

Monotone Bounded Depth Formula Complexity of Graph Homomorphism Polynomials

Balagopal Komarath¹ Rohit Narayanan¹

¹Indian Institute of Technology Gandhinagar

Abstract

We characterize the monotone bounded depth formula complexity for graph homomorphism and colored isomorphism polynomials using a graph parameter called the *cost of bounded product depth baggy elimination tree*. Using this characterization, for constant-degree polynomial families, we show an almost optimal separation between monotone circuits and monotone formulas for all fixed product depth and an almost optimal separation between monotone formulas of product depths Δ and $\Delta + 1$ for all $\Delta \geq 1$.

1 Introduction

The study of graph homomorphism polynomials has emerged as a surprisingly powerful lens for understanding fundamental questions in both algorithm design and algebraic complexity. As an algorithmic tool, efficient constructions for graph homomorphisms yield optimal algorithms for various pattern counting and detection problems [3]. In algebraic complexity, these same polynomials provide a unified framework for defining natural families that are complete for major complexity classes like VP and VNP [4, 5].

Komarath, Pandey, and Rahul [7] showed that the *monotone* complexity, where subtractions are disallowed in computation, of graph homomorphism polynomials is characterized by various structural parameters of the pattern graph H . Specifically, in the unbounded-depth setting, the monotone circuit complexity is characterized by H 's treewidth, the monotone ABP complexity by pathwidth, and the monotone formula complexity treedepth. Therefore, we can use known separations between these graph parameters to separate corresponding algebraic computational models.

The above characterizations naturally raise the question: how does this correspondence translate to *bounded-depth* computation? Recently, Bhargav, Chen, Curticapean and Dwivedi [1] provided a characterization for monotone bounded-depth circuits, linking their size to a depth-restricted variant of treewidth. However, a significant model remains uncharacterized: monotone bounded-depth formulas. This model is computationally weaker than circuits but fundamental in algebraic complexity theory. What graph-theoretic parameter, if any, governs the trade-off between depth and size for a formula computing $\text{Hom}_{H,n}$?

In this paper, we resolve this question. We introduce a graph decomposition, which we call the *Baggy elimination tree*, that are generalizations of elimination trees used to define treedepth. We define two measures for a baggy elimination tree: *product depth* (Δ) and

cost at product depth Δ (λ_Δ). We define $\lambda_\Delta(H)$ as the minimum possible cost that can be achieved for a given product depth Δ . Our main theorem proves that this graph-theoretic parameter characterizes bounded product depth monotone formula complexity for graph homomorphism polynomials.

The paper is organized as follows. We introduce definitions for the model and the polynomials in Section 2. We define baggy elimination trees, the parameter that characterized bounded product depth monotone formula complexity, in Section 3. We prove the characterization in Section 4. This characterization, which provides a precise characterization of the complexity in a well-studied model for a large class of important polynomials in terms of a structural graph parameter, is the main contribution of our work. In fact, for any fixed graph H , our upper and lower bounds only differ by constant factors. For constant-degree polynomial families, we prove an almost optimal separation between bounded product depth monotone circuits and monotone formulas in Section 5 and a depth hierarchy theorem for monotone formulas in Section 6. Better lower bounds and separations are known for monotone computation (See [6, 2]). However, our main contribution here is to show that these separations can be derived from easily provable separations between graph parameters continuing the line of work in [7] and [1].

2 Preliminaries

Definition 1. A polynomial over $\mathbb{Q}$ is called *monotone* if all its coefficients are non-negative. An *arithmetic formula* computing a polynomial in $\mathbb{Q}[x_1, \dots, x_n]$ is either a variable, a field constant, or $F_1 + \dots + F_k$, or $F_1 \dots F_k$ where F_i for $i \in [k]$ are arithmetic formulas computing polynomials in $\mathbb{Q}[x_1, \dots, x_n]$. The formula is called *monotone* if all constants in it are non-negative.

An arithmetic formula can be naturally represented as a rooted tree where the internal nodes (called gates) are labelled $+$ or $\times$ and the leaves (called input gates) are labelled by variables or constant. The *size* of a formula is then the number of edges in the tree. The *product depth* of the formula is the maximum number of gates labelled $\times$ over a path from the root to an input gate.

