Separating Decision Tree Complexity from Subcube Partition Complexity*

Robin Kothari[1,2], David Racicot-Desloges[3,4], and Miklos Santha[4,5]

1 **Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA, USA**
rkothari@mit.edu

2 **David R. Cheriton School of Computer Science and the Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada**

3 **Département de Mathématiques, Faculté des Sciences, Université de Sherbrooke, Sherbrooke, QC, Canada**
David.Racicot-Desloges@USherbrooke.ca

4 **Centre for Quantum Technologies, National University of Singapore, Singapore**

5 **LIAFA, CNRS, Université Paris Diderot, Paris, France**
miklos.santha@liafa.univ-paris-diderot.fr

Abstract

The subcube partition model of computation is at least as powerful as decision trees but no separation between these models was known. We show that there exists a function whose deterministic subcube partition complexity is asymptotically smaller than its randomized decision tree complexity, resolving an open problem of Friedgut, Kahn, and Wigderson ([7]). Our lower bound is based on the information-theoretic techniques first introduced to lower bound the randomized decision tree complexity of the recursive majority function.

We also show that the public-coin partition bound, the best known lower bound method for randomized decision tree complexity subsuming other general techniques such as block sensitivity, approximate degree, randomized certificate complexity, and the classical adversary bound, also lower bounds randomized subcube partition complexity. This shows that all these lower bound techniques cannot prove optimal lower bounds for randomized decision tree complexity, which answers an open question of Jain and Klauck ([9]) and Jain, Lee, and Vishnoi ([10]).

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Decision tree complexity, query complexity, randomized algorithms, subcube partition complexity

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2015.915

1 Introduction

The decision tree is a widely studied model of computation. While we have made significant progress in understanding this model (e.g., see the survey by Buhrman and de Wolf [5]), questions from over 40 years ago still remain unsolved [19].

* The research is partially funded by the Singapore Ministry of Education and the National Research Foundation, also through the Tier 3 Grant "Random numbers from quantum processes," MOE2012-T3-1-009; the European Commission IST STREP project Quantum Algorithms (QALGO) 600700; the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project); and the ARO grant Contract Number W911NF-12-1-0486.

18th Int'l Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'15) / 19th Int'l Workshop on Randomization and Computation (RANDOM'15).
Editors: Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim; pp. 915–930

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

In the decision tree model, we wish to compute a function $f : \{0,1\}^n \to \{0,1\}$ on an input $x \in \{0,1\}^n$, but we only have access to the input via a black box. The black box can be queried with an index $i \in [n]$, where $[n] = \{1, 2, \ldots, n\}$, and will respond with the value of x_i, the ith bit of x. The goal is to compute $f(x)$, while minimizing the number of queries made to the black box.

For a function $f : \{0,1\}^n \to \{0,1\}$, let $D(f)$ denote the deterministic query complexity (or decision tree complexity) of computing f, the minimum number of queries made by a deterministic algorithm that computes f correctly on all inputs. Let $R_0(f)$ denote the zero-error randomized query complexity of computing f, the minimum expected cost of a zero-error randomized algorithm that computes f correctly on all inputs. Finally, let $R(f)$ denote the bounded-error randomized query complexity of computing f, the number of queries made in the worst case by a randomized algorithm that outputs $f(x)$ on input x with probability at least $2/3$. More precise definitions can be found in Section 2.

Several lower bound techniques have been developed for query complexity over the years, most of which are based on the following observation: A decision tree that computes f and makes d queries partitions the set of all inputs, the hypercube $\{0,1\}^n$, into a set of monochromatic subcubes where each subcube has at most d fixed variables. A subcube is a restriction of the hypercube in which the values of some subset of the variables have been fixed. For example, the set of n-bit strings in which the first variable is set to 0 is a subcube of $\{0,1\}^n$ with one fixed variable. A subcube is monochromatic if f takes the same value on all inputs in the subcube. This idea is also the basis of many lower bound techniques in communication complexity [13], where a valid protocol partitions the space of inputs into monochromatic rectangles.

However, not all subcube partitions arise from decision trees, which naturally leads to a potentially more powerful model of computation. This model is called the subcube partition model in [7], but has been studied before under different names (see e.g., [4]). The deterministic subcube partition complexity of f, denoted by $D^{\text{sc}}(f)$, is the minimum d such that there is a partition of the hypercube into a set of monochromatic subcubes in which each subcube has at most d fixed variables. Since a decision tree making d queries always gives rise to such a partition, we have $D^{\text{sc}}(f) \leq D(f)$. Similarly, we define zero-error and bounded-error versions of subcube partition complexity, denoted by $R_0^{\text{sc}}(f)$ and $R^{\text{sc}}(f)$, respectively, and obtain the inequalities $R_0^{\text{sc}}(f) \leq R_0(f)$ and $R^{\text{sc}}(f) \leq R(f)$. As expected, we also have $R_0^{\text{sc}}(f) \leq D^{\text{sc}}(f)$ and $R^{\text{sc}}(f) \leq D^{\text{sc}}(f)$.

This brings up the obvious question of whether these models are equivalent. Separating them is difficult, precisely because most lower bound techniques for query complexity also lower bound subcube partition complexity. The analogous question in communication complexity is also a long-standing open problem (see [13, Open Problem 2.10] or [12, Chapter 3.2]). In fact, Friedgut, Kahn, and Wigderson [7, Question 1.1] explicitly ask if these measures are asymptotically different in the randomized model with zero error:

▶ **Question 1.** *Is there a function (family) $f = (f_n)$ such that $R_0^{\text{sc}}(f) = o(R_0(f))$?*

Similarly, one can ask the same question for bounded-error randomized query complexity. The main result of this paper resolves these questions:

▶ **Theorem 2.** *There exists a function $f = (f_h)$, with $f_h : \{0,1\}^{4^h} \to \{0,1\}$, such that $D^{\text{sc}}(f) \leq 3^h$, but $D(f) = 4^h$, $R_0(f) \geq 3.2^h$, and $R(f) = \Omega(3.2^h)$.*

This shows that query complexity and subcube partition complexity are asymptotically different in the deterministic, zero-error, and bounded-error settings. Besides resolving this

question, our result has another application. We know several techniques to lower bound bounded-error randomized query complexity, such as approximate polynomial degree [18], block sensitivity [17], randomized certificate complexity [1] and the classical adversary bound [15, 22, 2]. All these techniques are subsumed by the partition bound of Jain and Klauck [9], which in turn is subsumed by the public-coin partition bound of Jain, Lee, and Vishnoi [10]. Additionally, this new lower bound is within a quadratic factor of randomized query complexity. In other words, if $\mathrm{PPRT}(f)$ denotes the bounded-error public-coin partition bound for a function f, we have $\mathrm{PPRT}(f) \leq R(f)$ and also $R(f) = O(\mathrm{PPRT}(f)^2)$. This leaves open the intriguing possibility that this technique is optimal and is asymptotically equal to bounded-error randomized query complexity. Jain, Lee, and Vishnoi [10] indeed ask the following question:

▶ **Question 3.** *Is there a function (family) $f = (f_n)$ such that $\mathrm{PPRT}(f) = o(R(f))$?*

Our result also answers this question, because, as we show in Section 2, $\mathrm{PPRT}(f) \leq R^{\mathrm{sc}}(f)$. Thus, our asymptotic separation between $R^{\mathrm{sc}}(f)$ and $R(f)$ also separates $\mathrm{PPRT}(f)$ from $R(f)$.

