Theory of Computing Systems (2022) 66:1074–1098
https://doi.org/10.1007/s00224-022-10084-x

On the Decision Tree Complexity of Threshold Functions

Anastasiya Chistopolskaya[1] · Vladimir V. Podolskii[2]

Accepted: 4 May 2022 / Published online: 23 August 2022

Abstract

In this paper we study decision tree models with various types of queries. For a given function it is usually not hard to determine the complexity in the standard decision tree model (each query evaluates a variable). However in more general settings showing tight lower bounds is substantially harder. Threshold functions often have non-trivial complexity in such models and can be used to provide interesting examples. Standard decision trees can be viewed as a computational model in which each query depends on only one input bit. In the first part of the paper we consider natural generalization of standard decision tree model: we address decision trees that are allowed to query any function depending on two input bits. We show the first lower bound of the form $n-o(n)$ for an explicit function (namely, the majority function) in this model. We also show that in the decision tree model with AND and OR queries of arbitrary fan-in the complexity of the majority function is $n-1$. In the second part of the paper we address parity decision trees that are allowed to query arbitrary parities of input bits. There are various lower bound techniques for parity decision trees complexity including analytical techniques (degree over $\mathbb{F}_2$, Fourier sparsity, granularity) and combinatorial techniques (generalizations of block sensitivity and certificate complexity). These techniques give tight lower bounds for many natural functions. We give a new inductive argument tailored specifically for threshold functions. A combination of this argument with granularity lower bound allows

The results presented in Section 3 are supported by Russian Science Foundation (21-11-00318). The results presented in Section 4 were prepared within the framework of the HSE University Basic Research Program. The preliminary version of the paper has appeared in the proceedings of CSR 2020 conference.

✉ Vladimir V. Podolskii
podolskii@mi-ras.ru

Anastasiya Chistopolskaya
a.chistopolskaia@gmail.com

1 HSE University, Moscow, Russia

2 Steklov Mathematical Institute, Moscow, Russia

us to provide a simple example of a function for which all previously known lower bounds are not tight.

Keywords Decision tree · Parity decision tree · Granularity · Threshold function · Lower bound

1 Introduction

Decision trees is a computational model in which we compute a known Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ on an unknown input $x \in \{0,1\}^n$ and in one step we can query $q(x)$ for $q\colon \{0,1\}^n \to \{0,1\}$ from a fixed set of queries. In the standard and the most studied decision tree model we can query only individual variables of the input x [5, 12] (the complexity of f is denoted by $\mathsf{D}(f)$). The studies of this model among other things are related to the well-known sensitivity conjecture [10] that was recently resolved [11]. Among other cases studied in the literature are parity decision trees that can query any parity of input bits [28], linear decision trees in which the queries are linear threshold functions [8, 12] and decision trees with AND and OR queries [1].

In this paper we will mainly deal with threshold functions. Threshold function THR_n^k on n bits outputs 1 iff there are at least k ones in the input. The majority function MAJ_n is simply $\mathrm{THR}_n^{\lceil n/2\rceil}$.

The main goal of the first part of the paper is to study a natural generalization of standard decision tree model: we address decision trees that are allowed to query any function depending on two input bits. We denote the complexity in this model by $\mathsf{D}_{\mathrm{B}_2}(f)$. More generally, we can consider decision trees that can query arbitrary functions depending on at most r inputs, where r is a parameter. The standard decision tree model corresponds to the case $r = 1$.

This model can be viewed as a uniform version of multi-party Communication Complexity (see the book by Kushilevitz and Nisan [17] for details on Communication Complexity). In this model k players are trying to compute the function $f\colon \{0,1\}^{nk} \to \{0,1\}$ and the input is shared by the players. Each player is associated with a piece of input of size n. In the Number in Hand model (NIH) players see only the input bits associated with them. In the Number on the Forehead model (NOF) players see all input bits except those that are associated with them (thus the inputs visible to players have large overlaps). Decision tree model with $r = n$ for the computation of f can be viewed as a version of communication model where for each n bits there is a player seeing exactly these n bits. The decision tree model with $r = (k-1)n$ can be viewed as a generalization of NOF communication model.

The special case of this model was considered by Posobin [21] where the computation of MAJ_n with MAJ_k-queries was studied for $k < n$. There are results on a related model with non-Boolean counting queries [6, 13]. Related settings with non-Boolean domain also arise in algebraic decision tree model (see, e. g. [12, Section 14.8]).

We initiate the study of strong lower bounds for decision trees with queries of bounded fan-in considering the case of queries of fan-in 2. It is easy to see that the complexity $\mathsf{D}_{\mathrm{B}_2}(f)$ in this model is lower bounded by $\mathsf{D}(f)/2$ (each binary query can be simulated by two unary queries). However, in the view of generalization to larger r it is interesting to obtain lower bounds greater than $n/2$. We show that

$$\mathsf{D}_{\mathrm{B}_2}(\mathrm{MAJ}_n) \geq n - o(n).$$

We also show that if we allow additionally to query parities of three bits, the complexity of majority (as well as any symmetric function) drops to at most $2n/3$. Thus to obtain strong lower bounds for $r = 3$ more complicated functions need to be considered.

Also in this part of the paper we address the complexity of majority function MAJ in decision tree model with AND and OR queries (of arbitrary fan-in). We denote the complexity of a function f in this model by $\mathsf{D}_{\wedge,\vee}(f)$. The complexity of threshold functions in this model was studied by Ben-Asher and Newman [1] with the relation to a certain PRAM model. It was shown there that THR_n^k functions have complexity $\Theta(k/\log(n/k))$. In this paper we are interested in the precise complexity of functions in this model. We show that $\mathsf{D}_{\wedge,\vee}(\mathrm{MAJ}) = n - 1$.

In the second part of the paper we turn to parity decision tree model $\mathsf{D}_{\oplus}(f)$.

Apart from being natural and interesting on its own parity decision tree model was studied mainly in connection with Communication Complexity and more specifically, with Log-rank Conjecture. In Communication Complexity's most standard model there are two players Alice and Bob. Alice is given $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$ and they are trying to compute some fixed function $F\colon \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ on input (x, y). The question is how much communication is needed to compute $F(x, y)$ in the worst case. It is known that the deterministic communication complexity $\mathsf{D}^{\mathsf{cc}}(F)$ of the function F is lower bounded by $\log \operatorname{rank}(M_F)$, where M_F is the communication matrix of F [17]. It is a long standing conjecture and one of the key open problems in Communication Complexity, called Log-rank Conjecture [18], to prove that $\mathsf{D}^{\mathsf{cc}}(F)$ is upper bounded by a polynomial of $\log \operatorname{rank}(M_F)$.

An important special case of Log-rank Conjecture addresses the case of XOR-functions $F(x, y) = f(x \oplus y)$ for some f, where $x \oplus y$ is a bit-wise XOR of Boolean vectors x and y. On one hand, this class of functions is wide and captures many important functions (including equality and Hamming distance), and on the other hand the structure of XOR-functions allows to use analytic tools. For such functions $\operatorname{rank}(M_F)$ is equal to the Fourier sparsity $\mathsf{spar} f$, the number of non-zero Fourier coefficients of f. Thus, the Log-rank Conjecture for XOR-functions can be restated: is it true that $\mathsf{D}^{\mathsf{cc}}(F)$ is bounded by a polynomial of $\log \mathsf{spar} f$?

Given a XOR-function $f(x \oplus y)$ a natural way for Alice and Bob to compute the value of the function is to use a parity decision tree for f. They can simulate each query in the tree by computing parity of bits in their parts of the input separately and sending the results to each other. One query requires two bits of communication and thus $\mathsf{D}^{\mathsf{cc}}(F) \leq 2\mathsf{D}_{\oplus}(f)$, where by $\mathsf{D}_{\oplus}(f)$ we denote the parity decision tree complexity of f. This leads to an approach to establish Log-rank Conjecture for XOR-function [28]: show that $\mathsf{D}_{\oplus}(f)$ is bounded by a polynomial of $\log \mathsf{spar} f$.

This approach received a lot of attention in recent years and drew attention to parity decision trees themselves [9, 23–25, 27, 28]. In a recent paper [9] it was shown that actually $\mathsf{D}^{\mathsf{cc}}(F)$ and $\mathsf{D}_{\oplus}(f)$ are polynomially related. This means that the simple protocol described above is not far from being optimal and that the parity decision tree version of Log-rank Conjecture stated above is actually equivalent to the original Log-rank Conjecture for XOR-functions.

Known techniques for lower bounds for parity decision trees fall into one of the two categories: of analytical and combinatorial flavor. Analytical techniques include lower bounds on $\mathsf{D}_{\oplus}(f)$ through sparsity $\mathsf{spar}(f)$, granularity $\mathsf{gran}(f)$ and degree $\deg_2(f)$ over $\mathbb{F}_2$. The strongest lower bound among these is $\mathsf{D}_{\oplus}(f) \geq \mathsf{gran}(f) + 1$ (see details in Section 2).

Regarding combinatorial techniques, for standard decision trees there are several combinatorial measures known that lower bound decision tree complexity. Among them the most common are certificate complexity and block sensitivity. Zhang and Shi [28] generalized these measures to the setting of parity decision tree complexity.

Parity decision tree complexity versions of combinatorial measures are actually known to be polynomially related to parity decision tree complexity [28]. For analytical techniques it is known that existence of polynomial relation between $\mathsf{D}_{\oplus}(f)$ and $\mathsf{gran}(f)$ (or $\mathsf{spar}(f)$) is equivalent to Log-rank Conjecture for XOR-functions [9].

