Monotone Arithmetic Complexity of Graph Homomorphism Polynomials

Balagopal Komarath
Indian Institute of Technology Gandhinagar, India

Anurag Pandey
Department of Computer Science, Universität des Saarlances, Saarland Informatics Campus, Saarbrücken, Germany

Chengot Sankaramenon Rahul
School of Mathematics and Computer Science, Indian Institute of Technology Goa, India

Abstract

We study homomorphism polynomials, which are polynomials that enumerate all homomorphisms from a pattern graph H to n-vertex graphs. These polynomials have received a lot of attention recently for their crucial role in several new algorithms for counting and detecting graph patterns, and also for obtaining natural polynomial families which are complete for algebraic complexity classes VBP, VP, and VNP. We discover that, in the monotone setting, the formula complexity, the ABP complexity, and the circuit complexity of such polynomial families are exactly characterized by the treedepth, the pathwidth, and the treewidth of the pattern graph respectively.

Furthermore, we establish a single, unified framework, using our characterization, to collect several known results that were obtained independently via different methods. For instance, we attain superpolynomial separations between circuits, ABPs, and formulas in the monotone setting, where the polynomial families separating the classes all correspond to well-studied combinatorial problems. Moreover, our proofs rediscover fine-grained separations between these models for constant-degree polynomials.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Computational complexity and cryptography; Theory of computation → Circuit complexity; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Homomorphism polynomials, Monotone complexity, Algebraic complexity, Graph algorithms, Fine-grained complexity, Fixed-parameter algorithms and complexity, Treewidth, Pathwidth, Treedepth, Graph homomorphisms, Algebraic circuits, Algebraic branching programs, Algebraic formulas

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.83

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2011.04778`

Acknowledgements We thank Christian Engels and Marc Roth for bringing missing and incorrect references to our attention. We thank Markus Bläser for valuable discussions. We also thank anonymous reviewers for their valuable comments on the draft that helped improve the presentation of our results.

1 Introduction

This work is a culmination of exploration of four themes in combinatorics, algorithm design, and algebraic complexity – graph algorithms, homomorphism polynomials, graph parameters, and monotone computations. While each of these themes are of independent interest, a strong interplay among them has become quite apparent in recent years, and has lead to several new advancements in algorithms and complexity.

The first theme is *graph algorithms*, where the algorithms that are relevant to this work are those corresponding to pattern detection and counting. Loosely speaking, in such problems, we look for the "presence" of a graph H, called the *pattern graph* in another graph G, called the *host graph*[1]. The notions of presence of one graph in another graph that have been the most prevalent are subgraph isomorphism, induced subraph isomorphism, and homomorphism. In detection and counting algorithms for subgraph isomorphism (resp. induced subgraph isomorphism), we want to detect and count subgraphs (resp. induced subraphs) of an n-vertex host graph which is *isomorphic* to the pattern graph. Whereas, while looking for the occurrence of the pattern graph in the host graph, if we relax the mapping to allow multiple vertices in the pattern graph to be mapped to a single vertex in the host graph, while preserving the edge relations, we get a *homomorphism* of the pattern graph in the host graph. When the notion is homomorphism, we are interested in detecting and counting homomorphisms from the pattern graph to the host graph.

All the above mentioned problems have found many applications, both in theory and practice. For instance, detecting and counting (induced) subgraph isomorphisms are used in extremal graph theory in the study of dense graphs and quasirandom graphs [10, 29, 30], and in many applications which boil down to analyzing real-world graphs. This includes finding protein interactions in biomolecular networks [1, 34], finding active chemicals of interest in the research for drug synthesis [4], advertisement targeting by finding and counting certain social structure patterns in the analysis of social networks [24, 46], and also finding user patterns for recommender services on platforms like Amazon and Yelp [47]. The homomorphism counting problem appears for instance, in statistical physics via partition functions, in graph property testing, and extremal graph theory (see [5] for a survey). When both the host graph and the pattern graph are parts of the input, then it can be shown that all these pattern detection problems are NP-complete, since they generalize the CLIQUE problem, whereas the corresponding counting problems are all known to be #P-complete. On the other hand, in almost all the real-world applications pointed out above, we have a fixed pattern graph which we are trying to detect or count in a given host graph. Thus, in this work, we focus on the setting when the pattern graph is a fixed graph, and the host graph is a part of the input. Here, in the word-RAM model with $O(\log(n))$-bit words, since the pattern graph is of a fixed size, say its number of vertices is k, all the above problems can be solved using a trivial algorithm, based on exhaustive search, that runs in time $O(n^k)$ and space $O(1)$ where n is the number of vertices in the host graph. However, in almost all the real-world applications pointed out above, the host graph is massive. Hence, one desires faster algorithms, preferably linear or even sub-linear algorithms. Hence, there is a lot of interest and advances in improving upon this trivial algorithm using ideas from combinatorics, algebraic circuits, and machine learning (see [33] for a survey, also see [3, 28] and references therein). These problems are also very interesting from the perspective of complexity theory. For instance, in the Fixed Parameter Tractability community, it is conjectured that the best known algorithms for variants of these pattern detection and counting problems cannot be further improved. If true, this would imply $\mathsf{P} \neq \mathsf{NP}$ in a rather strong way (see [27, 32, 38, 26]). Recently, it was discovered by Curticapean, Dell and Marx [11] (also see [13]) that counting subgraph isomorphisms corresponds to counting linear combinations of homomorphisms, and hence establishing that it is sufficient to just consider the homomorphism counting problem. Several advances have been attained by considering homomorphism problems [3, 23, 25, 17, 44].

[1] In this paper, the pattern graph and the host graph are always simple, undirected graphs.

Our second theme – *homomorphism polynomials*, corresponds to one of the most successful ways by which progress has been made towards homomorphism problems, and hence towards (induced) subgraph isomorphism problems too. For a pattern graph H and a host graph G, the *homomorphism polynomial* is the polynomial whose monomials enumerate homomorphisms from H to G. For using homomorphism polynomials to obtain new algorithms for the above graph problems for a pattern graph H, it suffices to consider the homomorphism polynomial from H to K_n, the clique on n vertices. Thus, we call the homomorphism polynomial from H to K_n as the *homomorphism polynomial of H* (see Definition 10). It turns out that efficient *arithmetic circuit* (see Section 2 for the definition) constructions of homomorphism polynomials can be used to obtain almost all known best algorithms for detecting induced subgraph isomorphisms as well (See [3], also [23, 25, 17, 44]). Therefore, the study of homomorphism polynomials is a crucial area of study within graph pattern detection and counting. Homomorphism polynomials also turned out to be extremely useful in algebraic complexity theory in the quest for finding natural complete polynomial families for the complexity class VP (the algebraic complexity analog of P. See [39]). Homomorphism polynomials yield polynomial families that are complete for important algebraic complexity classes such as VBP, VP, and VNP through a single framework, simply by considering different pattern graphs [15, 31, 9], making it important not only from the perspective of algorithm design, but also from the perspective of complexity theory (also see [16], where homomorphism polynomials yield several dichotomy theorems in algebraic complexity theory). Homomorphism polynomials are thus the central focus of this work too.