We now provide a high-level overview of the techniques used in this paper. The main result is based on establishing the various complexities of a certain function. The function we choose is based on the quaternary majority function $\mathsf{4\text{-}MAJ} : \{0,1\}^4 \to \{0,1\}$, defined as the majority of the four input bits, with ties broken by the first bit. This function has low deterministic subcube complexity, $D^{\mathrm{sc}}(\mathsf{4\text{-}MAJ}) \leq 3$, but has deterministic query complexity $D(\mathsf{4\text{-}MAJ}) = 4$. From this function, we define an iterated function $\mathsf{4\text{-}MAJ}_h$ on 4^h variables by composing the function with itself h times, which gives us a function on 4^h bits. Since deterministic query complexity and deterministic subcube complexity behave nicely under composition, we have $D(\mathsf{4\text{-}MAJ}_h) = 4^h$ and $D^{\mathrm{sc}}(\mathsf{4\text{-}MAJ}_h) \leq 3^h$. The same function was also used by Savický [21], who studied this question in terms of decision tree size, as opposed to decision tree depth. These results are further discussed in Section 3. To prove Theorem 2, it remains to show that the randomized query complexity of this function is $\Omega(3.2^h)$.

We lower bound the randomized query complexity of $\mathsf{4\text{-}MAJ}_h$ using a strategy similar to the information-theoretic technique of Jayram, Kumar, and Sivakumar [11] and its simplification by Landau, Nachmias, Peres, and Vanniasegaram [14]. However, the original strategy was applied to lower bound a symmetric function (iterated 3-MAJ), whereas our function is not symmetric since the first variable of 4-MAJ is different from the rest. We modify the technique to apply it to asymmetric functions and establish the claimed lower bound. The lower bound relies on choosing a "hard distribution" of inputs and establishing a recurrence relation between the complexities of the function and its subfunctions on this distribution. Unlike 3-MAJ, where there is a natural candidate for a hard distribution, our chosen distribution is not obvious and is constrained by the fact that it must fit nicely into these recurrence relations. We prove this lower bound in Section 4. We end with some discussion and open problems in Section 5.

Subsequent work

Independent of our work, Göös, Pitassi, and Watson [8] improved the separation between deterministic query complexity and deterministic subcube complexity by exhibiting a function on n bits with deterministic query complexity $\tilde{\Omega}(n)$ and deterministic subcube complexity $\tilde{O}(n^{2/3})$. It may be possible to adapt their ideas, as done in [3], to improve the separation between randomized query complexity and subcube complexity.

2 Preliminaries

In this section, we formally define the various models of query complexity and subcube partition complexity, and the partition bound [9] and public-coin partition bound [10]. We then study the relationships between these quantities.

For the remainder of the paper, let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function on n bits and $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ be any input. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$ and let the support of a probability distribution p be denoted by $\mathrm{supp}(p)$. Lastly, we require the notion of composing two Boolean functions. If $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ are two Boolean functions, the *composed function* $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ acts on the Boolean string $y = (y_{11}, \ldots, y_{1m}, y_{21}, \ldots, y_{nm})$ as $f \circ g(y) = f(g(y_{11}, \ldots, y_{1m}), \ldots, g(y_{n1}, \ldots, y_{nm}))$.

2.1 Decision tree or query complexity

The deterministic query complexity of a function f, $D(f)$, is the minimum number of queries made by a deterministic algorithm that computes f correctly.

Formally, a *deterministic decision tree A on n variables* is a binary tree in which each leaf is labeled by either a 0 or a 1, and each internal node is labeled with a value $i \in [n]$. For every internal node of A, one of the two outgoing edges is labeled 0 and the other edge is labeled 1. On an input x, the algorithm A follows the unique path from the root to one of its leaves in the natural way: for an internal node labeled with the value i, it follows the outgoing edge labeled by x_i. The output $A(x)$ of the algorithm A on input x is the label of the leaf of this path. We say that the decision tree A *computes* f if $A(x) = f(x)$ for all x.

We define the *cost* of algorithm A on input x, denoted by $C(A, x)$, to be the number of bits queried by A on x, that is the number of internal nodes evaluated by A on x. The cost of an algorithm A, denoted $C(A)$, is the worst-case cost of the algorithm over all inputs x, that is $C(A) = \max_x C(A, x)$. Now, let $\mathcal{D}_n$ denote the set of all deterministic decision trees on n variables and let $\mathcal{D}(f) \subseteq \mathcal{D}_n$ be the set of all deterministic decision trees that compute f. We define the *deterministic query complexity of f* as $D(f) = \min_{A \in \mathcal{D}(f)} C(A)$.

One of the features of deterministic query complexity that we use in this paper is its composition property [23, 16]. This property is very intuitive: it asserts that the best way to compute the composition of f and g is to use optimal algorithms for f and g independently.

▶ **Proposition 4.** *For any two Boolean functions f and g, $D(f \circ g) = D(f)D(g)$.*

We can now move on to randomized analogs of deterministic query complexity. In a randomized algorithm, the choice of the queries might also depend on some randomness. Formally, a *randomized decision tree B on n variables* is defined by a probability distribution b over $\mathcal{D}_n$, that is by a function $b : \mathcal{D}_n \to [0,1]$ such that $\sum_{A \in \mathcal{D}_n} b(A) = 1$. On an input x, the algorithm B picks a deterministic decision tree A with probability $b(A)$ and outputs $A(x)$. Thus, for every x, the value $B(x)$ of B on x is a random variable.

We say that a randomized algorithm B computes f with *error* $\varepsilon \geq 0$ if $\Pr[B(x) = f(x)] \geq 1 - \varepsilon$ for all x, that is if $\sum_{A(x)=f(x)} b(A) \geq 1 - \varepsilon$ for all x. Let $\mathcal{R}_n$ be the set of all randomized decision trees over n bits and let $\mathcal{R}_\varepsilon(f) \subseteq \mathcal{R}_n$ be the set of all randomized decision trees that compute f with error ε. A randomized algorithm B then computes f with zero error if $\mathrm{supp}(b) \subseteq \mathcal{D}(f)$, that is the probability distribution b is completely supported on the set of deterministic decision trees that compute f. A zero-error randomized algorithm, also known as a Las Vegas algorithm, always outputs the correct answer. The cost of a zero-error randomized algorithm B on x is defined as $C(B, x) = \sum_{A \in \mathcal{D}_n} b(A)C(A, x) = \mathbb{E}[C(A, x)]$, the expected number of queries made on input x. The *zero-error randomized query complexity of*

f, denoted by $R_0(f)$, is defined as $R_0(f) = \min_{B \in \mathcal{R}_0(f)} \max_x C(B,x)$. From the definition of zero-error randomized query complexity, it is clear that $R_0(f) \leq D(f)$. The complexity $R_0(f)$ can be of strictly smaller order of growth than $D(f)$: there exists a function f for which $R_0(f) = o(D(f))$, e.g., the iterated NAND-function [20].

Randomized algorithms with error $\varepsilon > 0$ might give incorrect answer on their inputs with probability ε. We say that a randomized algorithm is of *bounded-error* (sometimes called a Monte Carlo algorithm) if on any input x, the probabilistic output is incorrect with probability at most $1/3$. The constant $1/3$ is not important and replacing it with any constant strictly between 0 and $1/2$ will only change the complexity by a constant multiplicative factor. For $\varepsilon > 0$, the cost of an ε-error randomized algorithm B on x is defined as $C(B,x) = \max_{A \in \mathrm{supp}(b)} C(A,x)$, the maximum number of queries made on input x by an algorithm in the support of b. Note how this definition differs from the one given for the zero-error case. We define the *ε-error randomized complexity of f* as $R_\varepsilon(f) = \min_{B \in \mathcal{R}_\varepsilon(f)} \max_x C(B,x)$, and the *bounded-error randomized query complexity of f* as $R(f) = R_{1/3}(f)$. Note that this definition is valid only for $\varepsilon > 0$ and does not coincide with $R_0(f)$ defined above for $\varepsilon = 0$. Setting $\varepsilon = 0$ in this definition simply gives us the deterministic query complexity $D(f)$. Nonetheless, it is true that $R(f) = O(R_0(f))$. This distinction is discussed in more detail in Section 5. Lastly, note that for all $\varepsilon > 0$, we have $R_\varepsilon(f) \leq D(f)$, and that there exist functions for which $R(f) = o(D(f))$ [20].

