Highlights

**On (Simple) Decision Tree Rank**
Yogesh Dahiya, Meena Mahajan

- Relating rank of decision tree for Boolean functions with depth (query complexity), size, sparsity, and certificate complexity.
- Relating rank of composed function with depth of outer function and rank of inner function.
- Decision tree size lower bound for composed functions in terms of depth of outer function and rank of inner function.
- Sandwiching depth of (AND,OR) decision trees between simple decision tree rank and log size.
On (Simple) Decision Tree Rank

Yogesh Dahiya*, Meena Mahajan*

*The Institute of Mathematical Sciences, (A CI of Homi Bhabha National Institute HBNI), Tharamani, Chennai, 600113, TN, India

Abstract

In the decision tree computation model for Boolean functions, the depth corresponds to query complexity, and the 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 allows us to easily recover known asymptotical 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.

Keywords: Boolean functions, Decision trees, query complexity, 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 the 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, 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 Depth$(f)$, $\text{Rank}(f)$ is not polynomially related with sensitivity or to certificate complexity $C(f)$, although it is bounded above by 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 $\text{Parity}_n$ function is significantly harder than the $\text{And}_n$ 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), up to 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 Depth$(f)$, Depth$(f)$ is bounded above by the product of Rank$(f)$ and $1 + \log \text{spart}(f)$ (Theorem 5.2).
3. Rank$(f)$ is bounded between the minimum certificate complexity of $f$ at any point, and $(C(f) - 1)^2 + 1$; Theorem 5.8. The upper bound (Lemma 5.4) is an improvement on the bound inherited from Depth$(f)$, and is obtained by adapting that construction.
4. For a composed function $f \circ g$, Rank$(f \circ g)$ is bounded above and below by functions of Depth$(f)$ and 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 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 Rank$(f)$ and DTSize$(f)$ from [9], the upper bound on log DTSize is asymptotically tight for the T^knes 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 Delaye, 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 a somewhat small degree. The original DNF is first converted to such a hybrid 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, 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 \( 2^{\text{Value}(F)} \). Subsequently, the results of [18, 11] yield the equivalence \( \text{Value}(F) = \text{Rank}(F) \), where \( \text{Rank}(F) \) 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 that 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 Cosi decision tree model. In Section 8 we examine the size-rank relationship for the Trans function. The bounds in Sections 4 to 6 and 8 are all obtained by direct inductive arguments决策树 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 \( f : \{0, 1\}^n \rightarrow \{0, 1\} \), a decision tree computing \( f \) is a binary tree with internal nodes labelled 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 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 \( \text{Depth}(T) \) of a decision tree \( T \) is the length of the longest root-to-leaf path, and its size \( \text{DTSIZE}(T) \) is the number of leaves. The decision tree complexity or the depth of \( f \), denoted by \( \text{Depth}(f) \), is defined to be the minimum depth of a decision tree computing \( f \). Equivalently, \( \text{Depth}(f) \) can also be seen as the minimum number of worst-case queries required to evaluate \( f \). The size of a function \( f \), denoted by \( \text{DTsize}(f) \), is defined similarly i.e. the minimum size of a decision tree computing \( f \). Since decision trees can be reduced, \( \text{Depth}(f) \leq n \) and \( \text{DTsize}(f) \leq 2^n \) for every \( n \)-variate function \( f \). A function is said to be evasive if its depth is maximal, \( \text{Depth}(f) = n \).
Using this definition, we can define several measures. For a decision tree $T$, its weighted depth with respect to the weight vector $[w_1, \ldots, w_n]$, denoted by $\text{Depth}_w(T, [w_1, w_2, \ldots, w_n])$, 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 $\text{Depth}_w(f, [w_1, w_2, \ldots, w_n])$, is the minimum weighted depth of a decision tree computing $f$. Note that $\text{Depth}(f)$ is exactly $\text{Depth}_w(f, [1, 1, \ldots, 1])$. 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 $i \in [n]$, 

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

**Certificate Complexity.** The certificate complexity of a function $f$, denoted $C(f)$, 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 $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and an input $a \in \{0, 1\}^n$, an $f$-certificate of $a$ is a subset $S \subseteq [1, \ldots, n]$ 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 $C(f, a)$ denote the minimum size of an $f$-certificate for the input $a$. That is,

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

Using this definition, we can define several measures.

For $b \in \{0, 1\}$, $C_b(f) = \max\{C(f, a) \mid a \in f^{-1}(b)\}$

$$C(f) = \max\{C(f, a) \mid a \in \{0, 1\}^n\} = \max\{C_0(f), C_1(f)\}$$

$$C_{avg}(f) = 2^{-n} \sum_{a \in \{0, 1\}^n} C(f, a)$$

$$C_{min}(f) = \min\{C(f, a) \mid a \in \{0, 1\}^n\}$$

**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 $f \circ (g_1, g_2, \ldots, g_n)$ is a function of arity $\sum_i m_i$, and is defined as follows: for $a' \in \{0, 1\}^m$ for each $i \in n$, $f \circ (g_1, g_2, \ldots, g_n)(a^1, a^2, \ldots, a^n) = f(g_1(a^1'), g_2(a^2'), \ldots, g_n(a^n'))$. We call $f$ the outer function and $g_1, \ldots, g_n$ the inner functions. For functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and $g : \{0, 1\}^m \rightarrow \{0, 1\}$, the composed function $f \circ g$ is the function $f \circ (g, g, \ldots, g) : \{0, 1\}^{nm} \rightarrow \{0, 1\}$. The composed function $\text{Or}_n \circ \text{And}_m$ has a special name, $\text{Trubes}_n^m$. Its dual is the function $\text{And}_n \circ \text{Or}_m$ that we denote $\text{Trubes}_n^d$. The dual of $f(x_1, \ldots, x_n)$ is the function $\neg f(\neg x_1, \ldots, \neg x_n)$.

For any function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, that we will call the base function, the iterated composed function $f^{\circ k} : \{0, 1\}^n \rightarrow \{0, 1\}$ is recursively defined as $f^{\circ 0} = f$, $f^{\circ 1} = f \circ f^{\circ (k-1)}$. The iterated composed functions for the base functions $\text{And}_2 \circ \text{Or}_2$ and $\text{Maj}_3$ 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 $f : \{0, 1\}^n \rightarrow \{0, 1\}$,

1. $f$ is evasive (has $\text{Depth}(f) = n$). (See eg. Lemma 14.19 [13].)
2. Hence, for any weights $w_i$, $\text{Depth}_w(f, [w_1, w_2, \ldots, w_n]) = \sum_i w_i$.