In our third theme, that is of *graph parameters*, the parameters relevant to this work are treewidth, pathwidth, and treedepth of graphs. Treewidth, loosely speaking, is a measure of how far a graph is from being a tree; similarly the pathwidth measures how far a graph is from being a path; and treedepth measures how far a graph is from being a star (see Section 2 for definitions). The connection between the parameters treewidth and pathwidth and counting homomorphisms was first explored in [13] (also see [3, 31]), where it is shown that when H has bounded treewidth, then there are small-sized arithmetic circuits for homomorphism polynomials, whereas when H has small pathwidth, then the corresponding homomorphism polynomials have small-sized *algebraic branching programs* (ABPs, defined in Section 2). These improved constructions for circuits and ABPs are one of the main source of advancement in finding improved algorithms for these pattern detection and counting problems [3, 23, 25, 17, 44]. The connection between the subgraph isomorphism problem and treedepth has also been explored in the context of Boolean computational models in [27] and [26], where they discuss treewidth and treedepth based upper bounds and lower bounds for counting subgraph isomorphisms for Boolean circuits and formulas. In the context of parameterized counting complexity of these problems, a stronger connection between treewidth and the complexity of counting homomorphisms was established by Dalmau and Jonsson [12] who showed a dichotomy theorem stating that when the pattern graph has bounded treewidth, then we can count homomorphisms in polynomial time. Otherwise, we can show #W[1]-hardness. Now we turn our attention to the other major application of homomorphism polynomials. It turns out that the parameters treewidth and pathwidth played a crucial role in the construction of natural complete polynomials for VP, VBP, and VNP too. More specifically, the complete polynomials for VP and VBP were homomorphism polynomials corresponding to specific pattern graphs of bounded treewidth and bounded pathwidth respectively. Thus, one sees that in the context of these pattern detection and counting problems, and in the homomorphism polynomials of interest, these graph parameters of the pattern graph ubiquitously pop up over and over again as the crucial complexity parameters. This made us wonder, to what extent do these parameters dictate the complexity of these problems and, in particular, the homomorphism polynomials.

Towards this, the starting point of our work is the observation that all the improved arithmetic circuit constructions of homomorphism polynomials based on treewidth and pathwidth, do not use any negative constants in the circuit. This brings us to our final theme, that is, of *monotone computation.*

In the Boolean setting, monotone computations refer to computations that do not involve the negation operation, the NOT gate. Similarly, in the algebraic setting for computing polynomials, monotone arithmetic circuits correspond to circuits that do not use negative constants or subtraction gates, that is, there is no cancellation involved in the circuit. Monotone computations are interesting for three main reasons. First reason is that monotone operations have favorable stability properties with respect to rounding errors [40]. Secondly, an improved monotone circuit requires combinatorial insights since it does not use any cancellations and algebraic identities. Such constructions often reveal interesting combinatorial structure about the problem at hand and lead to new combinatorial algorithms. for the problem at hand. For instance, Grenet's monotone algebraic branching program construction of the permanent polynomial [19]. Finally, the weakness of monotone models resulting from the lack of cancellations, makes them significantly easier to understand. Their computations are much better understood, both in the Boolean and the algebraic setting, and hence have been used as a starting point towards understanding the general model. Indeed, we know exponential lower bounds as well as separations between important complexity classes when we restrict ourselves to the monotone setting, in contrast to the embarrassingly poor understanding that we have in the general setting. In particular, Schnorr [40] showed an exponential lower bound for the clique polynomial, which happens to be a special case of homomorphism polynomial, where the pattern graph is a clique. Moreover, we know superpolynomial separations between complexity classes[2] mVF and mVBP [41], mVBP and mVP [20], and finally, very recently, between mVP and mVNP [45]. None of these separations are settled in the non-monotone setting. Moreover, the best known circuit lower bound in the non-monotone setting is just $\Omega(n \log n)$ [2]. Thus, it makes sense to study monotone models, both from the perspective of upper bounds – due to special interests in algorithms based on improved monotone circuits – as well as lower bounds – as a first step towards getting lower bounds for the general model. Many times, the best upper bounds come first in the monotone setting. For instance, the best known upper bound for the permanent polynomial for ABPs is a monotone construction [19]. We note that for computing homomorphism polynomials, Curticapean, Dell, and Marx gave an $O(n^{\omega})$ upper bound for all pattern graphs with treewidth $= 2$ [11], and an $O(n^{k\omega/3})$ upper bound is known for all k-vertex graphs [35], where ω is the exponent of matrix multiplication. However, for sufficiently large pattern graphs with treewidth > 2, which is almost all pattern graphs, the monotone constructions are the best known.

This motivated us to pursue the concrete question of whether these treewidth and pathwidth based improved monotone arithmetic circuits for homomorphisms polynomials can further be improved, and to what extent would these graph parameters dictate it. The answer that we discovered turns out to be conceptually quite satisfying, and settles the problem completely.

[2] mVF, mVBP and mVP refer to the class of polynomial families computable by poly-sized monotone arithmetic formulas, monotone algebraic branching programs, and monotone arithmetic circuits, respectively. mVNP refers to the monotone analog of the complexity class VNP.

1.1 Our contributions

In this work, we fully solve the question of monotone complexity of homomorphism polynomials by showing that they are completely determined by the aforementioned graph parameters.

For arithmetic circuits, we discover that the treewidth of the pattern graph exactly determines the monotone complexity of its homomorphism polynomial, by establishing that the treewidth based upper bound in [13, 31, 3] is also a lower bound.

In what follows, the monotone arithmetic circuit (resp. ABP or formula) complexity of a polynomial refers to the size of the smallest monotone arithmetic circuit (resp. ABP or formula) computing the polynomial. Here *size* of an arithmetic circuit (resp. ABP or formula) is the number of edges in the circuit (resp. ABP or formula). See Section 2 for detailed definitions. The underlying field is $\mathbb{Q}$.

▶ **Theorem 1.** *The monotone arithmetic circuit complexity of the homomorphism polynomial for a pattern graph H is $\Theta(n^{tw(H)+1})$, where $tw(H)$ is the treewidth of H.*

In the case of algebraic branching programs, we find that the pathwidth of the pattern graph is the parameter controlling the monotone complexity of its homomorphism polynomial, again, by proving a lower bound that exactly matches the pathwidth based upper bound in [13, 31, 3].

▶ **Theorem 2.** *The monotone ABP complexity of the homomorphism polynomial for a pattern graph H is $\Theta(n^{pw(H)+1})$, where $pw(H)$ is the pathwidth of H.*

Finally, for monotone formulas, it is the treedepth of the pattern graph which governs the complexity of its homomorphism polynomial.

▶ **Theorem 3.** *The monotone formula complexity of the homomorphism polynomial for a pattern graph H is $\Theta(n^{td(H)})$, where $td(H)$ is the treedepth of H.*

Hence, we resolve the question of monotone complexity of homomorphism polynomials completely by showing that treewidth, pathwidth and treedepth exactly characterize the complexity of homomorphism polynomials for arithmetic circuits, ABPs, and arithmetic formulas respectively. This also answers the conceptual question raised earlier asking to what extent do these graph parameters dictate the complexity of these homomorphism polynomials.

The characterization, in addition to giving several new lower bounds, in particular, also allows us to collect, through a unified framework, several classical and recent results for the monotone complexity of polynomials, obtained independently using different methods, over several decades. This is simply because the above theorems imply that for every family of pattern graph with high treewidth, pathwidth, or treedepth, the corresponding homomorphism polynomial family will have high circuit complexity, ABP complexity, or formula complexity, respectively.

- As a first example, applying the observations made in the proofs of Theorem 1 and Theorem 2 to polynomial families that count colored isomorphisms from pattern families (as opposed to fixed size pattern graphs), we obtain an alternative proof for the superpolynomial separation between monotone arithmetic circuits and monotone algebraic branching programs, first discovered by Hrubes and Yehudayoff [20]. We believe that our proofs are conceptually much simpler.

