Electronic Colloquium on Computational Complexity, Report No. 39 (2023)

Query Complexity of Search Problems

Arkadev Chattopadhyay
Tata Institute of Fundamental Research, Mumbai, India

Yogesh Dahiya
The Institute of Mathematical Sciences (HBNI), Chennai, India

Meena Mahajan
The Institute of Mathematical Sciences (HBNI), Chennai, India

Abstract

We relate various complexity measures like sensitivity, block sensitivity, certificate complexity for multi-output functions to the query complexities of such functions. Using these relations, we improve upon the known relationship between pseudo-deterministic query complexity and deterministic query complexity for total search problems: We show that pseudo-deterministic query complexity is at most the third power of its deterministic query complexity. (Previously a fourth-power relation was shown by Goldreich,Goldwasser,Ron (ITCS13).) We then obtain a significantly simpler and self-contained proof of a separation between pseudodeterminism and randomized query complexity recently proved by Goldwasser,Impagliazzo,Pitassi,Santhanam (CCC 2021). We also separate pseudodeterminism from randomness in And decision trees, and determinism from pseudodeterminism in Parity decision trees. For a hypercube colouring problem closely related to the pseudodeterministic complexity of a complete problem in TFNP^{dt}, we prove that either the monotone block-sensitivity or the anti-monotone block sensitivity is $\Omega(n^{1/3})$; previously an $\Omega(n^{1/2})$ bound was known but for general block-sensitivity.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees

Keywords and phrases Boolean functions, Decision trees, Randomness, Search problems, Pseudodeterminism

1 Introduction

The question of whether randomness adds computational power over determinism, and if so, how much, has been a question of great interest that is still not completely understood. Naturally, the answer depends on the computational model under consideration, but it also depends on the type of problems one hopes to solve. One may wish to compute some function of the input, a special case being decision problems where the function has just two possible values. There are also the search problems, where for some fixed relation $R \subseteq X \times Y$ and an input $x \in X$, one wishes to find a $y \in Y$ that is related to x; i.e. $(x, y) \in R$. If every $x \in X$ has at least one such y, we have a total search problem defined by R, the R-search problem. In the context of (total) search problems, a nuanced usage of randomness led to the beautiful notion of pseudo-determinism; see [11]. A function f solves the R-search problem if for every x, $(x, f(x)) \in R$. A randomized algorithm which computes such an f with high probability is said to be a pseudo-deterministic algorithm solving the R-search problem. Thus a pseudodeterministic algorithm uses randomness to solve a search problem and almost always provides a canonical solution per input.

The original papers introducing and studying pseudodeterminism examined both polynomial-time algorithms and sublinear-time algorithms; in the latter case, the computational resource measure is query complexity. In [13, 12], a maximal separation was established between pseudodeterministic and randomized query algorithms. Namely, for a specific search problem with randomized query complexity $O(1)$, it was shown that no pseudodeterministic algorithm has sublinear query complexity.

ISSN 1433-8092

Very recently, in [14], this separation was revisited. The separating problems in [13, 12] do not lie in the query-complexity analogue of NP (nondeterministic polylog query complexity, or polylog query complexity to deterministically verify a solution, TFNP^{dt}). This is a very natural class of search problems, and in [14], an almost-maximal separation between randomized and pseudo-deterministic search is established for a problem in this class. The problem in question is SEARCHCNF: given an assignment to the variables of a highly unsatisfiable k-CNF formula, to search for a falsified clause; this problem is very easy for randomized search a($O(1)$ queries), and solutions are easily verifiable. Theorem 7 of [14] establishes that for unsatisfiable k-CNF formulas on n variables with sufficiently strong expansion in the clause-variable incidence graph (in particular, for most random k-CNF formulas), the corresponding search problem has pseudodeterministic complexity $\Omega(\sqrt{n})$, even in the quantum query setting; its randomised complexity is $O(1)$. In [14], the size measure of decision trees in the pseudodeterministic setting was also studied. Lifting the query separation using a small gadget, a strong separation between randomized size and pseudodeterministic size was obtained: SearchCNF problem on random k-CNFs lifted with 2-bit XOR has randomized size $O(1)$ but require $\exp(\Omega(\sqrt{n}))$ size in pseudodeterministic setting.

Taking this study further, Theorem 3 of [14] shows that the promise problem PROMISEFIND1, of finding a 1 in an n-bit string with Hamming weight at least $n/2$, is in a sense complete for the class of search problems that are in TFNP^{dt} and have efficient randomized query algorithms. By relating this search problem to a certain combinatorial problem concerning colourings of the hypercube, and by using the lower bound for SEARCHCNF, a lower bound of $\Omega(\sqrt{n})$ on the pseudodeterministic complexity of PROMISEFIND1 is obtained (Theorem 14 and subsequent remark in [14]. The colouring problem on hypercubes states that any proper coloring of the hypercube contains a point with many 1s and with high block sensitivity. In [14], a point with block sensitivity $\Omega(\sqrt{n})$ is proven to exist (Theorem 14), and a point with block sensitivity $\Omega(n)$ is conjectured to exist (Conjecture 16).

Our contributions

Our first contribution is an improved derandomization of pseudodeterministic query algorithms.

For Boolean functions, randomized and deterministic query complexity are known to be polynomially related. Since deterministic query lower bounds are often easy to obtain using some kind of adversary argument, this provides a route to randomized query lower bounds for Boolean functions. For search problems, however, there is no such polynomial relation. Note that separating pseudodeterminism from randomness requires a lower bound against randomized query algorithms that provide canonical solutions. Such algorithms compute multi-output functions (following nomenclature from [14]) as opposed to Boolean functions. Thus what is required is randomized query lower bounds for multi-output functions. For such functions, too, lower bounds for deterministic querying are often relatively easy to obtain. And again, as for Boolean functions, deterministic and randomized query complexity for multi-output functions are known to be polynomially related; in [13, 12] (Theorem 4.1(3)), the authors show that the deterministic query complexity is bounded above by the fourth power (as opposed to cubic power for Boolean functions) of the randomized complexity. They also show that it is bounded above by the cubic power times a factor that depends on the size of the search problem's range. We revisit these relations, and further tighten them to a cubic power relation. Thus for search problems, deterministic query complexity is bounded above by the cubic power of its pseudodeterministic query complexity; Theorem 3.2.

Our next contribution is to give a significantly simpler, self-contained, proof of (a slightly weaker version of) the separation from [14] in the classical setting. For random k-CNF formulas, the randomized complexity of the search problem is easily seen to be $O(1)$; see Corollary 8 in [14]. The deterministic query complexity for the search problem is known to be $\Omega(n)$ and follows from [19, 5]; see also [17]. Using the relation from [13, 12], this immediately implies that pseudodeterministic query complexity is $\Omega(n^{1/4})$. (In fact, since the number of clauses is $\Theta(n)$, it even yields the bound $\Omega((n/\log n)^{1/3})$). Using instead our improved derandomization from Theorem 3.2 gives the lower bound $\Omega(n^{1/3})$. While these bounds are still not as strong as the lower bound of $\Omega(\sqrt{n})$ from [14], they certainly suffice to separate pseudodeterminism from randomness for this problem. We give a direct proof (Section 4 of the deterministic $\Omega(n)$ lower bound. This, along with Theorem 3.2, gives a self-contained proof that the pseudo-deterministic complexity of SearchCNF is $\Omega(n^{1/3})$.

However, the really significant feature of our separation is its simplicity, the way it is established. Even for classical (as opposed to quantum) queries, the lower bound proof in [14] is highly non-trivial. After connecting pseudodeterministic complexity for this problem to a notion in proof complexity, namely the degree of an Nullstellensatz refutation, it uses two "heavy hammers" – (1) known lower bounds on the degree of Nullstellensatz refutations for such formulas [1], and (2) the recently-proved sensitivity theorem [16], showing that sensitivity and degree are quadratically related, and then wraps up the proof with the fact that sensitivity gives lower bounds on randomized query complexity. The use of big tools seems necessitated by the fact that the authors of [14] directly give lower bounds on randomized algorithms for multi-output functions. By using the derandomization, our proof bypasses the use of both these known results, and relies on a lower bound for deterministic algorithms for multi-output functions; Proposition 2.8(2). Even for this lower bound, the already known proof uses other proof complexity results, namely, the connection between decision trees and tree-like resolution proofs [19], and the size of tree-like resolution proofs [2]. We give a direct proof framed entirely within the context of decision trees; this may be of independent interest. As an illustrative example, we first describe in Proposition 4.1 another relation (but not one in TFNP^{dt}) that separates pseudodeterminism from randomness.

Next, using the recent result from [9] that derandomized the size measures for total boolean functions, we establish a polynomial relationship between the log of pseudodeterministic size and the log of deterministic size, ignoring polylog factors in the input dimension. This gives us another way to separate randomized size from pseudodeterministic size: any total search problem which is easy with randomization but difficult for deterministic search will lead to a separation between pseudodeterministic size and randomized size; one such problem is SearchCNF on suitably expanding k-CNF formulas.

We also consider the complexity of search problems in two other, more general, query models. The first model is the And decision tree, where each query is a conjunction of variables. The second is the Parity decision trees, where each query reports the parity of some subset of variables. Both these models obviously generalise decision trees, and are much more powerful in the deterministic setting. We show the following:

1. For And decision trees, pseudo-determinism is still separated from randomness; Theorem 6.3. Furthermore, using the recent result from [9] which derandomized the And decision trees for total Boolean functions, we observe that pseudodeterminism and determinism are polynomially related in this setting, ignoring polylog$n$ factors;Corollary 6.5.
2. For Parity decision trees, determinism is separated from pseudo-determinism; Theorem 6.6. There is no polynomial relation between these two complexity measures.

In this setting, we do not know whether pseudo-determinism is separated from randomness.

Finally, in the same spirit of finding simpler proofs, we revisit the hypercube coloring problem from [14]. There, the existence of a point with large Hamming weight and block-sensitivity $\Omega(\sqrt{n})$ is established, using the previously established lower bound for SearchCNF. We give a completely combinatorial and constructive argument to show that a point with large Hamming weight and block-sensitivity $\Omega(n^{1/3})$ exists, Theorem 7.3 While we seemingly sacrifice stronger bounds in the quest for simplicity, our algorithm actually proves something that is stronger in a different way, and hence our result is perhaps incomparable with that of [14]. The difference is that we identify many sensitive blocks that are all 1s, or many sensitive blocks that are all 0s.

Our techniques.

We examine how the notions of sensitivity, block sensitivity, certificate complexity, originally defined for Boolean functions, extend to multi-output functions and what relationships can be established between them. Ignoring constant multiplicative factors, the same relationships continue to hold; see Theorem 3.1. These relationships are obtained by appropriately modifying the arguments that establish corresponding relationships for boolean functions. These relationships directly yield that that deterministic query complexity is bounded above by the cube of pseudodeterministic query complexity; Theorem 3.2.

To show directly that the search problem for a random k-CNF formula requires large deterministic query complexity (Section 4), we consider the notion of redundancy in and minimality of decision trees. In a decision tree for the Search CNF problem, a node querying a variable is redundant if in at least one of its two subtrees, no leaf is labelled by a clause containing that variable. Amongst all depth-optimal decision trees, the smallest tree is also minimal i.e. devoid of redundant nodes. We crucially use this property to show that the tree must have $\Omega(n)$ depth.

It is worth noting that the randomised lower bound from [14] for random k-CNF formulas uses neighbourhood expansion of the incidence graph. Our proof instead uses boundary expansion (also known as unique neighbour expansion) of the same graph; this makes the proof crisp. It can be seen as a reframing of the width lower bound for such formulas established in [5].