For a symmetric Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, let $f_0, f_1, \ldots, f_n \in \{0, 1\}$ 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,

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

Analogously, $\text{Gap}_{\min}(f)$ 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,

$$\text{Gap}_{\min}(f) = \min_{0 \leq k \leq n} \{ b - a : f_{a+1} \neq f_a = f_{a+1} = \ldots = f_b \neq f_{b+1} \}.$$
Proposition 2.5. For any binary tree $T$, the following proposition lists some known properties of the rank function for binary trees.

1. (Rank and Size relationship): $\text{Rank}(T) \leq \log(\text{DTSize}(T)) \leq \text{Depth}(T)$.
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 $\text{Rank}(T) \geq r$. 

*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 $\mathcal{V} = \{f : \{0,1\}^n \rightarrow \mathbb{R}\}$ 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 $\chi_S : \{0,1\}^n \rightarrow \{-1,1\}$ defined by $\chi_S(x) = (-1)^{\sum_{i \in S} x_i}$ is the $\pm 1$ 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 $\text{spar}(f)$, 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 $\overline{\text{spar}}(f) = |\{S \subseteq [n] : S \neq \emptyset, \hat{f}(S) \neq 0\}|$. For every $f$, $0 \leq \text{spar}(f) \leq \overline{\text{spar}}(f) \leq \text{spar}(f)+1$, and only the constant functions have $\text{spar} = 0$.

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 $f : \{0,1\}^n \rightarrow \{0,1\}$, $\log \text{spar}(f) \leq \log \text{DTSize}(f) + \text{Depth}(f) \leq 2\text{Depth}(f)$.

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, and its co-dimension is the number of bits set by the partial assignment. The subcube kill number of $f$, denoted by $K(f)$, is the size of a smallest partial assignment that makes the function constant. Formally, following the notation from [24, 20], a subcube $J$ is a partial assignment (to some of the $n$ variables) defined by $(S, \rho)$ where $S \subseteq [n]$ is the set of input bits fixed by $J$ and $\rho : S \rightarrow \{0,1\}$ 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 $\co\text{dim}(J)$. A function $f : \{0,1\}^n \rightarrow \{0,1\}$ restricted to $J = (S, \rho)$ is the function $f_{|J} : \{0,1\}^{\overline{\rho}(S)} \rightarrow \{0,1\}$ obtained by fixing variables in $S$ according to $\rho$. The following result quantifies the effect on the Fourier spectrum of subcube restriction.

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

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 $f$, $K(f)$, measures the (co-dimension of) a largest subcube over which $f$ is constant.

$K(f) = \min(\co\text{dim}(J)|f_{|J}$ 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): $\text{Rank}(T) \leq \log(\text{DTSize}(T)) \leq \text{Depth}(T)$.
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 $\text{Rank}(T) \geq r$. 


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 $\text{Rank}(f_S) \leq \text{Rank}(f)$.

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

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

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

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 $\log \text{DTSize}(f)$ up to 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 $f : \{0, 1\}^n \rightarrow \{0, 1\}$, $\text{Rank}(f) = n - \text{Gap}(f)$.

**Remark 2.9.** For (simple) deterministic possibly weighted decision trees, each of the measures $\text{DTSize}$, Depth, and Rank, is the same for a Boolean function $f$, its complement $\neg f$, and it’s 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 $\text{SearchR}$ is as follows: given an $x \in X$, find a $w \in W$ with $(x, w) \in R$. A decision tree for $\text{SearchR}$ is defined exactly as for Boolean functions; the only difference is that leaves are labelled with elements of $W$, and we require that for each input $x$, if the unique leaf reached on $x$ is labelled $w$, then $(x, w) \in R$.

The rank of the relation, $\text{Rank}(R)$, is the minimum rank of a decision tree solving the $\text{SearchR}$ problem.

A Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ 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 unsatisfiable 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 $\rho$ in rounds. Initially, $\rho$ is empty. In each round, the Prover queries a variable $x_i$ not set by $\rho$. The Delayer responds with a bit value 0 or 1 for $x_i$, or defers the choice to the Prover. In the latter 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 $\rho$, $(x, w) \in R$. (Thus, for a Boolean function $f$, the game ends when $f|_\rho$ is a constant function.) The value of the game, $\text{Value}(R)$, 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 $X = \{0, 1\}^n$, $\text{Rank}(R) = \text{Value}(R)$. In particular, for a boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, $\text{Rank}(f) = \text{Value}(R_f)$.

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 $\text{Rank}(R)$ points.
Lemma 3.3 (implicit in [11]).

As claimed.

In particular, if the Prover chooses a rank-optimal 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 a smaller rank (breaking ties arbitrarily), and descends into that subtree.

We claim that such a “tree-based” strategy restricts the Delayer’s score to $R(T)$. The proof is by induction on $\text{Depth}(T)$.

1. Base Case: $\text{Depth}(T) = 0$. 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: $\text{Depth}(T) \geq 1$. 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 $b$, $\text{Depth}(T_b) \leq \text{Depth}(T) - 1$, and $T_b$ is a decision tree for the search problem on $R_{i,b} := \{(x, w) \in R \mid x_i = b\} \subseteq X_i \times W$, where $X_i = \{x \in X \mid x_i = b\}$.

If the Delayer responds with a bit $b$, then by induction, the subsequent score of the Delayer is limited to $R(T_b) \leq R(T)$. Since the current round does not increase the score, the overall Delayer score is limited to $R(T)$.

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 $\min(R(T_0), R(T_1))$; by definition of rank, this is at most $R(T) - 1$. So the overall Delayer score is again limited to $R(T)$.

In particular, if the Prover chooses a rank-optimal tree $T_R$, then the Delayer’s score is limited to $R(T_R) = R(R)$ 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 $R(X)$ points.

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

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

1. Base Case: $n = 1$. If $R(X) = 1$, 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 the first variable queried by the prover.

If $R(R_{i,0}) = R(R_{i,1})$, then the Delayer defers, scoring one point in this round. Subsequently, suppose the Prover sets $x_i = b$. The game is now played on $R_{i,b}$, and by induction, the Delayer can subsequently score at least $R(R_{i,b})$ points. But also, because of the equality, we have $R(R(X)) \leq 1 + R(R_{i,b})$, 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 $R(R)$.

