On (Simple) Decision Tree Rank

Yogesh Dahiya ✉

The Institute of Mathematical Sciences, Chennai, (A CI of Homi Bhabha National Institute HBNI), India

Meena Mahajan ✉ ⓘ

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

──── **Abstract** ────────────────────────

In the decision tree computation model for Boolean functions, the depth corresponds to query complexity, and size corresponds to storage space. The depth measure is the most well-studied one, and is known to be polynomially related to several non-computational complexity measures of functions such as certificate complexity. The size measure is also studied, but to a lesser extent. Another decision tree measure that has received very little attention is the minimal rank of the decision tree, first introduced by Ehrenfeucht and Haussler in 1989. This measure is closely related to the logarithm of the size, but is not polynomially related to depth, and hence it can reveal additional information about the complexity of a function. It is characterised by the value of a Prover-Delayer game first proposed by Pudlák and Impagliazzo in the context of tree-like resolution proofs.

In this paper we study this measure further. We obtain an upper bound on depth in terms of rank and Fourier sparsity. We obtain upper and lower bounds on rank in terms of (variants of) certificate complexity. We also obtain upper and lower bounds on the rank for composed functions in terms of the depth of the outer function and the rank of the inner function. This allow us to easily recover known asympotical lower bounds on logarithm of the size for Iterated AND-OR and Iterated 3-bit Majority. We compute the rank exactly for several natural functions and use them to show that all the bounds we have obtained are tight. We also show that rank in the simple decision tree model can be used to bound query complexity, or depth, in the more general conjunctive decision tree model. Finally, we improve upon the known size lower bound for the Tribes function and conclude that in the size-rank relationship for decision trees, obtained by Ehrenfeucht and Haussler, the upper bound for Tribes is asymptotically tight.

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

Keywords and phrases Boolean functions, Decision trees, rank, certificate complexity, sparsity, iterated composition

1 Introduction

The central problem in Boolean function complexity is to understand exactly how hard it is to compute explicit functions. The hardness naturally depends on the computation model to be used, and depending on the model, several complexity measures for functions have been studied extensively in the literature. To name a few – size and depth for circuits and formulas, size and width for branching programs, query complexity, communication complexity, length for span programs, and so on. All of these are measures of the computational hardness of a function. There are also several ways to understand hardness of a function intrinsically, independent of a computational model. For instance, the sensitivity of a function, its certificate complexity, the sparsity of its Fourier spectrum, its degree and approximate degree, stability, and so on. Many bounds on computational measures are obtained by directly relating them to appropriate intrinsic complexity measures. See [13] for a wonderful overview of this area. Formal definitions of relevant measures appear in Section 2.

Every Boolean function f can be computed by a simple decision tree (simple in the sense that each node queries a single variable), which is one of the simplest computation models for Boolean functions. The most interesting and well-studied complexity measure in the

decision tree model is the minimal depth $\text{Depth}(f)$, measuring the query complexity of the function. This measure is known to be polynomially related to several intrinsic measures: sensitivity, block sensitivity, certificate complexity. But there are also other measures which reveal information about the function. The minimal size of a decision tree, $\text{DTSize}(f)$, is one such measure, which measures the storage space required to store the function as a tree, and has received some attention in the past.

A measure which has received relatively less attention is the minimal rank of a decision tree computing the function, first defined and studied in [9]; see also [1]. In general, the rank of a rooted tree (also known as its Strahler number, or Horton-Strahler number, or tree dimension) measures its branching complexity, and is a tree measure that arises naturally in a wide array of applications; see for instance [10]. The rank of a Boolean function f, denoted $\text{Rank}(f)$, is the minimal rank of a decision tree computing it. The original motivation for considering rank of decision trees was from learning theory – an algorithm, proposed in [9], and later simplified in [5], shows that constant-rank decision trees are efficiently learnable in Valiant's PAC learning framework [28]. Subsequently, the rank measure has played an important role in understanding the decision tree complexity of search problems over relations [23, 11, 18] – see more in the Related Work part below. The special case when the relation corresponds to a Boolean function is exactly the rank of the function. However, there is very little work focussing on the context of, and exploiting the additional information from, this special case. This is precisely the topic of this paper.

In this paper, we study how the rank of boolean functions relates to other measures. In contrast with $\text{Depth}(f)$, $\text{Rank}(f)$ is not polynomially related with sensitivity or to certificate complexity $\text{C}(f)$, although it is bounded above by $\text{Depth}(f)$. Hence it can reveal additional information about the complexity of a function over and above that provided by Depth. For instance, from several viewpoints, the PARITY_n function is significantly harder than the AND_n function. But both of them have the same Depth, n. However, Rank does reflect this difference in hardness, with $\text{Rank}(\text{AND}_n) = 1$ and $\text{Rank}(\text{PARITY}_n) = n$. On the other hand, rank is also already known to characterise the logarithm of decision tree size (DTSize), upto a $\log n$ multiplicative factor. Thus lower bounds on rank give lower bounds on the space required to store a decision tree explicitly. (However, the $\log n$ factor is crucial; there is no dimension-free characterisation. Consider e.g. $\log \text{DTSize}(\text{AND}_n) = \Theta(\log n)$.)

Our main findings can be summarised as follows:

1. $\text{Rank}(f)$ is equal to the value of the Prover-Delayer game of Pudlák and Impagliazzo [23] played on the corresponding relation R_f (Theorem 3.1). (This is implicit in earlier literature [18, 11].)

2. While Rank alone cannot give upper bounds on $\text{Depth}(f)$, $\text{Depth}(f)$ is bounded above by the product of $\text{Rank}(f)$ and $1 + \log \text{spar}(f)$ (Theorem 5.2).

3. $\text{Rank}(f)$ is bounded between the minimum certificate complexity of f at any point, and $(\text{C}(f) - 1)^2 + 1$; Theorem 5.8. The upper bound (Lemma 5.4) is an improvement on the bound inherited from $\text{Depth}(f)$, and is obtained by adapting that construction.

4. For a composed function $f \circ g$, $\text{Rank}(f \circ g)$ is bounded above and below by functions of $\text{Depth}(f)$ and $\text{Rank}(g)$; Theorem 6.6. The main technique in both bounds (Theorems 6.3 and 6.5) is to use weighted decision trees, as was used in the context of depth [21]. For iterated composed functions, these bounds can be used recursively (Corollary 6.8), and can be used to easily recover known bounds on Rank for some functions (Corollary 7.1).

5. The measures Rank and $\log \text{DTSize}$ for simple decision trees sandwich the query complexity in the more general decision tree model where each node queries a conjunction of literals (Theorem 7.4).

6. In the relation between $\mathrm{Rank}(f)$ and $\mathrm{DTSize}(f)$ from [9], the upper bound on $\log \mathrm{DTSize}(f)$ is asymptotically tight for the Tribes function (Section 8).

By calculating the exact rank for specific functions, we show that all the bounds we obtain on rank are tight. We also describe optimal strategies for the Prover and Delayer, for those more familiar with that setting.

Related work.

A preliminary version of this paper, with some proofs omitted or only briefly sketched, appears in the proceedings of the FSTTCS 2021 conference [8].

In [15] (Corollary 12), non-trivial learnability of s-term DNFs is demonstrated. The crucial result that allows this learning is the transformation of the DNF expression into a polynomial threshold function of not too large degree. An important tool in the transformation is the rank of a hybrid kind of decision tree; in these trees, each node queries a single variable, while the subfunctions at the leaves, though not necessarily constant, have somewhat small degree. The original DNF is fist converted to such a hyrid tree with a bound on its rank, and this is exploited to achieve the full conversion to low-degree polynomial threshold functions. This generalises an approach credited in [15] to Satya Lokam.

In [1], a model called k^{+}-decision trees is considered, and the complexity is related to both simple decision tree rank and to communication complexity. In particular, Theorems 7 and 8 from [1] imply that communication complexity lower bounds with respect to any variable partition (see [19]) translate to decision tree rank lower bounds, and hence by [9] to decision tree size lower bounds.

In [26], the model of linear decision trees is considered (here each node queries not a single variable but a linear threshold function of the variables), and for such trees of bounded rank computing the inner product function, a lower bound on depth is obtained. Thus for this function, in this model, there is a trade-off between rank and depth. In [27], rank of linear decision trees is used in obtaining non-trivial upper bounds on depth-2 threshold circuit size.

In [23], a 2-player game is described, on an unsatisfiable formula F in conjunctive normal form, that constructs a partial assignment falsifying some clause. The players are referred to in subsequent literature as the Prover and the Delayer. The value of the game, $\mathrm{Value}(F)$, is the maximum r such that the Delayer can score at least r points no matter how the Prover plays. It was shown in [23] that the size of any tree-like resolution refutation of F is at least $2^{\mathrm{Value}(F)}$. Subsequently, the results of [18, 11] yield the equivalence $\mathrm{Value}(F) = \mathrm{Rank}(F)$, where $\mathrm{Rank}(F)$ is defined to be the minimal rank of the tree underlying a tree-like resolution refutation of F. (Establishing this equivalence uses refutation-space and tree pebbling as intermediaries.) The relevance here is because there is an immediate, and well-known, connection to decision trees for search problems over relations: tree-like resolution refutations are decision trees for the corresponding search CNF problem. (See Lemma 7 in [3]). Note that the size lower bound from [23], and the rank-value equivalence from [18, 11], hold for the search problem over arbitrary relations, not just searchCNF. (See e.g. Exercise 14.16 in Jukna for the size bound.) In particular, for Boolean function f, it holds for the corresponding canonical relation R_f defined in Section 2. Similarly, the value of an asymmetric variant of this game is known to characterise the size of a decision tree for the search CNF problem [4], and this too holds for general relations and Boolean functions.

Organisation of the paper.

After presenting basic definitions and known results in Section 2, we describe the Prover-Delayer game from [23] in Section 3, and observe that its value equals the rank of the function. We also describe the asymmetric game from [4]. We compute the rank of some simple functions in Section 4. In Section 5, we describe the relation between rank, depth, Fourier sparsity, and certificate complexity. In Section 6, we present results concerning composed functions. In Section 7 we give two applications. Firstly, using our rank lower bound result, we prove the tight log size lower bound. Secondly, we prove a query lower bound in the CONJ decision tree model. In Section 8 we examine the size-rank relationship for the TRIBES function. The bounds in Sections 4–6 and 8 are all obtained by direct inductive arguments/decision tree constructions. They can also be stated using the equivalence of the game value and rank – while this does not particularly simplify the proofs, it changes the language of the proofs and may be more accessible to the reader already familiar with that setting. Hence we include such game-based arguments for our results in Section 9.

2 Preliminaries

Decision trees

For a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, a decision tree computing f is a binary tree with internal nodes labeled by the variables and the leaves labelled by $\{0,1\}$. To evaluate a function on an unknown input, 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 the left subtree, otherwise it proceeds in the right subtree. The label of the leaf so reached is the value of the function on that particular input. A decision tree is said to be reduced if no variable is queried more than once on any root-to-leaf path. Without loss of generality, any decision tree can be reduced, so in our discussion, we will only consider reduced decision trees. The depth $\mathrm{Depth}(T)$ of a decision tree T is the length of the longest root-to-leaf path, and its size $\mathrm{DTSize}(T)$ is the number of leaves. The decision tree complexity or the depth of f, denoted by $\mathrm{Depth}(f)$, is defined to be the minimum depth of a decision tree computing f. Equivalently, $\mathrm{Depth}(f)$ can also be seen as the minimum number of worst-case queries required to evaluate f. The size of a function f, denoted by $\mathrm{DTSize}(f)$, is defined similarly i.e. the minimum size of a decision tree computing f. Since decision trees can be reduced, $\mathrm{Depth}(f) \leq n$ and $\mathrm{DTSize}(f) \leq 2^n$ for every n-variate function f. A function is said to be evasive if its depth is maximal, $\mathrm{Depth}(f) = n$.

Weighted decision trees

Weighted decision trees describe query complexity in settings where querying different input bits can have differing cost, and arises naturally in the recursive construction. Formally, these are defined as follows: Let w_i be the cost of querying variable x_i. For a decision tree T, its weighted depth with respect to the weight vector $[w_1, \ldots, w_n]$, denoted by $\mathrm{Depth}_w(T, [w_1, w_2, ..., w_n])$, is the maximal sum of weights of the variables specified by the labels of nodes of T on any root-to-leaf path. The weighted decision tree complexity of f, denoted by $\mathrm{Depth}_w(f, [w_1, w_2, ..., w_n])$, is the minimum weighted depth of a decision tree computing f. Note that $\mathrm{Depth}(f)$ is exactly $\mathrm{Depth}_w(f, [1, 1, \ldots, 1])$. The following fact is immediate from the definitions.

