Electronic Colloquium on Computational Complexity, Report No. 1 (2022)

On (Simple) Decision Tree Rank

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

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

Abstract

In the decision tree computation model for Boolean functions, the depth corresponds to query complexity, and size corresponds to storage space. The depth measure is the most well-studied one, and is known to be polynomially related to several non-computational complexity measures of functions such as certificate complexity. The size measure is also studied, but to a lesser extent. Another decision tree measure that has received very little attention is the minimal rank of the decision tree, first introduced by Ehrenfeucht and Haussler in 1989. This measure is 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 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. 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 observe that the size-rank relationship for decision trees, obtained by Ehrenfeucht and Haussler, is tight upto constant factors.

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

Keywords and phrases Boolean functions, Decision trees, certificate complexity, rank

1 Introduction

The central problem in Boolean function complexity is to understand exactly how hard it is to compute explicit functions. The hardness naturally depends on the computation model to be used, and depending on the model, several complexity measures for functions have been studied extensively in the literature. To name a few – size and depth for circuits and formulas, size and width for branching programs, query complexity, communication complexity, length for span programs, and so on. All of these are measures of the computational hardness of a function. There are also several ways to understand hardness of a function intrinsically, independent of a computational model. For instance, the sensitivity of a function, its certificate complexity, the sparsity of its Fourier spectrum, its degree and approximate degree, stability, and so on. Many bounds on computational measures are obtained by directly relating them to appropriate intrinsic complexity measures. See [10] 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 $\mathrm{Depth}(f)$, 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, $\mathrm{DTSize}(f)$, 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 [6]; see also [1]. In general, the

ISSN 1433-8092

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 [7]. The rank of a Boolean function f, denoted $\mathrm{Rank}(f)$, 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 [6], and later simplified in [4], shows that constant-rank decision trees are efficiently learnable in Valiant's PAC learning framework [18]. Subsequently, the rank measure has played an important role in understanding the decision tree complexity of search problems over relations [14, 8, 11] – 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 $\mathrm{Depth}(f)$, $\mathrm{Rank}(f)$ is not polynomially related with sensitivity or to certificate complexity $\mathrm{C}(f)$, although it is bounded above by $\mathrm{Depth}(f)$. 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 $\textsc{Parity}_n$ function is significantly harder than the $\textsc{And}_n$ function. But both of them have the same Depth, n. However, Rank does reflect this difference in hardness, with $\mathrm{Rank}(\textsc{And}_n) = 1$ and $\mathrm{Rank}(\textsc{Parity}_n) = n$. On the other hand, rank is also already known to characterise the logarithm of decision tree size (DTSize), upto a $\log n$ multiplicative factor. Thus lower bounds on rank give lower bounds on the space required to store a decision tree explicitly. (However, the $\log n$ factor is crucial; there is no dimension-free characterisation. Consider e.g. $\log \mathrm{DTSize}(\textsc{And}_n) = \Theta(\log n)$.)

Our main findings can be summarised as follows:

1. $\mathrm{Rank}(f)$ is equal to the value of the Prover-Delayer game of Pudlák and Impagliazzo [14] played on the corresponding relation R_f. (This is implicit in earlier literature [11, 8].)
2. $\mathrm{Rank}(f)$ is bounded between the minimum certificate complexity of f at any point, and $(\mathrm{C}(f) - 1)^2 + 1$; Theorem 5.6. The upper bound (Lemma 5.2) is an improvement on the bound inherited from $\mathrm{Depth}(f)$, and is obtained by adapting that construction.
3. For a composed function $f \circ g$, $\mathrm{Rank}(f \circ g)$ is bounded above and below by functions of $\mathrm{Depth}(f)$ and $\mathrm{Rank}(g)$; 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 [13].
4. The relation between $\mathrm{Rank}(f)$ and $\mathrm{DTSize}(f)$ from [6] is tight, Section 7. In particular, for the $\textsc{Tribes}$ function, the $\log n$ multiplicative factor is necessary.