The families of polynomials that we look at in this paper enumerate graph homomorphisms or colored isomorphisms. The following definitions are from [7]:

Definition 2. 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 3. 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 .

Definition 4. 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 $u, v \in [n]$.

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

where ϕ ranges over all homomorphisms from H to K_n .

Computing the homomorphism polynomial is an important intermediate step in many algorithms related to finding and counting graph patterns. Instead of the homomorphism polynomial, we consider an equivalent polynomial (See Lemma 15 in [7]) called the colored isomorphism polynomial which enumerates all colored isomorphisms from a pattern to a host graph where there are n vertices of each color.

Definition 5. 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 $\{i, j\} \in E(H)$.

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

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

Monomials of the polynomial are computed using parse trees in formulas:

Definition 6. Let g be a gate in a formula F . A *parse tree* rooted at g is any rooted tree which can be obtained by the following procedure:

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

Any gate can occur at most once in any parse tree in F as F is a tree. Given a parse tree T that contains a gate g , we use T_g to denote the subtree of T rooted at g . 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 any subtree of T_g with the corresponding subtree of T'_g to obtain another parse tree. This is because all children of g in both parse trees are the same and therefore we can apply the aforementioned replacement. We call the tree obtained by removing T_g from T as the *tree outside T_g (or g) in T* .

Let $H = (V, E)$ be a graph. A vertex $v \in V(H)$ is called a *pendant vertex* if its degree is one. Throughout this paper, we assume that that pattern graph H has more than one edge and is connected.

3 Baggy Elimination Trees

In this section, we introduce the most important definition in this paper and work through some examples to understand the intuition behind this definition.

Definition 7 (Baggy Elimination Tree). For a graph H , a *baggy elimination tree* T is a rooted tree where each node in $V(T)$ is labeled with a non-empty "bag" of vertices and the tree satisfies: $\{u, v\} \in E(H)$ if and only if u and v are in the same bag or they are in bags that are in an ancestor-descendant relationship in T . A leaf node t_m in T is a *core leaf* if it contains some vertex that is not pendant in H . Otherwise, we call the leaf *non-core*.

The *product depth* of T is the maximum number of nodes on any root-to-leaf path $P = (t_1, \dots, t_m)$, excluding the leaf node t_m if t_m is a non-core leaf. The cost of a single path P is the sum of the cardinalities of the bags of all nodes on that path. The *cost of the tree* T is the maximum cost over all root-to-leaf paths in T . The Δ -product depth baggy elimination tree cost, denoted by $\lambda_\Delta(H)$, is the cost of the minimum cost baggy elimination tree of product depth at most Δ .

To motivate the above definition, we consider monotone formulas for $\text{Collso}_{P_7, n}$, where P_7 is the path on 7 vertices. This polynomial has n^7 monomials. So it has an n^7 -size monotone formula of product depth one. Komarath, Pandey, and Rahul [7] describe how to construct $O(n^3)$ -size monotone formulas for this polynomial. Their construction has a product depth of three.

Example 1. Consider the following formula for $\text{Collso}_{P_7, n}$:

$$\sum_{i_2, i_4, i_6 \in [n]} \left(\sum_{i_1 \in [n]} x_{\{(1, i_1), (2, i_2)\}} \sum_{i_3 \in [n]} x_{\{(2, i_2), (3, i_3)\}} x_{\{(3, i_3), (4, i_4)\}} \sum_{i_5 \in [n]} x_{\{(4, i_4), (5, i_5)\}} x_{\{(5, i_5), (6, i_6)\}} \sum_{i_7 \in [n]} x_{\{(6, i_6), (7, i_7)\}} \right)$$

This formula has size $O(n^4)$ and has product depth two. It corresponds to the baggy elimination tree of product depth two for P_7 shown in Figure 1. The size of the formula is determined by the cost of the tree and the nesting of product gates increasing the product depth to two is contributed only by the core leaves in the baggy elimination tree. This is the reason for distinguishing between core and non-core leaves in the definition of baggy elimination trees.