The separations for And and Parity decision trees are obtained through direct combinatorial arguments, using the notion of monotone sensitivity and the random subset sum principle respectively.

Related work.

For Boolean functions, the relations between many complexity measures and query complexity has been studied extensively in the literature. A consolidation of many known results appears in the survey [7] as well as in the classic book [17]. The degree and approximate degree of Boolean functions has also been a very useful measure, but is not directly relevant to this work.

The connection between decision trees and proof complexity is well-known for years; see for instance [19, 5, 4, 6]. However, this work aims to bypass proof complexity in giving lower bounds for query complexity.

Organisation of the paper.

After giving the definitions and listing relevant known results in Section 2, in Section 3 we establish the relationships between various measures for multi-output functions, and establish the polynomial relation between pseudodeterministic and deterministic query complexity for search problems. In Section 4 we give the simpler lower bound for random k-CNF formulas. Section 5 establishes a relationship between pseudodeterministic size and deterministic size. Section 6 discusses the complexity of search problems in And and Parity decision trees. Section 7 discusses the hypercube coloring problem from [14].

2 Preliminaries

Notation

For $x \in \{0,1\}^*$, and $b \in \{0,1\}$, $|x|$ denotes the length of x, and $|x|_b$ denotes the number of occurrences of b in x. We also use the notation $\mathrm{wt}(x)$ for $|x|_1$, since it is the Hamming weight of x. All logarithms in this paper are taken to the base 2. We use notations $\widetilde{O}(\cdot), \widetilde{\Theta}(\cdot), \widetilde{\Omega}(\cdot)$ to hide polylogarithmic factors in the input size (and not just polylogarithmic factors in the argument).

Search Problems

A search problem over domain $\mathcal{X}$ and range $\mathcal{Y}$ is a relation $S \subseteq \mathcal{X} \times \mathcal{Y}$. Given an input $x \in \mathcal{X}$, the task is to find a $y \in \mathcal{Y}$ such that $(x, y) \in S$, if such a y exists. If for every element $x \in \mathcal{X}$ there exist a $y \in \mathcal{Y}$ such that $(x, y) \in \mathcal{S}$, then $\mathcal{S}$ is said to be a total search problem.

A function $f : \mathcal{X} \to \mathcal{Y}$ solves a total search problem $\mathcal{S}$, denoted by $f \in_s \mathcal{S}$, if for every $x \in \mathcal{X}$, $(x, f(x)) \in S$. To emphasize that the range of f is some subset of $\mathcal{Y}$ and f is not necessarily a decision problem, we call such functions multi-output functions (following nomenclature from [14]).

Throughout this paper, we consider without loss of generality that $\mathcal{X} \subseteq \{0,1\}^*$ and $\mathcal{Y} \subseteq \mathbf{N}$. For $n \in \mathbf{N}$, $\mathcal{X}_n$ denotes the set $\mathcal{X} \cap \{0,1\}^n$, and $\mathcal{Y}_n = \{y \in \mathcal{Y} \mid \exists x \in \mathcal{X}_n : (x, y) \in S\}$. Further, S_n denotes the restriction of S to $\mathcal{X}_n$; that is, $S_n = \{(x, y) \in S \mid x \in \mathcal{X}_n\}$. The parameter $\ell_S(n)$ is the number of bits required to represent the range of the projection of S_n to $\mathcal{Y}$; that is, $\ell_S(n) = \log|\mathcal{Y}_n|$. Throughout this paper, we use $\mathcal{Y}_n = \{1, 2, ..., m_n\}$, and we drop the subscript n when clear from context. (Thus we often talk of $\mathcal{X} \subseteq \{0,1\}^n$ and $\mathcal{Y} = [m]$.)

Combinatorial Measures for Multi-output functions

For a multi-output function $f : \mathcal{X} \to \mathcal{Y}$, several complexity measures can be defined by adapting the corresponding definitions for Boolean functions ($\mathcal{X} = \{0,1\}^n$, $\mathcal{Y} = \{0,1\}$).

Certificate Complexity

For an input $a \in \mathcal{X}$, an f-certificate of a is a subset $B \subseteq \{1, ..., n\}$ such that

$$\forall a' \in \mathcal{X}, \left[\left(a'_j = a_j \forall j \in B\right) \implies f(a) = f(a')\right].$$

Such a certificate need not be unique. Let $\mathrm{C}(f, a)$ denote the minimum size of an f-certificate for the input a. Then

$$\text{For } b \in \mathcal{Y}, \quad \mathrm{C}_b(f) = \max\{\mathrm{C}(f, a) \mid a \in f^{-1}(b)\}$$

$$\mathrm{C}(f) = \max\{\mathrm{C}(f, a) \mid a \in \mathcal{X}\} = \max_{b \in \mathcal{Y}} \mathrm{C}_b(f)$$

Sensitivity and Block Sensitivity

For an $x \in \mathcal{X}$, $B \subseteq [n]$, and $b \in \{0, 1\}$, b_B is the n-bit string that is b at positions in B and $1 - b$ elsewhere. A (multi-output) function f is sensitive to block B on input x if $x \oplus 1_B \in \mathcal{X}$ and $f(x) \neq f(x \oplus 1_B)$. The block sensitivity of x with respect to f, $\mathrm{bs}(f, x)$, is the maximum integer r for which there exist r disjoint sensitive blocks of f at x. The block sensitivity of the function is defined as $\mathrm{bs}(f) = \max_{x \in \mathcal{X}} \mathrm{bs}(f, x)$.

By restricting the block sizes to 1, we get the notion of sensitivity. A bit $i \in [n]$ is sensitive for x with respect to f if the block $\{i\}$ is sensitive for x. The sensitivity of x with respect to f, $\mathrm{s}(f, x)$, is the number of sensitive bits for x. The sensitivity of the function is defined as $\mathrm{s}(f) = \max_{x \in \mathcal{X}} \mathrm{s}(f, x)$.

Next, we define variants of sensitivity and block sensitivity where one restricts changing input by only flipping 0's or by only flipping 1's. For $b \in \{0, 1\}$, a set $B \subseteq [n]$ is a *sensitive b-block of f at input x* if $x_i = b$ for each $i \in B$, $x \oplus 1_B \in \mathcal{X}$, and $f(x) \neq f(x \oplus 1_B)$. The *$b$-block sensitivity of f at x*, denoted by $\mathrm{bs}_b(f, x)$, is the maximum integer r for which there exist r disjoint sensitive b-blocks of f at x. The b-block sensitivity of f is $\mathrm{bs}_b(f) = \max_{x \in \mathcal{X}} \mathrm{bs}_b(f, x)$.

For $b \in \{0, 1\}$, the b-sensitivity of f at x, $\mathrm{s}_b(f, x)$, is the number of sensitive b-bits of x. The b-sensitivity of f is $\mathrm{s}_b(f) = \max_{x \in \mathcal{X}} \mathrm{s}_b(f, x)$. We note that $\mathrm{s}_0(f)$ and $\mathrm{bs}_0(f)$ are the same as the monotone sensitivity and monotone block sensitivity used in the work of [18] for studying a variant of standard decision trees, namely AND-decision trees.

For $d \in \mathcal{Y}$, we extend the notation, and denote $\mathrm{s}^d(f) = \max_{x \in f^{-1}(d)} \mathrm{s}(f, x)$ and $\mathrm{bs}^d(f) = \max_{x \in f^{-1}(d)} \mathrm{bs}(f, x)$.

Query Complexity Measures

Decision trees

For a search problem $\mathcal{S}$, a (deterministic) decision tree T computing $\mathcal{S}$ is a binary tree with internal nodes labelled by the variables and the leaves labelled by some $y \in \mathcal{Y}$. To evaluate $\mathcal{S}$ on an unknown input x, the process starts at the root of the decision tree and works down the tree, querying the variables at the internal nodes. If the value of the query is 0, the process continues in the left subtree, otherwise, it proceeds in the right subtree. Let the label of the leaf so reached be $T(x)$. For every $x \in \mathcal{X}$, $T(x)$ must belong to $\mathcal{S}(x)$. Every decision tree T computing $\mathcal{S}$ corresponds to a multioutput function $f : \mathcal{X} \to \mathcal{Y}$ solving $\mathcal{S}$, namely, the function which maps $x \in \mathcal{X}$ to $T(x)$. The depth of a decision tree T, denoted $\mathrm{Depth}(T)$, is the length of the longest root-to-leaf path, and its size $\mathrm{Size}(T)$ is the number of leaves.

Deterministic Query and Size Complexity

The deterministic query complexity of $\mathcal{S}$, denoted by $\mathrm{D}^{\mathrm{dt}}(\mathcal{S})$, is defined to be the minimum depth of a decision tree computing $\mathcal{S}$. Equivalently,

$$\mathrm{D}^{\mathrm{dt}}(\mathcal{S}) = \min_{f \in_s \mathcal{S}} \min_{T \text{ computes } f} \mathrm{Depth}(T)$$

i.e. the minimum number of worst-case queries required to evaluate any f solving $\mathcal{S}$. The deterministic size complexity of a $\mathcal{S}$, denoted by $\mathrm{DSize}^{\mathrm{dt}}(\mathcal{S})$, is defined similarly i.e.

$$\mathrm{DSize}^{\mathrm{dt}}(\mathcal{S}) = \min_{f \in_s \mathcal{S}} \min_{T \text{ computes } f} \mathrm{Size}(T)$$

Randomized and Distributional Query and Size Complexity

A randomized query algorithm/decision tree $\mathcal{A}$ is a distribution $\mathcal{D}_{\mathcal{A}}$ over deterministic decision trees. On input x, $\mathcal{A}$ starts by sampling a deterministic decision tree T according to $\mathcal{D}_{\mathcal{A}}$, and outputs the label of the leaf reached by T on x. Algorithm $\mathcal{A}$ computes $\mathcal{S}$ with error at most ϵ if for every input x, the probability that $A(x)$ belongs to $\mathcal{S}(x)$ is at least $1-\epsilon$. The complexity of the randomized algorithm is measured by the number of worst-case queries made by $\mathcal{A}$ on any input x i.e. maximum depth over all decision trees in the support of the distribution. The randomized query complexity of $\mathcal{S}$ for error ϵ, denoted by $\mathrm{R}^{\mathrm{dt}}_{\epsilon}(\mathcal{S})$, is the minimum number of worst-case queries required to compute $\mathcal{S}$ with error at most ϵ. That is,

$$\mathrm{R}^{\mathrm{dt}}_{\epsilon}(\mathcal{S}) = \min_{\mathcal{A} \text{ computes } \mathcal{S} \text{ with error } \leq \epsilon} \max_{T : \mathcal{D}_{\mathcal{A}}(T) > 0} \mathrm{Depth}(T).$$

When no ϵ is specified, it is assumed to be 1/3. The randomized size complexity of a search problem $\mathcal{S}$, denoted by $\mathrm{RSize}^{\mathrm{dt}}(\mathcal{S})$, is defined similarly i.e.

$$\mathrm{RSize}^{\mathrm{dt}}_{\epsilon}(\mathcal{S}) = \min_{\mathcal{A} \text{ computes } \mathcal{S} \text{ with error } \leq \epsilon} \max_{T : \mathcal{D}_{\mathcal{A}}(T) > 0} \mathrm{Size}(T).$$

For a probability distribution $\mathcal{D}$ over inputs $\mathcal{X}$, the $(\mathcal{D}, \epsilon)$-distributional query and size complexity of $\mathcal{S}$, denoted by $\mathrm{D}^{\mathrm{dt}}_{\mathcal{D},\epsilon}(\mathcal{S})$ and $\mathrm{DSize}^{\mathrm{dt}}_{\mathcal{D},\epsilon}(\mathcal{S})$ respectively, is the minimum depth/size of a deterministic decision tree that gives a correct answer on $1-\epsilon$ fraction of inputs weighted by $\mathcal{D}$. That is, with $x \sim \mathcal{D}$ denoting that x is sampled according to $\mathcal{D}$,

$$\mathrm{D}^{\mathrm{dt}}_{\mathcal{D},\epsilon}(S) = \min \left\{ \mathrm{Depth}(T) \mid T \text{ is a deterministic decision tree; } \Pr_{x \sim \mathcal{D}}[(x, T(x)) \notin \mathcal{S}] \leq \epsilon \right\}.$$

$$\mathrm{DSize}^{\mathrm{dt}}_{\mathcal{D},\epsilon}(S) = \min \left\{ \mathrm{Size}(T) \mid T \text{ is a deterministic decision tree; } \Pr_{x \sim \mathcal{D}}[(x, T(x)) \notin \mathcal{S}] \leq \epsilon \right\}.$$

Distributional query(size) complexity provides a technique to prove randomized query(size) lower bounds. It characterizes the randomized query(size) complexity completely.