- In the same spirit, we also manage to reattain the superpolynomial separation between algebraic branching programs and arithmetic formulas in the monotone setting, using simple applications of Theorem 2 and Theorem 3. This was previously known as a consequence of a result by Snir [41].
- Our characterization, in particular, also rediscovers the fine-grained separations between all these models for constant-degree polynomials. That is, for every constant d, we get a polynomial family such that it is computable by linear-sized monotone arithmetic circuits, but any monotone ABP computing it must be of size at least n^d. Analogously, for every constant d, we get a polynomial family computable by linear-sized monotone ABP but any monotone arithmetic formula computing it must be of size at least n^d. Earlier, such fine-grained separations could be obtained by applying the results of [41] and [20] together.
- Another simple application of Theorem 1 yields an exponential lower bound against monotone arithmetic circuits for the clique polynomial $\mathrm{CL}_{2n,n}$, which enumerates all cliques of size n in a $2n$-vertex clique, first proved by Schnorr [40, Theorem 4.4].

▶ Remark 4. Note that, Theorems 1, 2, and 3 have pattern graph H fixed and hence of constant size. Thus, the theorems pertain to constant degree polynomials and they do not yield superpolynomial lower bounds on homomorphism polynomials. Unlike the works that show exponential lower bounds on monotone polynomials, for instance [6, 7, 8, 42], our main goal is to obtain *tight* lower bounds on *homomorphism polynomials* for *fixed size* pattern graphs. Our proofs, however, go via the colored subgraph isomorphism polynomials for which we also get superpolynomial lower bounds by considering pattern graphs of non-constant size, see Definition 19, Theorem 20, and Theorem 21.

1.2 Limitations of known techniques

It is worth noting that in the Boolean setting, while there are known upper bounds and lower bounds based on these graph parameters for circuits and formulas for (colored) subgraph isomorphism problem, there is still an asymptotic gap between the upper bound and the corresponding lower bounds in these Boolean models (see [27, 26]). Thus, we cannot hope to translate the lower bounds in Boolean setting to the algebraic setting to get the lower bounds exactly matching the upper bounds.

An overwhelming majority of monotone arithmetic lower bounds exploit the so-called decomposition theorem (also refered to as normal form or representation theorem; see for e.g. [39, Lemma 8.7]) including the exponential monotone lower bounds for permanent [21, 37], separation between monotone VP and monotone VNP [45], as well as the superpolynomial separation between monotone arithmetic circuits and monotone ABPs [20]. However, a direct application of such a decomposition theorem fails to yield the *exact* characterization that the above theorems manage to achieve. We argue in Section 3 why these methods give asymptotically weaker lower bounds than what we desire to obtain.

For ABPs, Fournier, Malod, Szusterman, and Tavenas [18] recently gave a rank-based lower bound method for the monotone setting, inspired by [36] given in the non-commutative setting. However, in their models, the edge labels are homogeneous linear forms, which makes their method unsuitable for the tight lower bounds that we were looking for.

Finally, an approach that seems different from all these aforementioned approaches, including ours, is the one based on the concept of separating sets introduced by Schnorr [40], which also gave an exponential lower bound on the clique polynomial $\mathrm{CL}_{2n,n}$, which enumerates all cliques of size n in a $2n$-vertex clique [40, Theorem 4.4]. When the pattern is a clique, his technique gives the optimal lower bound. However, it can be shown that, for other pattern graphs H, his proof technique, too, falls short in proving the lower bound that matches the upper bound (see full version for details). In fact, unlike Schnorr's lower bound techniques, our proofs, at a very high level, follow an abstract argument that can be shown to always give the optimal lower bound on the number of addition gates (see full version for details).

1.3 Proof ideas

At a high level, proofs to all our main theorems follow the same strategy. We discuss the case of circuits, since it already illuminates the overall idea.

The starting point towards the discovery of our proof is the fact that the upper bounds for homomorphism polynomials are all monotone and they use an optimal *tree decomposition* (see Definition 13) of the pattern graph to build the circuit. Our initial observation, which is a simple consequence of the above fact, is that *every* parse tree of such a circuit reflects a tree decomposition used to build the circuit. Moreover, since in these upper bound constructions (see [3, 13]), the whole circuit construction stems from a single tree-decomposition, all the parse trees correspond to the same tree decomposition of the pattern graph.

The above observation led us to the key question that sets the path towards our proofs:

▶ **Problem 1.** *Does an arbitrary parse tree of an arbitrary monotone circuit computing the homomorphism polynomial for an arbitrary pattern H allows us to recover a tree-decomposition of H?*

It turns out that answering the above question for homomorphism polynomials in the affirmative is difficult and most likely false. This is due of presence of non-multilinear monomials in homomorphism polynomials which correspond to those homomorphic images of the pattern graphs which are simpler than the pattern graph itself (see Definition 10). These monomials may loose the information about the pattern graph altogether, since such monomials may also result from homomorphism polynomials of other patterns. Thus, in order to be able to recover the tree decomposition, it seems imperative to work with the subgraph isomorphism polynomial instead. At a very high-level, our proof successfully answers the question – *in how many monomial computations can a single gate participate?* To give a precise bound on this, we need an even more structured polynomial – *the colored subgraph isomorphism polynomial*, where a monomial apart from encoding the pattern graph we begin with, also encodes the precise mapping from the pattern graph to the host graph that gives rise to the monomial (see Definition 11).

Before going further into the proof, one might wonder whether by going from homomorphism polynomial to the colored subgraph isomorphism polynomial, we have made the polynomial too structured as compared to the homomorphism polynomial. That is, whether the subgraph isomorphism polynomial is much harder than the homomorphism polynomial and a lower bound against the former would not yield a desired lower bound for the latter. Our first technical contribution is establishing that this is not so. We show that, as long as the the pattern graph is of constant size, the colored[3] homomorphism polynomial and colored subgraph isomorphism polynomial have the same monotone circuit complexity, monotone ABP complexity and monotone formula complexity (Lemma 15). For this, we first note

[3] It can be easily shown that the homomorphism polynomial and its colored counterpart have the same complexity.

that going from colored subgraph isomorphism polynomial to homomorphism polynomial is straight-forward. For the other direction, we note that the colored homomorphism polynomial contains a superset of the set of monomials of the colored subgraph isomorphism polynomial. To remove the other monomials present, we carefully introduce weights corresponding to edge variables and then use partial derivatives w.r.t those weight variables. We finally show that such partial derivatives can indeed be done efficiently in the the monotone circuits, monotone ABPs, as well as monotone formulas; the case of formulas require the most care.

Having established that we can indeed work with the colored subgraph isomorphism polynomial, we move on to answer Problem 1 for the colored subraph isomorphism polynomial in affirmative. Our main technical contribution is to show that, given an arbitrary parse tree of an arbitrary circuit computing the colored subgraph isomorphism polynomial for a pattern graph, there exists simple algorithm to recover its tree decomposition. The algorithm works in a bottom up fashion starting with the leaves of the parse tree and moves towards the root using a precise way to combine the *bags* of the gates it encounters along the way (see Theorem 16).

Once we have the tree decomposition, we use this tree decomposition to show that if the number of vertices in H is k, and its treewidth is $tw(H)$, a gate can participate in the computation of at most $n^{k-tw(H)-1}$ monomials. We show this by using a weakness of monotone computation that, due to the absence of cancellations, any circuit computing a polynomial cannot compute an invalid submonomial[4] at any intermediate gate along the way. This weakness is exploited in almost all known monotone lower bounds, and dates back to Jerrum and Snir [21].

For proving the lower bounds in Theorem 2 and 3 for formulas and ABPs, we are able to use the same framework as above. Instead of constructing a tree decomposition using the parse trees, we construct a *path decomposition* (Definition 13) in case of ABPs, and an *elimination tree* (Definition 14) in case of formulas. If we start with an ABP instead, it is easy to see that the same algorithm for tree decomposition yields a path instead, and, hence, we get a path decomposition (see Theorem 17). For formulas, we give a different simple algorithm to show that an elimination forest of H can be constructed using the parse tree (see Theorem 18). Using the same weakness of monotone computation, we conclude that the number of monomials whose computation a gate can participate in is upper bounded by $n^{k-pw(H)-1}$ in case of ABPs, and by $n^{k-td(H)}$ in case of formulas, where $pw(H)$ and $td(H)$ denote the pathwidth and the treedepth of H respectively.

