On Isoperimetric Profiles and Computational Complexity

Pavel Hrubeš[*1] **and Amir Yehudayoff**[†2]

1 **Institute of Mathematics of CAS, Prague, Czech Republic**
hrubes@math.cas.cz
2 **Department of Mathematics, Technion-IIT, Haifa, Israel**
amir.yehudayoff@gmail.com

Abstract

The isoperimetric profile of a graph is a function that measures, for an integer k, the size of the smallest edge boundary over all sets of vertices of size k. We observe a connection between isoperimetric profiles and computational complexity. We illustrate this connection by an example from communication complexity, but our main result is in algebraic complexity.

We prove a sharp super-polynomial separation between monotone arithmetic circuits and monotone arithmetic branching programs. This shows that the classical simulation of arithmetic circuits by arithmetic branching programs by Valiant, Skyum, Berkowitz, and Rackoff (1983) cannot be improved, as long as it preserves monotonicity.

A key ingredient in the proof is an accurate analysis of the isoperimetric profile of finite full binary trees. We show that the isoperimetric profile of a full binary tree constantly fluctuates between one and almost the depth of the tree.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Monotone computation, separations, communication complexity, isoperimetry

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.89

1 Introduction

Computational complexity theory is about understanding the amount of resources required to compute a given function. One general framework for analyzing the limitations of a given computational device is to partition it to two or more parts and study the interactions between them (see [12, 2] and references trerein). This framework is directly related to communication complexity [24, 15], and appears in the study of branching programs (e.g. [4]), in algebraic complexity theory (e.g. [17, 10]), and more.

In this work, we observe the following phenomenon: the *exact* sizes of the parts in the partition matter. Some functions are "easy" to compute when the sizes can be chosen flexibly, but are "difficult" to compute when the sizes are chosen adversarially. The simplest illustration of this phenomenon comes from communication complexity, and is discussed below. Our most convincing application, however, comes from arithmetic circuit complexity, and forms the bulk of this paper.

* For the first author, the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Program (FP7/2007-2013) / ERC grant agreement no. 339691. The Institute of Mathematics is supported by RVO:67985840.
† For the second author, the research is supported by the ISF and BSF.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 89; pp. 89:1–89:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Isoperimetric profile of graphs

The aforementioned phenomenon is related to expansion properties of graphs. For a graph $G = (V(G), E(G))$ and $A \subseteq V(G)$, the size of the edge boundary of A is

$$e_G(A) = \big|\{\{a, b\} \in E(G) : a \in A, b \in B\}\big|,$$

where $B = V(G) \setminus A$. The edge isoperimetric profile of G is the function

$$\mathrm{eip}_G(k) = \min\{e_G(A) : A \subseteq V(G), |A| = k\}.$$

For example, G is connected if and only if $\mathrm{eip}(k) \geq 1$ for every non trivial k. Analyzing the isoperimetric profile of a graph is not a simple task. In fact, even understanding simple properties of the isoperimetric profile in manifolds is difficult (see [16] and references therein).

The high-level connection is as follows. Consider functions defined over some graph G. The vertices are labelled by variables, and the edges represent interactions between them. A partitioning of the variables into two parts yields a partition of the vertices of G. Intuitively, the size of the edge boundary represents the amount of interaction between the parts, and so a large boundary implies high complexity.

1.2 Communication complexity

Communication complexity studies the amount of communication two or more parties need to exchange in order to achieve a common goal. It was initiated by Yao in [24] in the context of distributed computing. For simplicity of the presentation, here we focus on the best-case deterministic two player model that was introduced by Papadimitriou and Sipser [18], and applied to study of very-large-scale integration (for more background and details, see the textbook [15]).

The deterministic communication complexity $D(f)$ of a boolean function $f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$ is the minimum number of bits two players need to exchange in order to determine the value of $f(x, y)$, where one player sees $x \in \{0,1\}^A$ and the other sees $y \in \{0,1\}^B$. Let $F : \{0,1\}^V \to \{0,1\}$. Given a partition of V to two sets A, B, we can define a function $F_{A,B} : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$ by $F_{A,B}(x, y) = F(z)$ where $z|_A = x$ and $z|_B = y$. The best-case deterministic communication complexity with partition-size k of F, denoted $D_k^{\mathrm{best}}(F)$, is the minimum of $D(F_{A,B})$ over all partitions of V to two sets A, B with $|A| = k$.

We observe the following connection between the best-case communication complexity and the edge isoperimetric profile for certain functions. Let $G = (V, E)$ be an undirected graph with n vertices and maximum degree Δ. Consider, e.g., the function $F : \{0,1\}^V \to \{0,1\}$ defined via G as[1]

$$F(z) = \bigoplus_{\{u,v\} \in E} z_u \wedge z_v,$$

where $\oplus$ is addition modulo two. Then, for every $1 \leq k \leq n$, it holds that[2]

$$\Omega(\mathrm{eip}_G(k)/\Delta^2) \leq D_k^{\mathrm{best}}(F) \leq O(\mathrm{eip}_G(k)).$$

[1] The definition of F is inspired by a private communication with Avi Wigderson following [19].
[2] In this text, big O and Ω notation means "up to a constant multiplicative factor".