▶ **Fact 2.1.** *For any reduced decision tree T computing an n-variate function, weights $w_1, \ldots, w_n$, and $i \in [n]$,*

$$Depth_w(T, [w_1, \ldots, w_{i-1}, w_i + 1, w_{i+1}, \ldots, w_n]) \leq Depth_w(T, [w_1, w_2, ..., w_n]) + 1.$$

Certificate Complexity

The certificate complexity of a function f, denoted $\mathrm{C}(f)$, measures the number of variables that need to be assigned in the worst case to fix the value of f. More precisely, for a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ and an input $a \in \{0,1\}^n$, an f-certificate of a is a subset $S \subseteq \{1, ..., n\}$ such that the value of $f(a)$ can be determined by just looking at the bits of a in set S. Such a certificate need not be unique. Let $\mathrm{C}(f, a)$ denote the minimum size of an f-certificate for the input a. That is,

$$\mathrm{C}(f, a) = \min\left\{ |S| \mid S \subseteq [n]; \forall a' \in \{0,1\}^n, \left[\left(a'_j = a_j \forall j \in S \right) \implies f(a') = f(a) \right] \right\}.$$

Using this definition, we can define several measures.

$$\text{For} \quad b \in \{0,1\}, \quad \mathrm{C}_b(f) = \max\{\mathrm{C}(f, a) \mid a \in f^{-1}(b)\}$$
$$\mathrm{C}(f) = \max\{\mathrm{C}(f, a) \mid a \in \{0,1\}^n\} = \max\{\mathrm{C}_0(f), \mathrm{C}_1(f)\}$$
$$\mathrm{C}_{avg}(f) = 2^{-n} \sum_{a \in \{0,1\}^n} \mathrm{C}(f, a)$$
$$\mathrm{C}_{\min}(f) = \min\{\mathrm{C}(f, a) \mid a \in \{0,1\}^n\}$$

Composed functions

For boolean functions $f, g_1, g_2, \ldots, g_n$ of arity $n, m_1, m_2, \ldots, m_n$ respectively, the composed function $f \circ (g_1, g_2, ..., g_n)$ is a function of arity $\sum_i m_i$, and is defined as follows: for $a^i \in \{0,1\}^{m_i}$ for each $i \in n$, $f \circ (g_1, g_2, ..., g_n)(a^1, a^2, ..., a^n) = f(g_1(a^1), g_2(a^2), \ldots, g_n(a^n))$. We call f the outer function and $g_1, \ldots, g_n$ the inner functions. For functions $f : \{0,1\}^n \longrightarrow \{0,1\}$ and $g : \{0,1\}^m \longrightarrow \{0,1\}$, the composed function $f \circ g$ is the function $f \circ (g, g, \ldots, g) : \{0,1\}^{mn} \longrightarrow \{0,1\}$. The composed function $\mathrm{OR}_n \circ \mathrm{AND}_m$ has a special name, $\mathrm{TRIBES}_{n,m}$, and when $n = m$, we simply write TRIBES_n. Its dual is the function $\mathrm{AND}_n \circ \mathrm{OR}_m$ that we denote $\mathrm{TRIBES}_{n,m}^d$. (The dual of $f(x_1, \ldots, x_n)$ is the function $\neg f(\neg x_1, \ldots, \neg x_n)$.)

For any function $f : \{0,1\}^n \longrightarrow \{0,1\}$, that we will call the base function, the iterated composed function $f^{\otimes k} : \{0,1\}^{n^k} \longrightarrow \{0,1\}$ is recursively defined as $f^{\otimes 1} = f$, $f^{\otimes k} = f \circ f^{\otimes(k-1)}$. The iterated composed functions for the base functions $\mathrm{AND}_2 \circ \mathrm{OR}_2$ and MAJ_3 will interest us later.

Symmetric functions

A Boolean function is symmetric if its value depends only on the number of ones in the input, and not on the positions of the ones.

▶ **Proposition 2.2.** *For every non-constant symmetric boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*
1. *f is evasive (has $Depth(f) = n$). (See eg. Lemma 14.19 [13].)*
2. *Hence, for any weights w_i, $Depth_w(f, [w_1, w_2, ..., w_n]) = \sum_i w_i$.*

For a symmetric Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, let $f_0, f_1, ..., f_n \in \{0,1\}$ denote the values of the function f on inputs of Hamming weight $0, 1, ..., n$ respectively. The Gap of f is defined as the length of the longest interval (minus one) where f_i is constant. That is,

$$\mathrm{Gap}(f) = \max_{0 \leq a \leq b \leq n} \{b - a : f_a = f_{a+1} = ... = f_b\}.$$

Analogously, $\mathrm{Gap_{min}}(f)$ is the length of the shortest constant interval (minus one); that is, setting $f_{-1} \neq f_0$ and $f_{n+1} \neq f_n$ for boundary conditions,

$$\mathrm{Gap_{min}}(f) = \min_{0 \leq a \leq b \leq n} \{b - a : f_{a-1} \neq f_a = f_{a+1} = ... = f_b \neq f_{b+1}\}.$$

Fourier Representation of Boolean functions

We include here some basic facts about Fourier representation relevant to our work. For a wonderful comprehensive overview of this area, see [22]. Consider the inner product space of functions $\mathcal{V} = \{f : \{0, 1\}^n \longrightarrow \mathbb{R}\}$ with the inner product defined as

$$\langle f, g \rangle = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)g(x).$$

For $S \subseteq [n]$, the function $\chi_S : \{0, 1\}^n \longrightarrow \{-1, 1\}$ defined by $\chi_S(x) = (-1)^{\sum_{i \in S} x_i}$ is the ± 1 parity of the bits in S and therefore is referred to as a parity function. The set of all parity functions $\{\chi_S : S \subseteq [n]\}$ forms an orthonormal basis for $\mathcal{V}$. Thus, every function $f \in \mathcal{V}$, in particular boolean functions, has a unique representation $f = \sum_{S \subseteq [n]} \hat{f}(S) \chi_S$. The coefficients $\{\hat{f}(S) : S \subseteq [n]\}$ are called the Fourier coefficients(spectrum) of f. The Fourier sparsity of f, denoted by $\mathrm{spar}(f)$, is the number of non-zero Fourier coefficients in the expansion of f, i.e. $|\{S \subseteq [n] : \hat{f}(S) \neq 0\}|$. It will be convenient for us to disregard the Fourier coefficient of the empty set. We therefore define $\tilde{\mathrm{spar}}(f) = |\{S \subseteq [n] : S \neq \emptyset; \hat{f}(S) \neq 0\}|$. For every f, $0 \leq \tilde{\mathrm{spar}}(f) \leq \mathrm{spar}(f) \leq \tilde{\mathrm{spar}}(f) + 1$, and only the constant functions have $\tilde{\mathrm{spar}} = 0$.

Sparsity is related to decision tree complexity; large sparsity implies large depth.

▶ **Proposition 2.3** (see Proposition 3.16 in [22])**.** *For a Boolean function $f : \{0, 1\}^n \longrightarrow \{0, 1\}$,* $\log \mathrm{spar}(f) \leq \log DTSize(f) + Depth(f) \leq 2Depth(f)$.

In our discussion, we will be interested in the effect of restrictions on the Fourier representation of a function. Of particular interest to us will be restrictions to subcubes. A subcube is a set of all inputs consistent with a partial assignment of n bits. Formally, a subcube J is a partial assignment (to some of the n variables) defined by (S, ρ) where $S \subseteq [n]$ is the set of input bits fixed by J and $\rho : S \longrightarrow \{0, 1\}$ is the map according to which the bits in S are fixed. A subcube is a special type of affine subspace; hence, inheriting notation from subspaces, for $J = (S, \rho)$, the cardinality of S is called the co-dimension of J, and is denoted by co-dim(J). A function $f : \{0, 1\}^n \longrightarrow \{0, 1\}$ restricted to $J = (S, \rho)$ is the function $f|J : \{0, 1\}^{n-|S|} \longrightarrow \{0, 1\}$ obtained by fixing variables in S according to ρ. The following result quantifies the effect on Fourier spectrum of subcube restriction.

▶ **Theorem 2.4.** *[[24, 20]] Let f be any Boolean function $f : \{0, 1\}^n \longrightarrow \{0, 1\}$. Fix any $S \subseteq [n]$, $S \neq \emptyset$. If $f|(S, \rho)$ is a constant, then for every $\rho' : S \longrightarrow \{0, 1\}$, $\tilde{\mathrm{spar}}(f|(S, \rho')) \leq \tilde{\mathrm{spar}}(f)/2$.*

This lemma follows from [24] (in the proof of Theorem 1.7 there) and [20] (see the discussion in Sections 2.1 and 3.1 there). Both papers consider affine subspaces, of which subcubes are a special case. Since the result is not explicitly stated in this form in either paper, for completeness we give a proof for the subcubes case in the appendix.

The subcube kill number of f, denoted by K(f), measures a largest subcube over which f is constant, and is defined as

$$\mathrm{K}(f) = \min\{\text{co-dim}(J)|f|J \text{ is constant}\}.$$

Decision Tree Rank

For a rooted binary tree T, the rank of the tree is the rank of the root node, where the rank of each node of the tree is defined recursively as follows: For a leaf node u, $\text{Rank}(u) = 0$. For an internal node u with children v, w,

$$\text{Rank}(u) = \begin{cases} \text{Rank}(v) + 1 & \text{if } \text{Rank}(v) = \text{Rank}(w) \\ \max\{\text{Rank}(v), \text{Rank}(w)\} & \text{if } \text{Rank}(v) \neq \text{Rank}(w) \end{cases}$$

The following proposition lists some known properties of the rank function for binary trees.

▶ **Proposition 2.5.** *For any binary tree T,*
1. *(Rank and Size relationship): $Rank(T) \leq \log(DTSize(T)) \leq Depth(T)$.*
2. *(Monotonicity of the Rank): Let T' be any subtree of T, and let T'' be an arbitrary binary tree of higher rank than T'. If T' is replaced by T'' in T, then the rank of the resulting tree is not less than the rank of T.*
3. *(Leaf Depth and Rank): If all leaves in T have depth at least r, then $Rank(T) \geq r$.*

For a Boolean function f, the rank of f, denoted $\text{Rank}(f)$, is the minimum rank of a decision tree computing f.

From Proposition 2.5(2), we see that the rank of a subfunction of f (a function obtained by assigning values to some variables of f) cannot exceed the rank of the function itself.

▶ **Proposition 2.6.** *(Rank of a subfunction): Let f_S be a subfunction obtained by fixing the values of variables in some set $S \subseteq [n]$ of f. Then $Rank(f_S) \leq Rank(f)$.*

The following rank and size relationship is known for boolean functions.

▶ **Proposition 2.7** (Lemma 1 [9]). *For a non-constant Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

$$Rank(f) \leq \log DTSize(f) \leq Rank(f) \log\left(\frac{en}{Rank(f)}\right).$$

It follows that $\text{Rank}(f) \in \Theta(\log \text{DTSize}(f))$ if and only if $\text{Rank}(f) = \Omega(n)$. However, even when $\text{Rank}(f) \in o(n)$, it characterizes $\log \text{DTSize}(f)$ upto a logarithmic factor, since for every f, $\text{Rank}(f) \in \Omega(\log \text{DTSize}(f)/\log n)$.

For symmetric functions, Rank is completely characterized in terms of Gap.

▶ **Proposition 2.8** (Lemma C.6 [1]). *For symmetric Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, $Rank(f) = n - Gap(f)$.*

▶ Remark 2.9. For (simple) deterministic possibly weighted decision trees, each of the measures DTSize, Depth, and Rank, is the same for a Boolean function f, its complement $\neg f$, and its dual f^d.

Relations and Search problems

A relation $R \subseteq X \times W$ is said to be X-complete, or just complete, if its projection to X equals X. That is, for every $x \in X$, there is a $w \in W$ with $(x, w) \in R$. For an X-complete relation R, where X is of the form $\{0,1\}^n$ for some n, the search problem SearchR is as follows: given an $x \in X$, find a $w \in W$ with $(x, w) \in R$. A decision tree for SearchR is defined exactly as for Boolean functions; the only diference is that leaves are labeled with elements of W, and we require that for each input x, if the unique leaf reached on x is labeled w, then $(x, w) \in R$. The rank of the relation, $\text{Rank}(R)$, is the minimum rank of a decision tree solving the SearchR problem.

A Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ naturally defines a complete relation R_f over $X = \{0,1\}^n$ and $W = \{0,1\}$, with $R_f = \{(x, f(x)) \mid x \in X\}$, and $\text{Rank}(f) = \text{Rank}(R_f)$.

3   Game Characterisation for Rank

In this section we observe that the rank of a Boolean function is characterised by the value of a Prover-Delayer game introduced by Pudlák and Impagliazzo in [23]. As mentioned in Section 1, the game was originally described for searchCNF problems on unsatsifiable clause sets. The appropriate analog for a Boolean function f, or its relation R_f, and even for arbitrary X-complete relations $R \subseteq X \times W$, is as follows:

The game is played by two players, the Prover and the Delayer, who construct a (partial) assignment ρ in rounds. Initially, ρ is empty. In each round, the Prover queries a variable x_i not set by ρ. The Delayer responds with a bit value 0 or 1 for x_i, or defers the choice to the Prover. In the later case, Prover can choose the value for the queried variable, and the Delayer scores one point. The game ends when there is a $w \in W$ such that for all x consistent with ρ, $(x, w) \in R$. (Thus, for a Boolean function f, the game ends when $f|_\rho$ is a constant function.) The value of the game, Value(R), is the maximum k such that the Delayer can always score at least k points, no matter how the Prover plays.

▶ **Theorem 3.1** (implied from [23, 18, 11]). *For any X-complete relation $R \subseteq X \times W$, where $X = \{0,1\}^n$, Rank(R) = Value(R). In particular, for a boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, Rank($f$) = Value($R_f$).*

The proof of the theorem follows from the next two lemmas.

▶ **Lemma 3.2** (implicit in [18]). *For an X-complete relation $R \subseteq \{0,1\}^n \times W$, in the Prover-Delayer game, the Prover has a strategy which restricts the Delayer's score to at most Rank(R) points.*

Proof. The Prover chooses a decision tree T for SearchR and starts querying variables starting from the root and working down the tree. If the Delayer responds with a 0 or a 1, the Prover descends into the left or right subtree respectively. If the Delayer defers the decision to Prover, then the Prover sets the variable to that value for which the corresponding subtree has smaller rank (breaking ties arbitrarily), and descends into that subtree.