To obtain the upper bounds claimed in Theorem 1, 2 and 3, we note that the upper bounds for circuits and ABPs that were already known are both monotone constructions, which go via the tree decomposition and the path decomposition of the pattern graph in case of circuits and ABPs respectively [13, 3, 31]. We give the formula upper bound using the elimination tree of the pattern graph. A treedepth based monotone formula upper bound is folklore in the Boolean setting.

To prove the separation between monotone complexity classes, we first note that the lower bounds on colored subgraph isomorphism polynomials also hold for pattern grpahs H of non-constant size. Consequently, pattern graphs with high pathwidth but low treewidth yield superpolynomial separation between circuits and ABPs (discussed in Theorem 20), whereas pattern graphs with high treedepth but low pathwidth give superpolynomial separation between ABPs and formulas (described in Theorem 21). For the first separation, we use a

[4] An invalid submonomial is a monomial m which does not divide any of the monomial of the target polynomial.

tree as the pattern graph, whereas for the second separation, we use path as the pattern graph. For an exponential lower bound on the clique polynomial, we simply apply Theorem 1 in combination with the fact that the treewidth of a clique on n vertices is $n-1$.

To simplify the presentation, we assume that the pattern graph is connected. The complexity for a disconnected pattern is the maximum of the complexity of its connected components.

2 Preliminaries

For basic notions in graph theory, we refer the readers to [43, 14]. We first give some definitions that set up our objects of computation and the models of computation.

▶ **Definition 5.** *A polynomial over $\mathbb{Q}$ is called* monotone *if all its coefficients are non-negative.*

Compact representations of polynomials such as the following are usually used by algorithms.

▶ **Definition 6.** *An* arithmetic circuit *over the variables $x_1, \ldots, x_n$ is a rooted DAG where each source node (also called an input gate) is labeled by one of the variables x_i or a constant $a \in \mathbb{Q}$. All other nodes (called gates) are labeled with either $+$ (addition) or $\times$ (multiplication). The circuit computes a polynomial over $\mathbb{Q}[x_1, \ldots, x_n]$ in the usual fashion. The circuit is called* monotone *if all constants are non-negative. The circuit is a* skew circuit *if for all $\times$ gates, at least one of the inputs is a variable or a constant. The circuit is a* formula *if all gates have out-degree at most one. The* size *of a circuit or skew circuit or formula is the number of edges in the circuit. The* depth *of a circuit is the number of edges in the longest path from the root to an input gate.*

Instead of skew circuits, a model that is equivalent in terms of power is usually studied in algebraic complexity.

▶ **Definition 7.** *An* Algebraic Branching Program (ABP) *is a DAG with a unique source node s and a unique sink node t. Each edge is labeled with a variable from $x_1, \ldots, x_n$ or a constant $a \in \mathbb{Q}$. Each path in the DAG from s to t corresponds to a term obtained by multiplying all the edge labels on that path. The polynomial computed by the ABP is the sum of all terms over all paths from s to t. The ABP is called* monotone *if all constants are non-negative. The* size *of the ABP is the number of edges.*

It is well-known that for any (monotone) polynomial, the size of the smallest (monotone) ABP and the size of the smallest (monotone) skew circuit are within constant factors of each other. In this paper, we will use the skew circuit definition in our proofs.

In this paper, we look at families of polynomials $(p_n)_{n \geq 0}$ and the optimal size and depth of the models computing them. In this case, the size and depth are functions of n and we are only interested in the asymptotic growth rate of these functions.

For simplicity, when proving lower bounds, we assume that all non-input gates have exactly two incoming edges (both may be from the same gate). This assumption increases the size by at most a constant factor and may increase the depth by at most a logarithmic factor.

The families of polynomials that we look at in this paper enumerate graph homomorphisms or colored isomorphisms. We first define these notions.

▶ **Definition 8.** *For graphs H and G, a* homomorphism *from H to G is a function $\phi : V(H) \mapsto V(G)$ such that $\{i, j\} \in E(H)$ implies $\{\phi(i), \phi(j)\} \in E(G)$. For an edge $e = \{i, j\}$ in H, we use $\phi(e)$ to denote $\{\phi(i), \phi(j)\}$.*

▶ **Definition 9.** *Let H be a k-vertex graph where its vertices are labeled by $[k]$ and let G be a graph where each vertex has a color in $[k]$. Then, a* colored isomorphism *of H in G is a subgraph of G isomorphic to H such that all vertices in the subgraph have different colors and for each edge $\{i, j\}$ in H, there is an edge in the subgraph between vertices colored i and j.*

Now, we are ready to define our main object of computation, the homomorphism polynomial.

▶ **Definition 10.** *For a pattern graph H on k vertices, the n-th* homomorphism polynomial *for H is a polynomial on $\binom{n}{2}$ variables x_e where $e = \{u, v\}$ for $u, v \in [n]$.*

$$\mathsf{Hom}_{H,n} = \sum_{\phi} \prod_{e \in E(H)} x_{\phi(e)}$$

where ϕ ranges over all homomorphisms from H to K_n.

Next, we define the colored isomorphism polynomial which will be crucial in our proofs. This polynomial enumerates all colored isomorphisms from a pattern to a host graph where there are n vertices of each color. This polynomial can be used to count colored isomorphisms in n-vertex host graphs by setting the variables corresponding to edges not in the host graph to 0.

▶ **Definition 11.** *For a pattern graph H on k vertices, the n-th* colored isomorphism polynomial *for H is a polynomial on $|E(H)|n^2$ variables x_e where $e = \{(i, u), (j, v)\}$ for $u, v \in [n]$ and $\{i, j\} \in E(H)$.*

$$\mathsf{ColIso}_{H,n} = \sum_{u_1,\ldots,u_k \in [n]} \prod_{\{i,j\} \in E(H)} x_{\{(i,u_i),(j,u_j)\}}$$

We notice that the labeling of H does not affect the complexity of ColIso_H. Given the polynomial ColIso_H for some labeling of H and if ψ is a relabeling of H, then the polynomial ColIso_H for the new labeling can be obtained by the substitution $x_{\{(i,u),(j,v)\}} \mapsto x_{\{(\psi(i),u),(\psi(j),v)\}}$.

For our main proofs, we need a way to analyze how a monomial is being computed in a circuit, an ABP, or a formula. For this, we use the notion of parse trees.

▶ **Definition 12.** *Let g be a gate in a circuit C. A* parse tree *rooted at g is any rooted tree which can be obtained by the following procedure, duplicating gates in C as necessary to preserve the tree structure.*

1. *The gate g is the root of the tree.*
2. *If there is a multiplication gate g in the tree, include all its children in the circuit as its children in the tree.*
3. *If there is an addition gate g in the tree, pick an arbitrary child of g in the circuit and include it in the tree.*

If the root gate of a parse tree is not mentioned, then it is assumed to be the output gate of the circuit. A parse tree witnesses the computation of some term. We note that if C is a formula, then any gate can occur at most once in any parse tree in C.

Given a parse tree T that contains a gate g, we use T_g to denote the subtree of T rooted at g. The tree obtained by removing T_g from T is called the tree outside T_g in T. Note that we can replace T_g in T with any parse tree rooted at g to obtain another parse tree.

Similarly, if we have two parse trees T and T' that both contain the same multiplication gate g from the circuit, then we can replace the left or right subtree of T_g with the left or right subtree of T'_g to obtain another parse tree. This is because both the left and right child of g in both parse trees are the same and therefore we can apply the aforementioned replacement.

Now, we define the graph parameters most crucial to our work. They are the parameters that turn out to exactly characterize the complexity of the above polynomial families in the models of computations defined above.