▶ **Proposition 2.1** ([20])**.** *$R^{\mathrm{dt}}_{\epsilon}(S) = \max_{\mathcal{D}} D^{\mathrm{dt}}_{\mathcal{D},\epsilon}(S)$ and $\mathrm{RSize}^{\mathrm{dt}}_{\epsilon}(S) = \max_{\mathcal{D}} \mathrm{DSize}^{\mathrm{dt}}_{\mathcal{D},\epsilon}(S)$.*

This is proved in [20] for Boolean functions, but it is easy to see that it also holds for multi-output functions and search relations. For an arbitrary distribution $\mathcal{D}$, $\mathrm{D}^{\mathrm{dt}}_{\mathcal{D},\epsilon} \leq \mathrm{R}^{\mathrm{dt}}_{\epsilon}$ ($\mathrm{DSize}^{\mathrm{dt}}_{\mathcal{D},\epsilon} \leq \mathrm{RSize}^{\mathrm{dt}}_{\epsilon}$), is easily shown using a weighted counting argument. The other direction, $\mathrm{R}^{\mathrm{dt}}_{\epsilon} \leq \max_{\mathcal{D}} \mathrm{D}^{\mathrm{dt}}_{\mathcal{D},\epsilon}$ ($\mathrm{RSize}^{\mathrm{dt}}_{\epsilon} \leq \max_{\mathcal{D}} \mathrm{DSize}^{\mathrm{dt}}_{\mathcal{D},\epsilon}$), was shown using linear programming duality. The easy direction of Proposition 2.1 gives us a way to prove randomized query lower bounds by proving a $(\mathcal{D}, \epsilon)$-distributional query complexity lower bound for some hard distribution $\mathcal{D}$. We note that this technique also works for other models of decision tree like And and Parity decision trees.

Pseudodeterministic Query and Size Complexity

A pseudodeterministic query algorithm/decision tree for a search problem $\mathcal{S}$, with error 1/3, is a randomized decision tree $\mathcal{A}$ computing $\mathcal{S}$ with the property that for every input x, there is a canonical value $y \in \mathcal{Y}$ such that with probability at least 2/3, $\mathcal{A}(x) = y$. Equivalently, a pseudodeterministic query algorithm is a randomized query algorithm that computes some multi-output function $f \in_s \mathcal{S}$ with error at most 1/3. The pseudodeterministic query complexity of $\mathcal{S}$, denoted by $\mathrm{psD}^{\mathrm{dt}}(\mathcal{S})$, is equal to $\min_{f \in_s \mathcal{S}} \mathrm{R}^{\mathrm{dt}}(f)$ and pseudodeterministic size complexity of $\mathcal{S}$, denoted by $\mathrm{psDSize}^{\mathrm{dt}}(\mathcal{S})$, is equal to $\min_{f \in_s \mathcal{S}} \mathrm{RSize}^{\mathrm{dt}}(f)$. Note the difference between pseudodeterministic and randomized query algorithms: randomized query algorithms on input x are not required to output a canonical value with high probability; they just need to output a value in $\mathcal{S}(x)$ with high probability.

The query-complexity analog of TFNP

TFNP is the class of total functions which can be solved in nondeterministic polynomial time, or for which the solution/value can be verified in deterministic polynomial time. Since every function is trivially computable with query complexity n, the analog of polynomial-time/efficient/tractable for query complexity is poly-logarithmic queries. The class TFNP^{dt} thus denotes total search problems for which solutions can be verified with polylogarithmic queries.

Known results

▶ **Proposition 2.2** ([17][7]). *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$,*

1. $s(f) \leq bs(f) \leq C(f) \leq s(f)bs(f)$.
2. $s(f) \leq bs(f) \leq 3R^{\mathrm{dt}}_{1/3}$.
3. $C(f) \leq D^{\mathrm{dt}}(f) \leq C(f)^2$.
4. $D^{\mathrm{dt}}(f) \leq C(f)bs(f)$.
5. $D^{\mathrm{dt}}(f) \in O((R^{\mathrm{dt}}(f))^3)$.

▶ **Proposition 2.3** (restated from [12]). *For a search relation S,*

1. $D^{\mathrm{dt}}(\mathcal{S}) \leq \left(psD^{\mathrm{dt}}(\mathcal{S})\right)^4$. *[Restated from Theorem 4.1(3) in [12]]*
2. $D^{\mathrm{dt}}(\mathcal{S}) \leq \left(psD^{\mathrm{dt}}(\mathcal{S})\right)^3 \ell_{\mathcal{S}}(n)$. *[Restated from Theorem 4.1(3) in [12]]*

▶ **Proposition 2.4.**
1. *[Corollary 4.2 in [12]] For the relation* $\textsc{ApproxHamWt} = \{(x, v) : |wt(x) - v| \leq n/10\}$, $psD^{\mathrm{dt}}(\textsc{ApproxHamWt}) \in \Omega(n)$ *and* $R^{\mathrm{dt}}(\textsc{ApproxHamWt}) \in O(1)$.
2. *[Theorem 4 in [14]] For the relation* $\textsc{PromiseFIND1} = \{(x, i) : wt(x) \geq |x|/2 \wedge x_i = 1\}$, $psD^{\mathrm{dt}}(\textsc{PromiseFIND1}) \in \Omega(\sqrt{n})$ *and* $R^{\mathrm{dt}}(\textsc{PromiseFIND1}) \in O(1)$.

Unsatisfiable k-CNF formulas

We consider random k-CNF formulas over n variables and $m = cn$ clauses. Let $\mathcal{F}^{k,n}_m$ be the distribution over random k-CNF formulas with m clauses, where each clause is sampled uniformly randomly with repetition from the set of all $2^k\binom{n}{k}$ clauses. To study these formulas, we need to study the underlying properties of the clause-variable incidence graph of these formulas.

▶ **Definition 2.5.** *Let $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a random k-CNF formula on n variables with m clauses. Consider the bipartite graph, $G_F = (V = [m], U = [n], E)$ with m left vertices, one for each clause C_i, and n right vertices, one for each variable, such that $(i, j) \in E$ if and only if clause C_i contains one of the literals $x_j, \neg x_j$. For any $V' \subseteq V$, the neighborhood of V' is the set $N(V') = \{u \in U \mid (v, u) \in E, v \in V'\}$, and the boundary of V' is the set $\partial V' = \{u \in U \mid |N(u) \cap V'| = 1\}$. A k-CNF formula F is said to be*

1. *(Matchability) r-matchable if in G_F, $\forall V' \subseteq V$ with $|V'| \leq r$, $|N(V')| \geq |V'|$.*
2. *(Neighborhood Expansion) an (r, ϵ)-expander if in G_F, $\forall V' \subset V$ with $r/2 \leq |V'| \leq r$, $|N(V')| \geq \epsilon|V'|$.*
3. *(Boundary Expansion) an (r, ϵ)-boundary expander if in G_F, $\forall V' \subset V$ with $r/2 \leq |V'| \leq r$, $|\partial V'| \geq \epsilon|V'|$.*

There are several notions of expansion in literature; they are similar but not exactly equivalent. We use boundary-expansion in our work. Boundary expansion is a stronger notion than neighborhood expansion, but neighborhood expansion does imply boundary expansion with some weakening in the expansion parameter. In particular, the following proposition can be easily verified.

▶ **Proposition 2.6.** *If a k-CNF formula, F, is an (r, ϵ)-expander, then it is an $(r, 2\epsilon - k)$-boundary expander.*

▶ **Proposition 2.7** ([10][2] [3]). *For a constant c large enough and $0 < \epsilon < 1/2$, there exist constants $\kappa_1, \kappa_2 \leq 1$, function of ϵ and c, such that following holds. For F a random 3-CNF formula on n variables with $m = cn$ clauses sampled from $\mathcal{F}_m^{3,n}$, with high probability, $1 - o(1)$,*

- *(F is highly unsatisfiable): Every assignment falsifies at least half of the clauses of F.*
- *(F is highly matchable): F is n-matchable.*
- *(F has expansion properties): F is $(\kappa_1 n, 1 + \epsilon)$-expander.*
- *(F has boundary expansion properties): F is $(\kappa_2 n, \epsilon)$-boundary expander.*

For an unsatisfiable CNF formula $F = \wedge_{i \in [m]} C_i$ on n variables, the SearchCNF relation is defined as $\textsc{SearchCNF}(F) = \{(a, i) \mid a \in \{0, 1\}^n, a$ falsifies clause $C_i\}$. It is known that for suitably expanding unsatisfiable formulas, the $\textsc{SearchCNF}$ relation has high deterministic and pseudo-deterministic query complexity.

▶ **Proposition 2.8.** *For F a random 3-CNF formula on n variables with $m = cn$ clauses sampled from $\mathcal{F}_m^{3,n}$, with probability $1 - o(1)$, F is unsatisfiable and furthermore,*

1. $R^{\mathrm{dt}}(\textsc{SearchCNF}(F)) = O(1)$. *(From Proposition 2.7.)*
2. $D^{\mathrm{dt}}(\textsc{SearchCNF}(F)) = \Omega(n)$. *(From [19, 5])*
3. $psD^{\mathrm{dt}}(\textsc{SearchCNF}(F)) = \Omega(\sqrt{n})$. *(Corollary 8 in [14])*
4. $\mathrm{DSize}^{\mathrm{dt}}(\textsc{SearchCNF}(F)) = \exp(\Omega(n))$. *(From [5])*
5. $\mathrm{psDSize}^{\mathrm{dt}}(\textsc{SearchCNF}(F)) = \exp(\Omega(\sqrt{n}))$. *(Theorem 22 in [14])*

3 Relating Measures for Multivalued functions

We show the analogs of Proposition 2.2(1-4) for multi-output functions.

▶ **Theorem 3.1.** *For a function $f : \{0, 1\}^n \to [m]$, the following relations hold.*

1. $C(f) \leq s(f)bs(f)$.
2. $s(f) \leq bs(f) \leq 3R^{\mathrm{dt}}_{1/3}(f)$
3. $C(f) \leq D^{\mathrm{dt}}(f) \leq C(f)^2$.
4. $D^{\mathrm{dt}}(f) \leq 2C(f)bs(f)$.

Proof. The proof idea is to do the necessary modifications to the analogous results in the Boolean function case. The first two items are completely straightforward, but are nonetheless included here for completeness.