By calculating the exact rank for specific functions, we show that all the bounds we obtain on rank are tight.

Related work.

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 [12]) translate to decision tree rank lower bounds, and hence by [6] to decision tree size lower bounds.

In [16], 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 [17], rank of linear decision trees is used in obtaining non-trivial upper bounds on depth-2 threshold circuit size.

In [14], 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 [14] that the size of any tree-like resolution refutation of F is at least $2^{\text{Value}(F)}$. Subsequently, the results of [11, 8] yield the equivalence Value(F) = Rank(F), where Rank(F) is defined to be the minimal rank of the tree underlying a tree-like resolution refutation of F. (Establishing this equivalence uses refutation-space and tree pebbling as intermediaries.) The relevance here is because there is an immediate, and well-known, connection to decision trees for search problems over relations: tree-like resolution refutations are decision trees for the corresponding search CNF problem. (See Lemma 7 in [2]). Note that the size lower bound from [14], and the rank-value equivalence from [11, 8], 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 [3], 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 [14] in Section 3, and observe that its value equals the rank of the function. We also describe the asymmetric game from [3]. We compute the rank of some simple functions in Section 4. In Section 5, we describe the relation between rank and certificate complexity. In Section 6, we present results concerning composed functions. Section 7 examines the size-rank relationship for the Tribes function. The bounds in Sections 4–7 are all obtained by direct inductive arguments/decision tree constructions. They can also be stated using the equivalence of the game value and rank – while this does not particularly simplify the proofs, it changes the language of the proofs and may be more accessible to the reader already familiar with that setting. Hence we include such game-based arguments for our results in Section 8.

2 Preliminaries

Decision trees

For a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, a decision tree computing f is a binary tree with internal nodes labeled by the variables and the leaves labelled by $\{0,1\}$. To evaluate a function on an unknown input, the process starts at the root of the decision tree and works down the tree, querying the variables at the internal nodes. If the value of the query is 0, the process continues in the the left subtree, otherwise it proceeds in the right subtree. The label of the leaf so reached is the value of the function on that particular input. A decision tree is said to be reduced if no variable is queried more than once on any root-to-leaf path. Without loss of generality, any decision tree can be reduced, so in our discussion, we will only consider reduced decision trees. The depth Depth(T) of a decision tree T is the length of the longest root-to-leaf path, and its size DTSize(T) is the number of leaves. The decision tree complexity or the depth of f, denoted by Depth(f), is defined to be the minimum depth of a decision tree computing f. Equivalently, 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 DTSize(f),

is defined similarly i.e. the minimum size of a decision tree computing f. Since decision trees can be reduced, $\mathrm{Depth}(f) \leq n$ and $\mathrm{DTSize}(f) \leq 2^n$ for every n-variate function f. A function is said to be evasive if its depth is maximal, $\mathrm{Depth}(f) = n$.

Weighted decision trees

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

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

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

Certificate Complexity

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

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

Using this definition, we can define several measures.

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

Composed functions

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

Symmetric functions

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

▶ **Proposition 2.2.** *For every non-constant symmetric boolean function* $f : \{0,1\}^n \longrightarrow \{0,1\}$,
1. *f is evasive (has $Depth(f) = n$). (See eg. Lemma 14.19 [10].)*
2. *Hence, for any weights w_i, $Depth_w(f, [w_1, w_2, ..., w_n])) = \sum_i w_i$.*

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

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

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

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

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, $\mathrm{Rank}(u) = 0$. For an internal node u with children v, w,

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

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

▶ **Proposition 2.3.** *For any binary tree T,*
1. *(Rank and Size relationship): $Rank(T) \le \log(DTSize(T)) \le Depth(T)$.*
2. *(Monotonicity of the Rank): Let T' be any subtree of T, and let T'' be an arbitrary binary tree of higher rank than T'. If T' is replaced by T'' in T, then the rank of the resulting tree is not less than the rank of T.*
3. *(Leaf Depth and Rank): If all leaves in T have depth at least r, then $Rank(T) \ge r$.*