Baggy elimination trees are a simple generalization of elimination trees used to define treedepth. In an elimination tree, each bag has to contain exactly one vertex. Observe that any baggy elimination tree of cost c can be converted into an elimination tree of cost c by replacing bags with more than one vertex with a path containing those vertices. Therefore, for any Δ , we have $\lambda_\Delta(H)$ is at most the treedepth of H for all H .

Remark 1. The graph that is just an edge has a formula of product depth zero that has linear size and this is optimal. Disconnected graphs can be characterized by defining *baggy elimination forests* instead of trees and by defining the product depth of such forests as one plus the max of product depths of trees in the forest if there is more than one tree in the forest. In the interest of simplicity, we omit this generalization from this paper.

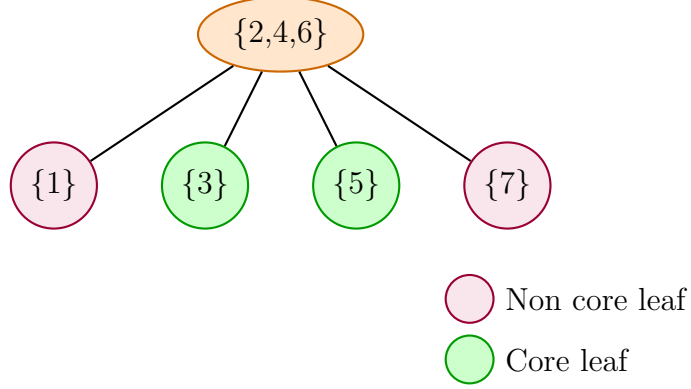


Figure 1: Baggy elimination tree of product depth two for P_7

4 Characterizing Bounded Product Depth Monotone Formulas

Our main theorem is as follows:

Theorem 1. For any connected graph H on more than two vertices, the polynomial family Collso_H has Δ -product depth monotone formula complexity of $\Theta(n^{\lambda_\Delta(H)})$.

Proof. We prove the upper-bound first. Let H be the pattern graph. For any $\Delta \geq 1$, we construct a monotone formula $\mathcal{F}$ for $\text{Collso}_{H,n}$ with product-depth Δ and size $O(n^{\lambda_\Delta(H)})$. Let T be a baggy elimination tree T for H with product depth Δ and cost $k = \lambda_\Delta(H)$.

We build a recursive formula $\mathcal{F}$ whose structure mirrors the structure of the tree T . For any node t in T , we use X_t to denote all vertices in H in the bag t and let $A(t)$ denote the set of t 's proper ancestors. Let $X_{A(t)} = \bigcup_{a \in A(t)} X_a$ be the set of all vertices in its ancestors. Let $\phi_{A(t)}$ be an assignment $\phi_{A(t)} : X_{A(t)} \rightarrow [n]$. We define a formula $\mathcal{F}(t \mid \phi_{A(t)})$ that computes the polynomial for the sub-problem induced by t and its descendants, given the fixed assignment $\phi_{A(t)}$ to all its ancestors.

First, we define short names for two classes of monomials for convenience. Let $\text{EM}(\phi_t, X_t)$ (Edge Monomial) be the product of variables for edges *within* the bag X_t :

$$\text{EM}(\phi_t, X_t) = \prod_{\substack{\{u,v\} \in E(H) \\ u,v \in X_t}} x_{\{\phi_t(u), \phi_t(v)\}}$$

and let $\text{ALM}(\phi_{A(t)}, \phi_t)$ (Ancestor Link Monomial) be the product of variables for all edges *between* the current bag and *an ancestor bag*:

$$\text{ALM}(\phi_{A(t)}, \phi_t) = \prod_{\substack{\{u,v\} \in E(H) \\ u \in X_{A(t)}, v \in X_t}} x_{\{\phi_{A(t)}(u), \phi_t(v)\}}$$