The upper bound is obtained by partitioning the variables according to the minimizer of the edge boundary, noting that the players need only to exchange inputs that appear on edges of the boundary. The lower bound holds for the following reason. For every partition A, B of the inputs with $|A| = k$, the edge boundary of A contains an induced matching of size $m \geq \Omega(\mathrm{eip}_G(k)/\Delta^2)$. Setting the variables not participating in the matching to zero, the lower bound follows from standard lower bounds on the inner product function, $\bigoplus_{i\in[m]} x_i \wedge y_i$.

In other words, the best-case communication complexity of F is determined by the isoperimetric profile of G, as long as G has constant degree. So, if the isoperimetric profile of G changes dramatically with k, then so does $D^{\mathrm{best}}_k(F)$. For example, there are functions F so that $D^{\mathrm{best}}_{n/2}(F) = O(1)$ whereas $D^{\mathrm{best}}_{n/3}(F) = \Omega(n)$.

1.3 Algebraic complexity

Algebraic complexity theory studies the complexity of computing polynomials over a field. For more background and details, see [7, 5, 21] and references within. We start by outlining definitions of the models we consider.

The most general model of computation in this area is that of *arithmetic circuit*. An arithmetic circuit is a directed acyclic graph with fan-in either zero or two. Vertices of fan-in zero are labelled by a field element or a variable. Vertices of fan-in two are labelled by either $+$ or $\times$. Every vertex in an arithmetic circuit computes a polynomial in the obvious way. A weaker model is an *arithmetic formula*: it is a circuit whose underlying graph is a tree. Another model is the *algebraic branching program*, ABP. An ABP is a directed acyclic graph with two special vertices $v_{\mathrm{start}}, v_{\mathrm{end}}$. Each edge e in it is labelled by $L(e)$ which is a variable times a field element. The program computes f, which is the following sum over all directed paths γ from v_{start} to v_{end}:

$$f = \sum_{\gamma : v_{\mathrm{start}} \to v_{\mathrm{end}}} \prod_{e\in\gamma} L(e).$$

The computation performed by ABPs corresponds to the iterated multiplication of matrices. The size parameter in each of the three models is the number of edges in the underlying graph.

We now discuss reductions between these models. Formulas can be losslessly simulated by an ABP, which in turn can be losslessly simulated by a circuit. In the opposite direction, Hyafil [11] showed that every circuit of size s computing a polynomial of degree r can be simulated by a formula of size $2^{O(\log(s)\log(r))}$. Valiant, Skyum, Berkowitz and Rackoff [23] significantly strengthened this result. In their seminal work, they showed that an arithmetic circuit of size s computing a polynomial of degree r can be simulated by a circuit of size $\mathrm{poly}(r, s)$ and depth $O(\log(r)\log(s))$. This result was later used in a sequence of works – Agrawal and Vinay [1], Koiran [13], and Tavenas [22] – to prove non-trivial simulations of circuits by circuits of depth four (and unbounded fan-in). This depth reductions sprung a long sequence of works on lower bounds for constant depth circuits (see [9, 14, 6] are references within).

An intriguing question is whether the simulations discussed above are sharp. It is possible they can be significantly improved. However, since we do not have strong enough methods for proving lower bounds for these models of computation, proving that the reductions are sharp is currently beyond us.

We therefore address this question in the restricted setting of monotone computations. A monotone arithmetic computation is a computation over the real numbers that uses only

non-negative numbers. The aforementioned reductions of Hyafil and Valiant et al. transform a monotone device to a monotone device. We call such reductions monotone.

Since we know how to prove lower bounds for monotone computation, we can in fact show that in the category of monotone reductions the reductions above are indeed sharp. Indeed, the reduction between ABPs and formulas is sharp since Shamir and Snir [20] proved that in order to multiply d matrices of size $n \times n$, we need a monotone formula of size $n^{\Omega(\log d)}$, but on the other hand this can be done via a monotone ABP of size $\mathrm{poly}(n, d)$. The reduction from circuits to ABPs was not previously studied. In this paper, we prove:

▶ **Theorem 1.** *There is a multilinear n-variate polynomial f with zero-one coefficients so that the following hold:*

1. *The polynomial f can be computed by a monotone arithmetic circuit of size $\mathrm{poly}(n)$.*
2. *Every monotone ABP computing f has size at least $2^{\Omega(\log^2(n))}$.*

The simulation of [23] shows that the lower bound in the theorem is tight, up to a constant in the exponent. Stated differently, the theorem shows that any monotone reduction from circuits to ABPs must incur a super-polynomial blowup. To the best of our knowledge, this is the first separation between algebraic branching programs and circuits. Prior to this work, it was conceivable that the monotone construction of [23] or other monotone variants of it can simulate a circuit by an ABP with only a polynomial increase in size.