In order to establish lower bounds on randomized query complexity, it is useful to take a distributional view of randomized algorithms [24], that is to consider the performance of randomized algorithms on a chosen distribution over inputs. Let μ be a probability distribution over all possible inputs of length n, and let B be a randomized decision tree algorithm. The cost of B under μ is $C(B,\mu) = \sum_{x \in \{0,1\}^n} \mu(x) C(B,x) = \mathbb{E}[C(B,x)]$. We define the *$\varepsilon$-error distributional complexity of f under μ* as $\Delta^\mu_\varepsilon(f) = \min_{B \in \mathcal{R}_\varepsilon(f)} C(B,\mu)$. The following simple fact is the basis of many lower bound arguments.

▶ **Proposition 5.** *For every distribution μ over $\{0,1\}^n$, and for all $\varepsilon \geq 0$, we have $\Delta^\mu_\varepsilon(f) \leq R_\varepsilon(f)$.*

Proof. This follows by expanding out the definitions and using the simple inequality between expectation and maximum:

$$\Delta^\mu_\varepsilon(f) = \min_{B \in \mathcal{R}_\varepsilon(f)} C(B,\mu) = \min_{B \in \mathcal{R}_\varepsilon(f)} \mathbb{E}[C(B,x)] \leq \min_{B \in \mathcal{R}_\varepsilon(f)} \max_x C(B,x) = R_\varepsilon(f). \tag{1}$$

◀

2.2 Subcube partition complexity

A subcube of the hypercube $\{0,1\}^n$ is a set of n-bit strings obtained by fixing the values of some subset of the variables. In other words, a subcube is the set of all inputs consistent with a partial assignment of n bits. Formally, a *partial assignment on n variables* is a function $a : I_a \to \{0,1\}$, with $I_a \subseteq [n]$. Given a partial assignment a, we call $S(a) = \{y \in \{0,1\}^n : y_i = a(i) \text{ for all } i \in I_a\}$ the *subcube generated by a*. A set $S \subseteq \{0,1\}^n$ is a *subcube* of the hypercube $\{0,1\}^n$ if $S = S(a)$ for some partial assignment on n variables a. Clearly, for every subcube, there exists exactly one such a. We denote by I_S the domain $I_a \subseteq [n]$ of a where $S = S(a)$. For example, the set $\{0100, 0101, 0110, 0111\}$ is a subcube of $\{0,1\}^4$. It is generated by the partial assignment $a : \{1,2\} \to \{0,1\}$, where $a(1) = 0$ and $a(2) = 1$. An alternative representation of a partial assignment is by an n-bit string where a position i takes the value $a(i)$ if $i \in I_a$ and takes the value $*$ otherwise. For this example, the subcube

$\{0100, 0101, 0110, 0111\}$ is generated by the partial assignment $01**$. Finally, another useful representation is in terms of a conjunction of literals, that is satisfied by all strings in the subcube. For example, the subcube $\{0100, 0101, 0110, 0111\}$ consists exactly of all 4-bit strings that satisfy the formula $\overline{x_1} \wedge x_2$.

The subcube partition model of computation, studied previously in [7, 4, 6], is a generalization of the decision tree model. A *partition* $\{S_1, \ldots, S_\ell\}$ of $\{0,1\}^n$ is a set of pairwise disjoint subsets of $\{0,1\}^n$ that together cover the entire hypercube, that is $\bigcup_i S_i = \{0,1\}^n$ and $S_i \cap S_j = \emptyset$ for $i \neq j$.

A *deterministic subcube partition P on n variables* is a partition of $\{0,1\}^n$ with a Boolean value $s \in \{0,1\}$ associated to each subcube, that is $P = \{(S_1, s_1), (S_2, s_2), \ldots, (S_\ell, s_\ell)\}$, where each S_i is a subcube and $\{S_1, \ldots, S_\ell\}$ is a partition of $\{0,1\}^n$. If the assignment a generates S_i for some i, we call a a *generating assignment for P*. For any x, we let S^x denote the subcube containing x, that is, if $x \in S_i$, then $S^x = S_i$. We define the value $P(x)$ of P on x as s_i.

We say that a deterministic subcube partition P computes f if $P(x) = f(x)$ for all x. Note that every deterministic decision tree algorithm A computing f induces a subcube partition computing f that consists of the subcubes generated by the partial assignments defined by the root–leaf paths of the tree and the Boolean values of the corresponding leaves. We define the cost of P on x as $C(P, x) = |I_{S^x}|$, analogous to the number of queries made on input x in query complexity. We define the worst-case cost as $C(P) = \max_x C(P, x)$. Let $\mathcal{D}^{\mathrm{sc}}_n$ be the set of all deterministic subcube partitions on n variables and let $\mathcal{D}^{\mathrm{sc}}(f) \subseteq \mathcal{D}^{\mathrm{sc}}_n$ be those partitions that compute f. We define the *deterministic subcube partition complexity of f* as $D^{\mathrm{sc}}(f) = \min_{P \in \mathcal{D}^{\mathrm{sc}}(f)} C(P)$. Deterministic subcube partition complexity also satisfies a composition theorem.

▶ **Proposition 6.** *For any $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, $D^{\mathrm{sc}}(f \circ g) \leq D^{\mathrm{sc}}(f) D^{\mathrm{sc}}(g)$.*

Proof. Let $P = \{(S_1, s_1), (S_2, s_2), \ldots, (S_p, s_p)\}$ and $Q = \{(T_1, t_1), \ldots, (T_q, t_q)\}$ be optimal deterministic subcube partitions computing f and g respectively. Suppose that S_h is generated by a_h for $h \in [p]$, and that T_j is generated by b_j for $j \in [q]$. Let $I_{a_h} = \{i_1, \ldots, i_{c_h}\}$. We define the deterministic subcube partition $P \circ Q$ on nm variables as follows. The generating assignments for $P \circ Q$ are $a_h \circ (b_{j_1}, \ldots, b_{j_{c_h}})$, for all $h \in [p]$, and $j_1, \ldots, j_{c_h} \in [q]$ that satisfy $a(i_k) = t_{j_k}$ for $k \in [c_h]$. When $|I_{b_{j_k}}| = d_k$, the assignment $e = a_h \circ (b_{j_1}, \ldots, b_{j_{c_h}})$ is defined by $I_e = \{(1,1), \ldots, (1, d_1), (2,1), \ldots, (c_h, d_{c_h})\}$, and $e(k, r) = b_{j_k}(r)$ for $1 \leq r \leq d_k$. The Boolean value associated with e is s_h. It is easy to check that $P \circ Q$ computes $f \circ g$ and that $C(P \circ Q) \leq C(P) C(Q)$. ◀

As in the case of query complexity, we extend deterministic subcube complexity to the randomized setting. A *randomized subcube partition R on n variables* is given by a distribution r over all deterministic subcube partitions on n variables. As for randomized decision trees, $R(x)$ is a random variable and we say that R computes f with error $\varepsilon \geq 0$ if $\Pr[R(x) = f(x)] \geq 1 - \varepsilon$ for all x. Let $\mathcal{R}^{\mathrm{sc}}_n$ be the set of all randomized subcube partitions over n variables and $\mathcal{R}^{\mathrm{sc}}_\varepsilon(f) \subseteq \mathcal{R}^{\mathrm{sc}}_n$ be the set of all randomized subcube partitions that compute f with error ε.

The cost of a zero-error randomized subcube partition R on x is defined by $C(R, x) = \mathbb{E}[C(P, x)]$, where the expectation is taken over R. For an ε-error subcube partition R, with $\varepsilon > 0$, the cost on x is $C(R, x) = \max_{P \in \mathrm{supp}(r)} C(P, x)$. For $\varepsilon \geq 0$, we define the *ε-error randomized subcube complexity of f* by $R^{\mathrm{sc}}_\varepsilon(f) = \min_{R \in \mathcal{R}^{\mathrm{sc}}_\varepsilon(f)} \max_x C(R, x)$.