The required formula is $\mathcal{F}(r \mid \emptyset)$, where r is the root node (it has no ancestors). We construct this formula inductively starting from the leaves of t . If t is a leaf node in T with

ancestors $A(t)$, the formula is:

$$\mathcal{F}(t \mid \phi_{A(t)}) = \sum_{\phi: X_t \rightarrow [n]} (\text{EM}(\phi, X_t) \cdot \text{ALM}(\phi_{A(t)}, \phi))$$

If t is an internal node with children $u_1, \dots, u_m$, its formula is:

$$\mathcal{F}(t \mid \phi_{A(t)}) = \sum_{\phi_t: X_t \rightarrow [n]} \left(\text{EM}(\phi_t, X_t) \cdot \text{ALM}(\phi_{A(t)}, \phi_t) \cdot \prod_{i=1}^m \mathcal{F}(u_i \mid \phi_{A(t)} \cup \phi_t) \right)$$

It is easy to see that the formula is correct using an induction. We now prove that it has product depth Δ and size $O(n^k)$. Observe that each node in the tree along any root to leaf path contributes at most one to the product depth. It now suffices to show that non-core leaves of T do not add to the product depth of the formula. Consider a non-core leaf t in T . It cannot contain more than one pendant vertex from H . Suppose for contradiction there are two pendant vertices u and v in t . Since H is connected and has more than one edge, there cannot be an edge between u and v . Therefore, u and v are adjacent to some vertex in a proper ancestor bag. We replace the node t in T with two non-core leaves t_1 and t_2 containing u and v respectively. Now, we can assume without loss of generality that $t = \{u\}$ for some pendant u in H . Therefore, in the formula $\mathcal{F}(t \mid \phi_{A(t)})$, we have $\text{EM}(\cdot, \cdot) = 1$ and that $\text{ALM}(\cdot, \cdot)$ is a single variable. So there are no multiplication gates in this formula. By definition, the longest core path in T has length Δ . Therefore, the constructed formula $\mathcal{F}$ has a product-depth of at most Δ .

We now prove the size upper-bound. Observe that the fan-in of each $+$ gate at the top-level in $\mathcal{F}(t \mid \cdot)$ is n^k where $k = |X_t|$. The fan-ins of all $\times$ gates are constant (independent¹ of n). Therefore, the total size of the formula is $O(n^{|X_{t_1}| + \dots + |X_{t_k}| = \lambda_\Delta(H)})$ where $t_1, \dots, t_k$ is the maximum cost path in T .

Now, we prove the lower bound. We prove that any monotone formula $\mathcal{F}$ for $\text{Collso}_{H,n}$ with product-depth Δ must have a size $\Omega(n^{\lambda_\Delta(H)})$. The proof is similar to the lower bound proof of Theorem 3 in [7]. Given a parse tree computing a monomial of $\text{Collso}_{H,n}$, we construct a baggy elimination tree for H from the parse tree. We then show that only a few monomials can be computed using a gate that corresponds to the leaf bag in the baggy elimination tree that achieves the maximum cost. This implies a lower bound on the total number of gates.

Let m be a monomial in $\text{Collso}_{H,n}$ and let T be a parse tree of $\mathcal{F}$ computing m after removing all $+$ gates from the parse tree and attaching the child of $+$ gate to the $+$ gate's parent. We construct a baggy elimination tree B from T as follows: For each vertex $i \in V(H)$, we find all leaves in T corresponding to variables that involve i (e.g., $x_{\{(i,f(i)), (j,f(j))\}}$ for all neighbors j of i in H). We put i into the bag that is the Least Common Ancestor (LCA) of these leaves in T , denoted by $t_i = \text{LCA}(i)$. After this process, if a non-root node is empty, we remove it from B , attaching any children to the removed nodes parent. We will see later that the root bag is either non-empty or has exactly one child after this process. If the root bag is empty and has one child, we can make the child the root. The nodes of our baggy

¹These constants depend on T and therefore H . But since we regard H as a fixed pattern graph, we can absorb this cost into the $O(\cdot)$ notation.