It is worth mentioning that Gupta, Kamath, Kayal and Saptharishi [8] showed how to simulate a circuit of size s and degree r over n variables by a depth three circuit of size $\exp(O(\sqrt{d \log(d) \log(n) \log(s)}))$, over fields of characteristic 0. Their simulation uses the reductions to depth four from [1, 13, 22] mentioned above, together with two other reductions. Most relevant to our work is that their simulation is not monotone. In fact, as they mention, a simulation to depth three achieving the same parameters can not be monotone.

We now briefly discuss the proof. All lower bounds we know of for monotone algebraic computation are based on combinatorial problems that are related to counting monomials. The lower bounds for circuits use the following structure, which is standard by now (see e.g. [20, 21, 10]). We denote by $\deg(f)$ the total degree of f.

▶ **Lemma 2.** *If f is a homogeneous polynomial of degree r that can be computed by a monotone circuit of size s, then for every integer $2 \leq k \leq r$, we can write*

$$f = \sum_{i=1}^{s} h_i g_i \tag{1}$$

where for each i, both h_i, g_i are homogeneous, monotone, and of degrees $k/3 \leq \deg(h_i) < 2k/3$ and $\deg(g_i) = r - \deg(h_i)$.

A typical monotone lower bound proceeds by showing that the number of monomials in each $h_i g_i$ is small and so s must be large. To separate monotone circuits from ABPs, we must find a polynomial that has a small circuit and use some other property of ABPs. We use the following.

▶ **Lemma 3.** *If f is a homogeneous polynomial of degree r that can be computed by a monotone ABP of size s, then for every integer $0 \leq k \leq r$, we can write*

$$f = \sum_{i=1}^{s} h_i g_i \tag{2}$$

where for each i, both h_i, g_i are homogeneous, monotone, and of degrees $\deg(h_i) = k$ and $\deg(g_i) = r - k$.

Notice the similarities between the two lemmas. The only difference between circuits and ABPs is that circuits are slightly more flexible about the degrees of h_i, g_i in (1), whereas ABPs are not. We exploit this weakness to prove the separation. Motivated by Section 1.2, the polynomial f in Theorem 1 will be defined over an underlying graph G. The setting will be such that for each k, large $\mathrm{eip}_G(k)$ implies a super-polynomial lower bound on s in (2), and hence gives a lower bound on the ABP-size. On the other hand, the graph must be chosen so that f is easy to compute via a monotone circuit. If G is too complex, such as an expander graph, then we do not expect to prove meaningful upper bounds. In fact, in order to circumvent the lower bound implied by (1), the value of $\mathrm{eip}_G(k)$ must be small for many integers k. It turns out that the right graph to choose is the full binary tree, which we discuss next.

1.4 Full binary trees

We analyze the isoperimetric profile of full finite binary trees. The case of the infinite binary tree was previously studied by Bharadwaj, Chandran and Das in [3], where it was shown to be related to meta-Fibonacci sequences and signed almost binary partitions.

For an integer $d \geq 0$, denote by T_d the full binary tree of depth d. It has precisely 2^d leaves and $2^{d+1} - 1$ vertices. It is clear that there are many values $k \leq |V(T_d)|$ for which $\mathrm{eip}_{T_d}(k) = 1$. It is more surprising that for some k, the value of $\mathrm{eip}_{T_d}(k)$ can be fairly large, almost as large as the depth of T_d. This is the content of the first theorem:

▶ **Theorem 4.** *Let $mip(T_d)$ be the maximum of $\mathrm{eip}_{T_d}(k)$ over all $0 \leq k \leq |V(T_d)|$. Then*

$$\frac{d}{2} - O(\log d) \leq mip(T_d) \leq \frac{d}{2} + O(1)\,.$$

In order to analyze the isoperimetric profile of T_d, we relate it to the binary representation of k. For an integer $k \geq 0$, denote the binary representation of k by

$$B(k) = (B_0(k), B_1(k), B_2(k), \ldots) \in \{0,1\}^{\mathbb{N}}.$$

I.e., $k = \sum_{i=0}^{\infty} B_i(k) 2^i$. Denote by $\mathrm{drops}(k)$ the number of drops in $B(k)$:

$$\mathrm{drops}(k) = |\{i \geq 0 : B_i(k) > B_{i+1}(k)\}|.$$

The number of drops is a quantity that is relatively easy to understand, and as the following theorem shows, it captures the isoperimetric profile of binary trees.

▶ **Theorem 5.** *For every $d \geq 0$ and $0 \leq k < |V(T_d)|$,*

$$\frac{\mathrm{drops}(k)}{2} \leq \mathrm{eip}_{T_d}(k) \leq 2\mathrm{drops}(k)\,.$$

Moreover, the lower bound can be improved to $\mathrm{drops}(k) - O(\log(\mathrm{drops}(k)))$.

The theorem provides an almost optimal description of the isoperimetric profile $\mathrm{eip}_{T_d}(k)$ of T_d. It implies that although $\mathrm{eip}_{T_d}(k) = 1$ for many values of k, for most values of k we have $\mathrm{eip}_{T_d}(k) \geq \Omega(d)$.