As mentioned before, a deterministic decision tree induces a deterministic subcube partition with the same cost and thus a randomized decision tree induces a randomized subcube partition with the same cost, which yields the following.

▶ **Proposition 7.** *For an n-bit Boolean function $f : \{0,1\}^n \to \{0,1\}$, we have that $D^{\mathrm{sc}}(f) \leq D(f)$ and, for all $\varepsilon \geq 0$, we have that $R^{\mathrm{sc}}_\varepsilon(f) \leq R_\varepsilon(f)$.*

2.3 Partition bounds

In 2010, Jain and Klauck [9] introduced a linear programming based lower bound technique for randomized query complexity called the partition bound. They showed that it subsumes all known general lower bound methods for randomized query complexity, including approximate polynomial degree [18], block sensitivity [17], randomized certificate complexity [1], and the classical adversary bound [15, 22, 2].

Recently, Jain, Lee, and Vishnoi [10] presented a modification of this method called the public-coin partition bound, which is easily seen to be stronger than the partition bound. Furthermore, they were able to show that the gap between this new lower bound and randomized query complexity can be at most quadratic. We define these lower bounds formally.

▶ **Definition 8** (Partition bound)**.** Let $f : \{0,1\}^n \to \{0,1\}$ be an n-bit Boolean function and let $\mathcal{S}_n$ denote the set of all subcubes of $\{0,1\}^n$. Then, for any $\varepsilon \geq 0$, let $\mathrm{prt}_\varepsilon(f)$ be the optimal value of the following linear program:

$$\underset{w_{S,z}}{\text{minimize:}} \quad \sum_{z=0}^{1} \sum_{S \in \mathcal{S}_n} w_{S,z} \cdot 2^{|I_S|} \tag{2}$$

$$\text{subject to:} \quad \sum_{S : x \in S} w_{S,f(x)} \geq 1 - \varepsilon \qquad (\text{for all } x \in \{0,1\}^n), \tag{3}$$

$$\sum_{S : x \in S} \sum_{z=0}^{1} w_{S,z} = 1 \qquad (\text{for all } x \in \{0,1\}^n), \tag{4}$$

$$w_{S,z} \geq 0 \qquad (\text{for all } S \in \mathcal{S}_n \text{ and } z \in \{0,1\}). \tag{5}$$

The *ε-partition bound* of f is defined as $\mathrm{PRT}_\varepsilon(f) = \frac{1}{2}\log_2(\mathrm{prt}_\varepsilon(f))$.

We now define the public-coin partition bound. Although our definition differs from the original definition [10], it is not too hard to see that they are equivalent. Before presenting the definition, recall that $\mathcal{D}^{\mathrm{sc}}_n$ is the set of deterministic subcube partitions on n variables, and $\mathcal{R}^{\mathrm{sc}}_\varepsilon(f)$ is the set of randomized subcube partitions that compute f with error at most $\varepsilon \geq 0$. For a randomized subcube partition $R \in \mathcal{R}^{\mathrm{sc}}_\varepsilon(f)$, we let r be the probability distribution over deterministic subcube partitions corresponding to R.

▶ **Definition 9** (Public-coin partition bound)**.** Let $f : \{0,1\}^n \to \{0,1\}$ be an n-bit Boolean function. Then, for any $\varepsilon \geq 0$, let $\mathrm{pprt}_\varepsilon(f)$ be the optimal value of the following linear program:

$$\underset{R}{\text{minimize:}} \quad \sum_{z=0}^{1} \sum_{S \in \mathcal{S}_n} \sum_{P : (S,z) \in P} r(P) \cdot 2^{|I_S|} \tag{6}$$

$$\text{subject to:} \quad R \in \mathcal{R}^{\mathrm{sc}}_\varepsilon(f). \tag{7}$$

The *ε-public-coin partition bound of f* is defined as $\mathrm{PPRT}_\varepsilon(f) = \frac{1}{2}\log_2(\mathrm{pprt}_\varepsilon(f))$.

Using the original definition, it is trivial that $\mathrm{prt}_\varepsilon(f) \leq \mathrm{pprt}_\varepsilon(f)$, since the public-coin partition bound is defined using the same linear program, with additional constraints. This statement also holds with the definitions given above, as we now prove.

▶ **Proposition 10.** *For any Boolean function f and for all $\varepsilon \geq 0$, we have that* $\mathrm{prt}_\varepsilon(f) \leq \mathrm{pprt}_\varepsilon(f)$.

Proof. Let R' be a randomized subcube partition achieving the optimal value for the linear program of $\mathrm{pprt}_\varepsilon(f)$ and r' be the corresponding probability distribution over deterministic subcube partitions. Then, for all (S, z) where S is a subcube and $z \in \{0, 1\}$, let

$$w'_{S,z} = \sum_{P:(S,z)\in P} r'(P). \tag{8}$$

This family of variables satisfies the conditions of the $\mathrm{pprt}_\varepsilon(f)$ linear program and is such that

$$\sum_{z=0}^{1} \sum_{S\in\mathcal{S}_n} w'_{S,z} \cdot 2^{|I_S|} = \sum_{z=0}^{1} \sum_{S\in\mathcal{S}_n} \sum_{P:(S,z)\in P} r'(P) \cdot 2^{|I_S|}. \tag{9}$$

◀

Recall that both partition bounds lower bound randomized query complexity, as shown in [10]. In particular, for all $\varepsilon > 0$, $\mathrm{PRT}_\varepsilon(f) \leq \mathrm{PPRT}_\varepsilon(f) \leq R_\varepsilon(f)$ and, when $\varepsilon = 0$, we have that $\mathrm{PRT}_0(f) \leq \mathrm{PPRT}_0(f) \leq D(f)$. It is not known if the zero-error partition bound also lower bounds zero-error randomized query complexity. However, as mentioned, the partition bounds also lower bound subcube partition complexity, which implies that they lower bound query complexity. The proof for query complexity easily extends to subcube partition complexity.

▶ **Proposition 11.** *For every Boolean function f and for all $\varepsilon > 0$, we have that* $\mathrm{PPRT}_\varepsilon(f) \leq R^{\mathrm{sc}}_\varepsilon(f)$ *and* $\mathrm{PPRT}_0(f) \leq D^{\mathrm{sc}}(f)$.

Proof. Let $R' \in \mathcal{R}^{\mathrm{sc}}_\varepsilon(f)$ be a randomized subcube partition that achieves $R^{\mathrm{sc}}_\varepsilon(f)$ and let r' be its corresponding probability distribution over deterministic subcube partitions. Let $P \in \mathrm{supp}(r')$. By definition, for every $(S, z) \in P$, we have that $|I_S| \leq C(P)$. Also by definition, $C(P) \leq R^{\mathrm{sc}}_\varepsilon(f)$. Furthermore, if $P = \{(S_1, z_1), (S_2, z_2), \ldots, (S_m, z_m)\}$, then

$$|P| \cdot 2^{n-C(P)} = m \cdot 2^{n-C(P)} \leq \sum_{i=1}^{m} 2^{n-|I_{S_i}|} = 2^n. \tag{10}$$