In view of this it is interesting to further study lower bounds for parity decision trees.

In this paper we prove a new lower bound for parity decision tree complexity of threshold functions. We show that

$$\mathsf{D}_{\oplus}(\mathrm{THR}_{n+2}^{k+1}) \geq \mathsf{D}_{\oplus}(\mathrm{THR}_{n}^{k}) + 1$$

for any k, n.

The combination of this result with granularity lower bound allows to show that for $n = 8k + 2$, $k > 0$ we have $\mathsf{D}_{\oplus}(\mathrm{THR}_{n}^{3}) = n - 1$, whereas all previous techniques give at most $n - 2$ lower bound. Thus, we give an example of a function, for which all known general techniques are not tight.

The rest of the paper is organized as follows. In Section 2 we provide necessary definitions, preliminary information and review lower bounds for parity decision trees. In Section 3 we study decision trees with binary queries as well as decision trees with AND and OR queries. In Section 4 we study parity decision tree complexity of threshold functions. In Section 5 we give concluding remarks.

2 Preliminaries

In many parts of the paper we assume that Boolean functions are functions of the form $f \colon \{0,1\}^n \to \{-1,1\}$, for $n \in \mathbb{N}$. That is, input bits are treated as 0 and 1 and to them we will usually apply operations over $\mathbb{F}_2$. Output bits are treated as -1 and 1 and the arithmetic will be over $\mathbb{R}$. The value -1 corresponds to 'true' and 1 corresponds to 'false'. In other parts of the paper it is more convenient to consider

Boolean functions in the form $f\colon \{-1,1\}^n \to \{-1,1\}$ with the same semantics of -1 and 1.

We denote the variables of functions by $x = (x_1, \dots, x_n)$. We use the notation $[n] = \{1, \dots, n\}$.

2.1 Boolean Fourier Analysis

We briefly review the notation and needed facts from Boolean Fourier analysis. For extensive introduction see [19].

For functions $f, g\colon \{0,1\}^n \to \mathbb{R}$ consider an inner product

$$\langle f, g\rangle = \mathbf{E}_x f(x)g(x),$$

where the expectation is taken over uniform distribution of x on $\{0,1\}^n$.

For a subset $S \subseteq [n]$ we denote by $\chi_S(x) = \prod_{i\in S}(-1)^{x_i}$ the Fourier character corresponding to S. We denote by $\widehat{f}(S) = \langle f, \chi_S\rangle$ the corresponding Fourier coefficient of f.

It is well-known that for any $x \in \{0,1\}^n$ we have $f(x) = \sum_{S\subseteq[n]} \widehat{f}(S)\chi_S(x)$.

If $f\colon \{0,1\}^n \to \{-1,1\}$ (that is, if f is Boolean) then the well-known Parseval's Identity holds:

$$\sum_{S\subset[n]} \widehat{f}(S)^2 = 1.$$

By the *support* of the Boolean function f we denote

$$\mathsf{Supp}(f) = \{S \subseteq [n] \mid \widehat{f}(S) \neq 0\}.$$

The *sparsity* of f is $\mathsf{spar}(f) = |\mathsf{Supp}(f)|$. Basically, the sparsity of f is the l_0-norm of the vector of its Fourier coefficients.

Consider a binary fraction α, that is α is a rational number that can be written in a form that its denominator is a power of 2. By the *granularity* $\mathsf{gran}(\alpha)$ of α we denote the minimal integer $k \geq 0$ such that $\alpha \cdot 2^k$ is an integer.

We will also frequently use the following closely related notation. For an integer L denote by $\mathsf{P}(L)$ the maximal power of 2 that divides L. It is convenient to set $\mathsf{P}(0) = \infty$.

Note that for Boolean f the Fourier coefficients of f are binary fractions. By the *granularity* of $f\colon \{0,1\}^n \to \mathbb{Z}$ we call the following value

$$\mathsf{gran}(f) = \max_{S\subseteq[n]} \mathsf{gran}(\widehat{f}(S)).$$

It is easy to see that for any $f\colon \{0,1\}^n \to \{-1,1\}$ it is true that $0 \leq \mathsf{gran}(f) \leq n-1$ and both of these bounds are achievable (for example, for $f(x) = \bigoplus_i x_i$ and $f(x) = \bigwedge_i x_i$ respectively).

It is known [7, 25] that $\mathsf{gran}(f)$ is always not far from the logarithm of $\mathsf{spar}(f)$:

$$\frac{\log \mathsf{spar}(f)}{2} \leq \mathsf{gran}(f) \leq \log \mathsf{spar}(f) - 1.$$

The first inequality can be easily obtained from Parseval's identity. The second is less trivial (see [25] or [19, Exercise 3.32]). Again, both inequalities are tight (the first one is tight for inner product $\mathrm{IP}(x, y) = \bigoplus_i (x_i \wedge y_i)$ or any other bent function [19]; the second one is tight for example for the conjunction of two variables).

For a Boolean function $f\{0,1\}^n \to \{-1,1\}$ denote by $\deg_2(f)$ the degree of the multilinear polynomial $p \in \mathbb{F}_2[x_1, \dots, x_n]$ computing f as a Boolean function, that is for all $x \in \mathbb{F}_2^n$ we have $p(x) = 1$ if $f(x) = -1$ and $p(x) = 0$ otherwise. It is well known that such multilinear polynomial p is unique for any f and thus $\deg_2(f)$ is well defined.

It is known that $\deg_2(f) \leq \log \mathsf{spar}(f)$ for any f [2]. We observe that the granularity is also lower bounded by the degree of the function.

Lemma 1 *For any* $f\colon \{0,1\}^n \to \{-1,1\}$ *we have* $\deg_2(f) \leq \mathsf{gran}(f) + 1$.

The proof strategy is similar to the one of [2]. We present the proof for the sake of completeness.

Proof For a function $f\colon \{0,1\}^n \to \{-1,1\}$ consider two subfunctions f_0 and f_1 on $n-1$ variables obtained from f by setting variable x_n to 0 and to 1 respectively. Note that for any $S \subseteq [n-1]$ we have

$$\begin{aligned}\widehat{f}(S) &= \mathbf{E}_{x\in\{0,1\}^n} f(x)\chi_S(x)\\ &= \frac{1}{2}\mathbf{E}_{x\in\{0,1\}^{n-1}} f_0(x)\chi_S(x) + \frac{1}{2}\mathbf{E}_{x\in\{0,1\}^{n-1}} f_1(x)\chi_S(x) = \frac{1}{2}\widehat{f_0}(S) + \frac{1}{2}\widehat{f_1}(S)\end{aligned}$$

and

$$\begin{aligned}&\widehat{f}(S\cup\{n\})\\ &= \mathbf{E}_{x\in\{0,1\}^n} f(x)\chi_S(x)\\ &= \frac{1}{2}\mathbf{E}_{x\in\{0,1\}^{n-1}} f_0(x)\chi_S(x) - \frac{1}{2}\mathbf{E}_{x\in\{0,1\}^{n-1}} f_1(x)\chi_S(x) = \frac{1}{2}\widehat{f_0}(S) - \frac{1}{2}\widehat{f_1}(S).\end{aligned}$$

Thus,

$$\widehat{f_0}(S) = \widehat{f}(S) + \widehat{f}(S\cup\{n\})$$

and

$$\widehat{f_1}(S) = \widehat{f}(S) - \widehat{f}(S\cup\{n\}).$$

In particular, the granularity of both f_0 and f_1 is not larger than the granularity of f. From this we conclude that the granularity of a subfunction of f is at most the granularity of f.

Denote $d = \deg_2(f)$ and consider a monomial of degree d in the polynomial p for f. For simplicity of notation assume that this is the monomial $x_1 \dots x_d$. Fix all variables x_i for $i > d$ to 0. We get a subfunction g of f of d variables and degree d. As discussed above $\mathsf{gran}(g) \leq \mathsf{gran}(f)$, so it is enough to show that $d \leq \mathsf{gran}(g)+1$. For this note that since the function g is of maximal degree we have that $|g^{-1}(-1)|$ is odd (see, e.g. [12, Section 2.1]). Thus,

$$\widehat{g}(\emptyset) = \mathbf{E}_{x\in\{0,1\}^d} g(x) = \frac{1}{2^d}\left(|g^{-1}(1)| - |g^{-1}(-1)|\right) = \frac{1}{2^d}\left(2^n - 2|g^{-1}(-1)|\right)$$

and the granularity of $\widehat{g}(\emptyset)$ is $d-1$. $\square$

2.2 Decision Trees

A decision tree T is a rooted directed binary tree. Each of its leaves is labeled by -1 or 1, each internal vertex v is labeled by some function $q_v \colon \{0,1\}^n \to \{-1,1\}$. Each internal node has two outgoing edges, one labeled by -1 and another by 1. A computation of T on input $x \in \{0,1\}^n$ is the path from the root to one of the leaves that in each of the internal vertices v follows the edge, that has label equal to the value of $q_v(x)$. Label of the leaf that is reached by the path is the output of the computation. The tree T *computes* the function $f \colon \{0,1\}^n \to \{-1,1\}$ iff on each input $x \in \{0,1\}^n$ the output of T is equal to $f(x)$.

Decision tree models differ by the types of functions q_v that are allowed in the vertices of the tree. For any set $\mathcal{Q}$ of functions the decision tree complexity of the function f is the minimal depth of a tree (that is, the number of edges in the longest path from the root to a leaf) using functions from Q and computing f. We denote this value by $\mathsf{D}_{\mathcal{Q}}(f)$.