We get an explicit choice of k for which $\mathrm{eip}(k)$ is large. For even $d \geq 0$, define

$$\sigma_d = 1 + 4 + 16 + \ldots + 2^d \approx \frac{2}{3} 2^{d+1}, \tag{3}$$

and for odd d define $\sigma_d = \sigma_{d-1}$. The number of drops in $B(\sigma_d)$ is at least $d/2$. The number σ_d can be thought of as the truncation of the binary expansion of $2/3$.

▶ **Corollary 6.** *For every $d \geq 0$, we have $\mathrm{eip}_{T_d}(\sigma_d) \geq d/2 - O(\log d) \geq d/4$.*

1.5 Organization

Section 3 contains the proof of the separation between monotone circuits and ABPs. Section 2 contains the proofs concerning the isoperimetric profile of the full binary tree that are needed in Section 3 (due to space limitations, some of the proofs that are not used in Section 3 will appear in the full version of the text).

2 Binary trees

In order to prove the lower bounds in Theorems 4 and Theorem 5, we will introduce the quantity $\mathrm{sbl}(k)$ which, roughly speaking, measures how far k is from powers of 2. We then relate $\mathrm{eip}_{T_d}(k)$ and $\mathrm{sbl}(k)$,

We say that k is a signed power of two if it is of the form $\pm 2^j$ for some integer $j \geq 0$. For $k \in \mathbb{Z}$, the signed binary length of k, denoted $\mathrm{sbl}(k)$, is the minimum b so that there exist b signed powers of two $k_1, \ldots, k_b$ so that $k = \sum_{i=1}^{b} k_i$, where repetitions are a priori allowed. For example, $\mathrm{sbl}(0) = 0$ and if $d \geq 1$ then

$$\mathrm{sbl}(1 + 2 + 4 + \ldots + 2^d) = \mathrm{sbl}(2^{d+1} - 1) = 2.$$

The following lemma gives a lower bound on $\mathrm{eip}_{T_d}(k)$ in terms of $\mathrm{sbl}(k)$.

▶ **Lemma 7.** *For every $d \geq 0$ and $0 < k \leq |V(T_d)|$,*

$$\mathrm{eip}_{T_d}(k) \geq \mathrm{sbl}(k) - O(\log(\mathrm{sbl}(k)))\,.$$

In addition, if $k < |V(T_d)|$ then $\mathrm{eip}_{T_d}(k) \geq \mathrm{sbl}(k)/2$.

Proof. For an integer k, let $\mathrm{st}(k)$ denote the smallest b such that there exist non-negative integers $j_1, \ldots, j_b$ and $\epsilon_1, \ldots, \epsilon_b \in \{-1, 1\}$ with[3]

$$k = \sum_{j=1}^{b} \epsilon_j \cdot (2^j - 1)\,. \tag{4}$$

Recall that if $j > 0$, then $2^j - 1$ is the number of vertices in T_{j-1}. We will prove the following:

▶ **Claim 8.** *For every $U \subseteq V(T_d)$ not containing the root of T_d, we have $\mathrm{st}(|U|) \leq e_{T_d}(U)$.*

Claim 8 implies that for every $U \subseteq V(T_d)$,

$$\mathrm{st}(|U|) \leq e_{T_d}(U) + 1\,. \tag{5}$$

Indeed, if U contains the root of T_d, then apply Claim 8 to the complement $\bar{U} = V(T_d) \setminus U$. Then $\bar{U}$ does not contain the root of T_d and $e_{T_d}(U) = e_{T_d}(\bar{U}) \geq \mathrm{st}(|\bar{U}|)$. But $|U| = |V(T_d)| - |\bar{U}|$ gives $\mathrm{st}(|U|) \leq \mathrm{st}(|\bar{U}|) + 1$.

Proof of Claim 8. The proof is by induction on d. The claim holds for $d = 0$ as $\mathrm{st}(0) = 0$. Assume $d > 0$. Let v_d be the root of T_d and for a node v, let $T(v)$ be the full subtree of T_d with root v. Let $U \subseteq V(T_d)$ be so that $v_d \notin U$. Let M be the subset of maximal vertices in

[3] This quantity previously appeared in [3].

U. That is, a vertex v is in M if $v \in U$ and the shortest path from v to v_d does not contain a vertex from U. For $v \in M$, let $U(v) = U \cap V(T(v))$. Hence,

$$e_{T_d}(U) = |M| + \sum_{v \in M} e_{T(v)}(U(v)).$$

Since $|U| = \sum_{v \in M} |U(v)|$,

$$\mathrm{st}(|U|) \leq \sum_{v \in M} \mathrm{st}(|U(v)|).$$

For every $v \in M$, the inductive assumption and (5) show that

$$\mathrm{st}(|U(v)|) \leq e_{T(v)}(U(v)) + 1.$$

Overall,

$$\mathrm{st}(|U|) \leq \sum_{v \in M} (1 + e_{T(v)}(U(v))) = |M| + \sum_{v \in M} e_{T(v)}(U(v)) = e_{T_d}(U). \quad \blacktriangleleft$$

To prove the lemma, it remains to estimate $\mathrm{sbl}(k)$ in terms of $\mathrm{st}(k)$. If k is written as in (4) with $b = \mathrm{st}(k)$, we have

$$\mathrm{sbl}(k) \leq \mathrm{sbl}(\sum_{j=1}^{b} \epsilon_j 2^j) + \mathrm{sbl}(\sum_{j=1}^{b} \epsilon_j) \leq \mathrm{st}(k) + \log_2(\mathrm{st}(k) + 1).$$

This gives $\mathrm{st}(k) \geq \mathrm{sbl}(k) - O(\log(\mathrm{sbl}(k)))$ and so (5) gives

$$\mathrm{eip}_{T_d}(k) \geq \mathrm{sbl}(k) - O(\log(\mathrm{sbl}(k)))\,,$$

as required.