We claim that such a "tree-based" strategy restricts the Delayer's score to Rank(T) points. The proof is by induction on Depth(T).
1. Base Case: Depth(T) = 0. This means that $\exists w \in W$, $X \times \{w\} \subseteq R$. Hence the game terminates with the empty assignment and the Delayer scores 0.
2. Induction Step: Depth(T) ≥ 1. Let x_i be the variable at the root node and T_0 and T_1 be the left and right subtree. The Prover queries the variable x_i. Note that for all b, Depth(T_b) $\leq$ Depth(T) − 1, and T_b is a decision tree for the search problem on $R_{i,b} \triangleq \{(x,w) \in R \mid x_i = b\} \subseteq X_{i,b} \times W$, where $X_{i,b} = \{x \in X \mid x_i = b\}$.
   If the Delayer responds with a bit b, then by induction, the subsequent score of the Delayer is limited to Rank(T_b) $\leq$ Rank(T). Since the current round does not increase the score, the overall Delayer score is limited to Rank(T).
   If the Delayer defers the decision to Prover, the Delayer gets one point in the current round. Subsequently, by induction, the Delayer's score is limited to min(Rank(T_0), Rank(T_1)); by definition of rank, this is at most Rank(T) − 1. So the overall Delayer score is again limited to Rank(T).

In particular, if the Prover chooses a rank-optimal tree T_R, then the Delayer's score is limited to Rank(T_R) = Rank(R) as claimed.   ◀

▶ **Lemma 3.3** (implicit in [11]). *For an X-complete relation $R \subseteq \{0,1\}^n \times W$, in the Prover-Delayer game, the Delayer has a strategy which always scores at least Rank(R) points.*

Proof. The Delayer strategy is as follows: When variable x_i is queried, the Delayer responds with $b \in \{0, 1\}$ if $\text{Rank}(R_{i,b}) > \text{Rank}(R_{i,1-b})$, and otherwise defers.

We show that the Delayer can always score $\text{Rank}(R)$ points using this strategy. The proof is by induction on the number of variables n. Note that if $\text{Rank}(R) = 0$, then there is nothing to prove. If $\text{Rank}(R) \geq 1$, then the prover must query at least one variable.

1. Base Case: $n = 1$. If $\text{Rank}(R) = 1$, then the prover must query the variable, and the Delayer strategy defers the choice, scoring one point.
2. Induction Step: $n > 1$. Let x_i be first variable queried by the prover.

   If $\text{Rank}(R_{i,0}) = \text{Rank}(R_{i,1})$, then the Delayer defers, scoring one point in this round. Subsequently, suppose the Prover sets x_i to b. The game is now played on $R_{i,b}$, and by induction, the Delayer can subsequently score at least $\text{Rank}(R_{i,b})$ points. But also, because of the equality, we have $\text{Rank}(R) \leq 1 + \text{Rank}(R_{i,b})$, as witnessed by a decision tree that first queries x_i and then uses rank-optimal trees on each branch. Hence the overall Delayer score is at least $\text{Rank}(R)$.

   If $\text{Rank}(R_{i,b}) > \text{Rank}(R_{i,1-b})$, then the Delayer chooses $x_i = b$ and the subsequent game is played on $R_{i,b}$. The subsequent (and hence overall) score is, by induction, at least $\text{Rank}(R_{i,b})$. But $\text{Rank}(R) \leq \text{Rank}(R_{i,b})$, as witnessed by a decision tree that first queries x_i and then uses rank-optimal trees on each branch.

◄

Lemmas 3.2 and 3.3 give us a way to prove rank upper and lower bounds for boolean functions. In a Prover-Delayer game for R_f, exhibiting a Prover strategy which restricts the Delayer to at most r points gives an upper bound of r on $\text{Rank}(f)$. Similarly, exhibiting a Delayer strategy which scores at least r points irrespective of the Prover strategy shows a lower bound of r on $\text{Rank}(f)$.

In [4], an aysmmmetric version of this game is defined. In each round, the Prover queries a variable x, the Delayer specifies values $p_0, p_1 \in [0, 1]$ adding up to 1, the Prover picks a value b, the Delayer adds $\log \frac{1}{p_b}$ to his score. Let ASym-Value denote the maximum score the Delayer can always achieve, independent of the Prover moves. Note that $\text{ASym-Value}(R) \geq \text{Value}(R)$; an asymmetric-game Delayer can mimic a symmetric-game Delayer by using $p_b = 1$ for choice b and $p_0 = p_1 = 1/2$ for deferring. As shown in [4], for the search CNF problem, the value of this asymmetric game is exactly the optimal leaf-size of a decision tree. We note below that this holds for the SearchR problem more generally.

▶ **Proposition 3.4** (implicit in [4]). *For any X-complete relation $R \subseteq X \times W$, where $X = \{0, 1\}^n$, $\log DTSize(R) = ASym\text{-}Value(R)$. In particular, for a boolean function $f : \{0, 1\}^n \longrightarrow \{0, 1\}$, $\log DTSize(f) = ASym\text{-}Value(R_f)$.*

(In [4], the bounds have $\log(S/2)$; this is because S there counts all nodes in the decision tree, while here we count only leaves.)

Thus we have the relationship

$$\text{Rank}(f) = \text{Value}(R_f) \leq \text{ASym-Value}(R_f) = \log \text{DTSize}(f).$$

This relationship explains where the slack may lie in the inequalities from Proposition 2.7 relating $\text{Rank}(f)$ and $\log \text{DTSize}(f)$. The symmetric game focusses on the more complex subtree, ignoring the contribution from the less complex subtree (unless both are equally complex), and thus characterizes rank. The asymmetric game takes a weighted contribution of both subtrees and thus is able to characterize size.

4   The Rank of some natural functions

For symmetric functions, rank can be easily calculated using Proposition 2.8. In Table 1 we tabulate various measures for some standard symmetric functions. As can be seen from the OR_n and AND_n functions, the $\text{Rank}(f)$ measure is not polynomially related with the measures $\text{Depth}(f)$ or certificate complexity $\text{C}(f)$.

f	Depth	C_0	C_1	C	Gap	Rank
0 or 1	0	0	0	0	n	0
AND_n	n	1	n	n	$n-1$	1
OR_n	n	n	1	n	$n-1$	1
PARITY_n	n	n	n	n	0	n
MAJ_{2k}	$2k$	k	$k+1$	$k+1$	k	k
MAJ_{2k+1}	$2k+1$	$k+1$	$k+1$	$k+1$	k	$k+1$
THR_n^k $(k \geq 1)$	n	$n-k+1$	k	$\max\left\{\begin{array}{l}n-k+1,\\ k\end{array}\right\}$	$\max\left\{\begin{array}{l}k-1,\\ n-k\end{array}\right\}$	$n - \text{Gap}$

Table 1 Some simple symmetric functions and their associated complexity measures

For two composed functions that will be crucial in our discussions in Section 5, we can directly calculate the rank as described below. (The rank can also be caluclated using Theorem 3.1; see Section 9, or using Theorem 6.6, which is much more general. We show these specific bounds here since we use them in Section 5.)

▶ **Theorem 4.1.** *For every $n \geq 1$,*
1. $Rank(\text{TRIBES}_{n,m}) = Rank(\text{TRIBES}_{n,m}^d) = n$ *for $m \geq 2$.*
2. $Rank(\text{AND}_n \circ \text{PARITY}_m) = n(m-1)+1$ *for $m \geq 1$.*

We prove this theorem by proving each of the lower and upper bounds separately in a series of lemmas below. The lemmas use the following property about the rank function.

▶ **Proposition 4.2.** *(Composition of Rank): Let T be a rooted binary tree with depth ≥ 1, rank r, and with leaves labelled by 0 and 1. Let T_0, T_1 be arbitrary rooted binary trees of ranks r_0, r_1 respectively. For $b \in \{0,1\}$, attach T_b to each leaf of T labeled b, to obtain rooted binary tree T' of rank r'.*
1. $r' \leq r + \max\{r_0, r_1\}$. *Furthermore, if T is a complete binary tree, and if $r_0 = r_1$, then this is an equality; $r' = r + r_0$.*
2. *If every non-trivial subtree (more than one leaf) of T has both a 0 leaf and a 1 leaf, then $r' \geq r + \max\{r_0, r_1\} - 1$. If, furthermore, T is a complete binary tree, then this is an equality when $r_0 \neq r_1$,*

Proof. The upper bound on r' follows from the definition of rank when $r_0 = r_1$, in which case it also gives equality for complete T. When $r_0 \neq r_1$, it follows from Proposition 2.5(2).

For non-trivially labeled T, we establish the lower bound by induction on $d = \text{Depth}(T)$.

In the base case $d = 1$, T has one 0-leaf and one 1-leaf, and $r = 1$. By definition of rank, r' satisfies the claimed inequality.

For the inductive step, let $\text{Depth}(T) = k > 1$. Let v be the root of T, and let T_ℓ, T_r be its left and right sub-trees respectively, with ranks r_ℓ and r_r respectively. Both $\text{Depth}(T_\ell)$

and $\text{Depth}(T_r)$ are at most $k-1$, and at least one of these is exactly $k-1 \geq 1$. Also, at least one of r_ℓ, r_r is non-zero.

Let T'_ℓ be the tree obtained by replacing 0 and 1 leaves of T_ℓ by T_0 and T_1 respectively; let its rank be r'_ℓ. Similarly construct T'_r, with rank r'_r. Then T' has root v with left and right subtrees T'_ℓ and T'_r.

If $r_\ell = 0$, then $r = r_r$ and $\text{Depth}(T_r) = k-1 \geq 1$. By the induction hypothesis, $r_r + \max\{r_0, r_1\} - 1 \leq r'_r$. Since $r' \geq r'_r$, the claimed bound follows.

If $r_r = 0$, a symmetric argument applies.

If both r_ℓ, r_r are positive, then by the induction hypothesis, $r_\ell + \max\{r_0, r_1\} - 1 \leq r'_\ell$ and $r_r + \max\{r_0, r_1\} - 1 \leq r'_r$. If $r_\ell = r_r$ then $r = r_\ell + 1$, and by definition of rank, $r' \geq 1 + \min\{r'_\ell, r'_r\} \geq r_\ell + \max\{r_0, r_1\} = r + \max\{r_0, r_1\} - 1$, as claimed. On the other hand, if $r_\ell \neq r_r$, then $r = \max\{r_\ell, r_r\}$, and by definition of rank, $r' \geq \max\{r'_\ell, r'_r\} \geq \max\{r_\ell, r_r\} + \max\{r_0, r_1\} - 1 = r + \max\{r_0, r_1\} - 1$, as claimed.

For complete binary tree T satisfying the labelling requirements, $r_\ell = r_r = r-1$. The same arguments, simplified to this situation, show the claimed equality: ◀

We first establish the bounds for $\text{TRIBES}_{n,m}^d = \bigwedge_{i \in [n]} \bigvee_{j \in [m]} x_{i,j}$.

▶ **Lemma 4.3.** *For every $n, m \geq 1$, $Rank(\text{TRIBES}_{n,m}^d) \leq n$.*

Proof. We show the bound by giving a recursive construction and bounding the rank by induction on n. In the base case, $n = 1$. $\text{TRIBES}_{1,m}^d = \text{OR}_m$, which has rank 1. For the inductive step, $n > 1$. For $j \leq n$, let $T_{j,m}$ denote the recursively constructed trees for $\text{TRIBES}_{j,m}^d$. Take the tree T which is $T_{1,m}$ on variables $x_{n,j}$, $j \in [m]$. Attach the tree $T_{n-1,m}$ on variables $x_{i,j}$ for $i \in [n-1]$, $j \in [m]$, to all the 1-leaves of T, to obtain $T_{n,m}$. It is straightforward to see that this tree computes $\text{TRIBES}_{n,m}^d$. Using Proposition 4.2 and induction, we obtain $\text{Rank}(T_{n,m}) \leq \text{Rank}(T_{1,m}) + \text{Rank}(T_{n-1,m}) \leq 1 + (n-1) = n$. ◀

▶ **Remark 4.4.** More generally, this construction shows that $\text{Rank}(\text{AND}_n \circ f) \leq n\text{Rank}(f)$.

▶ **Lemma 4.5.** *For every $n \geq 1$ and $m \geq 2$, $Rank(\text{TRIBES}_{n,m}^d) \geq n$.*

Proof. We prove this by induction on n. The base case, $n = 1$, is straightforward: $\text{TRIBES}_{1,m}^d$ is the function OR_m, whose rank is 1.

For the inductive step, let $n > 1$, and consider any decision tree Q for $\text{TRIBES}_{n,m}^d$. Without loss of generality (by renaming variables if necessary), let $x_{1,1}$ be the variable queried at the root node. Let Q_0 and Q_1 be the left and the right subtrees of Q. Then Q_0 computes the function $\text{AND}_n \circ (\text{OR}_{m-1}, \text{OR}_m, ..., \text{OR}_m)$, and Q_1 computes $\text{TRIBES}_{n-1,m}^d$, on appropriate variables. For $m \geq 2$, $\text{TRIBES}_{n-1,m}^d$ is a sub-function of $\text{AND}_n \circ (\text{OR}_{m-1}, \text{OR}_m, ..., \text{OR}_m)$, and so Proposition 2.6 implies that $\text{Rank}(Q_0) \geq \text{Rank}(\text{AND}_n \circ (\text{OR}_{m-1}, \text{OR}_m, ..., \text{OR}_m)) \geq \text{Rank}(\text{TRIBES}_{n-1,m}^d)$. By induction, $\text{Rank}(Q_1) \geq \text{Rank}(\text{TRIBES}_{n-1,m}^d) \geq n-1$. Hence, by definition of rank, $\text{Rank}(Q) \geq 1 + \min\{\text{Rank}(Q_0), \text{Rank}(Q_1)\} \geq n$. Since this holds for every decision tree Q for $\text{TRIBES}_{n,m}^d$, we conclude that $\text{Rank}(\text{TRIBES}_{n,m}^d) \geq n$, as claimed. ◀

Next, we establish the bounds for $\text{AND}_n \circ \text{PARITY}_m = \bigwedge_{i \in [n]} \bigoplus_{j \in [m]} x_{i,j}$. The upper bound below is slightly better than what is implied by Remark 4.4.