The standard decision tree model allows to query individual variables in the vertices of the tree. The complexity in this model is denoted simply by $\mathsf{D}(f)$. In the paper we also consider $\mathsf{D}_{\oplus}(f)$, $\mathsf{D}_{\wedge,\vee}(f)$, $\mathsf{D}_{\mathrm{B}_2}(f)$ standing for $\mathcal{Q}$ equal to the set of all parities, the set of all AND and OR functions and the set B_2 of all binary functions respectively.

2.3 Parity Decision Trees

There are various techniques known for lower bounds for parity decision trees (that is, decision trees with parity queries). Since we are not aware of the exposition of some bounds that follow from known techniques and the exposition on the connection of parity decision tree complexity with multiplicative complexity, we survey them here.

As discussed in the Introduction, one technique is via sparsity through communication complexity.

Lemma 2 *For any function* $f \colon \{0,1\}^n \to \{-1,1\}$ *we have* $\mathsf{D}_{\oplus}(f) \geq \frac{\log \mathsf{spar}(f)}{2}$.

Although, if Log-rank conjecture for XOR-functions is true, this approach gives optimal bounds up to a polynomial, in many cases it does not help to determine the precise parity decision tree complexity of Boolean functions. For example, this approach always gives bounds of at most $n/2$ for functions of n variables.

Another lower bound obtained through analytical approach is via granularity.

Lemma 3 *For any non-constant function* $f \colon \{0,1\}^n \to \{-1,1\}$ *we have* $\mathsf{D}_{\oplus}(f) \geq \mathsf{gran} f + 1$.

It is not hard to deduce this lemma from [19, Exercise 3.26]. However, as we have not seen this statement in the literature, we provide a proof.

Proof Along with the function $f\colon \{0,1\}^n \to \{-1,1\}$ consider the function $f'\colon \{0,1\}^n \to \{0,1\}$ such that $f'(x) = \frac{f(x)+1}{2}$. Clearly, the parity decision tree complexity of f and f' are equal. Also it is easy to see directly from the definition that for any non-constant f we have $\mathsf{gran}(f') = \mathsf{gran}(f) + 1$.

From Exercise 3.26 in [19] it follows that $\mathsf{D}_{\oplus}(f') \geq \mathsf{gran}(f')$.

Combining all of these together we get

$$\mathsf{D}_{\oplus}(f) = \mathsf{D}_{\oplus}(f') \geq \mathsf{gran}(f') = \mathsf{gran}(f) + 1. \qquad \square$$

Another standard approach is through the degree of polynomials. It is well known that the complexity of a function in standard decision trees model is lower bounded by the degree of the function over $\mathbb{R}$ (see, e.g. [5]). Completely analogously it can be shown that the parity decision tree complexity of a function is lower bounded by the degree of the function over $\mathbb{F}_2$.

Although it is very similar to analogous connection for standard decision trees, we have not seen it in the literature.

Lemma 4 *For any* $f\colon \{0,1\}^n \to \{-1,1\}$ *we have* $\mathsf{D}_{\oplus}(f) \geq \deg_2(f)$.

Proof The proof of this lemma follows closely the proof connecting standard decision tree complexity of a function with its degree over $\mathbb{R}$ (see, e.g. [5]).

Consider a parity decision tree T computing f with depth equal to $\mathsf{D}_{\oplus}(f)$. Consider arbitrary leaf l of this tree and consider the path in T leading from the root to l. For computation to follow this path on input x in each internal vertex v the input x must satisfy some linear restriction $L(x) = 1$ ($L(x)$ is the parity $L_v(x)$ labeling v if the path follows the edge labeled by -1 out of v and $L(x) = L_v(x) \oplus 1$ if the path follows the edge labeled by 1). Denote all these linear forms in these restrictions along the path by $L_1(x), \ldots, L_p(x)$, where $p \leq \mathsf{D}_{\oplus}(f)$. Thus, on input x we follow the path to l iff $L_1(x) \wedge \ldots \wedge L_p(x)$ is satisfied. Denote this expression by $T_l(x)$.

Denote by S the set of all leaves of T that are labeled by -1. For any input x we have that $f(x) = -1$ iff the computation path in T reaches a leaf labeled with -1 iff

$$\bigoplus_{l \in S} T_l(x) = 1.$$

It is left to observe that the latter expression is a multilinear polynomial over $\mathbb{F}_2$ of degree at most $\mathsf{D}_{\oplus}(f)$. $\square$

From the bounds discussed above it follows that the lower bound through granularity is stronger than the lower bounds through the sparsity and the degree. In fact, it allows to determine the exact complexity of some Boolean functions including majority and recursive majority. Since we have not seen these bounds presented in the literature, we present them here.

The majority function $\mathrm{MAJ}_n\colon \{0,1\}^n \to \{-1,1\}$ is defined as follows: $\mathrm{MAJ}_n(x) = -1 \Leftrightarrow \sum_{i=1}^{n} x_i \geq \frac{n}{2}$.

To state our result we will need the following notation: let $\mathsf{B}(n)$ be the number of ones in a binary representation of n.

Theorem 1 $\mathsf{D}_{\oplus}(MAJ_n) = n - \mathsf{B}(n) + 1$.

Recursive majority $\mathrm{MAJ}_3^{\otimes k}$ is a function on $n = 3^k$ variables and it can be defined recursively. For $k = 1$ we just let $\mathrm{MAJ}_3^{\otimes 1} = \mathrm{MAJ}_3$. For $k > 1$ we let

$$\mathrm{MAJ}_3^{\otimes k} = \mathrm{MAJ}_3\left(\mathrm{MAJ}_3^{\otimes k-1}, \mathrm{MAJ}_3^{\otimes k-1}, \mathrm{MAJ}_3^{\otimes k-1}\right),$$

where each $\mathrm{MAJ}_3^{\otimes k-1}$ is applied to a separate block of variables.

Theorem 2 $\mathsf{D}_{\oplus}(MAJ_3^{\otimes k}) = \frac{n+1}{2}$, *where $n = 3^k$ is the number of variables.*

The upper bound in Theorems 1 is a simple adaptation of the folklore algorithm. Other parts of the proofs of Theorems 1 and 2 are purely technical. For these reasons we move these proofs to Appendix A.1 and A.2.

Next we describe a connection to multiplicative complexity.

Multiplicative complexity $\mathsf{c}_{\wedge}(f)$ of a Boolean function f is the minimal number of AND-gates in a circuit computing f and consisting of AND, $\oplus$ and NOT gates, each gate of fan-in at most 2 (for formal definitions from Circuit Complexity see, e.g. [12]). This measure was studied in Circuit Compexity [3, 4, 14] as well as in connection to Cryptography [15, 26] and providing an explicit function f on n variables with $\mathsf{c}_{\wedge}(f) > n$ is an important open problem.

The following lemma was communicated to us by Alexander Kulikov [16] and with his permission we include it here.

Lemma 5 *For any f on n variables* $\mathsf{D}_{\oplus}(f) \leq \mathsf{c}_{\wedge}(f) + 1$.

Proof The proof is by induction on $s = \mathsf{c}_{\wedge}(f)$.

If $s = 0$, then f is computed by a circuit consisting of $\oplus$ and NOT gates and thus f is a linear form of its variables. We can compute it by one query in parity decision tree model.

For the step of induction, consider an arbitrary f and consider a circuit $\mathcal{C}$ computing f with the number of AND-gates equal to $\mathsf{c}_{\wedge}(f)$. Consider the first AND-gate g in $\mathcal{C}$. Both of its inputs compute linear forms over $\mathbb{F}_2$. Our decision tree algorithm queries one of inputs of g. Depending on the answer to the query, g computes either constant 0, or its second input. In both cases the gate g computes a linear form over $\mathbb{F}_2$, so we can simplify the circuit and obtain a new circuit $\mathcal{C}'$ computing the same function on inputs consistent with the answer to the first query and with at most $s - 1$ AND-gates. By induction hypothesis in both cases the function computed by $\mathcal{C}'$ is computable in parity decision tree model with at most s queries. Overall, we make $s + 1$ queries. □

As a corollary from Theorem 1 and Lemma 5 we get the following lower bound on the multiplicative complexity of majority.

Corollary 1 $\mathsf{c}_{\wedge}(MAJ_n) \geq n - \mathsf{B}(n)$.

This improves a lower bound of [4]. Previously this lower bound was known only for $n = 2^k$ for some k [4].

Returing to lower bounds for parity decision trees, another known approach is of a more combinatorial flavor. For standard decision trees there are several combinatorial measures known that lower bound decision tree complexity. Among them the most common are certificate complexity and block sensitivity. In [28] these measures were generalized to the setting of parity decision tree complexity. Parity decision tree complexity versions of these measures are actually known to be polynomially related to parity decision tree complexity [28]. It is also known that the certificate complexity provides a better bound [28]. The paper [20] provides an example of a function for which combinatorial measures give polynomially better lower bound than granularity.

Another more combinatorial approach goes through analogs of certificate complexity and block sensitivity for parity decision trees [28]. Since parity block sensitivity is always less or equal than parity certificate complexity and we are interested in lower bounds, we will introduce only certificate complexity here.

For a function $f : \{0, 1\}^n \to \{-1, 1\}$ and $x \in \{0, 1\}^n$ denote by $C_\oplus(f, x)$ the minimal co-dimension of an affine subspace in $\{0, 1\}^n$ that contains x and on which f is constant. The *parity certificate complexity* of f is $C_\oplus(f) = \max_x C_\oplus(f, x)$.

Lemma 6 [28] *For any function* $f : \{0, 1\}^n \to \{-1, 1\}$ *we have* $\mathsf{D}_\oplus(f) \geq C_\oplus(f)$.