Finally, assume that $U \neq V(T_d)$. If U does not contain the root, apply Claim 8 and the estimate $\mathrm{sbl}(k) \leq 2\mathrm{st}(k)$, to obtain $e_{T_d}(U) \geq \mathrm{sbl}(|U|)/2$. If U does contain the root, apply Claim 8 to $\bar{U} = V(T_d) \setminus U$ and note that $\mathrm{sbl}(|U|) \leq 2\mathrm{st}(|\bar{U}|)$ whenever $\bar{U} \neq \emptyset$. $\blacktriangleleft$

The next lemma shows that, up to a constant factor, $\mathrm{sbl}(k)$ and $\mathrm{drops}(k)$ are the same.

▶ **Lemma 9.** *For every $k \geq 0$, we have* $\mathrm{drops}(k) \leq \mathrm{sbl}(k) \leq 2\mathrm{drops}(k)$.

Proof. We start by showing that $\mathrm{sbl}(k) \leq 2\mathrm{drops}(k)$. First, note that k can be written as

$$k = \sum_{i=1}^{t} \sum_{r_i \leq j \leq \ell_i} 2^j\,, \tag{6}$$

where $t \leq \mathrm{drops}(k)$ and $r_1 \leq \ell_1 < r_2 \leq \ell_2 < \cdots < r_t \leq \ell_t$ are non-negative integers. Second, note that each sum $\sum_{r_i \leq j \leq \ell_i} 2^j$ can be expressed in terms of at most two signed powers of 2.

In order to prove the other inequality, we first note that for every $k, j \geq 0$

$$|\mathrm{drops}(k + 2^j) - \mathrm{drops}(k)| \leq 1\,. \tag{7}$$

To see (7), assume first that $B_j(k) = 0$. Then adding 2^j can introduce one drop if $B_{j+1}(k) = 0$ and delete one drop if $j > 0$ and $B_{j-1}(k) = 1$. If $B_j(k) = 1$, let ℓ be the smallest integer with $B_\ell(k) = 0$ and $\ell > j$. The binary representations of k and $k + 2^j$ are the same, except that $B_\ell(k + 2^j) = 1$ and $B_i(k + 2^j) = 0$ for every $j \leq i < \ell$. The number of drops in $k + 2^j$

can increase only if $B_{j-1}(k) = 1$ and $j > 0$, and increases at most by one. It can decrease only if $B_{\ell+1}(k) = 1$, and again at most by one.

By induction on $\mathrm{sbl}(k)$, we now prove that for every $k \geq 0$, $\mathrm{drops}(k) \leq \mathrm{sbl}(k)$. If $\mathrm{sbl}(k) = 0$, the statement holds. Otherwise, let $k_1, \ldots, k_b$, with $b = \mathrm{sbl}(k)$, be signed powers of two that sum to k. Choose k_i so that $k - k_i$ is non-negative. Hence, (7) shows that $\mathrm{drops}(k - k_i) \geq \mathrm{drops}(k) - 1$. Furthermore, we have $\mathrm{sbl}(k - k_i) = \mathrm{sbl}(k) - 1$. By the inductive assumption, $\mathrm{sbl}(k - k_i) \geq \mathrm{drops}(k - k_i)$. Altogether, $\mathrm{sbl}(k) = \mathrm{sbl}(k - k_i) + 1 \geq (\mathrm{drops}(k) - 1) + 1 = \mathrm{drops}(k)$, completing the proof. ◀

Proof of Theorems 4 and 5. The lower bounds in both theorems follow from Lemmas 7 and 9, since $\mathrm{drops}(\sigma_d) = d/2$ for σ_d as defined in (3). The proof of the upper bounds is by induction and is not presented due to space considerations. ◀

Remarks

First, none of the bounds in Theorem 5 is tight for all k. For example, the $2\mathrm{drops}(k)$ upper bound exceeds the upper bound from Theorem 4 for $k = \sigma_d$ from (3). Furthermore, we do not know which of the bounds in Theorem 4 is more accurate.

Second, we have $\mathrm{eip}_{T_d}(\sigma_d) \leq d/2 - \Omega(\log d)$. Hence, the lower bound of $\mathrm{drops}(k) - O(\log(\mathrm{drops}(k))$ from Theorem 5 cannot be improved as a function of $\mathrm{drops}(k)$. A similar statement holds for Lemma 7 and $\mathrm{sbl}(k)$.