This implies that $|P| \leq 2^{C(P)} \leq 2^{R^{\mathrm{sc}}_\varepsilon(f)}$ and, therefore, that

$$\mathrm{pprt}_\varepsilon(f) = \sum_{z=0}^{1} \sum_{S\in\mathcal{S}_n} \sum_{P:(S,z)\in P} r'(P) \cdot 2^{|I_S|} \;\leq\; 2^{R^{\mathrm{sc}}_\varepsilon(f)} \sum_{z=0}^{1} \sum_{S\in\mathcal{S}_n} \sum_{P:(S,z)\in P} r'(P) \tag{11}$$

$$= 2^{R^{\mathrm{sc}}_\varepsilon(f)} \sum_{P\in\mathrm{supp}(r')} r'(P) \cdot |P| \;\leq\; 2^{R^{\mathrm{sc}}_\varepsilon(f)} \cdot 2^{R^{\mathrm{sc}}_\varepsilon(f)} \sum_{P\in\mathrm{supp}(r')} r'(P) \tag{12}$$

$$= 2^{2R^{\mathrm{sc}}_\varepsilon(f)}. \tag{13}$$

The first inequality holds since $|I_S| \leq R^{\mathrm{sc}}_\varepsilon(f)$, and the second inequality uses the fact that $|P| \leq 2^{R^{\mathrm{sc}}_\varepsilon(f)}$. Setting $\varepsilon = 0$ gives $\mathrm{PPRT}_0(f) \leq D^{\mathrm{sc}}(f)$. ◀

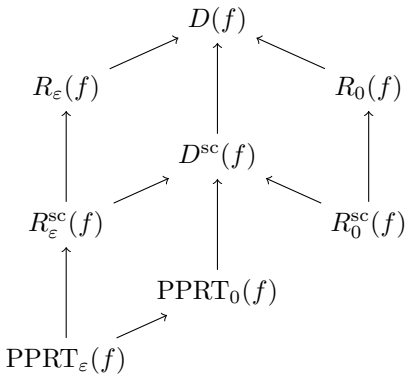

Figure 1 Relationships between the complexity measures introduced. An arrow from X to Y represents $X \leq Y$. For example, $D^{\mathrm{sc}}(f) \to D(f)$ means $D^{\mathrm{sc}}(f) \leq D(f)$.

The following theorem summarizes the known relations between the introduced complexity measures.

▶ **Theorem 12.** *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and for all $\varepsilon > 0$, the relations indicated in Figure 1 hold.*

Proof. The upper three vertical arrows represent the relations between query complexity and subcube partition complexity established in Proposition 7. The remaining vertical arrows represent the relations between the public-coin partition bounds and subcube partition complexity established in Proposition 11. The other inequalities are immediate and follow from their definitions. ◀

3 Iterated quaternary majority function

We now introduce the function we use to separate randomized query complexity from subcube partition complexity and establish some of its properties.

Let MAJ denote the Boolean majority function of its input bits when the number of bits is odd. The quaternary majority function 4-MAJ : $\{0,1\}^4 \to \{0,1\}$ is defined by $\text{4-MAJ}(x_1,x_2,x_3,x_4) = x_1(x_2 \vee x_3 \vee x_4) \vee x_2x_3x_4$. This function was introduced in [21]. We call it 4-MAJ, because the output of the function is the majority of its input bits, with the first variable breaking equality in its favor. In other words, the first variable has two votes, while the others have one, that is $\text{4-MAJ}(x_1,x_2,x_3,x_4) = \text{MAJ}(x_1,x_1,x_2,x_3,x_4)$. This function has previously been used to separate deterministic decision tree size from deterministic subcube partition size [21]. We use this function because its subcube partition complexity is smaller than its query complexity.

▶ **Proposition 13.** *We have $D^{\mathrm{sc}}(\text{4-MAJ}) = 3$ and $D(\text{4-MAJ}) = 4$.*

Proof. Observe that, for any choice of $w \in \{0,1\}$, we have that

$$\text{4-MAJ}(0,0,1,w) = \text{4-MAJ}(0,w,0,1) = \text{4-MAJ}(0,1,w,0) = \text{4-MAJ}(w,0,0,0) = 0$$

and that

$$\text{4-MAJ}(1,1,0,w) = \text{4-MAJ}(1,w,1,0) = \text{4-MAJ}(1,0,w,1) = \text{4-MAJ}(w,1,1,1) = 1.$$

The subcubes generated by these 8 partial assignments are disjoint and of size two, forming a partition of $\{0,1\}^4$. Thus, with the right Boolean values, they form a deterministic subcube

partition that computes 4-MAJ. Since all partial assignments have length 3, $D^{sc}(\text{4-MAJ}) \leq 3$. Although we do not use the inequality $D^{sc}(\text{4-MAJ}) \geq 3$ in our results, this can be verified by enumerating all deterministic subcube partitions with complexity 2. Furthermore, $D(\text{4-MAJ}) \leq 4$ since any function can be computed by querying all input bits. $D(\text{4-MAJ}) \geq 4$ can be shown either by enumerating all decision trees that make 3 queries or by using the lower bound in the next section. ◀

While our results only require us to show lower bounds on the randomized query complexity of 4-MAJ, we want to mention that the randomized query complexity of 4-MAJ is indeed smaller than its deterministic query complexity.

▶ **Proposition 14.** *For the* 4-MAJ *function,* $R_0(\text{4-MAJ}) \leq 13/4 = 3.25$.

Proof. The randomized algorithm achieving this complexity is simple: with probability 1/4, the algorithm queries the first variable and then it checks if the other variables all have the opposite value; with probability 3/4, it checks if the last three variables have all the same value and, if not, it queries the first variable. ◀

Since the 4-MAJ function separates deterministic subcube complexity from deterministic query complexity, a natural candidate for a function family that separates these measures is the iterated quaternary majority function, 4-MAJ_h, defined recursively on 4^h variables, for $h \geq 0$. In the base case, 4-MAJ_0 is the identity function on one bit. For $h > 0$, we define $\text{4-MAJ}_h = \text{4-MAJ} \circ \text{4-MAJ}_{h-1}$. In other words, for $h > 0$, let x be an input of length 4^h, and for $i \in \{1, 2, 3, 4\}$, let $x^{(i)}$ denote the i^{th} quarter of x, that is $|x^{(i)}| = 4^{h-1}$ and $x = x^{(1)}x^{(2)}x^{(3)}x^{(4)}$. Then, we have that $\text{4-MAJ}_h(x) = \text{4-MAJ}(\text{4-MAJ}_{h-1}(x^{(1)}), \text{4-MAJ}_{h-1}(x^{(2)}), \text{4-MAJ}_{h-1}(x^{(3)}), \text{4-MAJ}_{h-1}(x^{(4)}))$.

The function 4-MAJ_h inherits several properties from 4-MAJ. It has low deterministic subcube complexity, but high deterministic query complexity:

▶ **Proposition 15.** *For all* $h \geq 0$, $D^{sc}(\text{4-MAJ}_h) \leq 3^h$ *and* $D(\text{4-MAJ}_h) = 4^h$.

Proof. For $h = 0$, the statement is trivial and for $h = 1$, the statement is Proposition 13. Proposition 4 and Proposition 6 used recursively imply the result. ◀

We now introduce terminology that we use to refer to this function. We view 4-MAJ_h as defined by the read-once formula on the complete quaternary tree T_h of height h in which every internal node is a 4-MAJ gate. We identify the leaves of T_h from left to right with the integers $1, \ldots, 4^h$. For an input $x \in \{0,1\}^{4^h}$, the bit x_i defines the *value* of the leaf i. We then evaluate recursively the values of the internal nodes. The value of the root is $\text{4-MAJ}_h(x)$. For every internal node v in T_h, we denote its children by v_1, v_2, v_3 and v_4, from left to right. For any node v in T_h, let $Z(v)$ denote the set of variables associated with the leaves in the subtree rooted at v. We say that a node v is at level ℓ in T_h if the distance between v and the leaves is ℓ. The root is therefore at level h, and the leaves are at level 0. For $0 \leq \ell \leq h$, the set nodes at level ℓ is denoted by $\mathrm{T}_h(\ell)$.