If $R(R_{i,b}) > R(R_{i,1-b})$, 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 $R(R_{i,b})$. But $R(R(X)) \leq R(R_{i,b})$, 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 $R$, exhibiting a Prover strategy which restricts the Delayer to at most $r$ points gives an upper bound of $r$ on $R(f)$. Similarly, exhibiting a Delayer strategy which scores at least $r$ points irrespective of the Prover strategy shows a lower bound of $r$ on $R(f)$.

In [4], an asymmetric 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 $\text{ASym-Value}$ 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 \( X = \{0, 1\}^n \), \( \log \text{DTS} \text{ize}(R) = \text{ASym-Value}(R) \). In particular, for a boolean function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \), \( \log \text{DTS} \text{ize}(f) = \text{ASym-Value}(R_f) \).

(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

\[
\text{Rank}(f) = \text{Value}(R_f) \leq \text{ASym-Value}(R_f) = \log \text{DTS} \text{ize}(f).
\]

This relationship explains where the slack may lie in the inequalities from Proposition 2.7 relating \( \text{Rank}(f) \) and \( \log \text{DTS} \text{ize}(f) \). The symmetric game focuses 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 \( \text{Or}_n \) and \( \text{An} \text{d}_n \) functions, the \( \text{Rank}(f) \) measure is not polynomially related with the measures \( \text{Depth}(f) \) or certificate complexity \( \text{C}(f) \).

| \( f \)          | \( \text{Depth} \) | \( \text{C}_0 \) | \( \text{C}_1 \) | \( \text{C} \) | \( \text{Gap} \) | \( \text{Rank} \) |
|-----------------|-------------------|----------------|----------------|----------------|----------------|----------------|
| 0 or 1          | 0                 | 0              | 0              | 0              | \( n \)         | 0              |
| \( \text{And}_n \) | \( n \)            | 1              | \( n \)        | \( n \)        | \( n - 1 \)     | 1              |
| \( \text{Or}_n \)  | \( n \)            | \( n \)        | \( n \)        | \( n - 1 \)     | \( n \)         | 1              |
| \( \text{Par} \text{ity}_n \) | \( n \)            | \( n \)        | \( n \)        | \( 0 \)         | \( n \)         | 1              |
| \( \text{Maj}_{2k} \) | \( 2k \)           | \( k \)        | \( k + 1 \)    | \( k + 1 \)     | \( k \)         | \( k \)         |
| \( \text{Maj}_{2k+1} \) | \( 2k + 1 \)      | \( k + 1 \)    | \( k + 1 \)    | \( k \)         | \( k + 1 \)     | \( k \)         |
| \( \text{Thr} n \) (\( k \geq 1 \)) | \( n \)         | \( n - k + 1 \) | \( k \)        | \( \max \left\{ \frac{n - k + 1}{k}, k \right\} \) | \( \max \left\{ \frac{k - 1}{n - k}, n - k \right\} \) | \( \min \left\{ \frac{n - k + 1}{k}, k \right\} \) |

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 calculated 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. \( \text{Rank}(\text{Tri} \text{be}_{n,m}) = \text{Rank}(\text{Tri} \text{be}_{n,m}^d) = n \) for \( m \geq 2 \).
2. \( \text{Rank}(\text{And}_n \circ \text{Par} \text{ity}_m) = n(m - 1) + 1 \) 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 \( \geq 1 \), 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 \( T \) labelled \( b \), 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 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 labelled $T$, 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 $\text{Depth}(T_L)$ and $\text{Depth}(T_R)$ are at most $k-1$, and at least one of these is exactly $k-1$ 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 $r_L = 0$, then $r = r_L$ and $\text{Depth}(T_L) = k-1 \geq 1$. By the induction hypothesis, $r_L + \max\{r_0, r_1\} - 1 \leq r'_L$. Since $r' \leq r'_L$, the claimed bound follows.

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

If both $r_L, r_R$ are positive, then by the induction hypothesis, $r_L + \max\{r_0, r_1\} - 1 \leq r'_L$ and $r_R + \max\{r_0, r_1\} - 1 \leq r'_R$. If $r_R = r$, then $r = r_R + 1$, and by definition of rank, $r' \geq 1 + \min\{r'_L, r'_R\} \geq r_L + \max\{r_0, r_1\} - 1$, as claimed. On the other hand, if $r_L \neq r$, then $r = \max\{r_L, r_R\}$, and by definition of rank, $r' \geq \max\{r'_L, r'_R\} \geq \max\{r_L, r_R\} - 1 = r + \max\{r_0, r_1\} - 1$, as claimed.

For complete binary tree $T$ satisfying the labelling requirements, $r_L = r_R = r - 1$. The same arguments, simplified to this situation, show the claimed equality:

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

Lemma 4.3. For every $n, m \geq 1$, $\text{Rank}(\text{Trubes}^d_{n,m}) \leq n$.

Proof. We show the bound by giving a recursive construction and bounding the rank by induction on $n$. In the base case, $n = 1$. $\text{Trubes}^d_{1,m} = O_{r,m}$, which has rank 1. For the inductive step, $n > 1$. For $j \leq n$, let $T_{j,m}$ denote the recursively constructed trees for $\text{Trubes}^d_{j,m}$. Take the tree $T_{n-1,m}$ 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 $\text{Trubes}^d_{n,m}$. 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 $m \geq 2$, $\text{Rank}(\text{Trubes}^d_{n,m}) \geq n$.

Proof. We prove this by induction on $n$. The base case, $n = 1$, is straightforward: $\text{Trubes}^d_{1,m}$ is the function $O_{r,m}$, whose rank is 1.

For the inductive step, let $n > 1$, and consider any decision tree $Q$ for $\text{Trubes}^d_{n,m}$. 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 $\text{And}_n \circ (O_{r_{m-1}, \ldots, O_{r_m}})$, and $Q_1$ computes $\text{Trubes}^d_{n-1,m}$, on appropriate variables. For $m \geq 2$, $\text{Trubes}^d_{n-1,m}$ is a sub-function of $\text{And}_n \circ (O_{r_{m-1}, \ldots, O_{r_m}})$, and so Proposition 2.6 implies that $\text{Rank}(Q_0) \geq \text{Rank}(\text{And}_n \circ (O_{r_{m-1}, \ldots, O_{r_m}})) \geq \text{Rank}(\text{Trubes}^d_{n-1,m})$. By induction, $\text{Rank}(Q_1) \geq \text{Rank}(\text{Trubes}^d_{n-1,m}) \geq n - 1$. 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 $\text{Trubes}^d_{n,m}$, we conclude that $\text{Rank}(\text{Trubes}^d_{n,m}) \geq n$, 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 byRemark 4.4.