Third, the quantity $\mathrm{st}(k)$ from Claim 8 characterizes eip almost exactly: $\mathrm{st}(k) - 1 \leq \mathrm{eip}_{T_d}(k) \leq \mathrm{st}(k)$. This can be deduced from Claim 8 and Theorem 2.3 from [3]. It also implies $|\mathrm{eip}_{T_d}(k) - \mathrm{sbl}(k)| \leq O(\log k)$.

Fourth, we do not know of a simple procedure to compute $\mathrm{eip}_{T_d}(k)$, or to find a subset of size k that minimizes the boundary-size. The quantity $\mathrm{sbl}(k)$, however, can be computed exactly and quite easily, in time polynomial in $\log k$, which allows to efficiently approximate $\mathrm{eip}_{T_d}(k)$ more accurately.

3 Algebraic complexity

In this section, we prove the separation between monotone circuits and ABPs stated in Theorem 1, and the structure of ABPs stated in Lemma 3.

3.1 The tree function

We first describe a boolean function $\mathcal{T}_{d,m}$ which is later used to prove our separation. Recall that T_d is the full binary tree of depth d, and let V be the set of its vertices. For an integer $m > 1$, let $\mathbb{Z}_m$ be the additive group of integers modulo m, and let $\mathbb{Z}_m^V$ be the set of functions $\gamma : V \to \mathbb{Z}_m$.

A function $\gamma \in \mathbb{Z}_m^V$ will be called *legal*, if for every vertex v which is not a leaf and its two children v_1, v_2, we have

$$\gamma(v) = \gamma(v_1) + \gamma(v_2)\,.$$

The tree function $\mathcal{T}_{d,m} : \mathbb{Z}_m^V \to \{0,1\}$ takes $\gamma \in \mathbb{Z}_m^V$ and accepts if γ is legal.

For a node $v \in V$, let $T(v)$ be the full subtree of T_d rooted at v and let $\ell(T(v))$ be the set of its leaves. Then γ is legal iff

$$\gamma(v) = \sum_{u \in \ell(T(v))} \gamma(u) \tag{8}$$

holds for every vertex v in T.

The following lemma depicts the key property of $\mathcal{T}_{d,m}$ that we later use. For two functions f, g over the same domain, write $f \leq g$ if $f(x) \leq g(x)$ for all x. Denote by $\sup(f)$ the set of inputs x so that $f(x) = 1$.

▶ **Lemma 10.** *Let V_0, V_1 be a partition of V where $|V_0| = \sigma_d$, as defined in (3). If*

$$h_0 : \mathbb{Z}_m^{V_0} \to \{0,1\}, \quad h_1 : \mathbb{Z}_m^{V_1} \to \{0,1\}, \quad h_0 \wedge h_1 \leq \mathcal{T}_{d,m},$$

then $|\sup(h_0 \wedge h_1)| \leq m^{-d/16} |\sup(\mathcal{T}_{d,m})|$.

Proof. View T_d as directed from leaves to root. A directed path $v_1, \ldots, v_k$ in T_d from $v_1 \in \ell(T(v_k))$ to v_k will be called *pure* if $v_1 \neq v_k$ and there exists $i \in \{0,1\}$ so that $\{v_1, \ldots, v_k\} \cap V_i = \{v_k\}$. A node will be called *pure* if it is the last node of some pure path. Let $P \subset V$ be the set of pure nodes. Let E be the edge boundary of V_0.

To prove the lemma, we use the following two claims.

▶ **Claim 11.** $|P| \geq |E|/4$.

Proof of Claim 11. Let S be the set of nodes $v \in V$ so that the parent of v in T_d is pure. Since $|S| \leq 2|P|$, it is enough to show that $|S| \geq |E|/2$. Let T' be the minor of T_d obtained by contracting all the edges of T_d not in E. The tree T' is a (not necessarily binary) tree with $|E|$ edges. For $x \in V(T')$, let $[x] \subseteq V$ be the set of vertices that have been contracted to x. The root of T' is the vertex x so that the root of T_d is in $[x]$. View T' as directed from leaves to the root. For $x \in V(T')$, let $v(x) \in V$ be the vertex in $[x]$ that is closest to the root of T_d. This is well defined, since the set $[x]$ is connected in T_d. For every leaf $x \in V(T')$, we have $v(x) \in S$. For every $x \in V(T')$ with in-degree one which is not the root of T', the vertex $v(x)$ is also in S. Recall that T' can have at most $|E|/2$ vertices of in-degree at least two. Hence, $|S| \geq (|E|+1) - (|E|/2+1)$, as required. ◀

For every pure node v, fix a leaf $\tilde{v}$ such that the path from $\tilde{v}$ to v is pure. Let $\tilde{P} = \{\tilde{v} : v \in P\}$. The sizes of $\tilde{P}$ and P are the same.