▶ **Lemma 4.6.** *For every $n, m \geq 1$, $Rank(\text{AND}_n \circ \text{PARITY}_m) \leq n(m-1) + 1$.*

Proof. Recursing on n, we construct decision trees $T_{n,m}$ for $\text{AND}_n \circ \text{PARITY}_m$, as in Lemma 4.3. By induction on n, we bound the rank, also additionally using the fact that the rank-optimal decision tree for PARITY_m is a complete binary tree.

Base Case: $n = 1$. $\text{AND}_1 \circ \text{PARITY}_m = \text{PARITY}_m$. From Table 1, $\text{Rank}(\text{PARITY}_m) = m$; let $T_{1,m}$ be the optimal decision tree computing PARITY_m.

Inductive Step: $n > 1$. For $j \leq n$, let $T_{j,m}$ denote the recursively constructed trees for $\text{AND}_j \circ \text{PARITY}_m$. Take the tree T which is $T_{1,m}$ on variables $x_{n,j}$, $j \in [m]$. Attach the tree $T_{n-1,m}$ on variables $x_{i,j}$ for $i \in [n-1]$, $j \in [m]$, to all the 1-leaves of T, to obtain $T_{n,m}$. It is straightforward to see that this tree computes $\text{AND}_n \circ \text{PARITY}_m$.

By induction, $\text{Rank}(T_{n-1,m}) \leq (n-1)(m-1) + 1 \geq 1$. Since we do not attach anything to the 0-leaves of $T_{1,m}$ (or equivalently, we attach a rank-0 tree to these leaves), and since $T_{1,m}$ is a complete binary tree, the second statement in Proposition 4.2 yields $\text{Rank}(T_{n,m}) = \text{Rank}(T_{1,m}) + \text{Rank}(T_{n-1,m}) - 1$. Hence $\text{Rank}(T_{n,m}) \leq n(m-1) + 1$, as claimed. ◄

▶ **Lemma 4.7.** *For every* $n, m_1, m_2, \ldots, m_n \geq 1$, *and functions* $g_1, g_2, \ldots, g_n$ *each in* $\{\text{PARITY}_m, \neg\text{PARITY}_m\}$, $\text{Rank}(\text{AND}_n \circ (g_1, g_2, ..., g_n)) \geq (\sum_{i=1}^{n}(m_i - 1)) + 1$.
In particular, $\text{Rank}(\text{AND}_n \circ \text{PARITY}_m) \geq n(m-1) + 1$.

Proof. We proceed by induction on n. Let h be the function $\text{AND}_n \circ (g_1, g_2, ..., g_n)$.

Base Case: $n = 1$. $h = g_1$. Note that for all functions f, $\text{Rank}(f) = \text{Rank}(\neg f)$. So $\text{Rank}(h) = \text{Rank}(\text{PARITY}_{m_1}) = m_1$. Inductive Step: $n > 1$. We proceed by induction on $M = \sum_{i=1}^{n} m_i$.

1. Base Case: $M = n$. Each m_i is equal to 1. So h is the conjunction of n literals on distinct variables. (A literal is a variable or its negation.) Hence $\text{Rank}(h) = \text{Rank}(\text{AND}_n) = 1$.

2. Inductive Step: $M > n > 1$. Consider any decision tree Q computing h. Without loss of generality (by renaming variables if necessary), let $x_{1,1}$ be the variable queried at the root node. Let Q_0 and Q_1 be the left and the right subtrees of Q. For $b \in \{0, 1\}$, let g_{1b} denote the function g_1 restricted to $x_{1,1} = b$. Then Q_b computes the function $\text{AND}_n \circ (g_{1b}, g_2, \ldots, g_n)$ on appropriate variables.

   If $m_1 = 1$, then the functions g_{10}, g_{11} are constant functions, one 0 and the other 1. So one of Q_0, Q_1 is a 0-leaf, and the other subtree computes $\text{AND}_{n-1} \circ (g_2, ..., g_n)$. Using induction on n, we conclude

$$\text{Rank}(Q) \geq \text{Rank}(\text{AND}_{n-1} \circ (g_2, ..., g_n)) \geq \left[\sum_{i=2}^{n}(m_i - 1)\right] + 1 = \left[\sum_{i=1}^{n}(m_i - 1)\right] + 1.$$

   For $m_1 \geq 2$, $\{g_{10}, g_{11}\} = \{\text{PARITY}_{m_1-1}, \neg\text{PARITY}_{m_1-1}\}$. So one of Q_0, Q_1 computes $\text{AND}_n \circ (\text{PARITY}_{m_1-1}, g_2, ..., g_n)$, and the other computes $\text{AND}_n \circ (\neg\text{PARITY}_{m_1-1}, g_2, ..., g_n)$. Using induction on M, we obtain

$$\text{Rank}(Q) \geq 1 + \min_b \text{Rank}(Q_b) \geq 1 + (m_1 - 2) + \left[\sum_{i=2}^{n}(m_i - 1)\right] + 1 = \left[\sum_{i=1}^{n}(m_i - 1)\right] + 1.$$

   Since this holds for every decision tree Q for h, the induction step is proved.
◄

5   Relation between Rank and other measures

5.1 Relating Rank to Depth and Sparsity

From Propositions 2.5 and 2.7, we know that $\mathrm{Depth}(f)$ is at least $\mathrm{Rank}(f)$. In the other direction, the AND function with rank 1 and depth n shows that $\mathrm{Depth}(f)$ cannot be bounded from above by any function of $\mathrm{Rank}(f)$ alone. Similarly, we know from Proposition 2.3 that $\mathrm{Depth}(f)$ is bounded from below by $\log \mathrm{spar}(f)/2$, and yet, as witnessed by the PARITY function with depth n and sparsity 1, it cannot be bounded from above by any function of $\log \mathrm{spar}(f)$ alone. We show in this section that a combination of these two measures does bound $\mathrm{Depth}(f)$ from above. Thus, in analogy to Propositions 2.3 and 2.7, we see where $\mathrm{Depth}(f)$ is sandwiched:

$$\max\{\mathrm{Rank}(f), \log \mathrm{spar}(f)/2\} \leq \mathrm{Depth}(f) \leq \mathrm{Rank}(f)(1 + \log \mathrm{spar}(f))$$

To establish the upper bound, we first observe that subcube kill number is bounded above by rank.

▶ **Lemma 5.1.** *For every Boolean function f, $K(f) \leq \max_{subcube\ J} K(f|J) \leq Rank(f)$.*

Proof. The first inequality holds since $f = f|J$ for the unique subcube J of codimension 0.

Next, note that showing $K(g) \leq \mathrm{Rank}(g)$ for every boolean function g suffices to prove the lemma. This is because if we prove this, then for every subcube J, $K(f|J) \leq \mathrm{Rank}(f|J) \leq \mathrm{Rank}(f)$; the latter inequality follows by monotonicity of rank (Proposition 2.6).

To show $K(g) \leq \mathrm{Rank}(g)$, observe the following property of rank, which follows from the definition: For every internal node v in a tree, at least one of its children has rank strictly less than the rank of v. Now, let T be a rank-optimal tree for g, of rank r. Starting from the root in T, traverse down the tree in the direction of smaller rank until a leaf is reached. Using the above property, we see that we reach a leaf node in T at depth at most r. The variables queried on the path leading to the leaf node, and their settings consistent with the path, give a subcube of co-dimension at most r. On this subcube, since a decision tree leaf is reached, g becomes constant, proving the claim. ◀

Combining the Lemma 5.1 and Theorem 2.4, we show the following.

▶ **Theorem 5.2.** *For every Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

$$Depth(f) \leq Rank(f)(1 + \log(spar(f))).$$

The inequality is tight as witnessed by PARITY *function.*

Proof. Recall that $\mathrm{sp\tilde{a}r}(f)$ refers to the number of non-zero Fourier coefficients in the expansion of f apart from $\hat{f}(\emptyset)$. We prove the theorem by induction on $\mathrm{sp\tilde{a}r}(f)$.

When $\mathrm{sp\tilde{a}r}(f) < 1$, f is a constant function with $\mathrm{Depth}(f) = \mathrm{Rank}(f) = 0$, and the inequality holds.

Now assume that $\mathrm{sp\tilde{a}r}(f) \geq 1$. We give a recursive construction of a decision tree for f.

Choose a subcube $J = (S, \rho)$ of minimum co-dimension $|S| = K(f)$ on which f becomes constant. By Lemma 5.1, $|S| \leq \mathrm{Rank}(f)$. Start by querying all the variables indexed in S in any order. When the outcome of all these queries matches ρ, the function becomes a constant and the tree terminates at this leaf. On any other outcome ρ', the function is restricted to the subcube $J' = (S, \rho')$, and by Theorem 2.4, $\mathrm{sp\tilde{a}r}(f|J') \leq \mathrm{sp\tilde{a}r}(f)/2$. Proceed recursively to build the decision tree of $f|J'$ which is then attached to this leaf.

Each stage in the recursion makes at most $\mathrm{Rank}(f)$ queries and halves sparsity $\mathrm{sp\tilde{a}r}$. After at most $1 + \log \mathrm{sp\tilde{a}r}(f)$ stages, the sparsity of the restricted function drops to below 1 and the function becomes a constant.

Thus the overall depth of the entire tree is bounded by $\mathrm{Rank}(f) \cdot (1 + \log \mathrm{sp\tilde{a}r}(f))$, which is at most $\mathrm{Rank}(f) \cdot (1 + \log \mathrm{spar}(f))$, as claimed. ◀

5.2 Relation between Rank and Certificate Complexity

The certificate complexity and decision tree complexity are known to be related as follows.

▶ **Proposition 5.3** ([6],[12],[25], see also Theorem 14.3 in [13]). *For every boolean function* $f : \{0,1\}^n \longrightarrow \{0,1\}$,

$$C(f) \leq Depth(f) \leq C_0(f)C_1(f)$$

Both these inequalities are tight; the first for the OR and AND functions, and the second for the $\text{TRIBES}_{n,m}$ and $\text{TRIBES}_{n,m}^d$ functions. (For $\text{TRIBES}_{n,m}^d$, $C_0(\text{TRIBES}_{n,m}^d) = m$, $C_1(\text{TRIBES}_{n,m}^d) = n$ and $Depth(\text{TRIBES}_{n,m}^d) = nm$, see e.g. Exercise 14.1 in [13].)

Since Rank $\leq$ Depth, the same upper bound also holds for Rank as well. But it is far from tight for the $\text{TRIBES}_{n,m}$ function. In fact, the upper bound can be improved in general. Adapting the construction given in the proof of Proposition 5.3 slightly, we show the following.

▶ **Lemma 5.4.** *For every Boolean function* $f : \{0,1\}^n \longrightarrow \{0,1\}$,

$$Rank(f) \leq (C_0(f) - 1)(C_1(f) - 1) + 1$$

Moreover, the inequality is tight as witnessed by AND *and* OR *functions.*

Proof. The inequality holds trivially for constant functions since for such functions, Rank $= C_0 = C_1 = 0$. So assume f is not constant. The proof is by induction on $C_1(f)$.

Base Case: $C_1(f) = 1$. Let $S \subseteq [n]$ be the set of indices that are 1-certificates for some $a \in f^{-1}(1)$. We construct a decision tree by querying all the variables indexed in S. For each such query, one outcome immediately leads to a 1-leaf (by definition of certificate), and we continue along the other outcome. If all variables indexed in S are queried without reaching a 1-leaf, the restricted function is 0 everywhere and so we create a 0-leaf. This gives a rank-1 decision tree computing f.

For the inductive step, assume $Rank(g) \leq (C_0(g) - 1)(C_1(g) - 1) + 1$ is true for all g with $C_1(g) \leq k$. Let f satisfy $C_1(f) = k + 1$. Pick an $a \in f^{-1}(0)$ and a mimimum-size 0-certificate S for a. Without loss of generality, assume that $S = \{x_1, x_2, \ldots, x_\ell\}$ for some $\ell = |S| \leq C_0(f)$. Now, take a complete decision tree T_0 of depth l on these l variables. Each of its leaves corresponds to the unique input $c = (c_1, c_2, \ldots, c_l) \in \{0,1\}^l$ reaching this leaf. At each such leaf, attach a minimal rank decision tree T_c for the subfunction $f_c \triangleq f(c_1, c_2, \ldots, c_l, x_{l+1}, \ldots, x_n)$. This gives a decision tree T for f. We now analyse its rank.