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

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 $\text{Parity}_m$ is a complete binary tree.
For every Boolean function $f$, $K(f) \leq \text{Rank}(f)$. 

\textbf{5. Relation between Rank and other measures}

\textbf{5.1. Relating Rank to Depth and Sparsity}

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

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

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

\textbf{Lemma 5.1.} For every Boolean function $f$, $K(f) \leq \text{Rank}(f)$. 

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

\textbf{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 $\text{Parity}_m$.

\textbf{Inductive Step:} $n > 1$. For $j < n$, 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 $x_i, j \in [n-1], j \in [m]$, 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.

\textbf{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 $\{\text{Parity}_m, \neg \text{Parity}_m\}$, $\text{Rank}(\text{And}_n \circ (g_1, g_2, \ldots, g_n)) \geq (\sum_{i=1}^n (m_i - 1)) + 1$.

In particular, $\text{Rank}(\text{And}_n \circ \text{Parity}_m) \geq n(m-1) + 1$.

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

\textbf{Base Case:} $n = 1$. $h = g_1$. Note that for all functions $f$, $\text{Rank}(f) = \text{Rank}(\neg f)$. So $\text{Rank}(h) = \text{Rank}(\text{Parity}_m) = m_1$.

\textbf{Inductive Step:} $n > 1$. We proceed by induction on $M = \sum_{i=1}^n m_i$.

1. \textbf{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. \textbf{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 $Q_b$ denote the function $g_1$ restricted to $x_{1,1} = b$. Then $Q_b$ computes the function $\text{And}_n \circ (g_1, g_2, \ldots, g_n)$ on appropriate variables.

If $m_1 = 1$, then the functions $g_1, g_1$ 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 $\text{And}_{n-1} \circ (g_2, \ldots, g_n)$. Using induction on $n$, we conclude

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

For $m_1 \geq 2$, $g_1 = [\text{Parity}_{m_1-1}, \neg \text{Parity}_{m_1-1}]$. So one of $Q_0, Q_1$ computes $\text{And}_n \circ (\text{Parity}_{m_1-1}, g_2, \ldots, g_n)$, and the other computes $\text{And}_n \circ (\neg \text{Parity}_{m_1-1}, g_2, \ldots, g_n)$. 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.
Proof. 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 \( f \), 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, \( f \) 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 \( f : \{0, 1\}^n \rightarrow \{0, 1\} \),

\[
\text{Depth}(f) \leq \text{Rank}(f)(1 + \log(\text{spar}(f))) \leq \text{Rank}(f)(1 + \log(\text{spar}(f))).
\]

The first inequality is tight as witnessed by \textsc{Parity} function.

**Proof.** Recall that \text{spar}(f) refers to the number of non-zero Fourier coefficients in the expansion of \( f \) apart from \( f(\emptyset) \). The second inequality thus follows by definition. We prove the first inequality by induction on \text{spar}(f).

When \text{spar}(f) < 1, \( f \) is a constant function with \text{Depth}(f) = \text{Rank}(f) = 0, and the inequality holds.

Now assume that \text{spar}(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, \( |S| \leq \text{Rank}(f) \). Start by querying all the variables indexed in \( S \) in any order. When the outcome of all these queries matches \( \rho \), the function becomes a constant and the tree terminates at this leaf. On any other outcome \( \rho' \), the function is restricted to the subcube \( J' = (S, \rho') \). Proceed recursively to build the decision tree of \( f_{J,\rho} \) which is then attached to this leaf.

On any outcome \( \rho' \neq \rho \), we have \text{spar}(f_{J,\rho}) \leq \text{spar}(f)/2 by Theorem 2.4, and \( K(f_{J,\rho}) \leq \text{Rank}(f_{J,\rho}) \leq \text{Rank}(f) \) by Lemma 5.1 and monotonicity of rank (Proposition 2.6). Thus each stage in the recursion makes at most \text{Rank}(f) queries and halves sparsity \text{spar}. After at most \( 1 + \log \text{spar}(f) \) 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 \text{Rank}(f) \cdot (1 + \log \text{spar}(f)).

5.2. Relation between \text{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 \( f : \{0, 1\}^n \rightarrow \{0, 1\} \),

\[
C(f) \leq \text{Depth}(f) \leq C_0(f)C_1(f)
\]

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

Since \text{Rank} \leq \text{Depth}, the same upper bound also holds for \text{Rank} as well. But it is far from tight for the 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 \( f : \{0, 1\}^n \rightarrow \{0, 1\} \),

\[
\text{Rank}(f) \leq (C_0(f) - 1)(C_1(f) - 1) + 1
\]

Moreover, the inequality is tight as witnessed by And and Or functions.

**Proof.** The inequality holds trivially for constant functions since for such functions, \( \text{Rank} = C_0 = C_1 = 0 \). 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 $\text{Rank}(g) \leq (C_0(g) - 1)(C_1(g) - 1) + 1$ 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 minimum-size 0-certificate $S$ for $a$. Without loss of generality, assume that $S = \{x_1, x_2, \ldots, x_l\}$ for some $l = |S| \leq C_0(f)$. 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 is 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 $f_c \neq 0$, $\text{Rank}(T_c) \leq (C_0(f_c) - 1)(k - 1) + 1 \leq (C_0(f) - 1)(k - 1) + 1$. Thus $T$ is obtained from a rank-$\ell$ 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 $\text{Rank}(T) \leq \text{Rank}(T_0) \leq ((C_0(f) - 1)(k - 1) + 1) + (l - 1) \leq (C_0(f) - 1)(C_1(f) - 1) + 1$.