▶ **Claim 12.** *Assume $\sup(h_0) \neq \emptyset$. Then every $\beta \in \sup(h_1)$ is uniquely determined by its values on $(\ell(T_d) \cap V_1) \setminus \tilde{P}$.*

Proof of Claim 12. Fix $\alpha \in \sup(h_0)$. For the sake of contradiction, assume that there are two distinct $\beta_1, \beta_2 \in \sup(h_1)$ which agree on $(\ell(T_d) \cap V_1) \setminus \tilde{P}$. Since $h_0 \wedge h_1 \leq \mathcal{T}_{d,m}$, both $\alpha \cup \beta_1$ and $\alpha \cup \beta_2$ are legal maps satisfying (8). Hence, β_1 and β_2 differ on some leaf in V_1. They agree on leaves outside of $\tilde{P}$ and so there exists a pure node $v \in V_0$ such that $\beta_1(\tilde{v}) \neq \beta_2(\tilde{v})$. We can assume that v is minimal in that for every pure node $u \in T(v) \cap V_0$ with $u \neq v$, we have $\beta_1(\tilde{u}) = \beta_2(\tilde{u})$. Since every pure path that starts in $\ell(T(v)) \cap V_1$ must also end in $T(v)$, we obtain that $\beta_1(u) = \beta_2(u)$ for every leaf in $T(v) \cap V_1$ with the sole exception of $\tilde{v}$. But this is impossible since (8) implies that for every $i \in \{1,2\}$,

$$\alpha(v) - \sum_{u \in \ell(T(v)) \cap V_0} \alpha(u) = \beta_i(\tilde{v}) + \sum_{u \in (\ell(T(v)) \cap V_1) \setminus \{\tilde{v}\}} \beta_i(u)\,,$$

which gives $\beta_1(\tilde{v}) = \beta_2(\tilde{v})$. ◀

To conclude the lemma, assume that $\sup(h_0 \wedge h_1) \neq \emptyset$, otherwise we are done. Claim 12 shows that

$$|\sup(h_0)| \leq m^{|(\ell(T_d) \cap V_0) \setminus \tilde{P}|}\,, \quad |\sup(h_1)| \leq m^{|(\ell(T_d) \cap V_1) \setminus \tilde{P}|}\,.$$

Since $|\sup(\mathcal{T}_{d,m})| = m^{|\ell(T_d)|}$,

$$|\sup(h_0)| \cdot |\sup(h_1)| \leq m^{|\ell(T_d)|-|\tilde{P}|} = m^{-|\tilde{P}|}|\sup(\mathcal{T}_{d,m})|\,.$$

Finally, Claim 11 and Corollary 6 give $|\tilde{P}| = |P| \geq |E|/4 \geq e_{T_d}(V_0)/4 \geq d/16$. ◀

3.2 The tree polynomial

We now describe the polynomial that separates monotone circuits from ABPs. Recall that T_d is the full binary tree of depth d and $\mathcal{T}_{d,m} : \mathbb{Z}_m^V \to \{0,1\}$ is the tree function, where $V = V(T_d)$. For every $v \in T_d$ and $z \in \mathbb{Z}_m$, introduce the variable $x_{v,z}$. For a partial function $\gamma : V \to \mathbb{Z}_m$, define the monomial $x_\gamma = \prod_{v \in \mathrm{dom}(\gamma)} x_{v,\gamma(v)}$. Define the tree polynomial as

$$\mathcal{P}_{d,m} = \sum_{\gamma \in \mathbb{Z}_m^V : \mathcal{T}_{d,m}(\gamma)=1} x_\gamma\,.$$

It is homogeneous of degree $|V| = 2^{d+1} - 1$ and it has $m|V|$ variables.

We now prove a generalization of Theorem 1.

▶ **Proposition 13.** *The polynomial $\mathcal{P}_{d,m}$ can be computed by a monotone arithmetic circuit of size $O(m^2 2^d)$. However, every monotone ABP computing $\mathcal{P}_{d,m}$ has size at least $m^{\Omega(d)}$.*

Theorem 1 follows from Proposition 13 by setting $m = 2^d$.

Proof of Proposition 13. We first prove the lower bound. Assume that $\mathcal{P}_{d,m}$ has a monotone ABP of size s. Let σ_d be as defined in (3). By Lemma 3, we can write

$$\mathcal{P}_{d,m} = \sum_{i=1}^{s} h_i g_i\,,$$

where h_i, g_i are homogeneous and monotone, h_i has degree σ_d and g_i has degree $\deg(\mathcal{P}_{d,m}) - \sigma_d$. We can assume $h_i g_i \neq 0$ for every i. For a polynomial f, let $\mathrm{mon}(f)$ be the set of γ's such that x_γ has a non-zero coefficient in f, and let $f^\star$ be the boolean function with $\sup(f^\star) = \mathrm{mon}(f)$. Monotonicity guarantees that for every $i \in [s]$, there exists $V_i \subseteq V$ with $|V_i| = \sigma_d$ such $\mathrm{mon}(h_i) \subseteq \mathbb{Z}_m^{V_i}$ and $\mathrm{mon}(g_i) \subseteq \mathbb{Z}_m^{V \setminus V_i}$. Therefore,

$$\mathcal{P}^\star_{d,m} = \bigvee_{i=1}^{s} h_i^\star \wedge g_i^\star\,.$$

Since $\mathcal{P}^\star_{d,m} = \mathcal{T}_{d,m}$, Lemma 10 gives $s \geq m^{d/16}$.