▶ **Definition 13.** *A* tree decomposition *of H is a tree where each vertex (called a* bag*) in the tree is a subset of vertices of H. This tree must satisfy two properties.*

1. *For every edge $\{i, j\}$ in H, there must be at least one bag in the tree that contains both i and j.*
2. *For any vertex i in H, the subgraph of the tree decomposition induced by all bags containing i must be a subtree. This subtree is called the* subtree induced by i.

The size of a tree decomposition is the size of the largest bag minus one. The treewidth *of H is the size of a smallest tree decomposition of H.*

A tree decomposition is called a path decomposition *if it is a path. The* pathwidth *of H is the size of a smallest path decomposition.*

▶ **Definition 14.** *For a connected graph H, an* elimination tree *of H is a rooted, directed tree that can be constructed by arbitrarily picking a vertex u in H and adding edges from the roots of elimination trees of connected components of $H - u$ to the root vertex labeled u. In particular, if H is a single vertex, then the elimination tree of H is the same single vertex graph.*

The depth *of an elimination tree is the number of vertices in the longest path from a leaf to the root. The* treedepth *of H is the depth of the smallest depth elimination tree of H.*

We note that for any graph H, its elimination tree contains exactly $|V(H)|$ vertices. All edges in the tree are directed towards the root and the vertices of the tree are uniquely labeled with the vertices of H. If T is an elimination tree for the connected graph H, then all edges in H are between vertices that are in an ancestor-descendant relationship in T.

We now state some basic facts about treewidth, pathwidth, and treedepth. We combine these facts with the lower bounds in Section 3 to obtain, for any constant k, constant-degree polynomial families having linear size circuits (ABPs resp.) but requiring $\Omega(n^k)$ size ABPs (formulas resp.), thus giving us a fine-grained separation between these models. Note that, for constant-degree polynomials, such polynomial factor separations are the best one could ask for between formulas, ABPs, and circuits, since all constant-degree polynomials are computable by polynomial-sized formulas. Moreover, we use these facts to obtain superpolynomial separations between these models for high degree polynomials.

▶ **Fact 1.** *For all graphs H, we have $tw(H) \leq pw(H) \leq td(H) - 1$.*

▶ **Fact 2.** *For any number $p \geq 1$, there is a tree X_k on $k = \frac{1}{6}(5 \cdot 3^p - 3)$ vertices that have pathwidth p.*

Proof. The tree X_2 is simply an edge. The tree X_k for $p \geq 2$ is obtained by connecting three copies of the constructed trees with pathwidth $p - 1$ to a new root vertex. ◀

▶ **Fact 3.** *All paths have pathwidth 1 and the k-vertex path has treedepth $\lceil \log_2(k + 1) \rceil$.*

3 Algebraic complexity of homomorphism polynomials

In this section, we prove Theorems 1, 2 and 3. thus achieving our main result, that is, the exact characterization for the monotone complexity of homomorphism polynomials.

▶ **Lemma 15.** *For any fixed pattern H, both the homomorphism polynomial for H and the colored isomorphism polynomial for H have the same (asymptotically) monotone arithmetic circuit complexity, monotone ABP complexity, and monotone formula complexity.*

Proof. First, we show how to obtain a circuit that computes ColIso_H from one that computes Hom_H. We introduce new variables w_e for each $e \in E(H)$. Let C be a circuit that computes Hom_H over the vertex set $[k] \times [n]$. We substitute $x_{\{(i,u),(j,v)\}}$ with $x_{\{(i,u),(j,v)\}} w_{\{i,j\}}$ for all i, j, u, v if $\{i, j\} \in E(H)$. Otherwise, we set the variable to 0. Let C' be the resulting circuit. We then compute $\frac{\partial^{|E(H)|}}{\partial w_{e_1} \ldots \partial w_{e_{|E(H)|}}} C'$ using the sum and product rule for partial derivatives. Then, we set $w_e = 0$ for all e. The partial differentiation ensures that all monomials must have at least one edge of each color. Setting $w_e = 0$ ensures that all monomials can only have at most one edge of each color. Therefore, the remaining monomials contain each edge in H exactly once. These are exactly those homomorphisms that correspond to colored isomorphisms. For each colored isomorphism, there are $|aut(H)|$ ways to relabel the vertices of H to obtain the same edge set. So we divide by $|aut(H)|$ to obtain Hom_H exactly. Each partial differentiation increases the size of the circuit at most by a factor of 3. Therefore, if C has size s, the final circuit has size at most $c3^{|E(H)|}s$ for some constant c.

For the other direction, given a circuit that computes ColIso_H, we can replace each $x_{\{(i,u),(j,v)\}}$ with $x_{\{u,v\}}$ if $u \neq v$ and 0 otherwise. The circuit now computes Hom_H because the monomial corresponding to the homomorphism ϕ is generated by the corresponding colored isomorphism $i \mapsto (i, \phi(i))$. Also, every colored isomorphism on vertices (i, u_i) for $1 \leq i \leq k$ corresponds to a homomorphism to K_n as long as $u_i \neq u_j$ for $\{i, j\} \in E(H)$.

Notice that both constructions preserve monotonicity and yield a monotone ABP when the original circuit is a monotone ABP.

A straightforward application of the sum and product rule to compute the partial derivative does not necessarily yield a formula from a formula. While computing the partial derivative, we have to compute both f and $\partial f/\partial x$ for every sub-formula f. If $f = g + h$ for some formulas g and h, then we can simply compute $\partial f/\partial x = \partial g/\partial x + \partial h/\partial x$ without using any sub-formula more than once. When the formula $f = gh$ for some formulas g and h, we have to compute $\partial f/\partial x = g\partial h/\partial x + h\partial g/\partial x$. Therefore, we are using g and h twice, once for computing f and once for computing $\partial f/\partial x$. We can convert the resulting circuit into a formula by duplicating the sub-formulas g and h. For general formulas, this leads to a quadratic blowup in size. But, for monotone formulas computing constant-degree polynomials, we show that the size increases only by a constant factor when duplicating sub-formulas in this fashion.

First, we eliminate all gates in the formula that computes 0 and replace all gates that compute a constant by an input gate labeled with that constant. We call a multiplication gate f *trivial* if $f = ag$ for some constant a and sub-formula g. In this case, we have $\partial f/\partial x = a\partial g/\partial x$. Therefore, we do not have to make a copy of the non-constant sub-formula g. We make a copy of the input gate a here. But, we do not have to make additional copies of this input gate a later for any trivial gate encountered on the path from f to root. This is because f is not a constant and therefore it must be the other input to the trivial gate that is constant. We claim that if the monotone formula computes a polynomial of degree at most d, for any gate g, there are at most d non-trivial multiplication gates on the path from g to the root gate. This is because each non-trivial gate increases the degree by at least one and no cancellations can occur in a monotone formula. Therefore, we can compute the partial derivative using at most $d+2$ copies of each gate in the original formula. Since the degree of the polynomials we are differentiating is at most $2|E(H)|$ and we perform $|E(H)|$ differentiations, the final formula has size $O\big((2|E(H)|+2)^{|E(H)|}s\big)$ if the original formula has size s. ◀

From now on, we can use either ColIso_H or Hom_H to prove our results. For a pattern graph H, we say that an edge $\{i,j\}$ or a vertex i in H is present in a monomial over the variables of the colored isomorphism polynomial if there is a variable $x_{\{(i,u),(j,v)\}}$ for some u and v in that monomial. We also say that the edge $\{(i,u),(j,v)\}$ or the vertex (i,u) is present in that monomial in that case.