1. ($\mathrm{C}(f) \leq \mathrm{s}(f)\mathrm{bs}(f)$): The construction in the boolean function case works for multioutput functions as well. For completeness, we repeat the argument explicitly.
   For an arbitrary input $a \in \{0,1\}^n$, let $f(a) = i$. We show that $\mathrm{C}(f,a) \leq \mathrm{bs}(f,a)\mathrm{s}^i(f)$. Let $B_1, ..., B_k$ be disjoint minimal sets of blocks of variables that achieve $k = \mathrm{bs}(f,a)$. Then we claim that the set $B = B_1 \cup B_2 \cup ... \cup B_k$ is an f-certificate of a. Suppose not. Then there exists $b \in \{0,1\}^n$ which coincides with a on B, but $f(b) \neq f(a)$. Let B_{k+1} be the set of positions where b differs from a. Since b coincides with a on B, B_{k+1} is disjoint from b and is a sensitive block for a, contradicting $\mathrm{bs}(f,a) = k$.
   Hence $\mathrm{C}(f,a) \leq |B|$. Now, we just need to analyze the size of the certificate B. Note that $|B| \leq \mathrm{bs}(f,a) \max_{j\in[k]} |B_j|$. We bound $\max_{j\in[k]} |B_j|$ by showing that any minimal block to which a is sensitive w.r.t. to f cannot have more than $\mathrm{s}^i(f)$ variables. Let B_j be a minimal sensitive block for a and $a^{B_j} = a \oplus 1_{B_j}$. Now, observe that if we flip any variable in B_j, the function value flips from $f(a^{B_j})$ to $f(a) = i$. So, $|B_j| \leq \mathrm{s}^i(f, a^{B_j}) \leq \mathrm{s}^i(f)$. Since this holds for arbitrary minimal sensitive block B_j for a, we have $\max_{j\in[k]} |B_j| \leq \mathrm{s}^i(f)$. Thus $\mathrm{C}(f,a) \leq |B| \leq \mathrm{bs}(f,a)\mathrm{s}^i(f) \leq \mathrm{bs}(f)\mathrm{s}(f)$.
2. ($\mathrm{s}(f) \leq \mathrm{bs}(f) \leq 3\mathrm{R}^{\mathrm{dt}}_{1/3}(f)$): The first inequality follows form the definitions. The second inequality can be proven for the Boolean case in many ways. The proof via distributional query complexity works in the multi-output function setting as well, as follows.
   Let a be an input achieving the block sensitivity $k = \mathrm{bs}(f)$, and $B_1, B_2, ..., B_k$ be disjoint sensitive blocks for a. We demonstrate a hard distribution $\mathcal{D}$ such that $\mathrm{D}^{\mathrm{dt}}_{\mathcal{D},1/3}(f) \geq k/3$, thereby showing $\mathrm{R}^{\mathrm{dt}}_{1/3}(f) \geq k/3$. The hard distribution is as follows

$$\mathcal{D}(x) = \begin{cases} 1/2 & \text{if } x = a \\ 1/(2k) & \text{if } x = a \oplus 1_{B_i} \quad \text{for } i \in [k] \\ 0 & \text{Otherwise} \end{cases}$$

   Let T be any deterministic decision tree that gives correct answer for f on $2/3$ fraction of inputs weighted by $\mathcal{D}$. We argue that depth of T must be atleast $k/3$. Consider the path P traversed on a by T and let j be the label of the leaf l so reached. We argue that path P must query at least $k/3$ variables. Suppose not. Then there exist at least $s = (2k/3)+1$ blocks B_i's such that none of the variables from these block are queried by the path P. Without loss of generality, let these blocks be $B_1, B_2, ..., B_s$. So for all inputs in the set $A = \{a, a\oplus 1_{B_1}, a\oplus 1_{B_2}, ..., a\oplus 1_{B_s}\}$, the path P is traversed and the answer j is returned by T. Now, if $f(a) = j$, then T errors on the inputs $\{a\oplus 1_{B_1}, a\oplus 1_{B_2}, ..., a\oplus 1_{B_s}\}$, which together have probability mass more than 1/3.On the other hand, if $f(a) \neq j$, then T errs on a which has probability mass of $1/2$. Either way, this contradicts the assumption that T answers correctly on $2/3$ probability mass according to $\mathcal{D}$.
   Since the argument works for arbitrary T that is a $(\mathcal{D}, 1/3)$-distributional query algorithm for f, we have $k/3 \leq \mathrm{D}^{\mathrm{dt}}_{\mathcal{D},\epsilon}(f) \leq \mathrm{R}^{\mathrm{dt}}_{1/3}(f)$.
3. ($\mathrm{C}(f) \leq \mathrm{D}^{\mathrm{dt}}(f) \leq \mathrm{C}(f)^2$): The first inequality is easy to see. Given a decision tree T for f, on an input x, the variables queried by T on x form a valid certificate and so $\mathrm{C}(f) \leq \mathrm{D}^{\mathrm{dt}}(f)$.
   The construction for the upper bound is exactly same as the one in the boolean case, but the analysis has to be done more carefully for multi-output functions. For a multi-output function $f : \mathcal{X} \to [m]$, let $\vec{C} = (\mathrm{C}_1(f), \mathrm{C}_2(f), ..., \mathrm{C}_m(f))$. Let $\rho_1(f)$ and $\rho_2(f)$

denotes the largest and the second largest number in the tuple $\vec{C}$ respectively. We claim $\mathrm{D}^{\mathrm{dt}}(f) \leq \rho_1(f)\rho_2(f)$. Note that this proves our proposition since $\rho_1(f)\rho_2(f) \leq \mathrm{C}(f)^2$. We prove the claim by induction on $\rho_2(f)$. For the base case, when $\rho_2(f) = 0$, f is constant and so $\mathrm{D}^{\mathrm{dt}}(f) \leq \rho_1(f)\rho_2(f) = 0$. For the induction step, $\rho_2(f) > 0$, let $i \in [m]$ be the index such that $\mathrm{C}_i(f) = \rho_1(f)$. Pick an input a such that $f(a) = i$ (such an input exists because $\mathrm{C}_i(f) > 0$). Let S be the certificate for a and B be the set of variables in it. Without loss of generality, let $B = \{x_1, x_2, ..., x_k\}$. Take a complete binary tree T_0 querying all the variables in B. On one of the leaves of T_0, where variables in B match the bits of a, we know that the value of f is i. Each of the other leaves correspond to a unique setting ν of $x_1, ..., x_k$. Replace each leaf by the minimal depth decision tree for f restricted with ν, denoted by f_ν.

First, we claim that $\rho_2(f_\nu) \leq \rho_2(f) - 1$. This comes from the simple observation that for $h, l \in [m]$ with $h \neq l$, every h-certificate must intersect with every l-certificate of f. Since we queried an i-certificate of f, for all $j \neq i$, $\mathrm{C}_j(f_\nu) \leq \mathrm{C}_j(f) - 1$. Hence $\rho_2(f_\nu) \leq \rho_2(f) - 1$. Now applying the induction hypothesis for f_ν, $\mathrm{D}^{\mathrm{dt}}(f_\nu) \leq \rho_1(f_\nu)\rho_2(f_\nu) \leq \rho_1(f)(\rho_2(f) - 1)$. Putting things together, $\mathrm{D}^{\mathrm{dt}}(f) \leq \rho_1(f) + \rho_1(f)(\rho_2(f) - 1) \leq \rho_1(f)\rho_2(f)$.

4. ($\mathrm{D}^{\mathrm{dt}}(f) \leq 2\mathrm{C}(f)\mathrm{bs}(f)$): This part is different from the boolean function case. We give an algorithm to compute f, querying at most $2\mathrm{C}(f)\mathrm{bs}(f)$ variables. The algorithm is as follows

a. Repeat the following at most $2\mathrm{bs}(f)$ times: Pick an input with a certificate C that is consistent with the queries so far but still has unqueried variables. Query the unqueried variables of C.

If no such input exists, then the function under the restriction of queried variables has become constant. Return the appropriate constant and stop. Otherwise continue to the next step.

b. Pick any input y consistent with the variables queried so far, and return $f(y)$.

First note that the algorithm queries atmost $2\mathrm{bs}(f)\mathrm{C}(f)$ variables in the worst case. We must show the correctness of the algorithm.

If the algorithm stops in stage a, then we know that for all inputs, every certificate is either fully queried or inconsistent with the queries. Since certificates cannot be inconsistent for all inputs, we have an input x whose certificate is consistent and empty. This means that all the variables in the certificate have already been queried and checked, and so the function must evaluate to $f(x)$.

Now consider the case when the algorithm does not halt in stage a. We show that if the algorithm reaches stage b, then then all the consistent inputs y must have the same $f(y)$ value. Suppose, to the contrary, there exist y and z consistent with all variables queried in stage a, and with $f(y) \neq f(z)$. Let $t = 2\mathrm{bs}(f)$, $f(y) = l_y$, $f(z) = l_z$ and ρ be the partial assignment of variables queried so far. Let $C_1, C_2, ..., C_t$ be the certificates chosen in step a, and for $1 \leq i \leq t$, let B_i be the set of variables on which ρ disagrees with C_i. Even though $\rho \oplus 1_{B_i}$ is a partial assignment, it is consistent with the certificate C_i, and hence f becomes constant under partial assignment $\rho \oplus 1_{B_i}$. Thus $f(\rho \oplus 1_{B_i})$ is well-defined. Consider the following sets:

$$M_y = \{i \in [t] \mid f(\rho \oplus 1_{B_i}) \neq l_y\}.$$

$$M_z = \{i \in [t] \mid f(\rho \oplus 1_{B_i}) \neq l_z\}.$$

Then $M_y \cup M_z = [t]$, so $t \leq |M_y| + |M_z|$. Without loss of generality, let $|M_y| \geq |M_z|$; then $|M_y| \geq t/2 = \mathrm{bs}(f)$.

Let B be the set of positions where y and z differ.

By construction, each B_i can only have variables that are in C_i, but not queried in $\cup_{j<i} C_j$. Hence the blocks B_i for $i \in M_y$ are disjoint.
Also, B is disjoint from each B_i, since y and z are consistent with ρ.
Each block B_i for $i \in M_y$, and block B, are all sensitive blocks for y.
But this means that f is sensitive to $|M_y| + 1 \geq \mathrm{bs}(f) + 1$ disjoint blocks, a contradiction.
Thus, if the algorithm reaches stage b, all the inputs which are consistent with the queried variables must have the same function value. Hence the algorithm's output in stage (b) is correct.

◀

Using the above, we now show the analogs of Proposition 2.2(5) and Proposition 2.3 for multi-output functions and search problems.

▶ **Theorem 3.2.** *The following relations hold.*
1. *For a multi-output function f, $D^{\mathrm{dt}}(f) \in O((R^{\mathrm{dt}}(f))^3)$.*
2. *For a total search problem $\mathcal{S}$, $D^{\mathrm{dt}}(\mathcal{S}) \in O((psD^{\mathrm{dt}}(\mathcal{S}))^3)$.*

Proof. For a multi-output function f, using Theorem 3.1, we have

$$\mathrm{D}^{\mathrm{dt}}(f) \leq 2\mathrm{C}(f)\mathrm{bs}(f) \leq 2\mathrm{s}(f)\mathrm{bs}(f)^2 \leq 2\mathrm{bs}(f)^3 \leq 2\left(3\mathrm{R}^{\mathrm{dt}}_{1/3}(f)\right)^3.$$

For total search problem $\mathcal{S}$, let $\tilde{f}$ be a function solving $\mathcal{S}$, with $\mathrm{psD}^{\mathrm{dt}}(\mathcal{S}) = \mathrm{R}^{\mathrm{dt}}(\tilde{f})$. Then

$$\mathrm{D}^{\mathrm{dt}}(\mathcal{S}) = \min_{f \in_s \mathcal{S}} \mathrm{D}^{\mathrm{dt}}(f) \leq \mathrm{D}^{\mathrm{dt}}(\tilde{f}) \leq O((\mathrm{R}^{\mathrm{dt}}_{1/3}(\tilde{f})^3) = O(\mathrm{psD}^{\mathrm{dt}}(\mathcal{S})^3).$$

◀

4 Simpler separations between $\mathrm{psD}^{\mathrm{dt}}$ and R^{dt}

Using Theorem 3.2, we now provide simpler proofs of separations between randomized and pseudo-deterministic query complexity.