The described techniques allow to establish tight lower bounds for most standard functions. The complexity of both AND and OR is equal to n through, for example, certificate complexity. The exact complexity of MOD_3 function can be determined through the degree lower bound. Lower bounds for majority and recursive majority were discussed above.

3 Decision Trees with B_2-Queries

In this section we show a $n - o(n)$ lower bound for the complexity of MAJ_n function for B_2-queries. As a warm-up we start with the analysis of the complexity of MAJ_n in decision tree model with AND and OR queries of arbitrary fan-in. This model was studied in [1] with the relation to certain PRAM model.

In this section it will be convenient to switch to $\{-1, 1\}$ variables, that is we will consider $\text{MAJ}_n : \{-1, 1\}^n \to \{-1, 1\}$ that is equal to 1 iff $\sum_{i=1}^n x_i \geq 0$.

First we observe that the complexity of all monotone functions cannot be maximal.

Lemma 7 *For any monotone function* $f : \{-1, 1\}^n \to \{-1, 1\}$ *we have* $\mathsf{D}_{\wedge,\vee}(f) \leq n - 1$.

Proof We can query all variables one by one until two variables are left. Now, observe that a monotone function of two remaining variables is either a constant, or a variable, or AND_2, or OR_2. We can compute this function in at most one query. □

This upper bound is tight for MAJ_n.

Theorem 3 $\mathsf{D}_{\wedge,\vee}(MAJ_n) = n - 1$.

Proof The upper bound follows from Lemma 7.

For the lower bound we will argue by adversary argument, that is we will describe the strategy of query answering forcing the decision tree to make at least $n-1$ queries.

During the computation we will fix the values of some of the variables. We will maintain an undirected graph G on the variables that are not yet fixed. Each vertex in this graph will have degree either 0 or 1 (that is, our graph is a matching). Each edge in the graph is labeled by either 1 or -1. The intuition behind the edges is the following. We connect x_i and x_j by an edge labeled by a iff we add the restriction that at least one of the variables x_i and x_j is equal to a. That is, we are not allowed to fix both variables to $-a$ in the future.

In the beginning the set of vertices of G consists of all variables and there are no edges. In one query the number of connected components will reduce by at most 1, with only one exception (when we remove one connected component without making queries). We will answer the queries in such a way that to know the value of the function the decision tree should reduce our graph to an empty graph. From this it follows that at least $n-1$ queries are needed.

Along with the graph we maintain the parameter t that is equal to the sum of the values of already fixed variables. During most of the process we will have that $t \in \{-1, 0\}$.

Next we describe how to answer the queries. If the query asks the value of one of the variables x_i, there are two cases. If this variable is isolated in G, we fix the value of the variable in such a way that the new value of t still lies in $\{-1, 0\}$. If x_i was connected by an edge to x_j, we fix $x_i = 1$ and $x_j = -1$. The value of t does not change. In both cases we remove one connected component from G.

Next suppose the query asks AND or OR of several variables. Without loss of generality consider a query $\bigwedge_{i\in S} x_i$ for some $S \subseteq [n]$ with $|S| \geq 2$. The case of OR-query is symmetric. We can assume that none of the variables in S are already fixed, since otherwise we can either answer the query without fixing new variables, or simplify the query. Suppose there is an edge $\{x_i, x_j\}$ in G, such that $i \in S$. In this case we fix $x_i = 1$ and $x_j = -1$. The answer to the query is 1 (that is, 'false'). The number of connected components has reduced by 1 and t does not change. Next suppose that all vertices x_i with $i \in S$ are isolated. Since $|S| \geq 2$ we can consider two distinct variables x_i and x_j with $i, j \in S$. We connect these vertices by an edge with the label 1. Thus, we promise that at least one of the variables is 1 and the answer to the query is 1. Since we introduce one edge, the number of connected components reduces by 1. The value of t does not change since we do not fix any variables.

We maintain this query answering strategy until a certain condition is met. To describe this condition we need to introduce some notation. At an arbitrary point of computation denote by A the number of -1-edges, by B the number of 1-edges and by C the number of isolated vertices. Note that $A+B+C$ is the number of connected components in G.

At some point of the query answering we will have that $A+C=1$ or $B+C=1$. These cases are symmetric, without loss of generality suppose we have $A+C=1$. If at this point of the computation we have $t=0$ this might be a potential problem: note that none of 1-edges can change the balance to the negative side. If after fixing the last isolated vertex or the last -1-edge the balance does not decrease, it must be non-negative for all assignments of variables consistent with the current restrictions. So, to keep the function non-constant we will fix the last isolated vertex or -1-edge as soon as $A+C=1$. If $C=1$, we set the isolated vertex to -1 and we have $t=-1$ or $t=-2$. If $A=1$ and $t=0$, we set both of the variables connected by the edge to -1 and we have $t=-2$. If $A=1$ and $t=-1$ we set one of the variables to 1 and the other to -1 and we have $t=-1$.

In the rest of the process we have that all the remaining vertices are connected by 1-edges. Answering the queries as before we keep t the same. Thus, there is an input consistent with our answers such that MAJ_n is -1 on this input. On the other hand, if at least one of 1-edges is still present in the graph we can set both of its vertices to 1 and make the balance t non-negative. Thus, in this case there is also an assignment on which the value of the function is equal to 1. Thus, to make the function to be constant we should remove all connected components from G. □

We now proceed to the proof of the lower bound for binary queries. The main idea behind the proof is the same as in the previous theorem, but there are many technical problems we need to overcome.

Theorem 4 $\mathsf{D}_{\mathrm{B}_2}(MAJ_n) \geq n - O(\sqrt{n})$.

Proof Let us first classify the queries that can be made by functions in B_2. First note, that the queries q and $-q$ are equivalent. Next, there are functions in B_2 depending on at most one variable. They correspond to querying just one one variable. Next, there are OR-type functions in B_2, that is the function of the form $(x_i^a \vee x_j^b)$ for $a, b \in \{0, 1\}$, where $x_i^1 = x_i$ and $x_i^0 = -x_i$. Finally, there are two XOR-type functions in B_2. Due to the equivalence of a query and its negation, they correspond to the query $x_i \oplus x_j$. Note that the last query basically asks whether variables x_i and x_j are equal.

The proof strategy is similar to the one in the previous proof. During the computation we will maintain the graph G. But now the vertices of G are new fresh variables that we denote by $y_1, \ldots, y_k$ (here k is the number of vertices in G). To each of the vertices y_i we assign some integer weight c_i. Some of the vertices are connected by edges, labeled by 1 or -1. The edges form a matching. We will maintain that the weights of connected vertices are equal. We will maintain that $1 \leq c_i \leq \sqrt{n}$.

The intuition behind the graph is the following. At each point of the computation some of the original input variables x_i are fixed to constants and the rest are split into equivalence classes. All variables in each of the classes are fixed to some variable y_j, or to its negation $-y_j$. We will maintain the following relation

$$\sum_{i \,:\, x_i \text{ is unfixed}} x_i = \sum_{j=1}^{k} c_j y_j.$$

Basically, if at some point we fix all unfixed variables y_i, then the weighted sum of y_j's with weights c_j is the same as the sum of all x_i variables. In the beginning of the computation none of the variables x_i are fixed and each variable constitutes its own equivalence class. In other words, initially $k = n$, for all i we set $x_i = y_i$, $c_i = 1$ and there are no edges in the graph G.

We will answer queries in such a way that the number of connected components of G will reduce as slowly as possible. On almost all steps the number of connected components will reduce by 1, but sometimes we will have to reduce it by 2. We will show that such steps cannot happen too many times. We will show how to answer queries in such a way that to know the value of the function the decision tree must reduce the number of connected components to a small number.

We also maintain a parameter t that is equal to the sum of the values of already fixed variables x_i.

The computation will proceed in two phases. In the first phase we will maintain that $-\sqrt{n} \leq t \leq \sqrt{n}$.

We now explain how to answer the queries in the first phase. Note that each query to variables of x can be restated as a query to variables of y (since each x_i is fixed either to a constant or to some variable y_j). First we consider the case that the query addresses the variables of y that are isolated (as nodes of graph G).

Queries to Isolated Vertices Suppose the query asks the value of one of the variables y_i. We then fix the value of the variable in such a way that $c_i y_i$ and t have opposite signs. We remove the variable y_i from the graph. Since $c_i \leq \sqrt{n}$ the balance t is still at most $\sqrt{n}$ in absolute value.

Suppose the query asks whether $y_i = y_j$. Suppose first that $c_i \neq c_j$, suppose without loss of generality that $c_i > c_j$. Then the adversary reply with $y_i \neq y_j$, so we identify $y_j = -y_i$, remove the vertex y_j from G and subtract c_j from c_i. It is easy to see that all properties are maintained. The number of connected components reduces by 1.

If on the other hand $c_i = c_j$, then if $c_i > \sqrt{n}/2$, we fix $y_i = 1$ and $y_j = -1$, and remove both vertices from G. The number of connected components in this case reduces by 2. If on the other hand $c_i \leq \sqrt{n}/2$, we set $y_j = y_i$, remove y_j from G and add c_j to c_i. The number of connected components reduces by 1.

Suppose next that the query asks the function $y_i \vee \neg y_j$. In this case if $t \geq 0$ we set $y_i = -1$, otherwise we fix $y_j = 1$. In the first case we remove y_i from G and in the second we remove y_j. The answer to the query in both cases is -1. The number of connected components reduces by 1 and since $c_i, c_j \leq \sqrt{n}$ the balance t is still at most $\sqrt{n}$ in absolute value.