From Theorem 4.1, we see that the lower bound on $\text{Depth}$ in Proposition 5.3 does not hold for Rank; for $m > n$, $\text{Rank}(\text{Trues}^d_{n,m}) = n < m = C(\text{Trues}^d_{n,m})$. However, $\min\{\text{C_0(Trues}^d_{n,m}), C(\text{Trues}^d_{n,m})\} = n = \text{Rank}(\text{Trues}^d_{n,m})$. Further, for all the functions listed in Table 1, $\text{Rank}(f)$ 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 $\text{Rank}(f)$; for the symmetric function $f = \text{MAJ}_n \lor \text{PARITY}_n$, when $n > 4$, $\text{Rank}(f) < \min\{C_0(f), C_1(f)\}$.

**Proof.** Let $f$ be the function $\text{MAJ}_n \lor \text{PARITY}_n$, 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 $\text{Gap}(f) = n/2$, so by Proposition 2.8, $\text{Rank}(f) = n/2$.

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

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

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

Consider $g = \text{Trues}^d_{n,2}$ for $n > 2$. By Theorem 4.1, $\text{Rank}(g) = n$. 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

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

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

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

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

**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 the depth of each leaf in $T$ must be at least $\text{C_{min}(f)}$. By Proposition 2.5(3), $\text{Rank}(f) = \text{Rank}(T) \geq \text{C_{min}(f)}$.

Lemma 5.4 and Lemma 5.7 give these bounds sandwiching $\text{Rank}(f)$:
Theorem 5.8. \(C_{\min}(f) \leq \text{Rank}(f) \leq \text{C}_{1}(f) - 1 + 1 \leq \text{C}(f) - 1 \leq 1 + \text{C}(f) - 1 = 1\).

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 \(\text{Gap}_{\min}\). Using this relation, the upper bound on \(\text{Rank}(f)\) 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, \(\text{C}(f) = n - \text{Gap}_{\min}(f)\), and \(n - \text{C}(f) + 1 \leq \text{Rank}(f) \leq C(f)\). Both the inequalities on rank are tight for \(\text{Maj}_{2n+1}\).

Proof. We first show \(C(f) = n - \text{Gap}_{\min}(f)\). Consider any interval \([a, b]\) such that \(f_{a-1} \neq f_a = f_{a+1} = \ldots = f_b \neq f_{b+1}\). 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 than \(n - b\) 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,

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

As seen from Table 1, these bounds on rank are tight for \(\text{Maj}_{2n+1}\).

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 \(\text{Rank}\) from above; for \(k \geq 1\) and \(n = 4^k\) the function \(f = (\text{And}_2 \circ \text{Or}_2)^{2^k}\) on \(n\) variables has \(\text{Rank}(f) = \Omega(\text{C}(f)^2)\).

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 \(f, g\), \(\text{Depth}(f \circ g) = \text{Depth}(f)\text{Depth}(g)\).

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 \(\text{Rank}\) 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 \(\text{Rank}(f \circ g) = \text{Rank}(\text{Tribes}_n) = n\), 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 Depth(\( T_f \)Rank(\( T_g \)) = Depth(\( f, [r, r, \ldots, r] \)), 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 \( \text{Parry} \) function are used to show that the resulting tree’s rank is even smaller than Depth(\( f \)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 \( \text{Rank}(g_i) = r_i \), and for \( n \)-variate non-constant boolean function \( f \),

\[
\text{Rank}(f \circ (g_1, g_2, \ldots, g_n)) \leq \text{Depth}_w(f, [r_1, r_2, \ldots, r_n]).
\]

**Proof.** Let \( h \) denote the function \( f \circ (g_1, g_2, \ldots, g_n) \). For \( i \in [n] \), let \( m_i \) be the arity of \( g_i \). We call \( x_{i1}, x_{i2}, \ldots, x_{in} \), the \( i \)th 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:** Depth(\( T_f \)) = 0. Then \( f \) and \( h \) are the same constant function, so set \( T_h = T_f \).
2. **Recursion Step:** 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 \( i \)th 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 \( x_{i,b} \) for \( k \neq i \). Starting with the tree \( T_0 \), on the \( i \)th block of variables, attach tree \( T'_b \) to each leaf labelled \( b \) 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 Depth(\( T_f \)), we show that \( \text{Rank}(T_h) \leq \text{Depth}_w(T_f, [r_1, r_2, \ldots, r_n]) \).

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

\[
\begin{align*}
\text{Rank}(T_h) &\leq \text{Rank}(T_{g_i}) + \max_{b \in \{0,1\}} \{ \text{Rank}(T'_b), \text{Rank}(T'_b) \} \quad \text{(by Proposition 4.2)} \\
&= r_i + \max_{b \in \{0,1\}} \{ \text{Depth}_w(T_{g_i}, [r_1, r_2, \ldots, r_n]) \} \quad \text{(by induction)} \\
&\leq r_i + \max_{b \in \{0,1\}} \{ \text{Depth}_w(T_f, [r_1, r_2, \ldots, r_n]) \} \quad \text{by definition of } \text{Depth}_w
\end{align*}
\]

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

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 \( f, g \),

\[
\text{Rank}(f \circ g) \geq \max \{ \text{Rank}(f), \text{Rank}(g) \}.
\]

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 \(\text{Rank}(g_i) = r_i\), and for \(n\)-variable non-constant boolean function \(f\),

\[
\text{Rank}(f \circ (g_1, g_2, \ldots, g_n)) \geq \text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1
\]

\[
\geq \text{Depth}_w(f, [r_1, r_2, \ldots, r_n]) - (n - 1).
\]

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 \(\text{Depth}_w(f, [r_1, \ldots, r_n]) \leq \text{Depth}_w(T, [r_1, \ldots, r_n]) \leq \text{Depth}(T, [r_1 - 1, \ldots, r_n - 1]) + n = \text{Depth}_w(f, [r_1 - 1, \ldots, r_n - 1]) + n\), 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 \(f \circ (g_1, g_2, \ldots, g_n)\). 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 \(i\)th 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 \(\text{Depth}_w(f, [r_1 - 1]) = r_1 - 1\) and \(\text{Rank}(h) = \text{Rank}(g_1) = r_1\), 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 \(\text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1 = \text{Depth}_w(f, [0, 0, \ldots, 0]) + 1 = 1\). 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 \(f\) up to renaming of the literals. Hence \(\text{Rank}(h) = \text{Rank}(f) \geq 1\).