For at least one input c, we know that f_c is the constant function 0. For all leaves where f_c is not 0, $C_0(f_c) \leq C_0(f)$ since certificate size cannot increase by assigning some variables. Further, $C_1(f_c) \leq C_1(f) - 1$; this because of the well-known fact (see e.g. [13]) that every pair of a 0-certificate and a 1-certificate for f have at least one common variable, and T_0 has queried all variables from a 0-certificate. Hence, by induction, for each c with $f_c \neq 0$, $Rank(T_c) \leq (C_0(f_c) - 1)(k - 1) + 1 \leq (C_0(f) - 1)(k - 1) + 1$. Thus T is obtained from a rank-ℓ tree T_0 (with $\ell \leq C_0(f)$) by attaching a tree of rank 0 to at least one leaf, and attaching trees of rank at most $(C_0(f) - 1)(k - 1) + 1$ to all leaves. From Proposition 4.2, we conclude that $Rank(f) \leq Rank(T) \leq ((C_0(f) - 1)(k - 1) + 1) + (l - 1) \leq (C_0(f) - 1)(C_1(f) - 1) + 1$. ◀

From Theorem 4.1, we see that the lower bound on Depth in Proposition 5.3 does not hold for Rank; for $m > n$, $Rank(\text{TRIBES}_{n,m}^d) = n < m = C(\text{TRIBES}_{n,m}^d)$. However, $\min\{C_0(\text{TRIBES}_{n,m}^d), C_1(\text{TRIBES}_{n,m}^d)\} = n = Rank(\text{TRIBES}_{n,m}^d)$. Further, for all the functions listed in Table 1, $Rank(f)$ is at least as large as $\min\{C_0(f), C_1(f)\}$. However, even this is not a lower bound in general.

▶ **Lemma 5.5.** $\min\{C_0(f), C_1(f)\}$ *is not a lower bound on Rank(f); for the symmetric function* $f = \mathrm{MAJ}_n \vee \mathrm{PARITY}_n$, *when* $n > 4$, $Rank(f) < \min\{C_0(f), C_1(f)\}$.

Proof. Let f be the function $\mathrm{MAJ}_n \vee \mathrm{PARITY}_n$, for $n > 4$. Then $f(0^n) = 0$ and $C_0(f, 0^n) = n$, and $f(10^{n-1}) = 1$ and $C_1(f, 10^{n-1}) = n$. Also, f is symmetric, with $\mathrm{Gap}(f) = n/2$, so by Proposition 2.8, $\mathrm{Rank}(f) = n/2$. ◀

The average certificate complexity is also not directly related to rank.

▶ **Lemma 5.6.** *Average certificate complexity is neither a upper bound nor a lower bound on the rank of a function; there exist functions* f *and* g, *such that* $Rank(f) < C_{avg}(f)$ *and* $C_{avg}(g) < Rank(g)$.

Proof. Let f be the AND_n function for $n \geq 2$; we know that $\mathrm{Rank}(f) = 1$. Since the 1-certificate has length n and all minimal 0-certificates have length 0, the average certificate complexity of f is $C_{avg}(f) = 2^{-n}.n + (1 - 2^{-n}).1 = 1 + 2^{-n}(n-1)$.

Consider $g = \mathrm{TRIBES}_{n,2}^d$ for $n > 2$. By Theorem 4.1, $\mathrm{Rank}(g) = n$. Since $|g^{-1}(1)| = 3^n$ and each minimal 1-certificate has length n, and since $|g^{-1}(0)| = 4^n - 3^n$ and each minimal 0-certificate has length 2, we see that

$$C_{avg}(g) = \left(\frac{3}{4}\right)^n \cdot n + \left[1 - \left(\frac{3}{4}\right)^n\right] \cdot 2 < n = \mathrm{Rank}(g).$$

For a larger gap between Rank and C_{avg}, consider the function $h = \mathrm{AND}_n \circ \mathrm{PARITY}_n$. From Theorem 4.1, $\mathrm{Rank}(h) = n(n-1) + 1$. There are $2^{(n-1)n}$ 1-inputs, and all the 1-certificates have length n^2. Also, all minimal 0-certificates have length n. Hence $C_{avg}(h) = 2^{-n}n^2 + (1 - 2^{-n})n = n + o(1)$. ◀

What can be shown in terms of certificate complexity and rank is the following:

▶ **Lemma 5.7.** *For every Boolean function* f, $C_{\min}(f) \leq Rank(f)$. *This is tight for* OR_n.

Proof. Let T be a rank-optimal decision tree for f. Since the variables queried in any root-to-leaf path in T form a 0 or 1-certificate for f, we know that depth of each leaf in T must be at least $C_{\min}(f)$. By Proposition 2.5(3), $\mathrm{Rank}(f) = \mathrm{Rank}(T) \geq C_{\min}(f)$. ◀

Lemma 5.4 and Lemma 5.7 give these bounds sandwiching $\mathrm{Rank}(f)$:

▶ **Theorem 5.8.** $C_{\min}(f) \leq Rank(f) \leq (C_0(f) - 1)(C_1(f) - 1) + 1 \leq (C(f) - 1)^2 + 1$.

As mentioned in Proposition 2.8, for symmetric functions the rank is completely characterised in terms of Gap of f. How does Gap relate to certificate complexity for such functions? It turns out that certificate complexity is characterized not by Gap but by $\mathrm{Gap}_{\min}$. Using this relation, the upper bound on $\mathrm{Rank}(f)$ from Lemma 5.4 can be improved for symmetric functions to $C(f)$.

▶ **Lemma 5.9.** *For every symmetric Boolean function* f *on* n *variables,* $C(f) = n - Gap_{\min}(f)$ *and* $n - C(f) + 1 \leq Rank(f) \leq C(f)$. *Both the inequalities on rank are tight for* MAJ_{2k+1}.

Proof. We first show $C(f) = n - \mathrm{Gap}_{\min}(f)$. Consider any interval $[a, b]$ such that $f_{a-1} \neq f_a = f_{a+1} = ... = f_b \neq f_{b+1}$. Let x be any input with Hamming weight in the interval $[a, b]$. We show that $C(f, x) = n - (b - a)$.

1. Pick any $S \subseteq [n]$ containing exactly a bit positions where x is 1, and exactly $n - b$ bit positions where x is 0. Any y agreeing with x on S has Hamming weight in $[a, b]$, and hence $f(y) = f(x)$. Thus S is a certificate for x. Hence $C(f, x) \leq n - (b - a)$.

2. Let $S \subseteq [n]$ be any certificate for x. Suppose S contains fewer than a bit positions where x is 1. Then there is an input y that agrees with x on S and has Hamming weight exactly $a - 1$. (Flip some of the 1s from x that are not indexed in S.) So $f(y) \neq f(x)$, contradicting the fact that S is a certificate for x. Similarly, if S contains fewer that $n - b$ bit positions where x is 0, then there is an input z that agrees with x on S and has Hamming weight exactly $b + 1$. So $f(z) \neq f(x)$, contradicting the fact that S is a certificate for x.

Thus any certificate for x must have at least $a + (n - b)$ positions; hence $C(f, x) \geq n - (b - a)$. Since the argument above works for any interval $[a, b]$ where f is constant, we conclude that $C(f) = n - \text{Gap}_{\min}(f)$.

Next, observe that $\text{Gap}(f) + \text{Gap}_{\min}(f) \leq n - 1$. Hence,

$$n - C(f) + 1 = \text{Gap}_{\min}(f) + 1 \leq n - Gap(f) = \text{Rank}(f) \leq n - \text{Gap}_{\min}(f) = C(f).$$

As seen from Table 1, these bounds on Rank are tight for MAJ_{2k+1}.   ◄

Even for the (non-symmetric) functions in Theorem 4.1, $\text{Rank}(f) \leq C(f)$. However, this is not true in general.

▶ **Lemma 5.10.** *Certificate Complexity does not always bound Rank from above; for $k \geq 1$ and $n = 4^k$ the function $f = (\text{AND}_2 \circ \text{OR}_2)^{\otimes k}$ on n variables has $\text{Rank}(f) = \Omega(Cert(f)^2)$.*

This lemma shows that the relation between rank and certificate complexity (from Lemma 5.4) is optimal upto constant factors. The proof of the lemma is deferred to the end of Section 7.1, before which we develop techniques to bound the rank of composed functions.

6   Rank of Composed and Iterated Composed functions

In this section we study the rank for composed functions. For composed functions, $f \circ g$, decision tree complexity Depth is known to behave very nicely.

▶ **Proposition 6.1** ([21]). *For Boolean functions f, g, $Depth(f \circ g) = Depth(f)Depth(g)$.*

We want to explore how far something similar can be deduced about $\text{Rank}(f \circ g)$. The first thing to note is that a direct analogue in terms of Rank alone is ruled out.

▶ **Lemma 6.2.** *For general Boolean functions f and g, $\text{Rank}(f \circ g)$ cannot be bounded by any function of $\text{Rank}(f)$ and $\text{Rank}(g)$ alone.*

Proof. Let $f = \text{AND}_n$ and $g = \text{OR}_n$. Then $\text{Rank}(f) = \text{Rank}(g) = 1$. But $\text{Rank}(f \circ g) = \text{Rank}(\text{TRIBES}_n^d) = n$, as seen in Theorem 4.1.   ◄

For $f \circ g$, let T_f, T_g be decision trees for f, g respectively. One way to construct a decision tree for $f \circ g$ is to start with T_f, inflate each internal node u of T_f into a copy of T_g on the appropriate inputs, and attach the left and the right subtree of u as appropriate at the leaves of this copy of T_g. By Proposition 6.1, the decision tree thus obtained for $f \circ g$ is optimal for Depth if one start with depth-optimal trees T_f and T_g for f and g respectively. In terms of rank, we can also show that the rank of the decision tree so constructed is bounded above by $\text{Depth}(T_f)\text{Rank}(T_g) = \text{Depth}_w(f, [r, r, \ldots, r])$, where $r = \text{Rank}(T_g)$. (This is the construction used in the proofs of Lemmas 4.3 and 4.6, where further properties of the PARITY function are used to show that the resulting tree's rank is even smaller than $\text{Depth}(f)\text{Rank}(g)$.) In fact, we show below (Theorem 6.3) that this holds more generally,

when different functions are used in the composition. While this is a relatively straightforward generalisation here, it is necessary to consider such compositions for the lower bound we establish further on in this section.

▶ **Theorem 6.3.** *For non-constant boolean functions $g_1, \ldots, g_n$ with $Rank(g_i) = r_i$, and for n-variate non-constant booolean function f,*

$$Rank(f \circ (g_1, g_2, ..., g_n)) \leq Depth_w(f, [r_1, r_2, ..., r_n]).$$

Proof. Let h denote the function $f \circ (g_1, g_2, ..., g_n)$. For $i \in [n]$, let m_i be the arity of g_i. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m_i}$ the ith block of variables of h; g_i is evaluated on this block. Let T_f be any decision tree for f. For each $i \in [n]$, let T_{g_i} be a rank-optimal tree for g_i. Consider the following recursive construction of a decision tree T_h for h.

1. Base Case: $\text{Depth}(T_f) = 0$. Then f and h are the same constant function, so set $T_h = T_f$.
2. Recursion Step: $\text{Depth}(T_f) \geq 1$. Let x_i be the variable queried at the root node of T_f, and let T_0 and T_1 be the left and the right subtree of T_f, computing functions f_0, f_1 respectively. For notational convenience, we still view f_0, f_1 as functions on n variables, although they do not depend on their ith variable. Recursively construct, for $b \in \{0, 1\}$, the trees T'_b computing $f_b \circ (g_1, \ldots, g_{i-1}, b, g_{i+1}, \ldots, g_n)$ on the variables $x_{k,\ell}$ for $k \neq i$. Starting with the tree T_{g_i} on the ith block of variables, attach tree T'_b to each leaf labeled b to obtain the tree T_h.

From the construction, it is obvious that T_h is a decision tree for $f \circ (g_1, \ldots, g_n)$. It remains to analyse the rank of T_h. Proceeding by induction on $\text{Depth}(T_f)$, we show that $\text{Rank}(T_h) \leq D_w(T_f, [r_1, r_2, ..., r_n])$.

1. Base Case: $\text{Depth}(T_f) = 0$. Then $T_h = T_f$, so $\text{Rank}(T_h) = D_w(T_f, [r_1, r_2, ..., r_n]) = 0$.
2. Induction: $\text{Depth}(T_f) \geq 1$.

$$
\begin{aligned}
\text{Rank}(T_h) &\leq \text{Rank}(T_{g_i}) + \max\{\text{Rank}(T'_0), \text{Rank}(T'_1)\} \quad \text{(by Proposition 4.2)} \\
&= r_i + \max_{b \in \{0,1\}} \{\text{Rank}(T'_b)\} \\
&\leq r_i + \max_{b \in \{0,1\}} \{D_w(T_b, [r_1, r_2, ..., r_n])\} \quad \text{(by induction)} \\
&= D_w(T_f, [r_1, r_2, ..., r_n]) \quad \text{by definition of } D_w
\end{aligned}
$$

Picking T_f to be a tree for f that is optimal with respect to weights $[r_1, r_2, ..., r_n]$, we obtain $\text{Rank}(h) \leq \text{Rank}(T_h) \leq D_w(T_f, [r_1, r_2, ..., r_n]) = D_w(f, [r_1, r_2, ..., r_n])$. ◀

The really interesting question, however, is whether we can show a good lower bound for the rank of a composed function. This will help us understand how good is the upper bound in Theorem 6.3. To begin with, note that for non-constant Boolean functions f, g, both f and g are sub-functions of $f \circ g$. Hence Proposition 2.6 implies the following.

▶ **Proposition 6.4.** *For non-constant boolean functions f, g,*

$$Rank(f \circ g) \geq \max\{Rank(f), Rank(g)\}.$$

A better lower bound in terms of weighted depth complexity of f is given below. This generalises the lower bounds from Lemmas 4.5 and 4.7. The proofs of those lemmas crucially used nice symmetry properties of the inner function, whereas the bound below applies for any non-constant inner function. It is significantly weaker than the bound from Lemma 4.5 but matches that from Lemma 4.7.