In [12], the search problem ApproxHamWt was shown to demonstrate the limitations of pseudo-determinism over randomized querying. In a similar vein, the search problem BalancedFind1 defined below shows a similar separation, and (arguably) the lower bound is simpler to prove.

▶ **Proposition 4.1.** *Let $\mathcal{S}$ be the search problem*

$$\textsc{BalancedFind1} = \{(x, i) : (|x|_1 = |x|_0 \wedge x_i = 1) \ \text{ or } \ (|x|_1 \neq |x|_0)\}.$$

Then $R^{\mathrm{dt}}(\mathcal{S}) \in O(1)$, $D^{\mathrm{dt}}(\mathcal{S}) = n$, and $psD^{\mathrm{dt}}(\mathcal{S}) \in \Omega(n^{1/3})$.

Proof. First we show that $\mathrm{R}^{\mathrm{dt}}_{1/4}(\mathcal{S})$ is 2. For odd n, simply output 1 without querying anything. For even $n = 2m$, the randomized query algorithm is as follows: Randomly choose two distinct indices $i, j \in [n]$, and query them. If $x_i \vee x_j = 1$, output any index $k \in \{i, j\}$ with $x_k = 1$. Otherwise output 1. It is clear that for inputs x where $|x|_1 \neq |x|_0$, the algorithm is always correct. If $|x|_1 = |x|_0$, an error occurs only if both i, j are among the exactly m indices where x has a 0; this happens with probability $\binom{m}{2}/\binom{2m}{2} = \frac{1}{2}\cdot\frac{m-1}{2m-1} \leq 1/4$.

Next we show that $\mathrm{D}^{\mathrm{dt}}(\mathcal{S}) = \lfloor n/2 \rfloor = m$. It suffices to consider even n, since for odd n no queries are required. To see $\mathrm{D}^{\mathrm{dt}}(\mathcal{S}) \leq m$, consider the decision tree $T_{\mathcal{S}}$ which queries the first m variables, outputs the first index j for which $x_j = 1$, and if no such index exists it outputs

$m+1$. It is easy to verify that $T_{\mathcal{S}}$ solves $\mathcal{S}$. For the lower bound $\mathrm{D}^{\mathrm{dt}}(S) \geq m$, let T be any decision tree solving $\mathcal{S}$ on instances of length $n = 2m$. Consider the left-most path P in the tree, i.e. the path where all the queried variables are reported to be 0, and let it terminate at the leaf ℓ labelled i. We argue that this path must be of length at least m. Suppose not. Without loss of generality, let the variables queried on the path be $x_1, x_2, \ldots., x_k$ for some $k < m$. The set F of $m+1$ inputs defined as $F = \{0^{m-1}1^j01^{m-j} | 0 \leq j \leq m\}$ acts as a fooling set for T: since $k < m$, all the inputs in F are consistent with the variables queried on P, so for each $x \in F$, T reaches ℓ and outputs i; however, for each $i \in [2m]$, there exist $x \in F$ such that $x_i = 0$ and so $(x, i) \notin \mathcal{S}$. Hence T does not solves $\mathcal{S}$, a contradiction. Hence any decision tree T which solves $\mathcal{S}$ must have left-most path of length at least m and thus $\mathrm{D}^{\mathrm{dt}}(\mathcal{S}) \geq m = \lfloor n/2 \rfloor$.

From Theorem 3.2 and the fact that $\mathrm{D}^{\mathrm{dt}}(\mathcal{S}) = \Omega(n)$ as shown above, it follows that $\mathrm{psD}^{\mathrm{dt}}(\mathcal{S}) = \Omega(n^{1/3})$. ◀

Neither ApproxHamWt nor BalancedFind1 are in TFNP^{dt}. However, the search problem SearchCNF for random k-CNFs is in TFNP^{dt}, and as shown in [14], also separates pseudodeterminism from randomness. For the SearchCNF problem on suitably expanding kCNF formulas, the randomised query complexity is $O(1)$, while it is shown in [14] that the pseudo-deterministic query complexity is $\Omega(\sqrt{n})$. Note that already from the results of [19, 5], the deterministic complexity of SearchCNF for these formulas is $\Omega(n)$ (see Proposition 2.8). Hence from the results of [12] (see Proposition 2.3), it follows that pseudo-deterministic query complexity is $\Omega(n^{1/4})$ and even $\Omega((n/\log n)^{1/3})$ since $\ell_S(n) = O(n)$, giving the separation. The proof in [14] improves the lower bound to $\Omega(n^{1/2})$. At a very high level, the stages involved in their proof are as follows: ignoring constant multiplicative factors,

$$\begin{aligned}
\mathrm{psD}^{\mathrm{dt}}(\textsc{SearchCNF}) &= \mathrm{R}^{\mathrm{dt}}(f) && \text{choose } f \text{ computing canonical solutions optimally}\\
&\geq \max_i \mathrm{R}^{\mathrm{dt}}(f^i) && f^i\text{: Boolean indicator function for each } i \text{ in range}\\
&\geq \max_i \{s(f^i)\} && \text{known relation}\\
&\geq \max_i \{\sqrt{\deg(f^i)}\} && \text{by sensitivity theorem [16]}\\
&\geq \sqrt{\deg_{NS}(CNF)} && \text{construct Nullstellensatz refutation using } f^i\text{'s}\\
&\geq \sqrt{n} && \text{by NS-degree lower bound [1, 8, 15]}
\end{aligned}$$

The stage involving the Sensitivity theorem makes the connection between sensitivity and degree, and the stage involving Nullstellensatz degree lower bound uses expansion of random formulas.

Observe that by using Proposition 2.8(2) in conjunction with Theorem 3.2, we can already obtain a lower bound of $\Omega(n^{1/3})$ on $\mathrm{psD}^{\mathrm{dt}}$, marginally improving on the lower bound obtainable by using Proposition 2.8(2) in conjunction with Proposition 2.3. Of course, this is still not as strong as the lower bound from Proposition 2.8(3), but the proof is significantly simpler.

Below we present a direct proof of the deterministic lower bound from Proposition 2.8(2), using only Proposition 2.7. Though it does not show anything new, it is interesting because it directly operates on decision trees, and the tree manipulation techniques used may be useful in other contexts as well. This proof, along with the proof of Theorem 3.2, gives a complete self-contained proof of the fact that for SearchCNF, $\mathrm{psD}^{\mathrm{dt}} = \Omega(n^{1/3})$.

Proof. (Self-contained proof of the deterministic lower bound in Proposition 2.8(2).) Let F be a 3-CNF formula on n variables with $m = cn$ clauses such that F is highly unsatisfiable (i.e. each assignment falsifies at least half of the clauses), F is n-matchable, and F is a $(\kappa n, \epsilon)$-boundary expander for some $\epsilon > 0$. As noted in Proposition 2.7, for large enough c, a random formula chosen from $\mathcal{F}_m^{3,n}$ satisfies these properties with high probability.

Let T be any decision tree solving $\mathcal{S}$. Then T has the following properties

1. The leaves of T are labelled by the clauses of F. The subformula F', comprising of only the clauses appearing at leaves of T, must form an unsatisfiable system since on every assignment T leads to a falsified clause. Since F is n-matchable, Hall's theorem implies that any subset of at most n clauses of F can be matched to variables and thus can be satisfied by setting the variables appropriately. Hence F' must have at least $n+1$ clauses.
2. The partial assignment leading up to a leaf must falsify the clause labelled on the leaf. For example, if the leaf is labelled by the clause $x_1 \vee \neg x_2 \vee x_4$ then the partial assignment formed by querying the variables leading up to the leaf must have $x_1 = x_4 = 0$, $x_2 = 1$.

We show that any T solving $\mathcal{S}$ must have a node in T whose depth is at least $\epsilon\kappa n/2$. We do this by performing modifications on T, deleting some of the unnecessary query nodes of T, and reasoning about the modified tree. The modified decision tree is constructed as follows.

For each non-leaf node v in T, let x_v be the variable queried on v and let F_v^L and F_v^R be the set of clauses appearing at the leaves of the left and the right subtree of v respectively. We note below that the node v is **redundant** unless x_v appears in some clause of F_v^L as well as in some clause of F_v^R.

While T has redundant nodes, pick any such node v. Replace v by its left subtree if x_v does not appear in any clause in F_v^L, and by its right subtree if x_v does not appear in any clause in F_v^R.

Let T' be the tree obtained when no more deletion of nodes is possible; there are no redundant nodes. We observe the following properties about T'.

1. T' solves $\mathcal{S}$.
2. $\text{Depth}(T') \leq \text{Depth}(T)$.
3. For each node v in T', let F_v denote the set of clauses appearing at the leaves of subtree rooted at v. Let ∂F_v be the set of boundary variables, or unique-neighbour variables, associated with F_v. Then all the variables in ∂F_v must have been queried before node v. To see why this is so, let x be some variable in ∂F_v, and assume to the contrary that x is not queried by T on the path leading to v. By choice of x, there is a unique clause $C_x \in F_v$ containing either x or $\neg x$; without loss of generality assume it contains x. In particular, no clause $C \in F_v$ contains the literal $\neg x$. Let ℓ be a leaf in the subtree of v, labelled C_x. Since C_x is falsified by the partial assignment ρ that leads to ℓ, x must be set by ρ. Since it is not set upto v, there must be a node w on the path from v to ℓ that queries x. Since no clause in F_v has $\neg x$, the node w is redundant, a contradiction.

With the observations above, the only thing left to do is to find a node which has lots of boundary variables associated with it.

For the root node r, $|F_r| = |F'| =\geq n+1$ because of n-matchability. For a leaf node ℓ, $|F_\ell| = 1$. At each node v, $F_v = F_v^L \cup F_v^R$. Hence, there exists a node v with $\kappa n/2 \leq |F_v| \leq \kappa n$. (Start from the root node, and repeatedly move to the subtree with more clauses in its subtree until such a node is found.)

Since F is a $(\kappa n, \epsilon)$-boundary-expander, ∂F_v has size at least $\epsilon\kappa n/2$.

By observation 3 above, the path in T' leading to v queries all variables in ∂F_v. Along with observation 2, we put things together:

$$\text{Depth}(T) \geq \text{Depth}(T') \geq \text{Depth}_{T'}(v) \geq |\partial F_v| \geq \frac{\epsilon\kappa n}{2}.$$

Since this holds for an arbitrary decision tree T solving $\mathcal{S}$, hence $\text{D}^{\text{dt}}(\mathcal{S}) \geq \Omega(n)$. ◀

5 Pseudodeterministic Size vs Deterministic Size

In this section, we show a polynomial relationship, ignoring polylog n factors, between the log of pseudodeterministic size and the log of deterministic size for total search problems. But before we do that we look at an argument to extend results on Boolean functions to multi-output functions. We observe that a relationship between randomized and deterministic complexity in a query model for Boolean functions leads to an almost similar relationship between pseudodeterministic complexity and deterministic complexity for search problems. The result follows from a straightforward application of a binary search argument and also appears in the work of [13] for making a similar claim for the ordinary query model.