For the inductive step, \(M > n > 1\). Take a rank-optimal decision tree \(T_0\) for \(h\). We want to show that \(\text{Depth}_w(f, [r_1 - 1, \ldots, r_n - 1]) \leq \text{Rank}(T_0) - 1\). 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 \(T_0\). For \(b \in [0, 1]\), let \(g_i^b\) be the subfunction of \(g_i\) when \(x_{1,1}\) is set to \(b\). Note that \(T_0\) computes \(h_0 = f \circ (g_1^b, g_2, \ldots, g_n)\), a function on \(M - 1\) variables. We would like to use induction to deduce information about \(\text{Rank}(T_0)\). However, \(g_i^b\) may be a constant function, and then induction does not apply. So we do a case analysis on whether or not \(g_i^0\) and \(g_i^1\) are constant functions; this case analysis is lengthy and tedious but most cases are straightforward.

- Case 1: Both \(g_i^0\) and \(g_i^1\) are constant functions. Since \(g_1\) is non-constant, \(g_i^0 \neq g_i^1\), and \(r_i = \text{Rank}(g_i) = 1\). Assume that \(g_i^0 = 0\) and \(g_i^1 = 1\); 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 \(\text{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) = \text{Depth}_w(f, [0, r_2 - 1, \ldots, r_n - 1]) = 0\). Also, in this case, \(h\) is either \(x_{1,1}\) or \(\neg x_{1,1}\), so \(\text{Rank}(h) = 1\). 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 \(w_2, \ldots, w_n\), \(\text{Depth}_w(f, [0, w_2, \ldots, w_n]) \leq \text{Depth}_w(f_1, [w_2, \ldots, w_n])\); 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

\[
\text{Rank}(h) = \text{Rank}(h_1)
\]

\[
\geq \text{Depth}_w(h_1, [r_2 - 1, \ldots, r_n - 1]) + 1
\]

\[
= \text{Depth}_w(h_1, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1
\]

by induction hypothesis on \(n\)

by first observation above

since \(r_1 = 1\)

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

\[
\text{Rank}(h) \geq \text{max}(\text{Rank}(h_0), \text{Rank}(h_1))
\]

\[
\geq \text{max}(\text{Depth}_w(h_0, [r_2 - 1, \ldots, r_n - 1]) + 1)
\]

\[
\geq \text{Depth}_w(h_1, [r_2 - 1, \ldots, r_n - 1]) + 1
\]

by induction hypothesis on \(n\)

by def. of weighted depth

of a tree querying \(x_1\) first

since \(r_1 = 1\)

15
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 $\text{Rank}(g_1) = \text{Rank}(g_1^1)$: $\text{Rank}(g_1) \leq \text{Rank}(g_1^1)$ by Proposition 2.6, and $\text{Rank}(g_1^1) \leq \text{Rank}(g_1)$ 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

$$\text{Rank}(h) \geq \text{Rank}(h_1)$$

\[ \geq D_n(f, [\text{Rank}(g_1^1) - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{by induction on } M \]

\[ = D_n(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{since } \text{Rank}(g_1^1) = \text{Rank}(g_1) \]

Case 3: Both $g_1^0$ and $g_1^1$ are non-constant functions. Let $r_1^0 = \text{Rank}(g_1^0) \geq 1$. 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: $\max_b\{r_1^1\} = r_1 - 1$. 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 $r_1 - 1 = r_0^1 = r_1 = 0$, contradicting the fact that we are in Case 3. For $b \in \{0, 1\}$,

\[ \text{Rank}(h_b) = \text{Rank}(f \circ (g_1^b, g_2, ..., g_n)) \]

\[ \geq \text{Depth}_n(f, [r_1^b - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{by induction on } M \]

\[ = \text{Depth}_n(f, [r_1 - 2, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{since } r_1 - 1 = r_1^b. \]

Hence

\[ \text{Rank}(h) \geq 1 + \min_b \text{Rank}(h_b) \]

\[ \geq \text{Depth}_n(f, [r_1 - 2, r_2 - 1, ..., r_n - 1]) + 2 \quad \text{derivation above} \]

\[ \geq \text{Depth}_n(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{by Fact 2.1} \]

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

\[ \text{Rank}(h) \geq \max_b \text{Rank}(h_b) \]

\[ \geq \max_b \text{Depth}_n(f, [r_1^b - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{by induction on } M \]

\[ \geq \text{Depth}_n(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 \quad \text{since } \max_b\{r_1^1\} \geq r_1 \]

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 $f, g$,

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

Both inequalities are tight; the first for $\text{And}_n \circ \text{Parity}_n$ and the second for $\text{Tribes}_n$ and $\text{Tribes}_n^d$.

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 $g_1, ..., g_n$ with $\text{Rank}(g_i) = r_i$, and for $n$-variate symmetric non-constant boolean function $f$,

$$\sum_i r_i - (n - 1) \leq \text{Rank}(f \circ (g_1, g_2, ..., g_n)) \leq \sum_i r_i.$$
For iterated composed functions, we obtain the following corollary.

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

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

**Proof.** The result follows from Theorem 6.6 and application of 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

\[
\text{Rank}(f^\otimes k) = \text{Rank}(f \circ f^\otimes (k-1)) \\
\geq \text{Depth}(f) (\text{Rank}(f^\otimes (k-1)) - 1) + 1 \quad \text{by Theorem 6.6}
\]

\[
\geq \text{Depth}(f) (\text{Depth}(f)^{k-2} (\text{Rank}(f) - 1)) + 1 \quad \text{by induction on } k
\]

\[
= \text{Depth}(f)^{k-1} (\text{Rank}(f) - 1) + 1.
\]

\[
\text{Rank}(f^\otimes k) = \text{Rank}(f \circ f^\otimes (k-1)) \\
\leq \text{Depth}(f) \text{Rank}(f^\otimes (k-1)) \quad \text{by Theorem 6.6}
\]

\[
\leq \text{Depth}(f) (\text{Depth}(f)^{k-2} (\text{Rank}(f))) \quad \text{by induction on } k
\]

\[
= \text{Depth}(f)^{k-1} \text{Rank}(f).
\]