4 Randomized query complexity of 4-MAJ_h

In this section, we prove our main technical result, a lower bound on the randomized query complexity of 4-MAJ_h. We prove this by using distributional complexity, that is by using the inequality in Proposition 5. First, we define a "hard distribution" d_h for which we will show that $\Delta_\varepsilon^{d_h}(\text{4-MAJ}_h) \geq (1 - 2\varepsilon)(16/5)^h$, which implies our main result (Theorem 2).

4.1 The hard distribution

Intuitively, the distribution we use in our lower bound has to be one on which it is difficult to compute 4-MAJ$_h$. We start by defining a hard distribution for 4-MAJ and extend it to 4-MAJ$_h$ in the natural way: by composing it with itself.

The *hard distribution* d on inputs of length 4 is defined from d^0 and d^1, the respective hard distributions for 0-inputs and 1-inputs of length 4, by setting $d(x) = \frac{1}{2}d^b(x)$ when 4-MAJ$(x) = b$. We define d^0 as

$$d^0(1000) = \frac{2}{5}, \quad d^0(0011) = d^0(0101) = d^0(0110) = \frac{1}{6},$$
$$d^0(0001) = d^0(0010) = d^0(0100) = \frac{1}{30}, \text{ and } d^0(0000) = 0. \tag{14}$$

The definition of d^1 is analogous, or can be defined by $d^1(x_1, x_2, x_3, x_4) = d^0(1 - x_1, 1 - x_2, 1 - x_3, 1 - x_4)$. Given that the function 4-MAJ is symmetric in x_2, x_3, and x_4, there are only 4 equivalence classes of 0-inputs, to which we have assigned probability masses $2/5, 1/2, 1/10$, and 0, and then distributed the probabilities uniformly inside each class. The probabilities were chosen to make the recurrence relations in Lemma 17 and Lemma 18 work, while putting more weight on the intuitively difficult inputs. For example $x = 0000$ seems like an easy input since all inputs that are Hamming distance 1 from it are also 0-inputs, and thus reading any 3 bits of this input is sufficient to compute the function. In Lemma 18 we will give an equivalent characterisation of the hard distribution which is more directly related to the recurrence relations in the lemmas.

From this distribution we recursively define, for $h \geq 0$, the *hard distribution* d_h on inputs of length 4^h. In the base case, $d_0(0) = d_0(1) = \frac{1}{2}$. For $h > 0$, as for d, the distribution d_h is defined from d_h^0 and d_h^1, the respective hard distributions for 0-inputs and 1-inputs of length 4^h, by setting $d_h(x) = \frac{1}{2}d_h^b(x)$ when 4-MAJ$(x) = b$. Let $x = x^{(1)}x^{(2)}x^{(3)}x^{(4)}$ be a b-input, where $x^{(i)}$ is a b_i-input of length 4^{h-1}, for $i \in \{1, 2, 3, 4\}$. Then, $d_h^b(x) = d^b(b_1b_2b_3b_4) \cdot \Pi_{i=1}^4 d_{h-1}^{b_i}(x^{(i)})$. It is easily seen that according to d_h, for each node v in T_h, if the value of v is b, then the children of v have values distributed according to d^b. With the additional constraints that the root has uniform distribution over $\{0, 1\}$, this actually makes an alternative definition of d_h.

We will also require the notion of a minority path in our proof. For a given input, a minority path is a path from the root to a leaf in which each node has a value different from its parent's value. (Recall that the value of a node is the function 4-MAJ evaluated on the values of its children.) For example, for the 4-MAJ function, on input 1000 the unique minority path is the edge from the root to the first variable, whereas on input 1001 there are two minority paths from the root to the second and third variable. In general, since there may be multiple such paths, the minority path is defined to be a random variable over all root–leaf paths. Formally, for every input $x \in \{0, 1\}^{4^h}$, we define the *minority path* $M(x)$ as a random variable over all root–leaf paths in T_h as follows. First, the root is always in $M(x)$. Then, for any node v in $M(x)$, if there is a unique child w of v with value different from that of v, then $w \in M(x)$. Otherwise, there are exactly two children with different values, and we put each of them in $M(x)$ with probability $\frac{1}{2}$. Note that with this definition, if x is chosen from the hard distribution d_h, conditioned on the node v being in $M(x)$, the first child v_1 is in the minority path with probability $\frac{2}{5}$, and the child v_i is in the minority path with probability $\frac{1}{5}$, for $i \in \{2, 3, 4\}$.

4.2 Complexity of 4-MAJ$_h$ under the hard distribution

We can now lower bound the distributional complexity of 4-MAJ$_h$ under the hard distribution.

▶ **Theorem 16.** *For all $\varepsilon \geq 0$ and $h \geq 0$, we have $\Delta_\varepsilon^{d_h}(\text{4-MAJ}_h) \geq (1-2\varepsilon)(16/5)^h$.*

To show this, we need to define some quantities. For a deterministic decision tree algorithm A computing 4-MAJ$_h$, let $L_A(x)$ denote the set of variables queried by A on input x. Let B be a randomized decision tree algorithm that computes 4-MAJ$_h$ with error ε, and let b be its probability distribution over deterministic algorithms. For any two (not necessarily distinct) nodes of T_h, u and v, we define the function $E_B(v,u)$ as $E_B(v,u) = \mathbb{E}\big[|Z(v) \cap L_A(x)| \big| u \in M(x)\big]$, where the expectation is taken over b, d_h and the randomness in $M(x)$. In words, $E_B(v,u)$ is the expected number of queries below the node v over the randomness of B, the hard distribution and the randomness for the choice of the minority path, under the condition that u is in the minority path. For $0 \leq \ell \leq h$, we also define the functions $J_B^\varepsilon(h,\ell)$, $K_B^\varepsilon(h,\ell)$, $J^\varepsilon(h,\ell)$, and $K^\varepsilon(h,\ell)$ by

$$J_B^\varepsilon(h,\ell) = \sum_{v \in \mathrm{T}_h(\ell)} E_B(v,v), \tag{15}$$

$$K_B^\varepsilon(h,\ell) = \sum_{v \in \mathrm{T}_h(\ell)} \left(\frac{2}{5} \sum_{i=2}^{4} E_B(v_i, v_1) + \frac{1}{5} \sum_{j=2}^{4} \sum_{i \neq j} E_B(v_i, v_j) \right), \tag{16}$$

$$J^\varepsilon(h,\ell) = \min_{B \in \mathcal{R}_\varepsilon(\text{4-MAJ}_h)} J_B^\varepsilon(h,\ell) \quad \text{and} \quad K^\varepsilon(h,\ell) = \min_{B \in \mathcal{R}_\varepsilon(\text{4-MAJ}_h)} K_B^\varepsilon(h,\ell). \tag{17}$$

Observe that $J^\varepsilon(h,h) = \min_{B \in \mathcal{R}_\varepsilon(\text{4-MAJ}_h)} \mathbb{E}[C(B,x)] = \Delta_\varepsilon^{d_h}(\text{4-MAJ}_h)$.

The proof of Theorem 16 essentially follows from the following two lemmas.