Finally, suppose the query asks $y_i \vee y_j$ or $y_i \wedge y_j$. Suppose first that $c_i \neq c_j$, suppose without loss of generality that $c_i > c_j$. Then again we set $y_j = -y_i$, remove the vertex y_j from G and subtract c_j from c_i. It is easy to see that all properties are maintained. The number of connected components reduces by 1.

If on the other hand $c_i = c_j$ we connect y_i and y_j by an edge. We label the edge by -1 for the case of $y_i \vee y_j$ query and by 1 for the case of $y_i \wedge y_j$ query. The number of connected components reduces by 1.

Next we proceed to queries to non-isolated vertices.

Queries to Non-Isolated Vertices First consider arbitrary queries of the form y_i, $y_i \vee \neg y_j$, $y_i \vee y_j$ or $y_i \wedge y_j$ and suppose y_i is connected by an edge to some other vertex y_l (possibly $l = j$). For all these types of queries we can fix the answer to the query by fixing y_i to some constant. We also fix y_l to the opposite constant and remove both vertices from G. Since $c_i = c_l$ the balance t does not change. The number of connected components reduces by 1.

The only remaining case is the query of the form $y_i = y_j$ for the case when y_i is connected to some other vertex y_l by an edge. If $l = j$ we simply set $y_i = 1, y_j = -1$ and remove both vertices from G. The balance t does not change and the number of connected components reduces by 1. If $l \neq j$ we let $y_i = y_j$ and $y_l = -y_j$. We remove vertices y_i and y_l from the graph. Since c_i and c_l are equal the weight of y_j does not change. The number of connected components reduce by 1.

We have described how to answer queries in the first phase. Next we describe at which point this phase ends. For this denote by A the sum of weights of vertices connected by -1-edges, by B the sum of weights of vertices connected by 1-edges and by C the sum of weights of isolated vertices. The first phase ends once either $A+C \leq 3\sqrt{n}$ or $B+C \leq 3\sqrt{n}$. Without loss of generality assume that $A+C \leq 3\sqrt{n}$ (the other case is symmetric). Note that we can claim that $A + C > \sqrt{n}$. Indeed, note that in one step of the first phase at most two vertices are removed and the weight of each vertex is at most $\sqrt{n}$, so if $A + C \leq \sqrt{n}$, then on the previous step we already had $A + C \leq 3\sqrt{n}$.

At this step of the computation we fix all isolated vertices and all vertices connected by -1-edges to -1. Before that we had $-\sqrt{n} \leq t \leq \sqrt{n}$. Thus, since $\sqrt{n} < A + C \leq 3\sqrt{n}$, after this step we have $-4\sqrt{n} \leq t < 0$ (we could be more careful here, but this only results in a multiplicative constant factor in $O(\sqrt{n})$ in the theorem). After this the second phase of the computation starts. There are only vertices connected by 1-edges remained. We answer the queries as in the first phase. Note that the balance t does not change anymore. Thus if the sum of the weights of the remaining variables is at least $4\sqrt{n}$, then the function is non-constant: on one hand setting one vertex in each pair to 1 and the other to -1 we set function to -1 and setting all variables to 1 we set the function to 1. Thus the function becomes constant only once the total weight of the remaining vertices is below $4\sqrt{n}$, that is there are less than $2\sqrt{n}$ connected components.

Let us now calculate how many queries the decision tree needs to make to set the function to a constant. In the beginning G has n connected components and in the end it has at most $2\sqrt{n}$ connected components. On each step the number of connected components reduces by 1 with some exceptions that we consider below.

On the first phase there is the case when the number of connected components reduces by 2. Note that in this case the total weight of all vertices reduces by at least $\sqrt{n}$. Since originally the total weight is n and the total weight never increases, this step can occur at most $\sqrt{n}$ times.

Between the two phases we fix a lot of variables without answering any queries. Note that their total weight is at most $3\sqrt{n}$, thus the number of connected components reduces by at most $3\sqrt{n}$.

Thus, in total the decision tree needs to make at least $n - 2\sqrt{n} - \sqrt{n} - 3\sqrt{n} = n - O(\sqrt{n})$ queries to fix the function to a constant. □

We observe that the complexity of MAJ_n drops substantially if we allow to query parities of three variables.

Lemma 8 *Suppose $f\colon \{-1,1\}^n \to \{-1,1\}$ is symmetric function. Then there is a decision tree of depth $\lceil \frac{2n}{3} \rceil$ making queries only of the form AND_2, OR_2 and XOR_3 and computing f.*

Proof Split the variables in blocks of size 3. In each block query the parity of its variables. If the answer is -1, query AND_2 of any two variables in the block. If the answer to the first query is 1, query OR_2 of any two variables in the block. It is easy to see that after these two queries we know the number of -1 variables in the block. In the case when n is not divisible by 3, if there is a small block of size 1, it requires one query to handle. If there is block of size 2 we will handle it with two queries. Knowing the number of -1 variables in all blocks is enough to output the value of the symmetric function. □

4 Parity Decision Tree Complexity of Threshold Functions

In this section we show a new lower bound for parity decision tree complexity of threshold functions.

To show that all previous techniques are not tight for some threshold functions we need an approach to prove even better lower bounds. We will do this via the following theorem.

Theorem 5 *For any k and n we have $\mathsf{D}_{\oplus}(THR_{n+2}^{k+1}) \geq \mathsf{D}_{\oplus}(THR_n^k) + 1$.*

Proof Let $s = \mathsf{D}_{\oplus}(\mathrm{THR}_{n+2}^{k+1})$. We will construct a parity decision tree for THR_n^k making no more than $s-1$ queries.

Denote the input variables to THR_n^k by $x = (x_1, \ldots, x_n) \in \{0,1\}^n$. We introduce one more variable y (which we will fix later) and consider $x_1, \ldots, x_n, y, \neg y$ as inputs to the algorithm for THR_{n+2}^{k+1}. Note that $\mathrm{THR}_n^k(x) = \mathrm{THR}_{n+2}^{k+1}(x, y, \neg y)$. Our plan is to simulate the algorithm for THR_{n+2}^{k+1} on $(x, y, \neg y)$ (possibly reorded) and save one query on our way.

Consider the first query that the algorithm for THR_{n+2}^{k+1} makes. There are two substantially different cases: the first query asks parity of a proper subset of its inputs and the first query asks the parity of all inputs. We consider these two cases separately.

Suppose first that the query does not ask the parity of all input variables. Since the function THR_{n+2}^{k+1} is symmetric we can reorder the inputs in such a way that the query contains input y and does not contain $\neg y$, that is the query asks the parity $(\bigoplus_{i \in S} x_i) \oplus y$ for some $S \subseteq [n]$. Now it is time for us to fix the value of y. We let $y = \bigoplus_{i \in S} x_i$. Then the answer to the first query is 0, we can skip it and proceed to the second query. For each next query of the algorithm for THR_{n+2}^{k+1} if it contains y or $\neg y$ (or both) we substitute them by $\bigoplus_{i \in S} x_i$ and $(\bigoplus_{i \in S} x_i) \oplus 1$ respectively. The result is the parity of some variables among $x_1, \dots, x_n$ and we make this query to our original input x. Clearly the answer to the query to x is the same as the answer to the original query to $(x, y, \neg y)$. Thus, making at most $s - 1$ queries we reach the leaf of the tree for THR_{n+2}^{k+1} and thus compute $\mathrm{THR}_{n+2}^{k+1}(x, y, \neg y) = \mathrm{THR}_n^k(x)$.

It remains to consider the case when the first query to THR_{n+2}^{k+1} is $(\bigoplus_{i=1}^n x_i) \oplus y \oplus \neg y$. This parity is equal to $\bigoplus_{i=1}^n x_i$ and we make this query to x. Now we proceed to the second query in the computation of THR_{n+2}^{k+1} and this query does not query the parity of all input variables. We perform the same analysis as above for this query: rename the inputs, fix y to the parity of subset of x to make the answer to the query to be equal to 0, simulate further queries to $(x, y, \neg y)$. Again we save one query in this case and compute $\mathrm{THR}_n^k(x)$ in at most $s - 1$ queries. □

Next we analyze the decision tree complexity of THR_n^2 functions. For them the lower bound through granularity is tight. We need this analysis to use in combination with Theorem 5 to prove lower bound for THR_n^3.

Lemma 9 *For even n we have $\mathsf{D}_\oplus(THR_n^2) = n$ and for odd n we have $\mathsf{D}_\oplus(THR_n^2) = n - 1$.*

Proof We start with a lower bound.

Here we will need to consider two Fourier coefficients, $\widehat{\mathrm{THR}}_n^2(\emptyset)$ and $\widehat{\mathrm{THR}}_n^2([n])$. We start with the latter one.