▷ Claim 5.1. In a query model M, let D^{M}, R^{M}, and $\mathrm{psD}^{\mathrm{M}}$ denote deterministic, randomized and pseudodeterministic query complexities, respectively. And let $\mathrm{DSize}^{\mathrm{M}}$, $\mathrm{RSize}^{\mathrm{M}}$ and $\mathrm{psDSize}^{\mathrm{M}}$ denote deterministic, randomized and pseudodeterministic size complexities, respectively. Then,

1. If for all Boolean functions $f : \{0,1\}^n \to \{0,1\}$, $\mathrm{D}^{\mathrm{M}}(f) \leq q(\mathrm{R}^{\mathrm{M}}(f), n)$ for a function $q : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$, then for any search problem $\mathcal{S} \subseteq \{0,1\}^n \times [m]$, $\mathrm{D}^{\mathrm{M}}(\mathcal{S}) = O(q(\mathrm{psD}^{\mathrm{M}}(\mathcal{S}), n)\cdot \min(\log m, \mathrm{psD}^{\mathrm{M}}(\mathcal{S})))$.
2. If for all Boolean functions $f : \{0,1\}^n \to \{0,1\}$, $\log \mathrm{DSize}^{\mathrm{dt}}(f) \leq q(\log \mathrm{RSize}^{\mathrm{dt}}(f), n)$ for a function $q : \mathbb{R}\times\mathbb{N} \to \mathbb{N}$, then for any search problem $\mathcal{S} \subseteq \{0,1\}^n \times [m]$, $\log \mathrm{DSize}^{\mathrm{M}}(\mathcal{S}) = O(q(\log \mathrm{psDSize}^{\mathrm{M}}(\mathcal{S}), n) \cdot \min(\log m, \mathrm{psDSize}^{\mathrm{M}}(\mathcal{S})))$.

Proof. We prove (2) above; the proof of (1) follows along similar lines. First, we extend the relationship between randomized and deterministic complexities for Boolean functions to multi-output functions. Let $f : \{0,1\}^n \to [m]$ be a multi-output function. Without loss of generality, we assume that f is an onto function, i.e. for each $i \in [m]$, there exists a $x \in \{0,1\}^n$ such that $f(x) = i$. Let T be an optimal size randomized decision tree for f with size complexity $s = \log \mathrm{RSize}^{\mathrm{M}}(f)$. Consider the $\log m$ Boolean functions $f_0, f_1, ..., f_{\log m-1}$ such that for $x \in \{0,1\}^n$, $f_i(x) = 1$ if and only if the i-th bit in the binary representation of $f(x)$ is 1. For each function f_i, $\log \mathrm{RSize}^{\mathrm{M}}(f_i) \leq s$. Indeed, take T and replace the labels of the leaves to 0 if the i-th bit in the binary representation of the label is 0 and to 1 otherwise. By the given randomized and deterministic relationship, there exist deterministic decision trees $T_0, T_1, ..., T_{\log m-1}$ computing f_i's each with size at most $2^{q(s,n)}$. Composing T_i's, we obtain a deterministic decision tree for f of size at most $2^{q(s,n)\log m}$. So, for any multi-output function $f : \{0,1\}^n \to [m]$,

$$\log \mathrm{DSize}^{\mathrm{M}}(f) \leq q(\log \mathrm{RSize}^{\mathrm{M}}(f), n) \cdot \log m.$$

Next, we upper bound m by randomized size complexity $\mathrm{RSize}^{\mathrm{dt}}(f)$. We claim $m <= 2 \cdot \mathrm{RSize}^{\mathrm{M}}(f)$. Suppose not, $m > 2 \cdot \mathrm{RSize}^{\mathrm{M}}(f)$. For random coins r, the number of possible labels output by T is clearly upper bounded by $\mathrm{RSize}^{\mathrm{M}}(f)$. So, $\mathrm{E}_r[|\{\ell : \exists x \text{ s.t. } T(x,r) = \ell\}|] \leq \mathrm{RSize}^{\mathrm{M}}(f)$. For $m > 2 \cdot \mathrm{RSize}^{\mathrm{M}}(f)$, by averaging argument, there exist $i \in [m]$ such that $\Pr_r[T \text{ outputs } i] < 1/2$. Since for each $i \in [m]$ there exist x such that $f(x) = i$, choosing one such x we get that $\Pr_r[T(x) = f(x)] < 1/2$, contradicting the correctness of T. So for any multi-output function f we have, $\log \mathrm{DSize}^{\mathrm{M}}(f) \leq q(\log \mathrm{RSize}^{\mathrm{M}}(f), n) \cdot \log m \leq q(\log \mathrm{RSize}^{\mathrm{M}}(f), n) \cdot (\log \mathrm{RSize}^{\mathrm{M}}(f) + 1)$.

We utilize the relationship between deterministic and randomized complexities for multi-output functions to relate pseudodeterministic and deterministic complexities of search problems. For total search problem $\mathcal{S}$, let $\tilde{f}$ be a multi-output function solving $\mathcal{S}$, with

$\mathrm{psDSize}^{\mathrm{M}}(\mathcal{S}) = \mathrm{RSize}^{\mathrm{M}}(\tilde{f})$. Then

$$\begin{aligned}\log \mathrm{DSize}^{\mathrm{M}}(\mathcal{S}) &= \min_{f \in_s \mathcal{S}} \log \mathrm{DSize}^{\mathrm{M}}(f) \\ &\leq \log \mathrm{DSize}^{\mathrm{M}}(\tilde{f}) \\ &= O(q(\log \mathrm{RSize}^{\mathrm{M}}(\tilde{f}), n) \cdot \min(\log m, \log \mathrm{RSize}^{\mathrm{M}}(\tilde{f}))) \\ &= O(q(\log \mathrm{psDSize}^{\mathrm{M}}(\mathcal{S}), n) \cdot \min(\log m, \log \mathrm{psDSize}^{\mathrm{M}}(\mathcal{S}))).\end{aligned}$$

◀

Using the above result and a result from [9], we relate the log of deterministic size and the log of pseudodeterministic size for search problems. Recently it was shown in [9] that for all total Boolean functions, the log of deterministic size and the log of randomized size are polynomially related, ignoring a polylogarithmic factor in the input size.

▶ **Theorem 5.2** ([9, Theorem 3.1]). *For every total Boolean function $f : \{0,1\}^n \to \{0,1\}$, we have*

$$\log \mathrm{DSize}^{\mathrm{dt}}(f) = O((\log \mathrm{RSize}^{\mathrm{dt}}(f))^4 \log^3(n)).$$

We get the following result by applying Claim 5.1 to Theorem 5.2.

▶ **Corollary 5.3.** *For a total search problem $\mathcal{S} \subseteq \{0,1\}^n \times [m]$, we have*

$$\log \mathrm{DSize}^{\mathrm{dt}}(\mathcal{S}) = O(\log^4 \mathrm{psDSize}^{\mathrm{dt}}(\mathcal{S}) \cdot \log^3(n) \cdot \min(\log m, \log \mathrm{psDSize}^{\mathrm{dt}}(\mathcal{S})))$$

.

A separation between pseudodeterminism and randomized size was shown in [14]. For the SearchCNF problem on suitably expanding kCNF formulas lifted with 2-bit XOR gadget, the randomized size complexity is $O(1)$, while it was shown in [14] that the pseudo-deterministic size complexity is $\exp(\Omega(\sqrt{n}))$. We note that using the result from [5], which showed a $\exp(\Omega(n))$ lower bound on deterministic size complexity of SearchCNF on suitably expanding kCNF formulas(see Proposition 2.8), and applying Corollary 5.3, we obtain a lower bound of $\exp(\widetilde{\Omega}(n^{1/5}))$ on $\mathrm{psDSize}^{\mathrm{dt}}$ of SearchCNF on such formulas, giving us a separation between $\mathrm{RSize}^{\mathrm{dt}}$ and $\mathrm{psDSize}^{\mathrm{dt}}$ albeit not as strong as [14]. However, due to Corollary 5.3, we can now say that any total search problem which is easy for randomized size and hard for deterministic size will give us a separation between $\mathrm{RSize}^{\mathrm{dt}}$ and $\mathrm{psDSize}^{\mathrm{dt}}$.

6 More general decision trees

A variable is queried at each node of a decision tree. Generalising the class of permitted queries gives rise to many variants of decision trees that have been considered in different contexts. The two fundamental functions that are hard for decision tree depth are AND and PARITY, which are two of the most basic Boolean functions. It is thus natural to look at decision trees where query nodes can evaluate AND's or PARITY's of arbitrary subsets of input bits.
AND decision trees: Each node queries a conjunction of some variables.
PARITY decision trees: Each node queries the parity of some variables.
We denote the query complexity in these models, for different modes of computation, by $\mathrm{D}^{\wedge\text{-dt}}$, $\mathrm{psD}^{\wedge\text{-dt}}$, $\mathrm{R}^{\wedge\text{-dt}}$ and $\mathrm{D}^{\oplus\text{-dt}}$, $\mathrm{psD}^{\oplus\text{-dt}}$, $\mathrm{R}^{\oplus\text{-dt}}$.

Both these versions generalise decision trees and are much more powerful in the deterministic setting – the $\textsc{And}_n$ function has $\mathrm{D}^{\mathrm{dt}} = n$ and $\mathrm{D}^{\wedge\text{-dt}} = 1$, while the $\textsc{Parity}_n$ function has $\mathrm{D}^{\mathrm{dt}} = n$ and $\mathrm{D}^{\oplus\text{-dt}} = 1$.

Pseudodeterminism can be separated from randomness in AND decision trees. To establish the separation, we first give a technique to prove a pseudo-deterministic lower bound using monotone block sensitivity. The following theorem generalises Theorem 3.1(2) to AND decision trees. The same relation is proved for Boolean functions in [18], by reduction to a hard communication problem; here, we give a more direct proof.

▶ **Theorem 6.1.** *For a multi-output function f, $R^{\wedge\text{-dt}}_{1/3}(f) \geq mbs(f))/3$.*

Proof. Let a be an input with monotone block sensitivity $k = \mathrm{mbs}(f)$, and let $B_1, B_2, \ldots, B_k$ be sensitive disjoint 0-blocks of a. We describe a hard distribution $\mathcal{D}$ such that $\mathrm{D}^{\wedge\text{-dt}}_{\mathcal{D},1/3}(f) \geq k/3$, thereby showing $\mathrm{R}^{\wedge\text{-dt}}_{1/3}(f) \geq k/3$. The hard distribution is similar to the one used in Theorem 3.1(2).

$$\mathcal{D}(x) = \begin{cases} 1/2 & \text{if } x = a \\ 1/(2k) & \text{if } x = a \oplus 1_{B_i} \quad \text{for } i \in [k] \\ 0 & \text{otherwise} \end{cases}$$

We show that there is an adversary strategy $\mathcal{A}$ for responding to AND queries such that for any AND-decision tree T, if $\mathrm{Depth}(T) < k/3$, then the probability that T errs when following the responses of $\mathcal{A}$ is more than $1/3$.

The adversary, $\mathcal{A}$, maintains a partial assignment ρ consistent with his answers as follows: Firstly, adversary fixes all the variables not part of $\cup_i B_i$ according to a. Now, if T asks a query whose answer is already determined by ρ, $\mathcal{A}$ answers accordingly. Otherwise, the query asked must involve variables from at least one of the sensitive blocks not set in ρ yet. $\mathcal{A}$ picks one such block arbitrarily and sets all its variable to 0 in ρ, and returns 0 to T as the query reply.