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 \( \text{Conj} \) 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 \( \log \text{DTSize} \) 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 in the exponent. To prove this, the authors of [14] showed that iterated \( \text{AND}_2 \circ \text{OR}_2 \) and iterated \( \text{MAJ}_3 \) on \( n = 4^k \) and \( n = 3^k \) variables require decision trees of size \( \exp(O(\log^{\log^2 3} N)) \) and \( \exp(\Omega(\log^2 N)) \) respectively. It is easy to show that \( N((\text{AND}_2 \circ \text{OR}_2)^\otimes k) \) and \( N((\text{MAJ}_3)^\otimes k) \) is \( \exp(O(n^{1/3})) \) and \( \exp(O(n^{1/2})) \) respectively. So showing optimality essentially boiled down to showing that the decision tree size of iterated \( \text{AND}_2 \circ \text{OR}_2 \) and iterated \( \text{MAJ}_3 \) 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 \( n = 3^k \),

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

For \( k \geq 1 \) and \( n = 4^k \),

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

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

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

Recently (after the preliminary version of our work appeared), in [7], the rank of the iterated AND$_2$ \& 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 $g$,
\[
\log DTSize(f \circ g) \geq Depth(f)(\text{Rank}(g) - 1) + 1.
\]
\[
\log DTSize(f^\otimes k) \geq Depth(f)^{k-1}(\text{Rank}(f) - 1) + 1.
\]

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

It is easy to see that $C_0((\text{AND}_2 \circ \text{OR}_2)^\otimes 2) = C_1((\text{AND}_2 \circ \text{OR}_2)^\otimes 2) = 2$, and that for $k > 1$, $C_0((\text{AND}_2 \circ \text{OR}_2)^\otimes k) = 2C_0((\text{AND}_2 \circ \text{OR}_2)^\otimes (k-1))$. Thus $C_0((\text{AND}_2 \circ \text{OR}_2)^\otimes k) = C_1((\text{AND}_2 \circ \text{OR}_2)^\otimes k) = 2^k = \sqrt{n}$.

Hence, for $f = (\text{AND}_2 \circ \text{OR}_2)^\otimes k$, $\text{Rank}(f) \geq \frac{\log(n)}{2^{k-1}} + 1 = C(f)^2/4 + 1$.

### 7.2. Coni decision trees

In this section, we consider a generalization of the ordinary decision tree model, namely Coni decision trees. In the Coni decision tree model, each query is a conjunction of literals; and 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 Coni decision trees, (AND, OR)-decision trees, was investigated in [2] for determining the complexity of THR$^k_n$ 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$^k_n$ functions for all $k \geq 1$. It is easy to see that the (AND, OR)-query model is equivalent to the Coni query model up to a factor of 2. Let $\text{Depth}_{\wedge}$ and $\text{Depth}_{\wedge,\vee}(f)$ denote the query complexity of function $f$ in Coni and (AND, OR) query model respectively. Then

**Proposition 7.3.** For every boolean functions $f$,
\[
\text{Depth}_{\wedge}(f) \leq \text{Depth}_{\wedge,\vee}(f) \leq 2\text{Depth}_{\wedge}(f).
\]

Such a connection is not obvious for the rank of Coni and (AND, OR) decision trees.

Recently, in [17], a monotone version of Coni decision trees called Aso 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 Aso trees. Understanding monotone Aso 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 Aso 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 Coni decision tree model, up to a log $n$ factor. Formally,

**Theorem 7.4.** For every boolean functions $f$,
\[
\text{Rank}(f) \leq \text{Depth}_{\wedge}(f) \leq 4\log DTSize(f) \leq 4\text{Rank}(f)\log\left(\frac{en}{\text{Rank}(f)}\right).
\]

Consequently, if $\text{Rank}(f) = \Theta(n)$, then so is $\text{Depth}_{\wedge}(f)$.
Proof. First, we show that \( \text{Rank}(f) \leq \text{Depth}_d(f) \). This is the straightforward construction and its analysis. Let \( T_f \) be a depth-optimal \( \text{Conj} \) 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 \( \text{Depth}(T_f) = 0, \) set \( T = T_f \). In the recursion step, \( \text{Depth}(T_f) \geq 1 \). 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 \( \text{And} \) 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 \( T_1' \) to the unique leaf labelled 1 in \( T_Q \), to obtain \( T' \). Since \( f_0 \) can depend non-trivially on the variables in \( Q \), the tree \( T_0' \) may query these variables. However, these queries are redundant at the leaves of \( T_Q \). So process \( T' \) to remove any double queries, obtaining the reduced decision tree \( T \).

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

1. Base Case: \( \text{Depth}(T_f) = 0 \). Trivially true as \( T = T_f \) with rank 0.
2. Induction: \( \text{Depth}(T_f) \geq 1 \).

\[
\text{Rank}(T) \leq \text{Rank}(T_Q) + \max \{ \text{Rank}(T_0'), \text{Rank}(T_1') \} \quad \text{(by Proposition 4.2)}
\]

\[
= 1 + \max \{ \text{Rank}(T_b') \} = 1 + \max \{ \text{Depth}(f_0) \} \quad \text{(by induction)}
\]

\[
\leq 1 + (\text{Depth}(T_f) - 1) = \text{Depth}(T_f).
\]

Next, we show that \( \text{Depth}_d(f) \leq \log \text{DTSIZE}(f) \). The main idea is that an ordinary decision tree of size \( s \) can be balanced using \( \text{Conj} \) queries into a \( \text{Conj} \) decision tree of depth \( 4 \log s \).

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 reach 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 \( \text{Conj} \) 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 subroot rooted at \( v \), denoted by \( T_i \), 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 subroot rooted at the current node satisfies the condition.) Let \( J_v = (S, \rho) \) be the subcube associated with \( v \), and let \( Q \) be the \( \text{Conj} \) query testing membership in \( J_v \): \( Q = (\bigwedge_{x_i \in \rho, x_i = 1} x_i) \land (\bigwedge_{x_i \in \rho, x_i = 0} \neg x_i) \).

Note that \( v \) cannot be the root node of \( T_f \). 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 \( u \) in \( T_f \). Let \( T_i' \) be the tree obtained from \( T_f \) by removing the entire subroot \( T_i \), and attaching subtree \( T_u \) at \( w \). For all inputs not in \( J_v, T_{i'} \) and \( T_f \) compute the same value.

\( T \) starts by querying \( Q \). If \( Q \) evaluates to 1, proceed by recursively constructing the \( \text{Conj} \) decision tree for \( T_i \). If \( Q \) evaluates to 0, proceed by recursively constructing the \( \text{Conj} \) decision tree for \( T_{i'} \).