We have

$$\begin{aligned}\widehat{\mathrm{THR}}_n^2([n]) &= \frac{1}{2^n}\left(\sum_{i=0}^{1}(-1)^i\binom{n}{i} - \sum_{i=2}^{n}(-1)^i\binom{n}{i}\right)\\ &= \frac{1}{2^n}\left(2\sum_{i=0}^{1}(-1)^i\binom{n}{i} - \sum_{i=0}^{n}(-1)^i\binom{n}{i}\right) = \frac{1}{2^n}\left(2\sum_{i=0}^{1}(-1)^i\binom{n}{i} - 0\right).\end{aligned}$$

From this we can see that $\mathsf{gran}(\widehat{\mathrm{THR}}_n^2([n])) = n - \mathsf{P}\left(\sum_{i=0}^1(-1)^i\binom{n}{i}\right) - 1$ and thus

$$\mathsf{D}_\oplus(\mathrm{THR}_n^2) \geq n - \mathsf{P}\left(\sum_{i=0}^{1}(-1)^i\binom{n}{i}\right).$$

By the same analysis for $\widehat{\mathrm{THR}}_n^2(\emptyset)$ we can show that

$$\mathsf{D}_\oplus(\mathrm{THR}_n^2) \geq n - \mathsf{P}\left(\sum_{i=0}^{1}\binom{n}{i}\right).$$

Note that $\sum_{i=0}^1(-1)^i\binom{n}{i} = 1 - n$ and $\sum_{i=0}^1\binom{n}{i} = 1 + n$. From this for even n we clearly obtain a lower bound of $\mathsf{D}_\oplus(\mathrm{THR}_n^2) \geq n$. For odd n it is easy to see that one of the numbers $1 - n$ and $1 + n$ is not divisible by 4. Thus for odd n we obtain lower bound $\mathsf{D}_\oplus(\mathrm{THR}_n^2) \geq n - 1$.

It remains to prove that the lower bound is tight for odd n. To provide an algorithm making at most $n-1$ queries we again will split variables into blocks and again will assume that in the beginning all blocks are of size 1. We split all variables but one into pairs and check whether variables in each pair are equal. After this we have $(n-1)/2$ blocks of size 2 and one block of size 1. If there is a balanced block of size 2, again we can just query one variable from each of the remaining blocks thus learning the number of ones in the input. This allows us to compute the function in at most $n-1$ queries. If all blocks of size 2 contain equal variables, then note that the value of the function does not depend on the variable in the block of size 1. Indeed, $\mathrm{THR}_n^2(x) = 1$ iff $\sum_i x_i \geq 2$ iff there is a block of size 2 containing variables equal to 1. Thus it remains to query one variable from each block of size 2, which again alows us to compute the function with at most $n-1$ queries. □

Next we compute the granularity for threshold functions with threshold three.

Lemma 10 *For $n = 8m+2$ for integer m we have* $\mathsf{gran}(THR_n^3) = n-3$.

Proof For the upper bound we need to consider an arbitrary Fourier coefficient $\widehat{\mathrm{THR}}_n^3(S)$. We have

$$\widehat{\mathrm{THR}}_n^3([S]) = \frac{1}{2^n}\left(\sum_{x,|x|\leq 2}\chi_S(x) - \sum_{x,|x|\geq 3}\chi_S(x)\right)$$
$$= \frac{1}{2^n}\left(2\sum_{x,|x|\leq 2}\chi_S(x) - \sum_{x\in\{0,1\}^n}\chi_S(x)\right),$$

where by $|x|$ we denote $\sum_{i=1}^n x_i$. The second sum in the last expression is equal to either 2^n or 0 depending on S. Thus we have

$$\mathsf{gran}(\widehat{\mathrm{THR}}_n^3(S)) = n - \mathsf{P}\left(\sum_{x,|x|\leq 2}\chi_S(x)\right) - 1. \tag{1}$$

Denote the size of S by l. Then we have

$$\sum_{x,|x|\leq 2}\chi_S(x) = 1 - l + (n-l) + \frac{l(l-1)}{2} - l(n-l) + \frac{(n-l)(n-l-1)}{2},$$

where the first summand corresponds to x with $|x| = 0$, the next two summands correspond to $|x| = 1$ and the last three correspond to $|x| = 2$.

Rearranging this expression we obtain

$$\sum_{x,|x|\leq 2}\chi_S(x) = \frac{4l^2 + 2 + (n+1)(n-4l)}{2}.$$

We need to show that for $n \equiv 2 \pmod 8$ this number is divisible by 4, that is its numerator is divisible by 8. Since divisibility by 8 depends only on the remainder of n when divided by 8, it is enough to check divisibility of the numerator by 8 for $n = 2$. We have

$$4l^2 + 2 + (n+1)(n-4l) = 4l^2 + 2 + 3(2-4l) = 4(l^2 - 3l + 2),$$

which is clearly divisible by 8 for all l. Thus $\mathsf{P}\left(\sum_{x,|x|\leq 2} \chi_S(x)\right) \geq 2$ for $n = 8m+2$ and

$$\mathsf{gran}(\mathrm{THR}_n^3) \leq n-3.$$

For the lower bound on the granularity it is enough to consider Fourier coefficients $\widehat{\mathrm{THR}}_n^3(\emptyset)$ and $\widehat{\mathrm{THR}}_n^3([n])$. For them we have

$$\sum_{x,|x|\leq 2} \chi_\emptyset(x) = 1 + n + \frac{n(n-1)}{2} = \frac{2+n(n+1)}{2}$$

and

$$\sum_{x,|x|\leq 2} \chi_{[n]}(x) = 1 - n + \frac{n(n-1)}{2} = \frac{2+n(n-3)}{2}.$$

To show the lower bound it is enough to show that for any $n = 8m+2$ at least one of these expressions is not divisible by 8, that is their numerators are not divisible by 16. It is straightforward to check that for $n \equiv 2 \pmod{16}$ we have $2+n(n+1) \equiv 8 \pmod{16}$ and for $n \equiv 10 \pmod{16}$ we have $2+n(n-3) \equiv 8 \pmod{16}$. In both cases by (1) we found a Fourier coefficients with granularity at least $n-3$. □

We now show that for functions in Lemma 10 their decision tree complexity is greater than their granularity plus one. Note, that since granularity lower bound is not worse than the lower bounds through the sparsity and the degree, they also do not give tight lower bounds. Also it is easy to see that the certificate complexity does not give optimal lower bound as well (note that each input x lies in an affine subspace of dimension 2 on which the function is constant).

Theorem 6 *For $n = 8m+2$ for integer $m > 0$ we have $\mathsf{D}_\oplus(THR_n^3) = n-1$.*

Proof For the lower bound we note that $n-2$ is even and thus by Lemma 9 we have $\mathsf{D}_\oplus(\mathrm{THR}_{n-2}^2) \geq n-2$. Then by Theorem 5 we have $\mathsf{D}_\oplus(\mathrm{THR}_n^3) \geq n-1$.

For the upper bound we again view the inputs as blocks of size 1 and by checking equality of variables start combining all variables but two into larger blocks. We first combine them into blocks of size 2 and then combine all unbalanced blocks, except possibly one, to blocks of size 4. If in this process we ever encounter a balanced block we just query one variable from all other blocks thus learning the number of ones in the input in at most $n-1$ queries. If all blocks contain equal variables, then there is one block of size 2. As in the proof of Lemma 9 we observe that two variables in this block do not affect the value of the function. Indeed, $\mathrm{THR}_n^3(x) = 1$ iff $\sum_i x_i \geq 3$ iff there is a block of size 4 containing variables equal to 1. □

Thus, we have shown that previously known lower bounds are not tight for THR_{8m+2}^3. However, the gap between the lower bound and the actual complexity is 1.

Remark 1 We note that from our analysis it is straightforward to determine the complexity of THR_n^3 for all n. If $n = 4m$ or $4m+3$ for some m, then $\mathsf{D}_{\oplus}(\mathrm{THR}_n^3) = n$ and if $n = 4m + 1$ or $n = 4m + 2$, then $\mathsf{D}_{\oplus}(\mathrm{THR}_n^3) = n - 1$. The lower bounds (apart from the case covered by Theorem 6) follows from the consideration of $\widehat{\mathrm{THR}}_n^3(\emptyset)$ and $\widehat{\mathrm{THR}}_n^3([n])$ as in the proof of Lemma 10. The upper bound follows the same analysis as in the proof of Theorem 6.

5 Conclusion

The next natural question would be address the complexity of Boolean functions in decision tree model that can query functions of k variables for $k > 2$. In this model lower bounds of the form n/k are trivial, but it is not clear how to prove truly linear lower bounds. On the other hand, it is easy to show by counting argument that there are hard functions for this model. Recall, that the majority function has complexity at most $2n/3$ for $k = 3$. So more complicated functions are needed here.

Another important direction is further studies of lower bounds for parity decision trees. One of the key goals here is to show that parity decision tree complexity and sparsity are polynomially related. This would resolve Log-rank Conjecture for XOR-functions.

Appendix

A.1 Proof of Theorem 1

We start with an upper bound. The following lemma is a simple adaptation of the folklore algorithm (see, e.g. [22]).

Lemma 11

$$\mathsf{D}_{\oplus}(MAJ_n) \leq n - \mathsf{B}(n) + 1.$$

Proof Our parity decision tree will mostly make queries of the form $y \oplus z$ for a pair of variables. Note that such a query basically checks whether y and z are equal.

Our algorithm will maintain splitting of input variables into blocks of two types. We will maintain the following properties:

– the size of each block is a power of 2;
– all variables in each block of type 1 are equal;
– blocks of type 2 are balanced, that is they have equal number of ones and zeros.

In the beginning of the computation each variable forms a separate block of size one. During each step the algorithm will merge two blocks into a new one. Thus, after k steps the number of blocks is $n - k$.

The algorithm works as follows. On each step we pick two blocks of type 1 of equal size. We pick one variable from each block and query the parity of these two variables. If the variables are equal, we merge the blocks into a new block of type 1. If the variables are not equal, the new block is of type 2. The process stops when there are no blocks of type 1 of equal size.