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

From Proposition 2.3(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 2.4.** *(Rank of a subfunction): Let f_S be a subfunction obtained by fixing the values of variables in some set $S \subseteq [n]$ of f. Then $Rank(f_S) \le Rank(f)$.*

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

▶ **Proposition 2.5** (Lemma 1 [6]). *For a non-constant Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$,*

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

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

▶ **Proposition 2.6** (Lemma C.6 [1]). *For symmetric Boolean function* $f : \{0,1\}^n \longrightarrow \{0,1\}$, $Rank(f) = n - Gap(f)$.

▶ Remark 2.7. For (simple) deterministic possibly weighted decision trees, each of the measures DTSize, Depth, and Rank, is the same for a Boolean function f, its complement $\neg f$, and its dual f^d.

Relations and Search problems

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

A Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ naturally defines a complete relation R_f over $X = \{0,1\}^n$ and $W = \{0,1\}$, with $R_f = \{(x, f(x)) \mid x \in X\}$, and Rank$(f)$ = 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 [14]. As mentioned in Section 1, the game was originally described for searchCNF problems on unsatsifiable clause sets. The appropriate analog for a Boolean function f, or its relation R_f, and even for arbitrary X-complete relations $R \subseteq X \times W$, is as follows:

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

▶ **Theorem 3.1** (implied from [14, 11, 8]). *For any X-complete relation $R \subseteq X \times W$, where $X = \{0,1\}^n$, $Rank(R) = Value(R)$. In particular, for a boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, $Rank(f) = Value(R_f)$.*

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

▶ **Lemma 3.2** (implicit in [11]). *For an X-complete relation $R \subseteq \{0,1\}^n \times W$, in the Prover-Delayer game, the Prover has a strategy which restricts the Delayer's score to at most $Rank(R)$ points.*

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

We claim that such a "tree-based" strategy restricts the Delayer's score to $\mathrm{Rank}(T)$ points. The proof is by induction on $\mathrm{Depth}(T)$.

1. Base Case: $\mathrm{Depth}(T) = 0$. 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: $\mathrm{Depth}(T) \geq 1$. 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, $\mathrm{Depth}(T_b) \leq \mathrm{Depth}(T) - 1$, and T_b is a decision tree for the search problem on $R_{i,b} \triangleq \{(x, w) \in R \mid x_i = b\} \subseteq X_{i,b} \times W$, where $X_{i,b} = \{x \in X \mid x_i = b\}$.
   If the Delayer responds with a bit b, then by induction, the subsequent score of the Delayer is limited to $\mathrm{Rank}(T_b) \leq \mathrm{Rank}(T)$. Since the current round does not increase the score, the overall Delayer score is limited to $\mathrm{Rank}(T)$.
   If the Delayer defers the decision to Prover, the Delayer gets one point in the current round. Subsequently, by induction, the Delayer's score is limited to $\min(\mathrm{Rank}(T_0), \mathrm{Rank}(T_1))$; by definition of rank, this is at most $\mathrm{Rank}(T) - 1$. So the overall Delayer score is again limited to $\mathrm{Rank}(T)$.

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

▶ **Lemma 3.3** (implicit in [8]). *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 $\mathrm{Rank}(R)$ points.*