▶ **Lemma 17.** *For all $0 < l \leq h$, we have that $J^\varepsilon(h,\ell) \geq K^\varepsilon(h,\ell) + \frac{1}{5} J^\varepsilon(h,\ell-1)$.*

Proof. This proof mainly involves expanding the quantity $E_B(v,v)$ in terms of $E_B(v_i,v_j)$, where v_1, v_2, v_3, and v_4 are the children of v. Since, for every node v, the set of leaves below v is the disjoint union of the sets of leaves below its children, for every B we have that

$$J_B^\varepsilon(h,\ell) = \sum_{v \in \mathrm{T}_h(\ell)} \sum_{i=1}^{4} E_B(v_i, v). \tag{18}$$

By conditioning on the minority child of v, we get that

$$J_B^\varepsilon(h,\ell) = \sum_{v \in \mathrm{T}_h(l)} \sum_{i=1}^{4} \sum_{j=1}^{4} E_B(v_i,v_j) \Pr[v_j \in M(x) | v \in M(x)] \ . \tag{19}$$

As mentioned before, if x is chosen according to the distribution d_h, if $v \in M(x)$, then $v_1 \in M(x)$ with probability $\frac{2}{5}$ and $v_i \in M(x)$ with probability $\frac{1}{5}$, for $i \in \{2,3,4\}$. Substituting these values we get

$$J_B^\varepsilon(h,\ell) = K_B^\varepsilon(h,\ell) + \frac{1}{5} J_B^\varepsilon(h,\ell-1) + \frac{1}{5} E_B(v_1,v_1). \tag{20}$$

Discarding the last term on the right hand side, which is always non-negative, and taking the minimum over B for all remaining terms gives the result. ◀

Having established this, we need to relate $K^\varepsilon(h,\ell)$ with $J^\varepsilon(h-1,\ell-1)$. Informally, given a randomized algorithm that performs well on 4-MAJ$_h$ at depth ℓ, we construct another algorithm that performs well on 4-MAJ$_{h-1}$ at depth $\ell-1$.

▶ **Lemma 18.** *For all $0<\ell\le h$, we have that $K^\varepsilon(h,\ell)\ge 3J^\varepsilon(h-1,\ell-1)$.*

Proof. For any $B\in\mathcal{R}_\varepsilon(\text{4-MAJ}_h)$, we will construct $B'\in\mathcal{R}_\varepsilon(\text{4-MAJ}_{h-1})$ such that

$$\frac{1}{3}K^\varepsilon_B(h,\ell)=J^\varepsilon_{B'}(h-1,\ell-1). \tag{21}$$

Taking the minimum over all $B\in\mathcal{R}_\varepsilon(\text{4-MAJ}_h)$ implies the statement.

We start by giving a high level description of our construction of B' from B. First B' will choose a random injective mapping from $\{x_1,\dots,x_{4^{h-1}}\}$ to $\{x_1,\dots,x_{4^h}\}$, identifying each variable of T_{h-1} with some variable of T_h. Then, it will choose a random restriction for the remaining variables of T_h. Note that these choices are not made uniformly. Let B_r denote the algorithm for 4^{h-1} variables defined by B after the identification and the restriction according to randomness r. B' then simply executes B_r. Our embedding of the smaller instance into the larger instance is done in a way that preserves the output.

We now describe the random identification and restriction in detail. First, observe that there is a natural correspondence between the nodes of $\mathrm{T}_{h-1}(\ell-1)$ and $\mathrm{T}_h(\ell)$ (since they are of the same size): we simply map the ith node of $\mathrm{T}_{h-1}(\ell-1)$ from the left to the ith node of $\mathrm{T}_h(\ell)$ from the left. For every node $u\in\mathrm{T}_{h-1}(\ell-1)$, let $v\in\mathrm{T}_h(\ell)$ be its corresponding node. The algorithm B' makes the following independent random choices. To generate the random identification, B' randomly chooses a child w of v, where $w=v_1$ with probability $\frac{1}{5}$, and $w=v_i$ with probability $\frac{4}{15}$, for $i\in\{2,3,4\}$. Then, the variables of $Z(u)$ and the variables of $Z(w)$ are identified naturally, again from left to right.

For generating the random restriction, B' first generates random values for the three siblings of w. If $w=v_1$, then it chooses for (v_2,v_3,v_4) one of the six strings from the set $\{001,010,100,110,101,011\}$ uniformly at random. If $w\in\{v_2,v_3,v_4\}$, it chooses for v_1, a uniformly random value from $\{0,1\}$, and for the remaining two siblings, it picks the opposite value. From this, the restriction is generated as follows: for each sibling w' of w with value $b\in\{0,1\}$, a random string of length $4^{\ell-1}$ is generated according to $d^b_{\ell-1}$, and the variables in $Z(w')$ receive the values of this string. This finishes the description of B'.

We now show that $B'\in\mathcal{R}_\varepsilon(\text{4-MAJ}_{h-1})$. Because of the identification of the variables of $Z(u)$ and $Z(w)$, for every $x\in\{0,1\}^{4^{h-1}}$, the value of u coincides with the value of w. The random values chosen for the siblings of w are such that whatever value w gets, it is always a majority child of v. Therefore, for every input x, and for every randomness r, the value of u is the same as the value of v. This implies that for every x and every randomness r, the value of the roots of $\mathrm{T}_{h-1}(\ell-1)$ and $\mathrm{T}_h(\ell)$ are the same. Since B is an algorithm which computes 4-MAJ$_h$ with error at most ε, this means that B_r is an algorithm which computes 4-MAJ$_{h-1}$ with error at most ε, for every randomness r. From this, it follows that $B'\in\mathcal{R}_\varepsilon(\text{4-MAJ}_{h-1})$.

Finally we prove the equality in (21). For this, the main observation (which can be checked by direct calculation) is that when w gets a random Boolean value, the distribution of values generated by B' on the children of v is exactly the hard distribution d. Therefore,

$E_{B'}(u,u) = E_B(w,v)$. Consequently, we have that

$$\begin{aligned} J^{\varepsilon}_{B'}(h-1,\ell-1) &= \sum_{v\in \mathrm{T}_h(\ell)} E_B(w,v) = \sum_{v\in \mathrm{T}_h(\ell)} \sum_{i=1}^{4} E_B(v_i,v)\Pr[w=v_i | v\in M(x)] \\ &= \sum_{v\in \mathrm{T}_h(\ell)} \sum_{i=1}^{4}\sum_{j=1}^{4} E_B(v_i,v_j)\Pr[w=v_i]\Pr[v_j\in M(x) | w=v_i, v\in M(x)] \\ &= \frac{1}{3} K^{\varepsilon}_B(h,\ell). \end{aligned} \tag{22}$$

The third equality holds since the choice of w is independent from the fact that v is in the minority path. For the last equality, we used that the conditional probabilities evaluate to the following values:

$$\begin{aligned} &\Pr[v_j\in M(x) | w=v_j, v\in M(x)] = 0, && \text{for } j\in\{1,2,3,4\}; \\ &\Pr[v_j\in M(x) | w=v_1, v\in M(x)] = \frac{1}{3}, && \text{for } j\neq 1; \\ &\Pr[v_1\in M(x) | w=v_i, v\in M(x)] = \frac{1}{2}, && \text{for } i\neq 1; \\ &\Pr[v_j\in M(x) | w=v_i, v\in M(x)] = \frac{1}{4}, && \text{for } i,j\in\{2,3,4\} \text{ and } i\neq j. \end{aligned}$$ ◀

We can now return to proving Theorem 16.

Proof of Theorem 16. We claim that, for all $0\leq \ell\leq h$, we have that

$$J^{\varepsilon}(h,\ell) \geq (1-2\varepsilon)(16/5)^{\ell}. \tag{23}$$

The proof is done by induction on ℓ. For the base case $\ell = 0$, let $B\in\mathcal{R}_{\varepsilon}(\text{4-MAJ}_h)$. Then, we have that

$$J^{\varepsilon}_B(h,0) = \sum_{v\in \mathrm{T}_h(0)} \Pr[B \text{ queries } v \big| v\in M(x)]. \tag{24}$$

Observe that any randomized decision tree algorithm computing a nonconstant function with error at most ε must make at least one query with probability at least $1-2\varepsilon$, since otherwise it would output 0 or 1 with probability greater than ε, and thus on some input would err too much. Let therefore A be a deterministic algorithm from the support of B which makes at least one query. Then

$$\sum_{v\in \mathrm{T}_h(0)} \Pr[A \text{ queries } v \big| v\in M(x)] \geq \sum_{v\in \mathrm{T}_h(0)} \Pr[A \text{ first query is } v \big| v\in M(x)] = 1, \tag{25}$$

since in the summation the term corresponding to the first query of A is 1, whereas all other terms are 0. Thus, $J^{\varepsilon}(h,0)\geq 1-2\varepsilon$ for all $h\geq 0$.