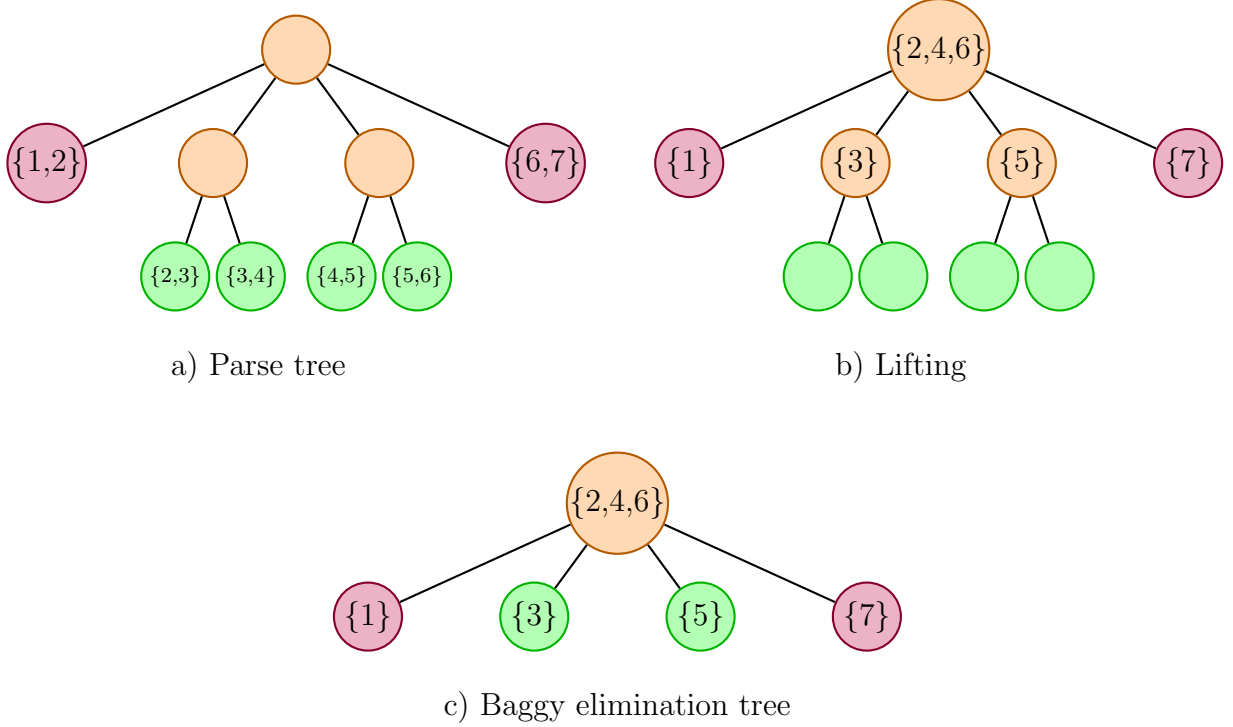


Figure 2: Parse tree to baggy elimination tree for P_7

tree B then correspond to gates $g \in T$ that are an LCA for at least one vertex. We denote the bag for a node g as $B(g) = \{i \in V(H) \mid \text{LCA}(i) = g\}$. The tree structure of B is thus inherited from T .

We claim that B is a valid baggy elimination tree of product depth Δ for H . For any edge $\{i, j\} \in E(H)$, in the initial tree (Refer Figure 2, Part (a)), the corresponding leaf in T is a descendant of both t_i and t_j , implying t_i and t_j must also be in an ancestor-descendant relationship. We will now prove our earlier claim about the root bag. Suppose the root bag is empty and has more than one child. Let u and v be two vertices of H in two distinct children of the root. Observe that all vertices adjacent to u or v in H are in the respective subtrees. So there is not path between u and v in H , which contradicts the assumption that H is connected.

We claim the product depth of T is at most Δ . A non-pendant vertex i of H appears in multiple leaves in T , forcing its t_i to be the LCA of distinct leaves in B , which must correspond to a multiplication gate. Therefore, any leaf in B that corresponds to an input gate can only contain pendant vertices in H . So the product depth of B is bounded by the product depth of $\mathcal{F}$.