▶ **Theorem 6.5.** *For non-constant boolean functions $g_1, \ldots, g_n$ with $Rank(g_i) = r_i$, and for n-variate non-constant boolean function f,*

$$Rank(f \circ (g_1, g_2, ..., g_n)) \geq Depth_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1$$
$$\geq Depth_w(f, [r_1, r_2, ..., r_n]) - (n - 1).$$

Proof. The second inequality above is straightforward: let T be a decision tree for f that is optimal with respect to weights $r_1 - 1, \ldots, r_n - 1$. Since T can be assumed to be reduced, repeated application of Fact 2.1 shows that the depth of T with respect to weights $r_1, \ldots, r_n$ increases by at most n. Thus $Depth_w(f, [r_1, \ldots, r_n]) \leq Depth_w(T, [r_1, \ldots, r_n]) \leq Depth_w(T, [r_1 - 1, \ldots, r_n - 1]) + n = Depth_w(f, [r_1 - 1, \ldots, r_n - 1]) + n$, giving the claimed inequality.

We now turn our attention to the first inequality, which is not so straightforward. We prove it by induction on n. Let h denote the function $f \circ (g_1, g_2, ..., g_n)$. For $i \in [n]$, let m_i be the arity of g_i. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m_i}$ the ith block of variables of h; g_i is evaluated on this block.

In the base case, $n = 1$. Since f is non-constant, f can either be x or $\neg x$; accordingly, h is either g_1 or $\neg g_1$. So $D_w(f, [r_1 - 1]) = r_1 - 1$ and $Rank(h) = Rank(g_1) = r_1$, and the inequality holds.

For the inductive step, when $n > 1$, we proceed by induction on $M = \sum_{i=1}^{n} m_i$. In the base case, $M = n$, and each m_i is equal to 1. Since all g_i's are non-constant, $r_i = 1$ for all i. So $D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 = D_w(f, [0, 0, ..., 0]) + 1 = 1$. Since all r_i's are 1, each g_i's is either $x_{i,1}$ or $\neg x_{i,1}$, Thus h is the same as f upto renaming of the literals. Hence $Rank(h) = Rank(f) \geq 1$.

For the inductive step, $M > n > 1$. Take a rank-optimal decision tree T_h for h. We want to show that $Depth_w(f, [r_1 - 1, \ldots, r_n - 1]) \leq Rank(T_h) - 1$. Without loss of generality, let $x_{1,1}$ be the variable queried at the root. Let T_0 and T_1 be the left and the right subtree of T_h. For $b \in \{0, 1\}$, let g_1^b be the subfunction of g_1 when $x_{1,1}$ is set to b. Note that T_b computes $h_b \triangleq f \circ (g_1^b, g_2, ..., g_n)$, a function on $M - 1$ variables. We would like to use induction to deduce information about $Rank(T_b)$. However, g_1^b may be a constant function, and then induction does not apply. So we do a case analysis on whether or not g_1^0 and g_1^1 are constant functions; this case analysis is lengthy and tedious but most cases are straightforward.

- Case 1: Both g_1^0 and g_1^1 are constant functions. Since g_1 is non-constant, $g_1^0 \neq g_1^1$, and $r_1 = Rank(g_1) = 1$. Assume that $g_1^0 = 0$ and $g_1^1 = 1$; the argument for the other case is identical. For $b \in \{0, 1\}$, let f_b be the function $f(b, x_2, \ldots, x_n)$; then $h_b = f_b \circ (g_2, \ldots, g_n)$. View f_b as functions on $n - 1$ variables.

  - Case 1a: Both f_0 and f_1 are constant functions. Then f is either x_1 or $\neg x_1$, so $Depth_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) = Depth_w(f, [0, r_2 - 1, ..., r_n - 1]) = 0$. Also, in this case, h is either $x_{1,1}$ or $\neg x_{1,1}$, so $Rank(h) = 1$. Hence the inequality holds.

  - Case 1b: Exactly one of f_0 and f_1 is a constant function; without loss of generality, let f_0 be a constant function. First, observe that for any weights $w_2, \ldots, w_n$, $D_w(f, [0, w_2, ..., w_n]) \leq D_w(f_1, [w_2, ..., w_n])$: we can obtain a decision tree for f witnessing this by first querying x_1, making the $x_1 = 0$ child a leaf labeled f_0, and attaching the optimal tree for f_1 on the $x_1 = 1$ branch. Second, note that since f_1 and

all g_i are non-constant, so is h_1. Now

$$\begin{aligned}
\text{Rank}(h) = \text{Rank}(h_1) && \text{since Rank}(h_0) = 0 \\
\geq D_w(f_1, [r_2 - 1, ..., r_n - 1]) + 1 && \text{by induction hypothesis on } n \\
\geq D_w(f, [0, r_2 - 1, ..., r_n - 1]) + 1 && \text{by first observation above} \\
= D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 && \text{since } r_1 = 1
\end{aligned}$$

- Case 1c: Both f_0 and f_1 are non-constant functions.

$$\begin{aligned}
\text{Rank}(h) \geq \max(\text{Rank}(h_0), \text{Rank}(h_1)) && \\
\geq \max_{b \in \{0,1\}} \{D_w(f_b, [r_2 - 1, ..., r_n - 1])\} + 1 && \text{by induction hypothesis on } n \\
\geq D_w(f, [0, r_2 - 1, ..., r_n - 1]) + 1 && \text{by def. of weighted depth} \\
&& \text{of a tree querying } x_1 \text{ first} \\
= D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 && \text{since } r_1 = 1
\end{aligned}$$

- Case 2: One of g_1^0 and g_1^1 is a constant function; assume without loss of generality that g_1^0 be constant. In this case, we can conclude that $\text{Rank}(g_1) = \text{Rank}(g_1^1)$: $\text{Rank}(g_1^1) \leq \text{Rank}(g_1)$ by Proposition 2.6, and $\text{Rank}(g_1) \leq \text{Rank}(g_1^1)$ as witnessed by a decision tree for g_1 that queries $x_{1,1}$ first, sets the $x_{1,1} = 0$ branch to a leaf labeled g_1^0, and attaches an optimal tree for g_1^1 on the other branch. Now

$$\begin{aligned}
\text{Rank}(h) \geq \text{Rank}(h_1) && \\
\geq D_w(f, [\text{Rank}(g_1^1) - 1, r_2 - 1, ..., r_n - 1]) + 1 && \text{by induction on } M \\
= D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 && \text{since Rank}(g_1^1) = \text{Rank}(g_1)
\end{aligned}$$

- Case 3: Both g_1^0 and g_1^1 are non-constant functions. Let $r_1^b = \text{Rank}(g_1^b) \geq 1$. A decision tree for g_1 that queries $x_{1,1}$ first and then uses optimal trees for g_1^0 and g_1^1 has rank $R \geq r_1$ and witnesses that $1 + \max\{r_1^0, r_1^1\} \geq R \geq r_1$. (Note that R may be more than r_1, since a rank-optimal tree for g_1 may not query $x_{1,1}$ first.)

  - Case 3a: $\max_b\{r_1^b\} = r_1 - 1$. Then $R = 1 + \max\{r_1^0, r_1^1\}$, which can only happen if $r_1^0 = r_1^1$, and hence $r_1^0 = r_1^1 = r_1 - 1$. We can further conclude that $r_1 \geq 2$. Indeed, if $r_1 = 1$, then $r_1 - 1 = r_1^0 = r_1^1 = 0$, contradicting the fact that we are in Case 3. For $b \in \{0, 1\}$,

$$\begin{aligned}
\text{Rank}(h_b) = \text{Rank}(f \circ (g_1^b, g_2, \ldots, g_n)) && \\
\geq \text{Depth}_w(f, [r_1^b - 1, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{by induction on } M \\
= \text{Depth}_w(f, [r_1 - 2, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{since } r_1 - 1 = r_1^b.
\end{aligned}$$

$$\begin{aligned}
\text{Hence Rank}(h) \geq 1 + \min_b \text{Rank}(h_b) && \\
\geq \text{Depth}_w(f, [r_1 - 2, r_2 - 1, \ldots, r_n - 1]) + 2 && \text{derivation above} \\
\geq \text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{by Fact 2.1}
\end{aligned}$$

  - Case 3b: $\max_b\{r_1^b\} > r_1 - 1$. So $\max_b\{r_1^b\} \geq r_1$.

$$\begin{aligned}
\text{Rank}(h) \geq \max_b \text{Rank}(h_b) && \\
\geq \max_b \text{Depth}_w(f, [r_1^b - 1, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{by induction on } M \\
\geq \text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{since } \max_b\{r_1^b\} \geq r_1
\end{aligned}$$

This completes the inductive step for $M > n > 1$ and completes the entire proof.   ◀

From Theorems 4.1, 6.3, and 6.5, we obtain the following:

▶ **Theorem 6.6.** *For non-constant boolean functions f, g,*

$$Depth(f)(Rank(g) - 1) + 1 \leq Rank(f \circ g) \leq Depth(f)Rank(g).$$

Both inequalities are tight; the first for $\text{AND}_n \circ \text{PARITY}_m$ and the second for TRIBES_n and TRIBES_n^d.

It is worth noting that in the above bounds, the role of Rank and Depth cannot be exchanged. With $f = \text{AND}_n$ and $g = \text{PARITY}_n$, $\text{Rank}(f)\text{Depth}(g) = n < n(n-1)+1 \leq \text{Rank}(f \circ g)$, and $\text{Rank}(g \circ f) \leq n < n(n-1) + 1 = \text{Rank}(g)(\text{Depth}(f) - 1) + 1 = \text{Depth}(f)(\text{Rank}(g) - 1) + 1$.

Since any non-constant symmetric function is evasive (Proposition 2.2), from Theorems 6.3 and 6.5, we obtain the following:

▶ **Corollary 6.7.** *For non-constant boolean functions $g_1, \ldots, g_n$ with $Rank(g_i) = r_i$, and for n-variate symmetric non-constant boolean function f,*

$$\sum_i r_i - (n-1) \leq Rank(f \circ (g_1, g_2, ..., g_n)) \leq \sum_i r_i.$$

For iterated composed functions, we obtain the following corollary.

▶ **Corollary 6.8.** *For $k \geq 1$ and non-constant boolean functions f,*

$$Depth(f)^{k-1}(Rank(f) - 1) + 1 \leq Rank(f^{\otimes k}) \leq Depth(f)^{k-1}Rank(f).$$

Proof. The result follows from Theorem 6.6 applying induction on k. The base case, k=1, is straightforward. For the induction step, $k > 1$, applying the recursive definition of iterated composed functions, we have

$$\begin{aligned}
\text{Rank}(f^{\otimes k}) &= \text{Rank}(f \circ f^{\otimes(k-1)}) \\
&\geq \text{Depth}(f)(\text{Rank}(f^{\otimes(k-1)}) - 1) + 1 \quad \text{by Theorem 6.6} \\
&\geq \text{Depth}(f)(\text{Depth}(f)^{k-2}(\text{Rank}(f) - 1)) + 1 \quad \text{by induction on } k \\
&= \text{Depth}(f)^{k-1}(\text{Rank}(f) - 1) + 1. \\
\text{Rank}(f^{\otimes k}) &= \text{Rank}(f \circ f^{\otimes(k-1)}) \\
&\leq \text{Depth}(f)\text{Rank}(f^{\otimes(k-1)}) \quad \text{by Theorem 6.6} \\
&\leq \text{Depth}(f)(\text{Depth}(f)^{k-2}(\text{Rank}(f)) \quad \text{by induction on } k \\
&= \text{Depth}(f)^{k-1}\text{Rank}(f).
\end{aligned}$$

◀

7   Applications

In this section, we give some applications of our results and methods. We first show how to obtain tight lower bounds on $\log \text{DTSize}$ for composed functions using the rank lower bound from Theorem 6.6. Next, we relate rank to query complexity in more general decision trees, namely CONJ decision trees, and show that rank (for ordinary decision trees) characterizes query complexity in this model up to $\log n$ factors.

7.1   Tight lower bounds for $\log \mathrm{DTSize}$ for Composed functions

It was shown in [9] that every boolean function f in n variables has a decision tree of size at most $\exp(O(\log n \log^2 N(f))$, where $N(f)$ is the total number of monomials in the minimal DNF for f and $\neg f$. Later, in [14], this relation was proved to be optimal up to $\log n$ factor. To prove this, the authors of [14] showed that iterated $\mathrm{AND}_2 \circ \mathrm{OR}_2$ and iterated MAJ_3 on $n = 4^k$ and $n = 3^k$ variables require decision trees of size $\exp(\Omega(\log^{\log_2 3} N))$ and $\exp(\Omega(\log^2 N))$ respectively. It is easy to show that $N((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k})$ and $N(\mathrm{MAJ}_3^{\otimes k})$ is $\exp(O(n^{1/\log_2 3}))$ and $exp(O(n^{1/2}))$ respectively. So showing optimality essentially boiled down to showing that the decision tree size of iterated $\mathrm{AND}_2 \circ \mathrm{OR}_2$ and iterated MAJ_3 on n variables is exponential $\exp(\Omega(n))$. This was established in [14] using spectral methods. We recover these size lower bounds using our rank lower bound for composed functions.