It is easy to see that all of the properties listed above are maintained. In the end of the process we have some blocks of the second type (possibly none of them) and some blocks of the first type (possibly none of them) of pairwise non-equal size. Note that the value of the majority function is determined by the value of variables in the largest block of type 1. Indeed, all blocks of type 2 are balanced and the largest block of type 1 has more variables then all other blocks of type 1 in total. Thus, to find the value of MAJ_n it remains to query one variable from the largest block of type 1. Note, that the case when there are no blocks of type 1 in the end of the process correspond to balanced input (and even n). In this case we can tell that the output is -1 without any additional queries.

Note that the sum of sizes of all blocks is equal to n. Since the size of each block is a power of 2, there are at least $\mathsf{B}(n)$ blocks in the end of the computation (one cannot break n in the sum of less then $\mathsf{B}(n)$ powers of 2). Thus, overall we make at most $n - \mathsf{B}(n) + 1$ queries and the lemma follows. □

Before proceeding with the lower bound through granularity we briefly discuss lower bounds that can be obtained by other approaches. It is known that $\mathsf{spar}(\mathrm{MAJ}_n) = 2^{n-1}$ [19]. Thus from the sparsity lower bound we can only get $\mathsf{D}_{\oplus}(\mathrm{MAJ}_n) \geq \log \mathsf{spar}(\mathrm{MAJ}_n)/2 = \frac{n-1}{2}$.

Note also that each input $x \in \{0,1\}^n$ to MAJ_n lies in the constant-valued subcube of dimension at least $\lceil \frac{n-1}{2} \rceil$. Indeed, if $\mathrm{MAJ}_n(x) = 1$ just pick a subcube on some subset of variables of size $\lceil \frac{n-1}{2} \rceil$ containing all ones of the input. The case $\mathrm{MAJ}_n(x) = -1$ is symmetric. Thus, in the approach through certificate complexity we get $\mathsf{D}_{\oplus}(\mathrm{MAJ}_n) \geq \lceil \frac{n-1}{2} \rceil$.

Finally, we observe that the degree approach also does not give a matching lower bound.

Lemma 12 *For any n we have $\deg(MAJ_n) = 2^p$ where p is the largest integer such that $2^p \leq n$.*

Proof Consider a multilinear polynomial p over $\mathbb{F}_2$ computing MAJ_n. For a set $S \subseteq [n]$ denote by c_S the coefficient of the monomial $\prod_{i \in S} x_i$ in p. Denote $|S| = k$ and denote by $x_S \in \{0,1\}^n$ the input such that $x_i = 1$ iff $i \in S$. By [12, Section 2.1] we have

$$c_S = \bigoplus_{x \leq x_S} \mathrm{MAJ}_n(x),$$

where the order on $\{0,1\}^n$ is coordinate-wise.

From this we obtain that

$$c_S = \sum_{i=\lceil \frac{n}{2} \rceil}^{k} \binom{k}{i} = \sum_{i=\lceil \frac{n}{2} \rceil}^{k} (-1)^i \binom{k}{i} \pmod 2,$$

where the second equation follows since changing the sign of an integer summand does not change its remainder when divided by 2.

Denote $l = \lceil \frac{n}{2} \rceil$. We can simplify the latter sum as follows:

$$\sum_{i=l}^{k}(-1)^i\binom{k}{i} = \sum_{i=l}^{k}(-1)^i\left(\binom{k-1}{i-1}+\binom{k-1}{i}\right) = (-1)^l\binom{k-1}{l-1}.$$

By Kummer's theorem $\binom{k-1}{l-1}$ is odd iff the summation process of $l-1$ and $k-l$ in binary representation does not have any carry bits. Note that both $l-1 = \lceil \frac{n}{2} \rceil - 1$ and $k-l \leq \lfloor \frac{n}{2} \rfloor$ are less or equal $n/2$. Thus their binary representations are one bit shorter than the binary representation of n. The maximal k for which $\binom{k-1}{l-1}$ is odd (and thus c_S is non-zero) is the one for which $k-l$ has a binary representation inverted compared to $l-1$, that is $(k-l)+(l-1) = k-1$ has a binary representation consisting of ones only. That is, k is a power of 2 not exceeding n. $\square$

It is not hard to see that this lower bound matches the upper bound of Lemma 11 only for $n = 2^r$ and $n = 2^r + 1$. On the other hand, for example it is far from optimal by approximately a factor of 2 for $n = 2^r - 1$ for some r.

We next show that Lemma 3 gives a tight lower bound for parity decision tree complexity of MAJ_n.

Lemma 13 $\mathsf{gran}(MAJ_n) = n - \mathsf{B}(n)$.

Proof We will show that $\mathsf{gran}(\mathrm{MAJ}_n) \geq n - \mathsf{B}(n)$. The inequality in the other direction follows from Lemma 11 and Lemma 3.

We consider the Fourier coefficient $\widehat{\mathrm{MAJ}}_n([n])$ and show that its granularity is at least $n - \mathsf{B}(n)$. Let $k = \lfloor (n+1)/2 \rfloor$. Note that k is the smallest number such that MAJ_n is -1 on inputs with k ones.

Then we have

$$\begin{aligned}\widehat{\mathrm{MAJ}}_n([n]) &= \tfrac{1}{2^n}\left(\textstyle\sum_{i=0}^{k-1}(-1)^i\binom{n}{i} - \sum_{i=k}^{n}(-1)^i\binom{n}{i}\right)\\ &= \tfrac{1}{2^n}\left(\textstyle\sum_{i=0}^{n}(-1)^i\binom{n}{i} - 2\sum_{i=k}^{n}(-1)^i\binom{n}{i}\right) = \tfrac{1}{2^n}\left(0 - 2\textstyle\sum_{i=k}^{n}(-1)^i\binom{n}{i}\right).\end{aligned}$$

From this we can see that

$$\mathsf{gran}(\widehat{\mathrm{MAJ}}_n([n])) = n - \mathsf{P}\left(2\sum_{i=k}^{n}(-1)^i\binom{n}{i}\right).$$

We proceed to simplify the sum of binomials (a very similar analysis is presented in [22]):

$$\sum_{i=k}^{n}(-1)^i\binom{n}{i} = \sum_{i=k}^{n}(-1)^i\left(\binom{n-1}{i-1}+\binom{n-1}{i}\right) = (-1)^k\binom{n-1}{k-1}.$$

Thus it remains to compute $\mathsf{P}(2\binom{n-1}{k-1})$. For even $n = 2h$ we have $k = h$ and $2\binom{n-1}{k-1} = 2\binom{2h-1}{h-1} = \binom{2h}{h}$. For odd $n = 2h+1$ we have $k = h+1$ and $2\binom{n-1}{k-1} = 2\binom{2h}{h}$.

By [22, Proposition 3.4] we have $\mathsf{P}(\binom{2h}{h}) = \mathsf{B}(h)$ (alternatively this can be seen from Kummer's theorem). Finally, notice that $\mathsf{B}(2h) = \mathsf{B}(h)$ and $\mathsf{B}(2h+1) = \mathsf{B}(h)+1$. It follows that

$$\mathsf{P}\left(2\sum_{i=k}^{n}(-1)^i\binom{n}{i}\right) = \mathsf{B}(n)$$

and

$$\mathsf{gran}(\widehat{\mathrm{MAJ}}_n([n])) = n - \mathsf{B}(n). \qquad \square$$

Overall, Theorem 1 follows.

A.2 Proof of Theorem 2

We start with an upper bound.

Lemma 14 $\mathsf{D}_{\oplus}(MAJ_3^{\otimes k}) \leq (n+1)/2$.

Proof Basically, recursive majority $\mathrm{MAJ}_3^{\otimes k}$ is a function computed by a Boolean circuit whose graph is a complete ternary tree of depth k, each internal vertex is labeled by the function MAJ_3 and each leaf is labeled by a (fresh) variable.

To construct an algorithm we first generalize the problem. We consider functions computed by Boolean circuits whose graphs are ternary tree, where each non-leaf has fan-in 3 and is labeled by MAJ_3, and each leaf is labeled by a fresh variable. We will show that if the number of non-leaf vertices in the circuit is l, then the function can be computed by a parity decision tree of size $l + 1$.

The proof is by induction on l. If $l = 1$, then the function in question is just MAJ_3 and by the results of Appendix A.1 it can be computed by a parity decision tree of depth 2.

For the step of induction consider a tree with l non-leaf vertices. Consider a non-leaf vertex of the largest depth. All of its three inputs must be variables, lets denote them by y, z and t, and in this vertex the function $\mathrm{MAJ}_3(y, z, t)$ is computed. Our first query will be $y \oplus z$. It will tell us whether y and z are equal. If $y = z$ are equal, then $\mathrm{MAJ}_3(y, z, t) = y$, and if $y \neq z$, then $\mathrm{MAJ}_3(y, z, t) = t$. Thus, we can substitute the gate in our vertex by the corresponding variable and reduce the problem to the circuit with $l - 1$ non-leaf vertices. By induction hypothesis, the function computed by this circuit can be computed by at most $(l - 1) + 1 = l$ queries. Thus, our original function is computable by $l + 1$ queries.

It is left to observe that a complete ternary tree of depth k has $3^{k-1}+\ldots+1 = \frac{3^k-1}{2}$ non-leaf vertices and for this tree our algorithm makes $\frac{3^k+1}{2} = \frac{n+1}{2}$ queries. $\square$

Before proceeding to the lower bound through granularity we again discuss lower bounds that can be obtained by other techniques.