Now let $\ell > 0$, and assume the statement holds for $\ell-1$. For $h\geq \ell$, using Lemma 17 and Lemma 18, we get that $J^{\varepsilon}(h,\ell)\geq 3J^{\varepsilon}(h-1,\ell-1) + \frac{1}{5}J^{\varepsilon}(h,\ell-1)$. Therefore, by the induction hypothesis, we have that

$$J^{\varepsilon}(h,\ell) \geq 3(1-2\varepsilon)\left(\frac{16}{5}\right)^{\ell-1} + \frac{1}{5}(1-2\varepsilon)\left(\frac{16}{5}\right)^{\ell-1} = (1-2\varepsilon)\left(\frac{16}{5}\right)^{\ell}. \tag{26}$$

The theorem follows when we set $h=\ell$ by noting that $J^{\varepsilon}(h,h)\leq \Delta^{d_h}_{\varepsilon}(\text{4-MAJ}_h)$. ◀

Combining Proposition 15 and Theorem 16 gives us our main result, an asymptotic separation between deterministic subcube partition complexity and randomized query complexity:

▶ **Theorem 2.** *There exists a function* $f = (f_h)$*, with* $f_h : \{0,1\}^{4^h} \to \{0,1\}$*, such that* $D^{\mathrm{sc}}(f) \leq 3^h$*, but* $D(f) = 4^h$*,* $R_0(f) \geq 3.2^h$*, and* $R(f) = \Omega(3.2^h)$*.*

We can also immediately deduce that the 4-MAJ$_h$ function positively answers both Question 1 and Question 3.

▶ **Corollary 19.** *We have that* $R_0^{\mathrm{sc}}(\text{4-MAJ}_h) = o(R_0(\text{4-MAJ}_h))$*.*

▶ **Corollary 20.** *For* $0 \leq \varepsilon \leq 1/3$*, we have that* $\mathrm{PPRT}_\varepsilon(\text{4-MAJ}_h) = o(R_\varepsilon(\text{4-MAJ}_h))$*.*

5 Discussion and open problems

Our main result is actually stronger than stated. In addition to the zero-error and ε-error randomized query complexities we defined, we can also define ε-error expected randomized complexity. In this model, we only charge for the expected number of queries made by the randomized algorithm, like in the zero-error case, but we also allow the algorithm to err. Formally, the *ε-error expected randomized query complexity of f* is $R_\varepsilon^{\mathrm{exp}}(f) = \min_{B \in \mathcal{R}_\varepsilon(f)} \max_x C(B,x))$. Observe that since this generalizes zero-error randomized query complexity, $R_0^{\mathrm{exp}}(f) = R_0(f)$, and it is immediate that, for all $\varepsilon \geq 0$, we have that $R_\varepsilon^{\mathrm{exp}}(f) \leq R_\varepsilon(f) \leq D(f)$.

Randomized query complexity is usually defined in the worst case [5], that is as $R_\varepsilon(f)$ instead of $R_\varepsilon^{\mathrm{exp}}(f)$. The main reason for not dealing with these measures separately is that worst case and expected randomized complexities are closely related. We have already observed that (obviously), in expectation, one can not make more queries than in the worst case. On the other hand, if for some constant $\eta > 0$ we let the randomized algorithm that achieves $R_\varepsilon^{\mathrm{exp}}(f)$ make $\frac{1}{2\eta} R_\varepsilon^{\mathrm{exp}}(f)$ queries, and give a random answer in case the computation is not finished, we get an algorithm of error $\varepsilon + \eta$ which never makes more than $\frac{1}{2\eta} R_\varepsilon^{\mathrm{exp}}(f)$ queries. Therefore, for all $\varepsilon \geq 0$ and $\eta > 0$, we have that $R_{\varepsilon+\eta}(f) \leq \frac{1}{2\eta} R_\varepsilon^{\mathrm{exp}}(f)$.

The result we show actually lower bounds $R_\varepsilon^{\mathrm{exp}}(f)$ as well. Thus, a stronger version of our result is the following: For all $\varepsilon \geq 0$, $R_\varepsilon^{\mathrm{exp}}(\text{4-MAJ}_h) \geq (1 - 2\varepsilon)(3.2)^h$.

We end with some open problems. It would be interesting to exactly pin down the randomized query complexity of 4-MAJ$_h$. For example we know that $R_0(\text{4-MAJ}_h) \geq 3.2^h$ and $R_0(\text{4-MAJ}_h) \leq 3.25^h$. The best separation between subcube partition complexity and query complexity remains open, even in the deterministic case.

Finally it would be interesting to know if the partition bounds also lower bound expected randomized query complexity, and in particular whether the zero-error partition bound lower bounds zero-error randomized query complexity.

References

1 S. Aaronson. Quantum certificate complexity. *SIAM Journal on Computing*, 35(4):804–824, 2006.

2 S. Aaronson. Lower bounds for local search by quantum arguments. *Journal of Computer and System Sciences*, 74(3):313–332, 2008.

3 A. Ambainis, K. Balodis, A. Belovs, T. Lee, M. Santha, and J. Smotrovs. Separations in query complexity based on pointer functions. *arXiv preprint* arXiv:1506.04719, 2015.

4 Y. Brandman, A. Orlitsky, and J. Hennessy. A spectral lower bound technique for the size of decision trees and two-level AND/OR circuits. *IEEE Transactons on Computers*, 39(2):282–287, February 1990.

5 H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

6 S. Chakraborty, R. Kulkarni, S. V. Lokam, and N. Saurabh. Upper bounds on Fourier entropy. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:52, 2013.

7 E. Friedgut, J. Kahn, and A. Wigderson. Computing graph properties by randomized subcube partitions. In *Randomization and Approximation Techniques in Computer Science*, volume 2483 of *Lecture Notes in Computer Science*, pages 105–113. Springer Berlin Heidelberg, 2002.

8 M. Göös, T. Pitassi, and T. Watson. Deterministic communication vs. partition number. *To appear in the Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

9 R. Jain and H. Klauck. The partition bound for classical communication complexity and query complexity. In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC'10, pages 247–258, 2010.

10 R. Jain, T. Lee, and N. Vishnoi. A quadratically tight partition bound for classical communication complexity and query complexity. *arXiv preprint* arXiv:1401.4512, 2014.

11 T. S. Jayram, R. Kumar, and D. Sivakumar. Two applications of information complexity. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing (STOC)*, STOC'03, pages 673–682, New York, NY, USA, 2003. ACM.

12 S. Jukna. *Boolean Function Complexity: Advances and Frontiers*. Algorithms and Combinatorics. Springer, 2012.

13 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 2006.

14 I. Landau, A. Nachmias, Y. Peres, and S. Vanniasegaram. The lower bound for evaluating a recursive ternary majority function: an entropy-free proof. Undergraduate Research Reports, Department of Statistics, University of California, Berkeley, 2006.

15 S. Laplante and F. Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. *SIAM Journal on Computing*, 38(1):46–62, 2008.

16 A. Montanaro. A composition theorem for decision tree complexity. *Chicago Journal of Theoretical Computer Science*, 2014(6), July 2014.

17 N. Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991.

18 N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 15(4):557–565, 1995.

19 A. L. Rosenberg. On the time required to recognize properties of graphs: a problem. *SIGACT News*, 5(4):15–16, 1973.

20 M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 29–38, 1986.

21 P. Savický. On determinism versus unambiguous nondeterminism for decision trees. *ECCC*, TR02-009, 2002.

22 R. Špalek and M. Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006.

23 A. Tal. Properties and applications of boolean function composition. In *Proc. of the 4th Conf. on Innovations in Theoretical Computer Science*, ITCS'13, pages 441–454, 2013.

24 A. Yao. Probabilistic computations: Toward a unified measure of complexity. *Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.