▶ **Corollary 7.1.** *For $k \geq 1$ and $n = 3^k$,*

$$\log DTSize(\mathrm{MAJ}_3^{\otimes k}) \geq Rank(\mathrm{MAJ}_3^{\otimes k}) \geq n/3 + 1.$$

For $k \geq 1$ and $n = 4^k$,

$$\log DTSize((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) \geq Rank((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) \geq n/4 + 1.$$

Proof. The MAJ_3 function has depth 3 and rank 2. Applying Corollary 6.8, we see that $\mathrm{Rank}(\mathrm{MAJ}_3^{\otimes k}) \geq 3^{k-1} + 1 = n/3 + 1$. Since rank is a lower bound on $\log \mathrm{DTSize}$ (Proposition 2.7), we get the desired size lower bound for iterated MAJ_3.

The $\mathrm{AND}_2 \circ \mathrm{OR}_2$ function has depth 4 and rank 2. Again applying Corollary 6.8, we get $\log \mathrm{DTSize}((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) \geq \mathrm{Rank}((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) \geq 4^{k-1} + 1 = n/4 + 1$, giving the size lower bound for iterated $\mathrm{AND}_2 \circ \mathrm{OR}_2$. ◀

The size lower bound for iterated $\mathrm{AND}_2 \circ \mathrm{OR}_2$ on n variables from [14], in conjunction with the rank-size relation from Proposition 2.7, implies that the rank of the iterated function is $\Omega(n)$. Corollary 7.1 demonstrates that these tight rank and size lower bounds can be recovered simultaneously Corollary 6.8.

Recently (after the preliminary version of our work appeared), in [7], the rank of the iterated $\mathrm{AND}_2 \circ \mathrm{OR}_2$ function on n variables was revisited, in the context of separating rank from randomised rank. Using the Prover-Delayer game-based characterisation of rank from Theorem 3.1, it was shown there that this function has rank exactly $(n + 2)/3$. While an $\Omega(n)$ bound is now easy to obtain as in Corollary 7.1, getting the exact constants required significantly more work.

The arguments given in [14] and [7] are tailored to the specific functions being considered, and do not work in general. On the other hand, our rank lower bound from Theorem 6.6 implies size lower bounds for composed functions in general. For completeness, we state our rank lower bound of Theorem 6.6 in terms of size.

▶ **Corollary 7.2.** *For $k \geq 1$ and non-constant boolean functions f and g,*

$$\log DTSize(f \circ g) \geq Depth(f)(Rank(g) - 1) + 1.$$

$$\log DTSize(f^{\otimes k}) \geq Depth(f)^{k-1}(Rank(f) - 1) + 1.$$

Using Corollary 7.1, we can now complete the proof of Lemma 5.10.

Proof. (of Lemma 5.10) From Corollary 7.1, we know that $\mathrm{Rank}((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) \geq n/4 + 1$.

It is easy to see that $\mathrm{C}_0((\mathrm{AND}_2 \circ \mathrm{OR}_2)) = \mathrm{C}_1((\mathrm{AND}_2 \circ \mathrm{OR}_2)) = 2$, and that for $k > 1$, $\mathrm{C}_0((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) = 2\mathrm{C}_0((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k-1})$ and $\mathrm{C}_1((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) = 2\mathrm{C}_1((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k-1})$. Thus $\mathrm{C}_0((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) = \mathrm{C}_1((\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}) = 2^k = \sqrt{n}$.

Hence, for $f = (\mathrm{AND}_2 \circ \mathrm{OR}_2)^{\otimes k}$, $\mathrm{Rank}(f) \geq \frac{\mathrm{C}_0}{2} \frac{\mathrm{C}_1}{2} + 1 = \mathrm{C}(f)^2/4 + 1$. ◀

7.2   Conj **decision trees**

In this section, we consider a generalization of the ordinary decision tree model, namely Conj decision trees. In the Conj decision tree model, each query is a conjunction of literals, where a literal is a variable or its negation. (In [15], such a tree where each conjunction involves at most k literals is called a k-decision tree; thus in that notation these are n-decision trees. A 1-decision tree is a simple decision tree.) A model essentially equivalent to Conj decision trees, (And, Or)-decision trees, was investigated in [2] for determining the complexity of Thr_n^k functions. (And, Or)-decision trees query either an And of a subset of variables or an Or of a subset of variables. It was noted in [2] that the (And, Or) query model is related to the computation using Ethernet channels, and a tight query lower bound was shown in this model for Thr_n^k functions for all $k \geq 1$. It is easy to see that the (And, Or)-query model is equivalent to the Conj query model upto a factor of 2. Let $Depth_{\bar{\wedge}}$ and $Depth_{\wedge,\vee}(f)$ denote the query complexity of function f in Conj and (And, Or) query model respectively. Then

▶ **Proposition 7.3.** *For every boolean functions f,*

$$Depth_{\bar{\wedge}}(f) \leq Depth_{\wedge,\vee}(f) \leq 2Depth_{\bar{\wedge}}(f).$$

Such a connection is not obvious for the rank of Conj and (And, Or) decision trees.

     Recently, in [17], a monotone version of Conj decision trees called And decision trees is studied, where queries are restricted to And of variables, not literals. To emphasize the difference, we refer to these trees as monotone And trees. Understanding monotone And decision trees in [17] led to the resolution of the log-rank conjecture for the class of And functions (any function composed with the 2-bit And function), up to a $\log n$ factor. As remarked in [16], understanding these more general decision tree models has shed new light on central topics in communication complexity, including restricted cases of the log-rank conjecture and the query-to-communication lifting methodology.

     In this section, we show that simple decision tree rank characterizes the query complexity in the Conj decision tree model, up to a $\log n$ factor. Formally,

▶ **Theorem 7.4.** *For every boolean functions f,*

$$Rank(f) \leq Depth_{\bar{\wedge}}(f) \leq \log DTSize(f) \leq Rank(f) \log \left(\frac{en}{Rank(f)} \right).$$

Consequently, if $Rank(f) = \Theta(n)$, then so is $Depth_{\bar{\wedge}}(f)$.

Proof. First, we show that $Rank(f) \leq Depth_{\bar{\wedge}}(f)$. This is the straightforward construction and its analysis. Let T_f be a depth-optimal Conj decision tree for f of depth d. We give a recursive construction of an ordinary decision tree T for f of rank at most d. In the base case, when $Depth(T_f) = 0$, set $T = T_f$. In the recursion Step, $Depth(T_f) \geq 1$. Let Q be the literal-conjunction queried at the root node of T_f, and let T_0 and T_1 be the left and right subtree of T_f, computing f_0 and f_1 respectively. Recursively construct ordinary decision trees T_0' and T_1' for f_0 and f_1. Let T_Q be the ordinary decision tree obtained by querying the variables in Q one by one to evaluate the query Q. Note that T_Q evaluates the And function on literals in Q; hence it has rank 1, and has exactly one leaf labelled 1 . Attach T_0' to each leaf labelled 0 in T_Q, and and T_1' to the unique leaf labelled 1 in T_Q, to obtain T.

     From the construction, it is clear that T evaluates f. To analyse the rank of T, proceed by induction on $Depth(T_f)$.

1. Base Case: $Depth(T_f) = 0$. Trivially true as $T = T_f$ with rank 0.

2. Induction: $\text{Depth}(T_f) \geq 1$.

$$\text{Rank}(T) \leq \text{Rank}(T_Q) + \max\{\text{Rank}(T'_0), \text{Rank}(T'_1)\} \quad \text{(by Proposition 4.2)}$$
$$= 1 + \max_{b \in \{0,1\}} \{\text{Rank}(T'_b)\}$$
$$\leq 1 + \max_{b \in \{0,1\}} \{\text{Depth}_{\bar{\wedge}}(f_b)\} \quad \text{(by induction)}$$
$$\leq 1 + (\text{Depth}(T_f) - 1) = \text{Depth}(T_f).$$

Next, we show that $\text{Depth}_{\bar{\wedge}}(f) \leq \log \text{DTSize}(f)$. The main idea is that an ordinary decision tree of size s can be balanced using Conj queries into a Conj decision tree of depth $O(\log s)$.

Let T_f be a size-optimal simple tree for f of size s. Associate with each node v of T_f a subcube J_v containing all the inputs that reaches node v. The root node has the whole subcube $\{0,1\}^n$. For a node v, J_v is defined by the variables queried on the path leading to the node v. The recursive construction of a Conj decision tree T of depth at most $2 \log_{3/2} s$ proceeds as follows. If $s = 1$, set $T = T_f$. Otherwise, in the recursion step, $s > 1$. Obtain a node v in T_f such that number of leaves in the subtree rooted at v, denoted by T_v, in the range $[s/3, 2s/3]$. (Such a node necessarily exists, and can be found by starting at the root and traversing down to the child node with more leaves until the number of leaves in the subtree rooted at the current node satisfies the condition.) Let $J_v = (S, \rho)$ be the subcube associated with v, and let Q be the Conj query testing membership in J_v; $Q = (\bigwedge_{i \in S : \rho(i)=1} x_i)(\bigwedge_{i \in S : \rho(i)=0} \neg x_i)$.

Note that v cannot be the root node of T. Let u be the sibling of v in T_f, and let T_u be the subtree rooted at u. Let w be the parent of v and w in T_f. Let T'_f be the tree obtained from T_f by removing the entire subtree T_v, removing the query at w, and attaching subtree T_u at w. For all inputs not in J_v, T'_f and T_f compute the same value.

T starts by querying Q. If Q evaluates to 1, proceed by recursively constructing the Conj decision tree for T_v. If Q evaluates to 0, proceed by recursively constructing the Conj decision tree for T'_f.

The correctness of T is obvious. It remains to estimate the depth of T. Let $D(s)$ be the number of queries made by the constructed Conj decision tree. By construction, we have $D(s) \leq 1 + D(2s/3)$ giving us $D(s) = 2 \log_{3/2} s$, thereby proving our claim.

The last inequality about size and rank, $\log \text{DTSize}(f) \leq \text{Rank}(f) \log \left(\frac{en}{\text{Rank}(f)} \right)$, comes from Proposition 2.7. ◄

8 Tightness of Rank and Size relation for Tribes

In Proposition 2.7, we saw a relation between rank and size. The relationship is essentially tight. As remarked there, whenever $\text{Rank}(f) = \Omega(n)$, the relation is tight. The function $f = \text{Parity}_n$ is one such function that witnesses the tightness of both the inequalities. Since $\text{Rank}(\text{Parity}) = n$, Proposition 2.7 tells us that $\log \text{DTSize}(\text{Parity})$ lies in the range $[n, n \log e]$, and we know that $\log \text{DTSize}(\text{Parity}) = n$.

For the Tribes_n function, which has $N = n^2$ variables, we know from Theorem 4.1 that $\text{Rank}(\text{Tribes}_n) = n \in o(N)$. Thus Proposition 2.7 tells us that $\log \text{DTSize}(\text{Tribes}_n)$ lies in the range $[n, n \log(en)]$. (See also Exercise 14.9 [13] for a direct argument showing $n \leq \log \text{DTSize}(\text{Tribes}_n)$). But that still leaves a $(\log(en))$-factor gap between the two quantities. We show that the true value is closer to the upper end. To do this, we establish a stronger size lower bound for decision trees computing Tribes_n^d.

▶ **Lemma 8.1.** *For every $n, m \geq 1$, every decision tree for $\text{TRIBES}_{n,m}^d$ has at least m^n 1-leaves and n 0-leaves.*

Proof. Recall that $\text{TRIBES}_{n,m}^d = \bigwedge_{i \in [n]} \bigvee_{j \in [m]} x_{i,j}$. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$ the ith block of variables. We consider two special kinds of input assignments: 1-inputs of minimum Hamming weight, call this set S_1, and 0-inputs of maximum Hamming weight, call this set S_0. Each $a \in S_1$ has exactly one 1 in each block; hence $|S_1| = m^n$. Each $b \in S_0$ has exactly m zeroes, all in a single block; hence $|S_0| = n$. We show that in any decision tree T for $\text{TRIBES}_{n,m}^d$, all the inputs in $S = S_1 \cup S_0$ go to pairwise distinct leaves. Since all inputs in S_1 must go to 1-leaves of T, and all inputs of S_0 must go to 0-leaves, this will prove the claimed statement.

Let a, b be distinct inputs in S_1. Then there is some block $i \in [n]$, where they differ. In particular there is a unique $j \in [m]$ where $a_{i,j} = 1$, and at this position, $b_{i,j} = 0$. The decision tree T must query variable $x_{i,j}$ on the path followed by a, since otherwise it will reach the same 1-leaf on input a' that differs from a at only this position, contradicting the fact that $\text{TRIBES}_{n,m}^d(a') = 0$. Since $b_{i,j} = 0$, the path followed in T along b will diverge from a at this query, if it has not already diverged before that. So a, b reach different 1-leaves.

Let a, b be distinct inputs in S_0. Let i be the unique block where a has all zeroes; b has all 1s in this block. On the path followed by a, T must query all variables from this block, since otherwise it will reach the same 0-leaf on input a'' that differs from a only at an unqueried position in block i, contradicting $\text{TRIBES}_{n,m}^d(a'') = 1$. Since a and b differ everywhere on this block, b does not follow the same path as a, so they go to different leaves of T.   ◀

We thus conclude that the second inequality in Proposition 2.7 is also asymptotically tight for the TRIBES_n^d function.

The size lower bound from Lemma 8.1 can also be obtained by specifying a good Delayer strategy in the asymmetric Prover-Delayer game and invoking Proposition 3.4.; see Section 9.

9   Proofs using Prover-Delayer Games

In this section we give Prover-Delayer Game based proofs of our results.