It is clear that the ρ maintained by the adversary is consistent with his answers to queries. Also, at each stage, each of the sensitive blocks is either set entirely to 0s in ρ, or entirely unset in ρ. Each query results in at most one of the sensitive blocks being set.

If $\mathrm{Depth}(T) < k/3$, then T asks less than $k/3$ queries and returns an answer L on a leaf l. More than $2k/3$ blocks thus remain unset when l is reached; w.l.o.g. let $B_1, B_2, ..., B_s$ be these blocks, for some $s > 2k/3$. On all the inputs in the set $\{a, a \oplus 1_{B_1}, a \oplus 1_{B_2}, ..., a \oplus 1_{B_s}\}$, T will reach l and output answer L. However, $f(a \oplus 1_{B_i}) \neq f(a)$ for each $i \in [s]$. If $L \neq f(a)$, then $\Pr_{x\sim\mathcal{D}}[T(x) \neq f(x)] \geq \mathcal{D}(a) = 1/2$. On the other hand, if $L = f(a)$, then

$$\Pr_{x\sim\mathcal{D}}[T(x) \neq f(x)] \geq \sum_{i\in[s]} \mathcal{D}(a \oplus 1_{B_i}) = s \times \frac{1}{2k} > \frac{2k}{3}\frac{1}{2k} = \frac{1}{3}.$$

Thus, either way, if $\mathrm{Depth}(T) < k/3 = \mathrm{mbs}(f)/3$ then $\Pr_{x\sim\mathcal{D}}[T(x) \neq f(x)] > 1/3$. It follows that $\mathrm{D}^{\wedge\text{-dt}}_{\mathcal{D},1/3}(f) \geq \mathrm{mbs}(f)/3$. By Proposition 2.1, $\mathrm{R}^{\wedge\text{-dt}}_{1/3}(f) \geq \mathrm{mbs}(f)/3$. ◀

From this theorem and the definition of pseudodeterminism, we obtain the following corollary.

▶ **Corollary 6.2.** *For a total search problem $\mathcal{S}$, $psD^{\wedge\text{-dt}}_{1/3}(\mathcal{S}) \geq \min_{f\in_s\mathcal{S}} mbs(f)/3$.*

Using this result, we can now separate randomised and pseudodeterministic complexity for AND decision trees.

▶ **Theorem 6.3.** *Let $\mathcal{S}$ be the search problem* $\textsc{ApproxHamWt} = \{(x, v) : |wt(x) - v| \leq n/10\}$*, where $wt(x)$ is the Hamming weight of x. Then $R^{\wedge\text{-dt}}(\mathcal{S}) = R^{\text{dt}}(\mathcal{S}) = O(1)$, while $psD^{\wedge\text{-dt}}(\mathcal{S}) \in \Omega(n)$.*

Proof. It is easy to see, and already noted in Corollary 4.2 of [12], that $\mathrm{R}^{\text{dt}}(\mathcal{S}) = O(1)$.

To show $\mathrm{psD}^{\wedge\text{-dt}}(\textsc{ApproxHamWt}) = \Omega(n)$, we will show that any f solving $\textsc{ApproxHamWt}$ must have monotone sensitivity of at least $4n/5$. This too follows the proof outline from Corollary 4.2 of [12], where a lower bound on psD^{dt} was obtained. But using Corollary 6.2, we draw the stronger conclusion that $\mathrm{psD}^{\wedge\text{-dt}}(\textsc{ApproxHamWt}) \geq 4n/5$.

Suppose that for some f solving $\textsc{ApproxHamWt}$, $\mathrm{ms}(f) < 4n/5$. We start with $x^0 = 0^n$ and create a sequence of inputs $\langle x^i \rangle$ such that $wt(x^i) = i$ and $f(x^i) = f(0^n)$. Because f solves $\textsc{ApproxHamWt}$, $n/10 \geq f(0^n) = f(x^1) = f(x^2) = \ldots = f(x^l) \geq l - n/10$. Thus is we are able to create such a sequence of length at least $l = n/5 + 1$, then we already have a contradiction.

The only thing left is to create the sequence x^i. For $0 \leq i \leq n/5$, given x^i with $f(x^i) = f(0^n)$, we need to find a suitable x^{i+1}. Note that x^i has exactly $n - i$ 0-bit positions, of which at most $\mathrm{ms}(f)$ are sensitive, so at least $s = n - i - \mathrm{ms}(f)$ 0-bit positions are not sensitive. Since $\mathrm{ms}(f) < 4n/5$ and $i \leq n/5$, $s > 0$, so x^i has at least one non-sensitive 0-bit position. Pick any such position, say j, and define $x^{i+1} = x^i \oplus 1_{\{j\}}$. Note that x^{i+1} satisfies the desired properties we are looking for i.e. $f(x^{i+1}) = f(x^i) = f(0^n)$ and $wt(x^{i+1}) = i + 1$. ◀

Recently it was shown in [9] that the deterministic $\textsc{And}$ query complexity and randomized $\textsc{And}$ query complexity for total boolean functions are polynomially related, ignoring polylog$n$ factors.

▶ **Proposition 6.4** ([9, Theorem 4.5])**.** *For every total Boolean function $f : \{0,1\}^n \to \{0,1\}$, $D^{\wedge\text{-dt}}(f) = O(R^{\wedge\text{-dt}}(f)^3 \log^4(n))$.*

Using this along with Claim 5.1, we get a polynomial relationship between $\mathrm{psD}^{\wedge\text{-dt}}$ and $\mathrm{D}^{\wedge\text{-dt}}$.

▶ **Corollary 6.5.** *For a total search problem $\mathcal{S} \subseteq \{0,1\}^n \times [m]$, we have*

$$D^{\wedge\text{-dt}}(\mathcal{S}) = O(psD^{\wedge\text{-dt}}(\mathcal{S})^3 \cdot \log^4(n) \cdot \min(\log m, psD^{\wedge\text{-dt}}(\mathcal{S})))$$

.

For $\textsc{Parity}$ decision trees we show that such a relation does not hold; pseudodeterminism adds significant power.

▶ **Theorem 6.6.** *Let $\mathcal{S}$ be the search problem*

$$\textsc{SearchOR} = \{(x, v) : (x_v = 1) \text{ or } (x = 0^n \wedge v = n + 1)\}.$$

Then $D^{\oplus\text{-dt}}(\mathcal{S}) = n$ whereas $psD^{\oplus\text{-dt}}(\mathcal{S}) = O(\log n \log\log n)$.

Proof. $\mathrm{D}^{\oplus\text{-dt}}(\mathcal{S}) \leq n$ is trivial; we show $\mathrm{D}^{\oplus\text{-dt}}(\mathcal{S}) \geq n$. Let T be any parity decision tree solving $\mathcal{S}$. Consider the left-most path P in the tree, i.e. the path where all the queries are reported to be 0, and let it terminate at the leaf ℓ. We claim that this path must be of length n. Suppose not. Let $L_1, L_2, ..., L_k$, for $k < n$, be the set of parities queried by T on the path P. Now, note that all the inputs on which T reaches leaf ℓ form an affine subspace $\mathcal{A}$ of co-dimension at most k defined by $L_1 = 0, L_2 = 0, \ldots, L_k = 0$. Since $k < n$, it contains at least $2^{n-k} \geq 2$ points. Clearly, 0^n is in $\mathcal{A}$, but it must contain at least one more point, x, other than 0^n. Since $\mathcal{S}(0) \cap \mathcal{S}(x) = \emptyset$, T must err on either x or 0 (or both). Thus, Depth(T) must be at least n. Hence $\mathrm{D}^{\oplus\text{-dt}}(\mathcal{S}) = n$.

Next, we show that $\mathrm{psD}^{\oplus\text{-dt}}(\mathcal{S}) = O(\log n \log\log n)$. Let f be the multi-output function which returns $n+1$ on input 0^n, and on all other inputs it returns the bit position of the first 1. Note that f solves SearchOR. We give a randomized algorithm for f making $O(\log n \log\log n)$ queries, thereby showing that $\mathrm{psD}^{\oplus\text{-dt}}(\mathcal{S}) = O(\log n \log\log n)$. The main idea for the randomized algorithm is to perform binary search for the bit position of the first 1. The algorithm is as follows:

1. Initialise the search space $\mathcal{C}$ to $[1, 2, ..., n]$. C is an ordered set.
2. Repeat until the search space C contains exactly one bit position: Let $C = [p, p+1, ..., p+s]$ at the current stage. For $k = 2\log\log n$, sample k random parities $L_1, L_2, \ldots, L_k$ independently over the variables $x_p, x_{p+1}, ..., x_{p+\lfloor s/2 \rfloor}$. That is, for $i \in [k]$ and $p \leq j \leq p + \lfloor s/2 \rfloor$, each L_i independently contains x_j with probability 1/2. Query $L_1, \ldots, L_k$, and if any one of them evaluates to 1, update the search space C to $[p, p+1., , p+\lfloor s/2 \rfloor]$. Otherwise update C to $[p+\lfloor s/2 \rfloor + 1, p + \lfloor s/2 \rfloor + 2, ..., p+s]$.
3. Let p be the only bit position in $\mathcal{C}$ at this stage. If $x_p = 0$ return $n+1$ otherwise return p.

First, note that the algorithm makes at most $O(\log n \log\log n)$ queries, since the search space reduces by half in each iteration of step 2 and each iteration of step 2 makes $2\log\log n$ queries. We now show the correctness.

On the all-zero input 0^n, with probability 1 the algorithm is correct (since it reaches step 3 with $p = n$).

Let x be an input which contains at least one bit set to 1, and let q be the first such bit position. The algorithm performs a binary search trying to find q. It maintains in C the potential search space which should contain q. Certainly, in the beginning, C contains q. The algorithm reduces the search space to half by querying random parities over variables from the first half of the search space. We argue that with good enough probability, the algorithm reduces the search space correctly i.e. if C contained q before an iteration of step 2, then with the good probability it contains q after the operation. Observe that if the first half of the search space contains q, then each L_i independently evaluates to 1 with probability 1/2. Since we query $k = 2\log\log n$ parities, with probability $1 - \frac{1}{2^k} = 1 - \frac{1}{(\log n)^2}$, the algorithm detects the correct half of the search space containing q. If the first half of the search space does not contain q, then all queries report 0, and so with probability 1, the algorithm detects the correct half of the search space containing q. Thus any one iteration erroneously discards q from the search space with probability at most $\frac{1}{(\log n)^2}$. If the algorithm reduces the search space correctly in each of the $\log n$ iterations of step 2, then it will return the correct answer for x. By the union bound, the algorithm is correct on x with probability at least $1 - \frac{1}{\log n}$. ◀

The separation between randomness and pseudodeterminism remains unclear in Parity decision tree model.

7 A combinatorial proof of a Combinatorial Problem

In [14], the authors studied the pseudodeterministic query complexity of a promise problem (PromiseFIND1). Here the input bit string has 1s in at least half the positions, and the task is to find a 1. They observed that PromiseFIND1 is a complete problem for easily-verifiable search problems with randomized query algorithms (see Theorem 3 in [14]), and proved a $\Omega(\sqrt{n})$ lower bound on its pseudodeterministic query complexity. They conjectured that the pseudodeterminisitic query lower bound for PROMISEFIND1 can be improved to $\Omega(n)$. Towards understanding the PROMISEFIND1 problem better, they introduced a natural colouring problem on hypercubes which states that any proper coloring of the hypercube contains a point with many 1s and with high block sensitivity.