In our proofs, we restrict our attention to the multiplication gates and the input gates in parse trees. This is because our proofs consider the computation of individual monomials separately and in this case, the addition gates play no role in the computation as all of them have exactly one child in the parse tree. While proving lower bounds, we also assume that all addition and multiplication gates have exactly two incoming edges. Since we define the size of a circuit as the number of edges, this transformation changes the size only by a constant factor. In addition, with this restriction, the number of gates and the number of edges are related by constant factors. Therefore, we may lower bound either to lower bound the size of the circuit.

▶ **Theorem 16** (Theorem 1 restated). *The monotone arithmetic circuit complexity of* Hom_H *is* $\Theta\big(n^{tw(H)+1}\big)$.

Proof. The upper bound is already known (See [13], [3]).

Let H be of treewidth t. For proving the lower bound, we consider the n-th colored isomorphism polynomial for H, $\mathsf{ColIso}_{H,n} = \sum_{u_1,\ldots,u_k} \prod_{i,j} x_{\{(i,u_i),(j,u_j)\}}$, where $u_1,\ldots,u_k \in [n]$ and $\{i,j\} \in E(H)$ (see Definition 11). Consider any monomial m on the vertices (i,u_i) for $1 \le i \le k$ and its associated parse tree. We assume without loss of generality that both sides of any multiplication gate computes a non-constant term (If they compute a constant, it has to be 1 and we can simply ignore this subtree). We build a tree decomposition of H from this parse tree in a bottom-up fashion. In this tree decomposition, each gate is associated with one or two bags in the decomposition. Exactly one of these bags is designated as the *root bag* of that gate. An example of this construction is shown in Figure 1 where the 4-vertex cycle is the pattern.

1. For an input gate $x_{\{(i,u),(j,v)\}}$. Add the bag $\{i,j\}$ as a leaf of the tree decomposition. For example, in Figure 1, the input gate labeled $x_{\{(1,u),(2,v)\}}$ is associated with the bag $\{1,2\}$.
2. For a multiplication gate g. Let A and B be the contents of the root bags of its left and right subtrees. Add a bag containing $A \cup B$ as the parent of those roots. For example, in Figure 1, the gate g is first associated with a bag containing $\{1,2,3\} = \{1,2\} \cup \{2,3\}$.
3. If at any gate, there are vertices (i,u) such that the monomial computed at that gate includes all edges incident on (i,u). Then add a new bag that excludes exactly all such vertices i from the current root bag and add it as a parent of the current root bag. This new bag is now considered the root bag associated with the gate g. For example, in Figure 1, we observe that at gate g, all edges incident on the vertex $(2,v)$ are included and therefore we add a new bag that removes 2 and make it the root bag associated with g.

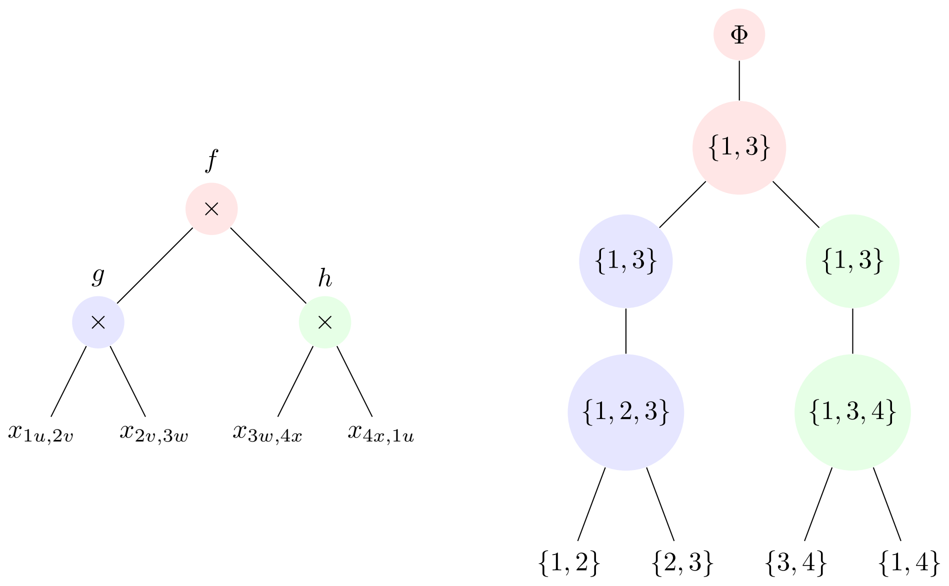

Figure 1 A tree decomposition of C_4 from a parse tree. $u, v, w, x \in [n]$.

The result is a tree decomposition of H. All edges of H are covered because all edges of H must be present in the monomial. Since a vertex is forgotten only after all edges incident on it have been multiplied, the subgraph of the tree decomposition induced by any vertex in H is a subtree.

Since H has treewidth t, the tree decomposition that we constructed must have a bag with at least $t+1$ vertices. We consider some gate g in the parse tree that is associated with a bag that contains at least $t+1$ vertices. In the following proof, we only consider the case where the bag contains exactly $t+1$ vertices. If there are more than $t+1$ vertices, then we get a better lower bound. We assume without loss of generality that these vertices are $1, \ldots, t+1$. We now claim that the gate g can only be present in parse trees of monomials m' such that m' contains vertices (i, u_i) for $1 \le i \le t+1$. Suppose for contradiction there is a monomial m' with a parse tree T' that contains g and has vertex (i, v_i) where $v_i \neq u_i$ for some such i. Let T be the parse tree for m. There are two cases:

1. The vertex i is forgotten at a bag associated with g: This means that both left and right subtrees of T_g compute monomials that contain (i, u_i). Now if T' contains (i, v_i) on the left subtree of T'_g, then replace right subtree of T'_g with right subtree of T_g. Else, the tree T' contains (i, v_i) on the right subtree of T'_g or outside T'_g. In both cases we replace the left subtree of T'_g with the left subtree of T_g.
2. Vertex i is not forgotten at a bag associated with g: This means that (i, u_i) appears in at least one of the subtrees of T_g and outside T_g in T. In T', if (i, v_i) appears in T'_g, then replace the tree T_g with T'_g in T. Otherwise, the vertex (i, v_i) must appear outside T'_g in T'. In this case, replace T'_g with T_g in T'.

All these new parse trees yield monomials that contain both (i, u_i) and (i, v_i). A contradiction. Notice that we can obtain exactly n^{k-t-1} colored isomorphisms that fix $t+1$ vertices. Therefore, at most n^{k-t-1} monomials of the polynomial can contain g in its parse tree. Since there are n^k monomials, this gives the required lower bound. ◀

▶ **Theorem 17** (Theorem 2 restated). *The monotone algebraic branching program complexity of* Hom_H *is* $\Theta\big(n^{pw(H)+1}\big)$.

Proof. The upper bound is already known (See [13], [3]). For the lower bound, we modify the proof of Theorem 16 to obtain a path decomposition instead. Recall that for any (monotone) polynomial, the size of the smallest (monotone) ABP and the size of the smallest (monotone) skew circuit are within constant factors of each other. We consider the parse tree for a monomial m in a skew circuit computing Hom_H and start building the path decomposition from a deepest input gate.

We give a detailed proof in the full version. ◀

▶ **Theorem 18** (Theorem 3 restated). *The monotone formula complexity of* Hom_H *is* $\Theta\big(n^{td(H)}\big)$.

Proof. We first prove the upper bound[5]. Let T be an elimination tree of depth d for H. We show how to construct a formula of size n^d for $\mathsf{Collso}_{H'}$ in a bottom-up fashion where H' is the graph obtained from T by adding all possible edges $\{i, j\}$ where i is an ancestor of j in T. Note that H' and H has the same treedepth by construction. The polynomial Collso_H can be obtained by setting extra variables in this polynomial to 1.