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

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

1. Base Case: $n = 1$. If $\mathrm{Rank}(R) = 1$, then the prover must query the variable, and the Delayer strategy defers the choice, scoring one point.
2. Induction Step: $n > 1$. Let x_i be first variable queried by the prover.
   If $\mathrm{Rank}(R_{i,0}) = \mathrm{Rank}(R_{i,1})$, then the Delayer defers, scoring one point in this round. Subsequently, suppose the Prover sets x_i to b. The game is now played on $R_{i,b}$, and by induction, the Delayer can subsequently score at least $\mathrm{Rank}(R_{i,b})$ points. But also, because of the equality, we have $\mathrm{Rank}(R) \leq 1 + \mathrm{Rank}(R_{i,b})$, as witnessed by a decision tree that first queries x_i and then uses rank-optimal trees on each branch. Hence the overall Delayer score is at least $\mathrm{Rank}(R)$.
   If $\mathrm{Rank}(R_{i,b}) > \mathrm{Rank}(R_{i,1-b})$, then the Delayer chooses $x_i = b$ and the subsequent game is played on $R_{i,b}$. The subsequent (and hence overall) score is, by induction, at least $\mathrm{Rank}(R_{i,b})$. But $\mathrm{Rank}(R) \leq \mathrm{Rank}(R_{i,b})$, as witnessed by a decision tree that first queries x_i and then uses rank-optimal trees on each branch.

◀

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

In [3], an aysmmmetric version of this game is defined. In each round, the Prover queries a variable x, the Delayer specifies values $p_0, p_1 \in [0, 1]$ adding up to 1, the Prover picks a value b, the Delayer adds $\log \frac{1}{p_b}$ to his score. Let ASym-Value denote the maximum score the Delayer can always achieve, independent of the Prover moves. Note that $\text{ASym-Value}(R) \geq \mathrm{Value}(R)$;

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 [3], 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 [3]). *For any X-complete relation $R \subseteq X \times W$, where $X = \{0,1\}^n$, $\log DTSize(R) = ASym\text{-}Value(R)$. In particular, for a boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$, $\log DTSize(f) = ASym\text{-}Value(R_f)$.*

(In [3], 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

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

4 The Rank of some natural functions

For symmetric functions, rank can be easily calculated using Proposition 2.6. In Table 1 we tabulate various measures for some standard symmetric functions. As can be seen from the $\textsc{Or}_n$ and $\textsc{And}_n$ functions, the $\mathrm{Rank}(f)$ measure is not polynomially related with the measures $\mathrm{Depth}(f)$ or certificate complexity $\mathrm{C}(f)$.

f	Depth	C_0	C_1	C	Gap	Rank
0 or 1	0	0	0	0	n	0
$\textsc{And}_n$	n	1	n	n	$n-1$	1
$\textsc{Or}_n$	n	n	1	n	$n-1$	1
$\textsc{Parity}_n$	n	n	n	n	0	n
$\textsc{Maj}_{2k}$	$2k$	k	$k+1$	$k+1$	k	k
$\textsc{Maj}_{2k+1}$	$2k+1$	$k+1$	$k+1$	$k+1$	k	$k+1$
$\textsc{Thr}_n^k$ $(k \geq 1)$	n	$n-k+1$	k	$\max\left\{\begin{matrix} n-k+1, \\ k \end{matrix}\right\}$	$\max\left\{\begin{matrix} k-1, \\ n-k \end{matrix}\right\}$	$n - \mathrm{Gap}$

Table 1 Some simple symmetric functions and their associated complexity measures

For two composed functions that will be crucial in our later discussions, we can directly calculate the rank as described below. (The rank can also be caluclated using Theorem 3.1; see Section 8.)

▶ **Theorem 4.1.** *For every $n \geq 1$,*

1. $Rank(\textsc{Tribes}_{n,m}) = Rank(\textsc{Tribes}^d_{n,m}) = n$ *for* $m \geq 2$.
2. $Rank(\textsc{And}_n \circ \textsc{Parity}_m) = n(m-1)+1$ *for* $m \geq 1$.