Prover strategy for $\text{TRIBES}_{n,m}$, proving Lemma 4.3

We give a Prover strategy which restricts the Delayer to n points, proving the upper bound on $\text{Rank}(\text{TRIBES}_{n,m})$.

Whenever the Delayer defers a decision, the Prover chooses 1 for the queried variable.

The Prover queries variables $x_{i,j}$ in row-major order. In each row of variables, the Prover queries variables until some variable is set to 1 (either by the Delayer or by the Prover). Once a variable is set to 1, the Prover moves to the next row of variables.

This Prover strategy allows the Delayer to defer a decision for at most one variable per row; hence the Delayer's score at the end is at most n.

Delayer strategy for $\text{TRIBES}_{n,m}$, proving Lemma 4.5

We give a Delayer strategy which always score at least n points, proving the lower bound.

On a query $x_{i,j}$, the Delayer defers the decision to the Prover unless all other variables in row i have already been queried. In that case Delayer responds with a 1.

Note that with this strategy, the Delayer ensures that the game ends with function value 1. (No row has all variables set to 0.) Observe that to certify a 1-input of the function, the

Prover must query at least one variable in each row. Since $m \geq 2$, the Delayer gets to score at least one point per row, and thus has a score of at least n at the end of the game.

Prover strategy for $\text{AND}_n \circ \text{PARITY}_m$, proving Lemma 4.6

We give a Prover strategy which restricts Delayer to $n(m-1)+1$ points. The Prover queries variables in row-major order. If on query $x_{i,j}$ the Delayer defers a decision to the Prover, the Prover chooses arbitrarily unless $j = m$. If $j = m$, then the Prover chooses a value which makes the parity of the variables in row i evaluate to 0.

Let j be the first row such that the Delayer defers the decision on $x_{j,m}$ to the Prover. (If there is no such row, set $j = n$.) With the strategy above, the Prover will set $x_{j,m}$ in such a way that the parity of the variables in j-th row evaluates to 0, making f evaluate to 0 and ending the game. The Delayer scores at most $m - 1$ points per row for rows before this row j, and at most m points in row j. Hence the Delayer's score is at most $(j-1)(m-1)+m$ points. Since $j \leq n$, the Delayer is restricted to $n(m-1)+1$ points at the end of the game.

Delayer strategy for $\text{AND}_n \circ \text{PARITY}_m$, proving Lemma 4.7

We give a Delayer strategy which always scores at least $n(m-1)+1$ points.

On query $x_{i,j}$, if this is the last un-queried variable, or if there is some un-queried variable in the same i-th row, the Delayer defers the decision to the Prover. Otherwise the Delayer responds with a value that makes the parity of the variables in row i evaluate to 1.

This strategy forces the Prover to query all variables to decide the function. The Delayer picks up $m - 1$ points per row, and an additional point on the last query, giving a total score of $n(m-1)+1$ points.

Prover strategy in terms of certificate complexity, proving Lemma 5.4

We give a Prover strategy which restricts the Delayer to $(C_0(f) - 1)(C_1(f) - 1) + 1$ points. Let $\tilde{f}$ be the function obtained by assigning values to the variables queried so far. As long as $C_1(\tilde{f}) > 1$, Prover picks an $a \in \tilde{f}^{-1}(0)$ and its 0-certificate S, and queries all the variables in S one by one. If at any point the Delayer defers a decision to the Prover, the Prover chooses the value according to a. When $C_1(\tilde{f})$ becomes 1, the Prover picks an $a \in \tilde{f}^{-1}(1)$ and its 1-certificate $\{i\}$ and queries the variable x_i. If the Delayer defers the decision, the Prover chooses a_i.

The above strategy restricts the Delayer to $(C_0(f) - 1)(C_1(f) - 1) + 1$ points; the proof is essentially same as Lemma 5.4.

Prover and Delayer strategies for composed functions, proving Theorem 6.6

For showing the upper bound, the Prover strategy is as follows: the Prover chooses a depth-optimal tree T_f for f and moves down this tree. Let X^i denote the ith block of variables; i.e. the set of variables $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$. The Prover queries variables blockwise, choosing to query variables from a particular block according to T_f. If x_i is the variable queried at the current node of T_f, the Prover queries variables from X^i following the optimal Prover strategy for the function g, until the value of $g(X^i)$ becomes known. At this point, the Prover moves to the corresponding subtree in T_f.

For lower bound, the Delayer strategy is as follows: When variable y_k is queried, the Delayer responds with $b \in \{0, 1\}$ if $\text{Rank}(h_{k,b}) > \text{Rank}(h_{k,1-b})$, and otherwise defers. Here $h_{k,b}$ is the sub-function of h when y_k is set to b.

The proof that above strategies give the claimed bounds is essentially what constitutes the proof of Theorem 6.6.

Delayer strategy in asymmetric game in $\textsc{Tribes}_n^d$, proving Lemma 8.1

We give a Delayer strategy in an asymmetric Prover-Delayer game which scores at least $n \log n$. On query x_{ij}, Delayer responds with $(p_0, p_1) = (1 - \frac{1}{k}, \frac{1}{k})$, where k is the number of free variables in row i at the time of the query.

We show that the strategy above scores at least $n \log n$ points. The game can end in two possible ways:

1. Case 1: The Prover concludes with function value 0. In this case, the Prover must have queried all variables in some row, say the i-th row, and chosen 0 for all of them. For the last variable queried in the i-th row, the Delayer would have responded with $(p_0, p_1) = (0, 1)$, and hence scored ∞ points in the round and the game.

2. Case 2: The Prover concludes with function value 1. In this case, the Prover must have set a variable to 1 in each row. We show that the Delayer scores at least $\log n$ points per row. Pick a row arbitrarily, and let k be the number of free variables in the row when the first variable in that row is set to 1. The Prover sets $n - k$ variables in this row to 0 before he sets the first variable to 1. For $b \in \{0, 1\}$, let $p_{b,j}$ represents the p_b response of the Delayer when there are j free variables in the row. That is, $p_{0,j} = 1 - \frac{1}{j} = \frac{j-1}{j}$ and $p_{1,j} = \frac{1}{j}$. The contribution of this row to the overall score is at least

$$\log \frac{1}{p_{0,n}} + \log \frac{1}{p_{0,n-1}} + \ldots + \log \frac{1}{p_{0,k+1}} + \log \frac{1}{p_{1,k}} = \log \left(\frac{1}{p_{0,n}} \frac{1}{p_{0,n-1}} \ldots \frac{1}{p_{0,k+1}} \frac{1}{p_{1,k}} \right) = \log n.$$

Since each row contributes at least $\log n$ points, the Delayer scores at least $n \log n$ points at the end of the game.

10   Conclusion

The main thesis of this paper is that the minimal rank of a decision tree computing a Boolean function is an interesting measure for the complexity of the function, since it is not related to other well-studied measures in a dimensionless way. Whether bounds on this measure can be further exploited in algorithmic settings like learning or sampling remains to be seen.

11   Acknowledgments

───── **References** ─────

1   James Aspnes, Eric Blais, Murat Demirbas, Ryan O'Donnell, Atri Rudra, and Steve Uurtamo. k $^+$ decision trees - (extended abstract). In *6th International Workshop on Algorithms for Sensor Systems, Wireless Ad Hoc Networks, and Autonomous Mobile Entities, ALGO-SENSORS*, volume 6451 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2010. full version on author's webpage, http://www.cs.cmu.edu/ odonnell/papers/k-plus-dts.pdf.

2   Yosi Ben-Asher and Ilan Newman. Decision trees with boolean threshold queries. *Journal of Computer and System Sciences*, 51(3):495–502, 1995.

3   Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, 2004.

4 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113(18):666–671, 2013.

5 Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters*, 42(4):183–185, 1992.

6 Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 118–126. IEEE, 1987.

7 Arjan Cornelissen, Nikhil S Mande, and Subhasree Patro. Improved quantum query upper bounds based on classical decision trees. *arXiv preprint arXiv:2203.02968*, 2022.

8 Yogesh Dahiya and Meena Mahajan. On (Simple) Decision Tree Rank. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213 of *LIPIcs*, pages 15:1–15:16, 2021.

9 Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231 – 246, 1989.

10 Javier Esparza, Michael Luttenberger, and Maximilian Schlund. A brief history of Strahler numbers. In *Language and Automata Theory and Applications - 8th International Conference LATA*, volume 8370 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014.

11 Juan Luis Esteban and Jacobo Torán. A combinatorial characterization of treelike resolution space. *Information Processing Letters*, 87(6):295–300, 2003.

12 Juris Hartmanis and Lane A Hemachandra. One-way functions and the nonisomorphism of NP-complete sets. *Theoretical Computer Science*, 81(1):155–163, 1991.

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

14 Stasys Jukna, A Razborov, Petr Savicky, and Ingo Wegener. On p versus np ∩ co-np for decision trees and read-once branching programs. *Computational Complexity*, 8(4):357–370, 1999.

15 Adam R. Klivans and Rocco A. Servedio. Learning DNF in time $2^{\tilde{o}(n^{1/3})}$. *J. Comput. Syst. Sci.*, 68(2):303–318, 2004.

16 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Guest column: Models of computation between decision trees and communication. *ACM SIGACT News*, 52(2):46–70, 2021.

17 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.

18 Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. *Electron. Colloquium Comput. Complex.*, (41), 1999.

19 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

20 Nikhil S. Mande and Swagato Sanyal. On parity decision trees for fourier-sparse boolean functions. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

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

22 Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

23 Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for k-SAT (preliminary version). In *Proceedings of the eleventh annual ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 128–136, 2000.

24 Amir Shpilka, Avishay Tal, and Ben lee Volk. On the structure of boolean functions with small spectral norm. *Comput. Complex.*, 26(1):229–273, 2017.

25 Gábor Tardos. Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A? *Combinatorica*, 9(4):385–392, 1989.

26   György Turán and Farrokh Vatan. Linear decision lists and partitioning algorithms for the construction of neural networks. In *Foundations of Computational Mathematics*, pages 414–423, Berlin, Heidelberg, 1997. Springer.

27   Kei Uchizawa and Eiji Takimoto. Lower bounds for linear decision trees with bounded weights. In *41st International Conference on Current Trends in Theory and Practice of Computer Science SOFSEM*, volume 8939 of *LNCS*, pages 412–422. Springer, 2015.

28   Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

A   Proof of Theorem 2.4

▶ **Theorem 2.4.** *[[24, 20]] Let f be any Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$. Fix any $S \subseteq [n]$, $S \neq \emptyset$. If $f|(S,\rho)$ is a constant, then for every $\rho' : S \longrightarrow \{0,1\}$, $\tilde{spar}(f|(S,\rho')) \leq \tilde{spar}(f)/2$.*

Proof. Let J denote the subcube (S,ρ), and let $\bar{S}$ denote the set $[n] \setminus S$.

Partition the Fourier support of f (sets W for which $\hat{f}(W) \neq 0$) into buckets, one for each $T \subseteq \bar{S}$, as follows:

Bucket for T, $B_T = \{W \subseteq [n] : W \cap \bar{S} = T; \hat{f}(W) \neq 0\}$.

Note that $\tilde{spar}(f) = |B_\emptyset \setminus \{\emptyset\}| + \sum_{T \subseteq \bar{S}; T \neq \emptyset} |B_T|$.

In the Fourier expansion of f, group together terms with the same signature outside S. That is,

$$f(x) = \sum_{T \subseteq \bar{S}} \left[\sum_{R \subseteq S} \hat{f}(T \cup R)\chi_{T \cup R}(x) \right] = \sum_{T \subseteq \bar{S}} \underbrace{\left[\sum_{R \subseteq S} \hat{f}(T \cup R)\chi_R(x) \right]}_{c_T(x)} \chi_T(x) = \sum_{T \subseteq \bar{S}} c_T(x)\chi_T(x)$$

Inside any subcube of the form $J' = (S,\rho')$, the functions $c_T(x)$ are independent of x, since $\chi_R|J'(x) = (-1)^{\sum_{i \in R} \rho'(i)}$.

In particular, for the subcube J, $f|J$ is a constant, say c. Considering the Fourier expansion of $g = f|J$, we see that

$$c = g = \sum_{T \subseteq \bar{S}} (c_T|J)(x)\chi_T(x).$$

Since the Fourier representation of a function is unique, it follows that $c_\emptyset|J = c$, and $c_T|J = 0$ for all non-empty T. Fix any non-empty T. Then, by definition of c_T and B_T,

$$(c_T|J)(x) = \sum_{R \subseteq S} \hat{f}(T \cup R)(\chi_R|J) = \sum_{W \in B_T} \hat{f}(W)(\chi_R|J).$$

Since $c_T|J = 0$, it must have none or at least two non-zero terms to achieve a cancellation. Thus for each non-empty T, if $B_T \neq \emptyset$, then $|B_T| \geq 2$.

Now we can bound the sparsity of $f|J'$ for any subcube (S,ρ'). Since $f|J'(x) = \sum_{T \subseteq \bar{S}} (c_T|J')\chi_T(x)$, we see that $\tilde{spar}(f|J')$ is at most the number of non-empty buckets B_T

for non-empty T. Hence

$$\tilde{\text{spar}}(f|J') = \text{number of non-empty } T \text{ with non-empty bucket } B_T$$
$$= \frac{1}{2} \sum_{T \subseteq \bar{S}, T \neq \emptyset, B_T \neq \emptyset} 2$$
$$\leq \frac{1}{2} \sum_{T \subseteq \bar{S}, T \neq \emptyset, B_T \neq \emptyset} |B_T|$$
$$\leq \frac{1}{2} \tilde{\text{spar}}(f).$$

◀