First note that each input $x \in \{0, 1\}^n$ lies in the subspace of co-dimension at most 2^k on which the function is constant. For this it is enough to show that in each x we can flip $3^k - 2^k$ variables without changing the value of the function. This is easy to check by induction on k. For $k = 1$ there are two variables that are equal to each other and we can flip the third variable without changing the value of the function. For $k > 1$ consider inputs to the MAJ_3 at the top of the circuit. Two of them are equal and by induction hypothesis we can flip $3^{k-1} - 2^{k-1}$ variables in each of them without changing the value of the function. The last input to the top gate does not affect the value of the function and we can flip all 3^{k-1} variables in it. Overall this gives us $3^k - 2^k$ variables. This gives us $\mathsf{C}_{\oplus}(\mathrm{MAJ}_3^{\otimes k}) \leq 2^k = n^{\log_3 2}$ which does not give a matching lower bound.

Also note that the polynomial computing MAJ_3 is $p(x) = x_1x_2 \oplus x_2x_3 \oplus x_1x_3$. The polynomial for $\mathrm{MAJ}_3^{\otimes k}$ can be computed by a simple composition of p with itself. It is easy to see that its degree is $2^k = n^{\log_3 2}$. Thus, an approach through polynomials over $\mathbb{F}_2$ does not give strong lower bounds.

For Fourier analytic considerations it is convenient to switch to $\{-1, 1\}$ Boolean inputs. For a variable $y \in \{0, 1\}$ let us denote by $y' \in \{-1, 1\}$ the variable $y' = 1 - 2y$. For now we will use new variables as inputs to Boolean functions.

The Fourier decomposition of MAJ_3 is

$$\mathrm{MAJ}_3(y', z', t') = \frac{1}{2}\left(y' + z' + t' - y'z't'\right). \tag{2}$$

From this the Fourier decomposition of $\mathrm{MAJ}_3^{\otimes k}$ can be obtained by recursion:

$$\begin{aligned}\mathrm{MAJ}_3^{\otimes k}(x^1, x^2, x^3) = \ & \tfrac{1}{2}(\mathrm{MAJ}_3^{\otimes k-1}(x^1) + \mathrm{MAJ}_3^{\otimes k-1}(x^2) + \mathrm{MAJ}_3^{\otimes k-1}(x^3) \\ & -\mathrm{MAJ}_3^{\otimes k-1}(x^1) \cdot \mathrm{MAJ}_3^{\otimes k-1}(x^2) \cdot \mathrm{MAJ}_3^{\otimes k-1}(x^3)),\end{aligned} \tag{3}$$

where x^1, x^2, x^3 are blocks of 3^{k-1} variables.

Lemma 2 can give lower bounds up to $n/2$ and thus in principle might give at least almost matching lower bound. However, this is not the case as we discuss below.

Note that since there is no free coefficient in the polynomial (2), Fourier coefficients arising from all three summands in the right-hand side of (3) will not cancel out with each other: no two of them have equal set of variables. Thus, if we denote $S(k) = \mathsf{spar}(\mathrm{MAJ}_3^{\otimes k})$ we have that $S(1) = 4$ and

$$S(k) = 3S(k-1) + S(k-1)^3 \tag{4}$$

for $k > 1$. On one hand, this means that $S(k) > S(k-1)^3$. This gives $S(k) > 2^{2\cdot 3^{k-1}}$. Thus $\log \mathsf{spar}(\mathrm{MAJ}_3^{\otimes k}) > 2 \cdot 3^{k-1} = 2n/3$ and $\mathsf{D}_{\oplus}(\mathrm{MAJ}_3^{\otimes k}) > n/3$.

On the other hand if we let $S'(k) = S(k) + 1/2$, it is easy to check that (4) implies

$$S'(k) < S'(k-1)^3.$$

Since $S'(1) = 9/2$ this gives $S'(k) < 2^{(\log_2 \frac{9}{2})\cdot 3^{k-1}}$. Thus,

$$\log \mathsf{spar}(\mathrm{MAJ}_3^{\otimes k}) < \left(\log_2 \frac{9}{2}\right) \cdot \frac{n}{3} < 0.723 \cdot n.$$

Thus Lemma 2 can give us a lower bound of at most $0.362 \cdot n$. We note that this upper bound on the sparsity can be further improved by letting $S'(k) = S(k) + \alpha$ for smaller α.

Now we proceed to the tight lower bound. Again we will estimate $\mathsf{gran}(\widehat{\mathrm{MAJ}_3^{\otimes k}}[n])$. Observe that this Fourier coefficient can be easily computed from (2) and (3). Indeed, from (2) we have that $\left|\widehat{\mathrm{MAJ}}_3^{\otimes 1}[n]\right| = \frac{1}{2}$. From (3) we have that

$$\left|\widehat{\mathrm{MAJ}}_3^{\otimes k}[n]\right| = \left|\frac{1}{2}(\widehat{\mathrm{MAJ}}_3^{\otimes k-1}[n])^3\right|.$$

The numerator of this Fourier coefficient equals to 1 for any k. Thus, denoting $G(n) = \mathsf{gran}(\widehat{\mathrm{MAJ}}_3^{\otimes k}[n])$ for $n = 3^k$ we have $G(3) = 1$ and

$$G(n) = 3G\left(\frac{n}{3}\right) + 1.$$

It is straightforward to check that $G(n) = \frac{n-1}{2}$. From this, Lemma 3 and Lemma 14 Theorem 2 follows.

Acknowledgments We would like to thank Alexander Kulikov for letting us know about the connection between parity decision trees and multiplicative complexity and for permission to add the proof to the paper. We also would like to thank Alexander for drawing our attention to the connection of parity decision tree complexity to the degree over $\mathbb{F}_2$.

References

1. Ben-Asher, Y., Newman, I.: Decision trees with boolean threshold queries. J. Comput. Syst Sci. **51**(3), 495–502 (1995)
2. Bernasconi, A., Codenotti, B.: Spectral analysis of boolean functions as a graph eigenvalue problem. IEEE Trans. Comput. **48**(3), 345–351 (1999)
3. Boyar, J., Find, M.G.: The relationship between multiplicative complexity and nonlinearity. In: Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II, pp. 130–140 (2014)
4. Boyar, J., Peralta, R.: Tight bounds for the multiplicative complexity of symmetric functions. Theor. Comput. Sci. **396**(1-3), 223–246 (2008)
5. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: A survey. Theor. Comput. Sci. **288**(1), 21–43 (2002)
6. Eppstein, D., Hirschberg, D.S.: From discrepancy to majority. Algorithmica **80**(4), 1278–1297 (2018)
7. Gopalan, P., O'Donnell, R., Servedio, R.A., Shpilka, A., Wimmer, K.: Testing fourier dimensionality and sparsity. SIAM J. Comput. **40**(4), 1075–1100 (2011)
8. Gröger, H.D., Turän, G.: On linear decision trees computing boolean functions. In: Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings, pp. 707–718 (1991)
9. Hatami, H., Hosseini, K., Lovett, S.: Structure of protocols for XOR functions. In: IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA, pp. 282–288 (2016)
10. Hatami, P., Kulkarni, R., Pankratov, D.: Variations on the sensitivity conjecture. Theor. Comput. Grad. Surv. **4**, 1–27 (2011)
11. Huang, H.: Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. arXiv:abs/1907.00847 (2019)
12. Jukna, S.: Boolean Function Complexity - Advances and Frontiers, volume 27 of Algorithms and Combinatorics. Springer (2012)
13. Knop, D., Pilipczuk, M., Wrochna, M.: Tight complexity lower bounds for integer linear programming with few constraints. In: 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany, pp. 44:1–44:15 (2019)
14. Kojevnikov, A., Kulikov, A.S.: Circuit complexity and multiplicative complexity of boolean functions. In: Ferreira, F., Löwe, B., Mayordomo, E., Gomes, L.M. (eds.) Programs, Proofs, Processes, 6th Conference on Computability in Europe, CiE 2010, Ponta Delgada, Azores, Portugal, June 30 - July

4, 2020. Proceedings, volume 6158 of Lecture Notes in Computer Science, pp. 239–245. Springer (2010)
15. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations, pp. 486–498 (2008)
16. Kulikov, A.S.: Personal communication
17. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press (1997)
18. Lovász, L., Saks, M.E.: Lattices Möbius functions and communication complexity. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988, pp 81–90 (1988)
19. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press (2014)
20. O'Donnell, R., Wright, J., Zhao, Y., Sun, X., Tan, L.-Y.: A composition theorem for parity kill number. In: IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014, pp. 144–154 (2014)
21. Posobin, G.: Computing majority with low-fan-in majority queries. arXiv:abs/1711.10176 (2017)
22. Saks, M.E., Werman, M.: On computing majority by comparisons. Combinatorica **11**(4), 383–387 (1991)
23. Shpilka, A., Tal, A., Volk, B.L.: On the structure of boolean functions with small spectral norm. In: Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014, pagesp. 37–48 (2014)
24. Tsang, H.Y., Wong, C.H., Xie, N., Zhang, S.: Fourier sparsity, spectral norm, and the log-rank conjecture. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, pp. 658–667 (2013)
25. Tsang, H.Y., Xie, N., Zhang, S.: Fourier sparsity of GF(2) polynomials. In: Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings, pp. 409–424 (2016)
26. Vaikuntanathan, V.: Computing blindfolded: New developments in fully homomorphic encryption. In: Ostrovsky, R. (ed.) IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011, pp. 5–16. IEEE Computer Society (2011)
27. Yao, P.: Parity decision tree complexity and 4-party communication complexity of xor-functions are polynomially equivalent. Chicago J. Theor. Comput. Sci., 2016 (2016)
28. Zhang, Z., Shi, Y.: On the parity complexity measures of boolean functions. Theor. Comput. Sci. **411**(26-28), 2612–2618 (2010)