Consider a monomial m and a parse tree T computing it in $\mathcal{F}$. After the construction of the baggy elimination tree from T as detailed above, let g be the gate in T that corresponds to the leaf in T that maximizes the cost. Let $d = \lambda_\Delta(H)$. Assume without loss of generality that $1, \dots, d$ are the vertices of H appearing in g and its ancestors in the baggy elimination tree. The monomial m fixes an assignment for all vertices in H to $[n]^d$. Let $\phi_m(i) = u_i$ for $i \in [d]$ in this assignment. We claim that for any monomial m' for which g appears in a parse

tree computing m' , we must have $\phi_{m'}(i) = u_i$ for $i \in [d]$ as well. Suppose for contradiction that $\phi(m') = v_i \neq u_i$ for some $i \in [d]$. Let T' be the parse tree for m' . Let g' be the gate in T such that i is in the bag corresponding to g' . Since there is a unique path from g to g' in $\mathcal{F}$, the gate g' must appear in T' as well. We say that a vertex i in H is contained in a subtree T of some parse tree if there is some input gate labelled by $x_{\{(i, \cdot), (j, \cdot)\}}$ in the subtree T . We now split the proof into cases:

- If g' is an input gate, then it is labelled $x_{\{(i, w_i), (j, w_j)\}}$ for some $w_i, w_j \in [n]$ and $j \in V(H)$. Since g' is in the parse tree for both m and m' , it must be that $w_i = u_i$ and $w_i = v_i$, a contradiction.
- The gate g' is a multiplication gate. The vertex i occurs in at least two subtrees of g' as we lifted i to g' . We split into two cases:
 - If $T'_{g'}$ contains i , then we use one such subtree to replace the corresponding subtree in T . The resulting parse tree computes a monomial that contains variables indexed by (i, u_i) and (i, v_i) , a contradiction.
 - If $T'_{g'}$ does not contain i , then i must occur in the tree outside $T'_{g'}$ in T' . We replace $T'_{g'}$ with $T_{g'}$ in T' . The resulting parse tree again computes a monomial that contains variables indexed by (i, u_i) and (i, v_i) , a contradiction.

Now, we prove the lower bound. The polynomial $\text{Collso}_{H,n}$ has n^k monomials where $k = |V(H)|$. Any monomial can be mapped to a gate as above such that a gate in the image has at most $n^{k-\lambda\Delta(H)}$ pre-images. Therefore, there must be at least $n^{\lambda\Delta(H)}$ such gates. $\square$

5 Separating Circuits from Formulas in Bounded Depth

The full b -ary tree of depth Δ denoted $F_{b,\Delta}$ is defined as follows: $F_{b,1}$ is the single node tree. $F_{b,\Delta+1}$ is a root node with b subtrees each isomorphic to $F_{b,\Delta}$. We note that the treedepth of $F_{b,\Delta}$ is Δ for $b \geq 2$. It is at most Δ because the tree itself is an elimination tree. For $\Delta = 1$, the lower bound is true. For $\Delta > 1$, since $b \geq 2$, at least one $F_{b,\Delta-1}$ subtree is untouched in the root. So the depth of any elimination tree is at least $1 + (\Delta - 1) = \Delta$.

Bhargav et. al. [1] showed that the Δ -product depth monotone circuit complexity of $\text{Collso}_{F_{b,\Delta+1}}$ is $\Theta(n^2)$. Theorem 1 and the above lower bound on treedepth shows that the Δ -product depth monotone formula complexity of this polynomial family is $\Theta(n^\Delta)$. Therefore, we get the following theorem separating the power of circuits and formulas of bounded product depth.

Theorem 2. For any $\Delta \geq 1$, there is a constant-degree (does not depend on N) polynomial family that is computable by $O(N)$ -size monotone circuits of product depth Δ but requires $\Omega(N^{\Delta/2})$ monotone formulas of product depth Δ , where N is the number of variables in the formula.

Remark 2. Observe that a product depth Δ monotone circuit of size s can be converted into a product depth Δ monotone formula of size $O(s^{2\Delta})$ by simply duplicating gates as

needed. Therefore, this separation is optimal up to constant factors independent of Δ in the exponent of n .