Let i be the label of a node in T such that the path from root to i is labeled $i_1, \ldots, i_p$ and the children of i are labeled $\ell_1, \ldots, \ell_s$. We use the notation $(i_j, u_j)_j$ where $j \in [p]$ to denote the p pairs $(i_1, u_1), \ldots, (i_p, u_p)$. We construct the formula:

$$f_i^{\{(i_j,u_j)_j\}} = \sum_u \Big((\prod_j x_{\{(i_j,u_j),(i,u)\}}) \prod_t f_{\ell_t}^{\{(i,u),(i_j,u_j)_j\}} \Big)$$

where $j \in [p]$ and $t \in [s]$.

We use induction on the height c of the node i in the elimination tree to prove the size upper bound. We claim that the formula that corresponds to such a node has size $O(n^c)$ (constants hidden by the O notation depend only on H). For the base case, if i is a leaf node, this formula has size $O(n)$. If i has depth c in T, then this formula has size $O(n) \times O(n^{c-1}) = O(n^c)$ by the induction hypothesis.

If $\{i, j\}$ is not an edge in H, then we set all $x_{\{(i,u),(j,v)\}}$ to 1 in the formula. The formula corresponding to the root node in T is the required polynomial. To prove this, consider the colored subgraph isomorphism containing the vertices (i, u_i) for $1 \le i \le k$. This monomial has the parse tree obtained by setting the variable u in the outermost summation for each f_i^* to u_i. For the other direction, the parse tree obtained by setting u to u_i in the outermost summation of each f_i^* generates the monomial that corresponds to the colored subgraph isomorphism on vertices (i, u_i).

We now prove the lower bound. Let d be the treedepth of H. We consider a parse tree for a monomial m in a formula computing Collso_H and build an elimination tree for H from it. An example of this construction is shown in Figure 2 where the 4-vertex path is the pattern. We associate a set of rooted trees with each gate g as follows: If the gate g is the lowest gate in the parse tree such that all edges incident on vertices $i_1, \ldots, i_r$ are present in the monomial computed at g, then make i_1 the parent of all roots of trees from the children of g. Now, make i_{j+1} the parent of i_j for $1 \le j \le r-1$. We call the vertices $i_1, \ldots, i_r$ to be associated with g. If there are no such vertices for a gate g, then the forest associated

[5] We believe this construction is already known as folklore. But we present it here for the sake of completion since we couldn't find a reference for the construction.

with g is simply the union of the forests of its children. We start with empty forests initially. For any edge $\{i, j\} \in E(H)$, the gates that corresponds to i and j in this tree must belong to the path from the input gate for this edge to the root. This shows that this tree is an elimination tree for H.

In Figure 2, the input gate labeled z is such that the only edge incident on $(1, u)$ is already multiplied in at z. Therefore, we associate the vertex 1 with z and the set of trees associated with z is just the one-vertex tree 1. Also, at gate f, we have multiplied in all edges incident on $(3, w)$ and therefore, we associate 3 with f. Also, note that this vertex 3 is the parent of the roots of elimination trees from g and h.

For proving the lower bound, we consider some gate g in the parse tree of m such that g is associated with a leaf at a depth of at least d in the elimination tree. We can assume that the depth is exactly d. If it is more, then we obtain a better lower bound. Let this vertex be d. Assume without loss of generality that $1, \ldots, d$ are the vertices on the path from the root to d in the elimination tree. Let $(1, u_1), \ldots, (d, u_d)$ be the corresponding vertices in m. We claim that any monomial m' for which g appears in its parse tree must also have vertices u_j of color j for $1 \leq j \leq d$.

Suppose for contradiction that there is a monomial m' with g in its parse tree and m' has vertex (i, v_i) where $u_i \neq v_i$ for some $1 \leq i \leq d$. Let g' be the gate in T such that i is associated with g'. Now, recall that if T is an elimination tree for the connected graph H, then all edges in H are between vertices that are in an ancestor-descendant relationship in T. Thus, g' must be an ancestor of g (g' could be the same as g). Let T' be the parse tree for m'. Then, the tree T' must contain g' as well because it contains g and in a formula there is a unique path from any gate to the root. There are two cases:

1. The gate g' is an input gate: In this case, $u_i = v_i$ because a monomial cannot have two different vertices of the same color and (i, u_i) is present in g'.
2. The gate g' is a multiplication gate: In this case, the vertex (i, u_i) must appear in both subtrees of $T_{g'}$. If the vertex (i, v_i) appears in the right (left) subtree of $T'_{g'}$, then we replace the left (right) subtree of $T'_{g'}$ with the left (right) subtree of $T_{g'}$. Otherwise, the vertex (i, v_i) appears outside $T'_{g'}$ in T'. In this case, we replace the subtree $T'_{g'}$ with the subtree $T_{g'}$.

In all cases, we obtain a monomial that contains both vertices (i, u_i) and (i, v_i). A contradiction. Therefore, at most n^{k-d} monomials of the polynomial can contain the gate g in their parse tree. Since there are a total of n^k monomials, the lower bound follows.

In Figure 2, we can infer using the above argument that the input gate z can be a part of parse trees of at most n different monomials. ◀

We show how to use our characterizations to prove superpolynomial separations between various monotone models. The polynomials are *not* based on fixed pattern graphs but a natural extension of our polynomials to patterns that grow in size with the host graph.

▶ **Definition 19.** *Let $H = (H_n)$ be a family of* pattern graphs *where H_n is some graph on n vertices. Then, we define the n-th polynomial in the family* ColIso_H *as:*

$$\mathsf{ColIso}_{H_n,n} = \sum_{u_1,\ldots,u_n} \prod_{\{i,j\}} x_{\{(i,u_i),(j,u_j)\}}$$

where $u_1, \ldots, u_n \in [n]$ and $\{i, j\} \in E(H_n)$.

It is easy to see that our lower bounds for monotone circuits, monotone ABPs, and monotone formulas hold for ColIso_H for any family of pattern graphs H, where the lower bound for the n-th polynomial is given by the treewidth, pathwidth, and treedepth of H_n

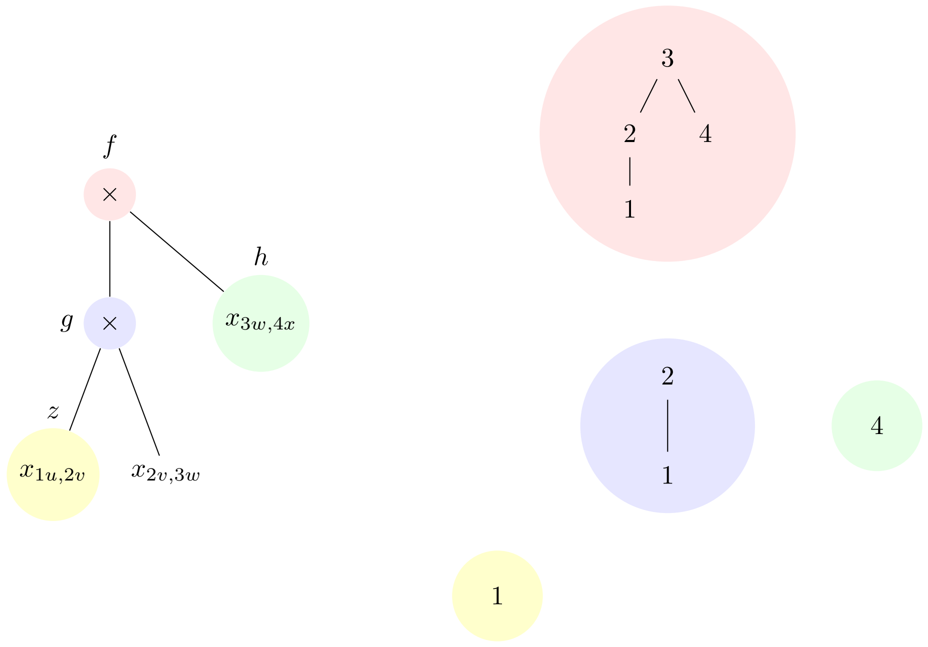

Figure 2 An elimination tree for P_4 from a parse tree. $u, v, w, x \in [n]$.

respectively. The lower bounds for Hom_H, however, does not hold because Lemma 15 causes a size blow-up that is exponential in the size of the pattern graph while constructing $\mathsf{Collso}_{H_n,n}$ from $\mathsf{Hom}_{H_n,n}$.