For the upper bound, fix m and proceed by induction on d. For $a \in \mathbb{Z}_m$, let $F_{a,d}$ be the polynomial defined as $\mathcal{P}_{d,m}$, except γ range over legal maps with $\gamma(v_d) = a$, where v_d is the root. Hence, $\mathcal{T}_{d,m} = \sum_{a \in \mathbb{Z}_m} F_{a,d}$.

We will show that $F_{d,a}$, for all $a \in \mathbb{Z}_m$, can be simultaneously computed by a circuit of size $s(d) = O(m^3 2^d)$. For $d = 0$, we have $s(d) = O(m)$. Assume that $d > 0$. Let T_ℓ be the left subtree of T_d of depth $d-1$, and let T_r be the right subtree of depth $d-1$. Let C_ℓ, C_r be the two circuits guaranteed by induction on T_ℓ, T_r. For $a \in \mathbb{Z}_m$, denote by $f_{\ell,a}$ the polynomial computed in C_ℓ so that the root of T_ℓ is mapped to a, and similarly define $f_{r,a}$. To define the circuit for $F_{d,a}$, use the following:

$$F_{d,a} = \sum_{a_\ell, a_r \in \mathbb{Z}_m : a = a_\ell + a_r} f_{\ell,a_\ell} f_{r,a_r} x_{v_d,a},$$

Overall, we get the recursion

$$s(d) \leq O(m^2) + 2 \cdot S(d-1),$$

showing that $s(d) = O(m^2 2^d)$. ◀

3.3 The structure of ABPs

We end this section by proving Lemma 3.

Proof of Lemma 3. Let P be an ABP computing a homogeneous polynomial f of degree r. For a vertex v in P, denote by h_v the polynomial

$$h_v = \sum_{\gamma : v_{\text{start}} \to v} \prod_{e \in \gamma} L_P(e)$$

and denote by g_v the polynomial

$$g_v = \sum_{\gamma : v \to v_{\text{end}}} \prod_{e \in \gamma} L_P(e).$$

Since the computation is monotone and f homogeneous, both h_v, g_v are homogeneous and the sum of their degrees is r. Denote by U the set of vertices v in P so that the degree of h_v is k. Every directed path from v_{start} to v_{end} in P passes through U exactly once. It follows that

$$f = \sum_{v \in U} h_v g_v,$$

as needed. ◀

Acknowledgements We thank Pavel Pudlák for helpful discussions.

References

1. M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *FOCS*, pages 67–75, 2008.
2. S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
3. B. S. Bharadwaj, L. S. Chandran, and A. Das. Isoperimetric problem and meta-fibonacci sequences. In *Computing and Combinatorics*, pages 22–30. Springer, 2008.
4. A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3(1):1–18, 1993.
5. P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315. Springer Science and Business Media, 1997.
6. H. Fournier, N. Limaye, G. Malod, and S. Srinivasan. Lower bounds for depth 4 formulas computing iterated matrix multiplication. In *STOC*, pages 128–135, 2014.
7. J. von zur Gathen. Algebraic complexity theory. *Annual review of computer science*, 3:317–347, 1988.
8. A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. Arithmetic circuits: A chasm at depth three. In *FOCS*, pages 578–587, 2013.
9. A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. Approaching the chasm at depth four. *Journal of the ACM*, 61(6):33, 2014.

10 P. Hrubes and A. Yehudayoff. Monotone separations for constant degree polynomials. *Information Processing Letters*, 110(1):1–3, 2009.

11 L. Hyafil. On the parallel evaluation of multivariate polynomials. *SIAM J. Comput.*, 8(2):120–123, 1979.

12 S. Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science and Business Media, 2012.

13 P. Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448:56–65, 2012.

14 M. Kumar and S. Saraf. The limits of depth reduction for arithmetic formulas: It's all about the top fan-in. In *STOC*, pages 136–145, 2014.

15 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

16 F. Morgan. Manifolds with density. *Notices of the AMS*, pages 853–858, 2005.

17 N. Nisan. Lower bounds for non-commutative computation. In *STOC*, pages 410–418, 1991.

18 C. H. Papadimitriou and M. Sipser. Communication complexity. In *STOC*, pages 196–200, 1982.

19 R. Raz and A. Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. *J. Comput. Syst. Sci.*, 77(1):167–190, 2011.

20 E. Shamir and M. Snir. Lower bounds on the number of multiplications and the number of additions in monotone computations. Technical Report RC-6757, IBM, 1977.

21 A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5:207–388, 2010. URL: `10.1561/0400000039`, `doi:10.1561/0400000039`.

22 S. Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Information and Computation*, 240:2–11, 2015.

23 L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. on Computing*, 12(4):641–644, 1983.

24 A. C. Yao. Some complexity questions related to distributive computing (preliminary report). In *STOC*, pages 209–213, 1979.