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 \( \text{Conj} \) decision tree. By construction, we have \( D(s) \leq 1 + D(2s/3) \) giving us \( D(s) \leq 2 \log_{3/2}s \leq 4 \log s \), thereby proving our claim.

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

8. Tightness of Rank and Size relation for \text{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 inequalities. Since \( \text{Rank}(\text{Parity}_n) = n \), Proposition 2.7 tells us that \( \log \text{DTSIZE}(\text{Parity}_n) \) lies in the range \([n, n \log \epsilon]\), and we know that \( \log \text{DTSIZE}(\text{Parity}_n) = n \).

For the \( \text{Tribes}_d \) function, which has \( N = n^2 \) variables, we know from Theorem 4.1 that \( \text{Rank}(\text{Tribes}_d) = n \in o(N) \). Thus Proposition 2.7 tells us that \( \log \text{DTSIZE}(\text{Tribes}_d) \) lies in the range \([n, n \log \epsilon(n)] \). (See also Exercise 14.9 [13] for a direct argument showing \( n \leq \log \text{DTSIZE}(\text{Tribes}_d) \). But that still leaves a \((\log \epsilon(n))\)-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 \( \text{Tribes}_d^\epsilon \).
Lemma 8.1. For every \( n,m \geq 1 \), every decision tree for \( \text{Trubes}_{n,m}^d \) has at least \( m^n \) 1-leaves and \( n \) 0-leaves.

Proof. Recall that \( \text{Trubes}_{n,m}^d = \bigwedge_{i \in [n]} \bigvee_{j \in [m]} x_{i,j} \). We call \( x_{i,1}, x_{i,2}, \ldots, x_{i,m} \) the \( i \)th 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 \( \text{Trubes}_{n,m}^d \), 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 \( \text{Trubes}_{n,m}^d(a') = 0 \). 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 \( \text{Trubes}_{n,m}^d(a'') = 1 \). 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 \( \text{Trubes}_{n,m}^d \) 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 \( \text{Trubes}_{n,m}^d \), proving Lemma 4.3. We give a Prover strategy which restricts the Delayer to \( n \) points, proving the upper bound on \( \text{Rank(Trubes}_{n,m}^d) \).

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 \( \text{Trubes}_{n,m}^d \), proving Lemma 4.5. We give a Delayer strategy which always scores 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 a 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 block, since \( |S_1| = m^n \). 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 \( \text{Trubes}_{n,m}^d \), 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 \( \text{Trubes}_{n,m}^d(a') = 0 \). 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 \( \text{Trubes}_{n,m}^d(a'') = 1 \). 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 \( \text{Trubes}_{n,m}^d \) 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.
Delayer strategy for $\text{And}_m \circ \text{Parity}_m$, 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 $\overline{f}$ be the function obtained by assigning values to the variables queried so far. As long as $C_1(\overline{f}) > 1$, Prover picks an $a \in \overline{f}^{-1}(0)$ 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 $C_1(\overline{f})$ becomes 1, the Prover picks an $a \in \overline{f}^{-1}(1)$ 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 requires the Delayer to $(C_0(f) - 1)(C_1(f) - 1) + 1$ points; the proof is essentially the 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 $i$th 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 the 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 the above strategies give the claimed bounds is essentially what constitutes the proof of Theorem 6.6.

Delayer strategy in asymmetric game in $\text{Tribes}_m^n$, proving Lemma 8.1. We give a Delayer strategy in an asymmetric Prover-Delayer game which scores at least $n \log n$. On query $x_{i,j}$, 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 $\infty$ 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 $n \log n$ 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 $p_{0,j}$ 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.
Acknowledgments

The authors thank Anna Gál and Srikanth Srinivasan for interesting discussions about rank at the Dagstuhl seminar 22371.

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, ALGOSENSORS, 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 Iian 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:2203.02906, 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 Lutenberger, and Maximilian Schlund. A brief history of Strahler numbers. In 41st International Conference on Current Trends in Theory and Practice of Computer Science SOFSEM, volume 8939 of LNCS, pages 412–422. Springer, 2015.

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

[12] Javier Esparza, Michael Lutenberger, 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.

[13] Javier Esparza, Michael Lutenberger, 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.

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

[15] Yusuf Avni, Wern Hua, and Jordan T. Bastian. A composition theorem for decision tree complexity. Chicago Journal of Theoretical Computer Science, 2014(6), July 2014.

[16] Ryan O’Donnell. Analysis of boolean functions. Cambridge University Press, 2014.

[17] 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.

[18] 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.

[19] 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.

[20] Leslie G. Valiant. A theory of the learnable. Communications of the ACM, 27(11):1134–1142, 1984.
Appendix A. Proof of theorem 2.4

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

Proof. Let $J$ denote the subcube $(S, \rho)$, and let $\tilde{S}$ 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 $T \subseteq \tilde{S}$, as follows:

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

Note that $\text{spar}(f) = |B_0 \setminus \{\emptyset\}| + \sum_{T \subseteq \tilde{S}, T \neq \emptyset} |B_T|$.

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

$$f(x) = \sum_{T \subseteq S} \left| \sum_{R \subseteq S} \hat{f}(T \cup R) \chi_{T \cup R}(x) \right| = \sum_{T \subseteq S} \left( \sum_{R \subseteq S} \hat{f}(T \cup R) \chi_R(x) \right) \chi_T(x) = \sum_{T \subseteq S} c_T(x) \chi_T(x)$$

Inside any subcube of the form $J' = (S, \rho')$, the functions $c_T(x)$ are independent of $x$, since $\chi_{T \cup R}(x) = (-1)^{\sum_{i \in T} \rho'(i)}$.

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

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

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 $B_T$,

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

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 $f_{J|T}$ for any subcube $(S, \rho')$. Since $f_{J|T}(x) = \sum_{T \subseteq \tilde{S}} (c_{T|J})(x) \chi_T(x)$, we see that $	ext{spar}(f|_{J|T})$ is at most the number of non-empty buckets $B_T$ for non-empty $T$. Hence

$$\text{spar}(f|_{J|T}) = \text{number of non-empty } T \text{ with non-empty bucket } B_T$$

$$= \frac{1}{2} \sum_{T \subseteq \tilde{S}, T \neq \emptyset} 2$$

$$\leq \frac{1}{2} \sum_{T \subseteq \tilde{S}, T \neq \emptyset} |B_T|$$

$$\leq \frac{1}{2} \text{spar}(f).$$