We now show that there are pattern families that prove a super-polynomial separation between the size complexity of monotone circuits and monotone ABPs.

▶ **Theorem 20** (Compare [20, Theorem 1, Proposition 13]). *For any $p \geq 1$ and $n = \frac{1}{6}(5 \cdot 3^p - 3)$, there is a polynomial family of degree $n - 1$ on $n^3 - n$ variables such that the family has linear size monotone circuits but require $n^{\log_2(n+1)}$ size monotone ABPs.*

For the separation, the n-th polynomial in the family is simply Collso_{X_n}, where X_n are the trees whose existence is guaranteed from Fact 2. See full version for a detailed proof.

We now show that there are pattern families that prove a super-polynomial separation between the size complexity of monotone circuits and monotone ABPs.

▶ **Theorem 21** (See [41, Theorem 3.2]). *For $n \geq 2$, there is a polynomial family of degree $n - 1$ on $n^3 - n$ variables such that the family has linear size monotone ABPs but require $n^{\lceil \log_2(n+1) \rceil}$ size monotone formulas.*

The n-th polynomial in the family is simply Collso_{P_n}, the path on n vertices. See full version for a detailed proof.

4 Discussion

We note that our lower bounds for colored isomorphism polynomials also hold for *counting circuits* studied by Jukna [22]. Such circuits produce the same evaluation as arithmetic circuits when variables are substituted from $\{0, 1\}$. Informally, a counting circuit is an arithmetic circuit that computes a polynomial modulo the axiom $x^2 = x$. Note that this is sufficient for counting homomorphisms or colored isomorphisms. However, the counting circuit lower bounds do not extend to the homomorphism polynomials since the reduction between the colored isomorphism polynomial and the homomorphism polynomial in Lemma 15 use partial derivatives.

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and Süleyman Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008*, pages 241–249, 2008. doi:10.1093/bioinformatics/btn163.

2 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

3 Markus Bläser, Balagopal Komarath, and Karteek Sreenivasaiah. Graph pattern polynomials. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 18:1–18:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.18.

4 C. Borgelt and M. R. Berthold. Mining molecular fragments: finding relevant substructures of molecules. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 51–58, 2002.

5 Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006.

6 Bruno Pasqualotto Cavalar, Mrinal Kumar, and Benjamin Rossman. Monotone circuit lower bounds from robust sunflowers. In *LATIN 2020: Theoretical Informatics: 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, pages 311–322, 2021. doi:10.1007/978-3-030-61792-9_25.

7 Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. Negations provide strongly exponential savings. *Electron. Colloquium Comput. Complex.*, page 191, 2020. URL: https://eccc.weizmann.ac.il/report/2020/191.

8 Arkadev Chattopadhyay, Rajit Datta, and Partha Mukhopadhyay. *Lower Bounds for Monotone Arithmetic Circuits via Communication Complexity*, pages 786–799. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3406325.3451069.

9 Prasad Chaugule, Nutan Limaye, and Aditya Varre. Variants of homomorphism polynomials complete for algebraic complexity classes. In *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2019. doi:10.1007/978-3-030-26176-4_8.

10 F. R. K. Chung, R. L. Graham, and R. M. Wilson. Quasi-random graphs. *Combinatorica*, 9(4):345–362, 1989. doi:10.1007/BF02125347.

11 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *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.

12 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoret. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

13 Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Counting h-colorings of partial k-trees. *Theor. Comput. Sci.*, 281(1-2):291–309, 2002. doi:10.1016/S0304-3975(02)00017-8.

14 R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006. URL: https://books.google.de/books?id=aR2TMYQr2CMC.

15 Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. Homomorphism polynomials complete for VP. *Chic. J. Theor. Comput. Sci.*, 2016, 2016. URL: `http://cjtcs.cs.uchicago.edu/articles/2016/3/contents.html`.

16 Christian Engels. Dichotomy theorems for homomorphism polynomials of graph classes. *Journal of Graph Algorithms and Applications*, 20(1):3–22, 2016. `doi:10.7155/jgaa.00382`.

17 Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015. `doi:10.1137/140978211`.

18 Hervé Fournier, Guillaume Malod, Maud Szusterman, and Sébastien Tavenas. Nonnegative Rank Measures and Monotone Algebraic Branching Programs. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.FSTTCS.2019.15`.

19 Bruno Grenet. An upper bound for the permanent versus determinant problem, 2012.

20 Pavel Hrubes 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*, pages 89:1–89:12, 2016. `doi:10.4230/LIPIcs.ICALP.2016.89`.

21 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, July 1982. `doi:10.1145/322326.322341`.

22 Stasys Jukna. Lower bounds for monotone counting circuits. *Discrete Applied Mathematics*, 213:139–152, 2016.

23 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. `doi:10.1016/S0020-0190(00)00047-8`.

24 Xiangnan Kong, Jiawei Zhang, and Philip S. Yu. Inferring anchor links across multiple heterogeneous social networks. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 179–188, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2505515.2505531`.

25 Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discrete Math.*, 27(2):892–909, 2013. `doi:10.1137/110859798`.

26 Deepanshu Kush and Benjamin Rossman. Tree-depth and the formula complexity of subgraph isomorphism. *CoRR*, abs/2004.13302, 2020. `arXiv:2004.13302`.

27 Yuan Li, Alexander A. Razborov, and Benjamin Rossman. On the ac^0 complexity of subgraph isomorphism. *SIAM J. Comput.*, 46(3):936–971, 2017. `doi:10.1137/14099721X`.

28 Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 1959–1969, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3394486.3403247`.

29 László Lovász. *Large networks and graph limits*, volume 60 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 2012. `doi:10.1090/coll/060`.

30 László Lovász and Vera T. Sós. Generalized quasirandom graphs. *J. Combin. Theory Ser. B*, 98(1):146–163, 2008. `doi:10.1016/j.jctb.2007.06.005`.

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

32 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6:85–112, 2010. `doi:10.4086/toc.2010.v006a005`.

33 Dániel Marx and Michal Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 542–553. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

34 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

35 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

36 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 410–418, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/103418.103462.

37 Ran Raz and Amir Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *J. Comput. System Sci.*, 77(1):167–190, 2011. doi:10.1016/j.jcss.2010.06.013.

38 Benjamin Rossman. Lower bounds for subgraph isomorphism. In *Proceedings of the International Congress of Mathematicians—Rio de Janeiro 2018. Vol. IV. Invited lectures*, pages 3425–3446. World Sci. Publ., Hackensack, NJ, 2018.

39 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. *Github survey*, 2015.

40 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.

41 Marc Snir. On the size complexity of monotone formulas. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 621–631, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg.

42 Srikanth Srinivasan. Strongly exponential separation between monotone vp and monotone vnp. *ACM Trans. Comput. Theory*, 12(4), September 2020. doi:10.1145/3417758.

43 Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, 2 edition, September 2000.

44 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.

45 Amir Yehudayoff. Separating monotone vp and vnp. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 425–429, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316311.

46 J. Zhang and G. Wu. Targeting social advertising to friends of users who have interacted with an object associated with the advertising, Dec. 15 2010. US Patent App. 12/968,786.

47 Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. Meta-graph based recommendation fusion over heterogeneous information networks. In *KDD*, pages 635–644, 2017. doi:10.1145/3097983.3098063.