6 Product Depth Hierarchy for Monotone Formulas

We show that $F_{b,\Delta+2}$ has high cost for product depth Δ baggy elimination trees.

Theorem 3. For any $b > \Delta \geq 1$, we have $\lambda_\Delta(F_{b,\Delta+2}) \geq b + \Delta$.

Proof. We prove by induction on Δ . For $\Delta = 1$, note that the root bag must contain all non-leaf vertices of $F_{b,3}$ and there are $b + 1$ such vertices.

For $\Delta > 1$, we split the proof into two cases:

- If the root bag does not contain any vertex from at least one $F_{b,\Delta+1}$ subtree, then by the induction hypothesis, the $\Delta - 1$ product depth baggy elimination tree for this subtree contributes at least $b + \Delta - 1$ to the cost. The root bag contains at least one vertex. So the total cost is $b + \Delta$.
- Otherwise, the root bag contains at least b vertices. We claim that there are no vertices in the root bag from at least one of the b^2 subtrees isomorphic to $F_{b,\Delta}$. Otherwise, root bag itself will contain $b^2 \geq 2b \geq b + \Delta$ vertices. But the subtree $F_{b,\Delta}$ has cost at least Δ for any product depth. Therefore, the total cost is at least $b + \Delta$.

□

We can now prove a depth hierarchy theorem for monotone formulas.

Theorem 4. For any $\Delta \geq 1$, for any constant $k \geq 2$, there is a constant-degree (does not depend on N) polynomial family that has $O(s(N))$ -size monotone formulas of product depth Δ but requires $\Omega(s(N)^k)$ -size monotone formulas of product depth $\Delta - 1$, where N is the number of variables in the formula.

Proof. The family is $\text{Collso}_{F_{b,\Delta+1}}$ with $b = (k - 1)\Delta + 1$. The corresponding colored isomorphism polynomial has $O(n^\Delta)$ size monotone formulas of product depth Δ but need $\Omega(n^{b+\Delta})$ for product depth $\Delta - 1$. □

Remark 3. Observe that any polynomial family where degree is bounded by constant d has product depth one monotone formulas of size $O(n^d)$. Therefore, polynomial separations as above are the best one could hope for. However, it is possible that the above separations can be achieved using polynomials with lower (constant) degree.

References

- [1] C. S. Bhargav, Shiteng Chen, Radu Curticapean, and Prateek Dwivedi. *Monotone Bounded-Depth Complexity of Homomorphism Polynomials*. arXiv preprint arXiv:2505.22894. 2025.

- [2] Arkadev Chattopadhyay, Utsab Ghosal, and Partha Mukhopadhyay. “Robustly Separating the Arithmetic Monotone Hierarchy via Graph Inner-Product”. In: *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*. Vol. 250. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 12:1–12:20. DOI: 10.4230/LIPIcs.FSTTCS.2022.12.
- [3] Radu Curticapean. “Homomorphism Tensors and Linear Equations”. In: *56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015*. 2015, pp. 326–339.
- [4] Arnaud Durand, Meena Mahajan, Guillaume Malod, and Nitin Saurabh. “Completeness for VP and VNP from Graph Homomorphism”. In: *40th International Symposium on Mathematical Foundations of Computer Science, MFCS 2015*. 2015, pp. 308–319.
- [5] Prateek Dwivedi and Nitin Saxena. “Graph Homomorphism and Natural VNP-Complete Families”. In: *Theory Comput. Syst.* 62.2 (2018), pp. 441–464.
- [6] Maria Klawe, Wolfgang J. Paul, Nicholas Pippenger, and Mihalis Yannakakis. “On monotone formulae with restricted depth”. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing (STOC '84)*. New York, NY, USA: Association for Computing Machinery, 1984, pp. 480–487. DOI: 10.1145/800057.808717.
- [7] Balagopal Komarath, Anurag Pandey, and C. S. Rahul. “Monotone Arithmetic Complexity of Graph Homomorphism Polynomials”. In: *Algorithmica* 85.9 (2023), pp. 2554–2579.