Negation can be exponentially powerful. *Theoretical Computer Science*, 12(3):303–314, 1980. `doi:10.1016/0304-3975(80)90060-2`.

44   Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983. `doi:10.1137/0212043`.

45   H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21(4):655–670, 1992. `doi:10.1137/0221040`.

46   Amir Yehudayoff. Separating monotone VP and VNP. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 425–429. ACM, 2019. `doi:10.1145/3313276.3316311`.