▶ **Definition 7.1.** *A proper coloring of the n-dimensional hypercube is any function* $\phi : \{0,1\}^n - \{0^n\} \longrightarrow [n]$ *such that for all* $\beta \in \{0,1\}^n - \{0^n\}$, $\beta_{\phi(\beta)} = 1$.

We say a proper coloring ϕ is d-sensitive if there exists a $\beta \in \{0,1\}^n$ such that $|\beta|_1 \geq n/2$ and β has block sensitivity at least d with respect to ϕ. That is, there are d disjoint blocks of inputs, $B_1, ..., B_d$ such that for all $i \in [d]$, $\phi(\beta) \neq \phi(\beta \oplus 1_{B_i})$. The hypercube coloring problem is about proving lower bound on the (block) sensitivity of every proper coloring. In [14] it was shown that every proper coloring is $\Omega(\sqrt{n})$-sensitive.

▶ **Theorem 7.2** (Restated from Theorem 14 [14])**.** *Every proper coloring of the Boolean cube is* $\Omega(\sqrt{n})$*-sensitive.*

The hypercube coloring problem is closely related to the pseudodeterministic query complexity of PROMISEFIND1. It is a straightforward observation that showing every proper coloring is d-sensitive implies a lower bound of d on the pseudo-deterministic query complexity of PROMISEFIND1. To prove Theorem 7.2, [14] converted their sensitivity lower bound for the search problem associated with a random unsat k-XOR formula into a block sensitivity lower bound for the hypercube coloring problem.

We give a self-contained combinatorial solution to the coloring problem. Our solution shows that every proper coloring of hypercube has a $\beta \in \{0,1\}^n$ with Hamming weight $\geq n/2$ and with block sensitivity $\Omega(n^{1/3})$. In fact, we show that either the 1-block sensitivity or the 0-block sensitivity (or both) is $\Omega(n^{1/3})$. Thus this appears incomparable with the bound from [14].

Our solution is constructive: we describe an algorithm that finds the required high-weight high-block-sensitivity point, by querying ϕ at various points. It is not an efficient algorithm, since it involves computing block-sensitivity at various points. But it finds the required point, hence proving that such a point exists. On the other hand, the solution in [14] independently proves the existence of such a point, and so a brute-force search algorithm can find one.

▶ **Theorem 7.3.** *Every proper coloring* ϕ *of the Boolean hypercube has a* $\beta \in \{0,1\}^n$ *with* $|\beta| \geq n/2$ *satisfying* $bs_0(\phi, \beta) = \Omega(n^{1/3})$ *or* $bs_1(\phi, \beta) = \Omega(n^{1/3})$.

In particular, this implies a $\Omega(n^{1/3})$ lower bound on the block sensitivity of the hypercube coloring problem and on the pseudodeterministic query complexity of PROMISEFIND1. While our bound is not as strong as the lower bound of $\Omega(\sqrt{n})$ from [14], it is simple and self-contained, and we hope that it will add to our understanding of PROMISEFIND1 problem.

Proof. In Algorithm 1, we describe a procedure to find the required point β. To prove that the algorithm is correct, we need to prove that if it returns $\beta \in \{0,1\}^n$ and blocks $D_1, D_2, \ldots, D_r$, then

1. $\beta \in \mathcal{X}$ (i.e. β has Hamming weight at least $n/2$),
2. $D_1, D_2, \ldots, D_r$ are disjoint sensitive blocks of ϕ at β, and
3. either all these blocks are 1-blocks of β or all these blocks are 0-blocks.
4. $r \in \Omega(n^{1/3})$,

Observe that by construction, for each $i \in [t+1]$ where β^i is constructed by the algorithm, β^i has 0s in B_j for $j < i$ and 1s in B_i (in fact, 1s elsewhere); hence the blocks $B_1, \ldots, B_{i-1}$ are disjoint.

Further, by construction, each complete iteration of the for loop adds fewer than t^2 positions to C: there are fewer than t blocks (otherwise the algorithm would terminate at line 12) and each block has size less than t (otherwise the algorithm would terminate at line 16). Thus, since $|C_0| = 0$, if the algorithm reaches line 18 in iteration i, then C_i has size less than $i \cdot t^2$. Hence β^{i+1} has hamming weight $n - |C_i| \ > n - it^2 \geq \ n - t^3 > n - n/2 \geq n/2$ and is in $\mathcal{X}$.

Algorithm 1 Algorithm to find the sensitive point

Require: A proper coloring ϕ. i.e.

1: For $\mathcal{X} = \{x \in \{0,1\}^n \mid \sum_i x_i \geq n/2,\ \phi : \mathcal{X} \to [n]$ satisfying $\forall x \in \mathcal{X},\ x_{\phi(x)} = 1$.
2: $t \leftarrow \lfloor (n/2)^{1/3} \rfloor$
3: $C_0 \leftarrow \emptyset$
4: **for** i from 1 to t **do**
5: $\quad \beta^i \leftarrow 0_{C_{i-1}}$ ▷ Reference input for which we try to find t sensitive 1-blocks.
6: $\quad \ell \leftarrow \phi(\beta^i)$
7: $\quad s \leftarrow \mathrm{bs}_1(\phi, \beta^i)$ ▷ $\{\ell\}$ is a 1-sensitive block of β^i, so $s \geq 1$
8: $\quad B_{i,1}, B_{i,2}, \ldots, B_{i,s}$: disjoint, minimally-sensitive
9: $\quad$ 1-blocks achieving the 1-block sensitivity s.
10: $\quad B_i \leftarrow \cup_{j=1}^{s} B_{i,j}$ ▷ ℓ is a sensitive bit of β^i and s is maximum number of disjoint 1-sensitive blocks, $\ell \in B_i$.
11: $\quad$ **if** $s \geq t$ **then**
12: $\quad\quad$ **return** β^i and $\{B_{i,1}, B_{i,2}, \ldots, B_{i,s}\}$ ▷ $\mathrm{bs}_1(\phi, \beta^i) \geq t$
13: $\quad$ **end if**
14: $\quad$ **if** $\max_{j \in [s]} |B_{i,j}| \geq t$ **then**
15: $\quad\quad$ Pick any such $j \in [s]$ with $|B_{i,j}| \geq t$.
16: $\quad\quad$ **return** $\beta^i \oplus 1_{B_{i,j}}$ and $\{\{k\} \mid k \in B_{i,j}\}$ ▷ $\mathrm{s}_0(\phi, \beta^i \oplus 1_{B_{i,j}}) \geq t$
17: $\quad$ **end if**
18: $\quad C_i \leftarrow C_{i-1} \cup B_i$ ▷ We show: C_i forms a ϕ-certificate for β^i
19: **end for**
20: $\beta^{t+1} \leftarrow 0_{C_t}$
21: **return** β^{t+1} and $\{B_1, B_2, \ldots, B_t\}$ ▷ $\mathrm{bs}_0(\phi, \beta^{t+1}) \geq t$

If the algorithm terminates at line 12 in the ith iteration of the for loop, then by the choice in line 9 the returned blocks are disjoint 1-sensitive blocks of $\beta = \beta^i$, and there are at least t of them. Similarly, if the algorithm terminates at line 16 in the ith iteration of the for loop, then by minimality of the sensitive block $B_{i,j}$ chosen in line 15, each position in $B_{i,j}$ is a 0-sensitive location in $\beta = \beta^i \oplus 1_{B_{i,j}}$, and there are at least t of them.

If the algorithm terminates at line 21, then each B_i is a 0-block of $\beta = \beta^{t+1}$ and there are t such blocks. It remains to prove that each B_i is sensitive for $\beta = \beta^{t+1}$. To show this, we will first show that each C_i is a certificate for β^i, and then show that this implies each B_i is sensitive for β.

For the first part, suppose for some $i \in [t]$, C_i is not a certificate for β^i. Then there exists an $\alpha \in \mathcal{X}$ such that $\forall j \in C_i, \alpha_j = \beta^i_j$, but $\phi(\alpha) \neq \phi(\beta^i)$. Let B be the set of positions where α and β^i differ i.e. $\alpha = \beta^i \oplus 1_B$. Since α and β^i agree on C_i, B must be disjoint from C_i. Since $\phi(\beta^i) \neq \phi(\alpha) = \phi(\beta^i \oplus 1_B)$, B is a 1-sensitive block of ϕ at β^i. By the choice in line 9 at the ith iteration, β^i has no 1-sensitive blocks disjoint from the blocks $B_{i,1}, \ldots, B_{i,s}$. But B_i is precisely the union of the these blocks, and is contained in C_i, so B is disjoint from B_i, a contradiction. Hence C_i is indeed a ϕ-certificate for β^i.

For the second part, note that for each $i \in [t]$, β and β^i agree on C_{i-1} and $\beta \oplus B_i$ and β^i agree on C_i. Since C_i is a certificate for β^i, $\phi(\beta \oplus B_i) = \phi(\beta^i) = \ell$, say. By the definition of proper coloring, $\{\ell\}$ is a 1-sensitive block of β^i, and since the blocks chosen in line 9 are the maximum possible 1-sensitive blocks, $\ell \in B_i$. But $\phi(\beta) \neq \ell$ because $\beta = 0_{C_t}$ and has only 0s in B_i. Thus $\phi(\beta) \neq \phi(\beta \oplus B_i)$, and hence B_i is a 0-sensitive block for β.

Finally, by choice of t, we see that $r \in \Omega(n^{1/3})$. This completes the proof of correctness of the algorithm. ◀

References

1. Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. In *42nd Annual Symposium on Foundations of Computer Science FOCS*, pages 190–199. IEEE Computer Society, 2001.
2. Paul Beame, Richard Karp, Toniann Pitassi, and Michael Saks. The efficiency of resolution and Davis–Putnam procedures. *SIAM Journal on Computing*, 31(4):1048–1075, 2002.
3. Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures & Algorithms*, 23(1):92–109, 2003.
4. Eli Ben-Sasson*, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, 2004.
5. Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2):149–169, 2001.
6. Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113(18):666–671, 2013.
7. Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.
8. Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *J. Comput. Syst. Sci.*, 62(2):267–289, 2001.
9. Arkadev Chattopadhyay, Yogesh Dahiya, Nikhil Mande, Jaikumar Radhakrishnan, and Swagato Sanyal. Randomized versus deterministic decision tree size. *Electron. Colloquium Comput. Complex.*, TR22-185, 2022.
10. Vasek Chvátal and Endre Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4):759–768, 1988.
11. Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, page 136, 2011.
12. Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. *Electron. Colloquium Comput. Complex.*, page 101, 2012. extended abstract in proceedings of ITCS 2013.
13. Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science ITCS*, pages 127–138. ACM, 2013. See also ECCC Vol. 19, T.R. 12-101, 2012.

14 Shafi Goldwasser, Russell Impagliazzo, Toniann Pitassi, and Rahul Santhanam. On the pseudo-deterministic query complexity of NP search problems. In Valentine Kabanets, editor, *36th Computational Complexity Conference CCC*, volume 200 of *LIPIcs*, pages 36:1–36:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

15 Dima Grigoriev. Tseitin's tautologies and lower bounds for nullstellensatz proofs. In *39th Annual Symposium on Foundations of Computer Science FOCS*, pages 648–652. IEEE Computer Society, 1998.

16 Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *CoRR*, abs/1907.00847, 2019.

17 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012.

18 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for and-functions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 197–208, 2021.

19 László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM Journal on Discrete Mathematics*, 8(1):119–132, 1995.

20 Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In *24th Annual Symposium on Foundations of Computer Science FOCS*, pages 420–428. IEEE Computer Society, 1983.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il