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

▶ **Proposition 4.2.** *(Composition of Rank): Let T be a rooted binary tree with depth ≥ 1, rank r, and with leaves labelled by 0 and 1. Let T_0, T_1 be arbitrary rooted binary trees of ranks r_0, r_1 respectively. For $b \in \{0,1\}$, attach T_b to each leaf of T labeled b, to obtain rooted binary tree T' of rank r'.*

1. $r' \leq r + \max\{r_0, r_1\}$. *Furthermore, if T is a complete binary tree, and if $r_0 = r_1$, then this is an equality; $r' = r + r_0$.*
2. *If every non-trivial subtree (more than one leaf) of T has both a 0 leaf and a 1 leaf, then $r' \geq r + \max\{r_0, r_1\} - 1$. If, furthermore, T is a complete binary tree, then this is an equality when $r_0 \neq r_1$,*

Proof. The upper bound on r' follows from the definition of rank when $r_0 = r_1$, in which case it also gives equality for complete T. When $r_0 \neq r_1$, it follows from Proposition 2.3(2).

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

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

For the inductive step, let $\text{Depth}(T) = k > 1$. Let v be the root of T, and let T_ℓ, T_r be its left and right sub-trees respectively, with ranks r_ℓ and r_r respectively. Both $\text{Depth}(T_\ell)$ and $\text{Depth}(T_r)$ are at most $k - 1$, and at least one of these is exactly $k - 1 \geq 1$. Also, at least one of r_ℓ, r_r is non-zero.

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

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

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

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

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

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

▶ **Lemma 4.3.** *For every $n, m \geq 1$, Rank($\textsc{Tribes}^d_{n,m}$) $\leq n$.*

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

▶ **Lemma 4.5.** *For every $n \geq 1$ and $m \geq 2$, Rank($\textsc{Tribes}^d_{n,m}$) $\geq n$.*

Proof. We prove this by induction on n. The base case, $n = 1$, is straightforward: $\textsc{Tribes}^d_{1,m}$ is the function $\textsc{Or}_m$, whose rank is 1.

For the inductive step, let $n > 1$, and consider any decision tree Q for $\textsc{Tribes}^d_{n,m}$. 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 $\text{AND}_n \circ (\text{OR}_{m-1}, \text{OR}_m, ..., \text{OR}_m)$, and Q_1 computes $\text{TRIBES}^d_{n-1,m}$, on appropriate variables. For $m \geq 2$, $\text{TRIBES}^d_{n-1,m}$ is a sub-function of $\text{AND}_n \circ (\text{OR}_{m-1}, \text{OR}_m, ..., \text{OR}_m)$, and so Proposition 2.4 implies that $\text{Rank}(Q_0) \geq \text{Rank}(\text{AND}_n \circ (\text{OR}_{m-1}, \text{OR}_m, ..., \text{OR}_m)) \geq \text{Rank}(\text{TRIBES}^d_{n-1,m})$. By induction, $\text{Rank}(Q_1) \geq \text{Rank}(\text{TRIBES}^d_{n-1.m}) \geq n-1$. 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 $\text{TRIBES}^d_{n,m}$, we conclude that $\text{Rank}(\text{TRIBES}^d_{n,m}) \geq n$, 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 by Remark 4.4.

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

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

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

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

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

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

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

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

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

1. Base Case: $M = n$. Each m_i is equal to 1. So h is the conjunction of n literals on distinct variables. (A literal is a variable or its negation.) Hence $\text{Rank}(h) = \text{Rank}(\text{AND}_n) = 1$.
2. Inductive Step: $M > n > 1$. Consider any decision tree Q computing h. Without loss of generality (by renaming variables if necessary), let $x_{1,1}$ be the variable queried at the root node. Let Q_0 and Q_1 be the left and the right subtrees of Q. For $b \in \{0,1\}$, let g_{1b} denote the function g_1 restricted to $x_{1,1} = b$. Then Q_b computes the function $\text{AND}_n \circ (g_{1b}, g_2, \ldots, g_n)$ on appropriate variables.

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

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

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

Using induction on M, we obtain

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

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

◀

5 Relation between Rank and Certificate Complexity

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

▶ **Proposition 5.1** ([5],[9],[15], see also Theorem 14.3 in [10]). *For every boolean function* $f : \{0,1\}^n \longrightarrow \{0,1\}$,

$$C(f) \leq \mathit{Depth}(f) \leq C_0(f)C_1(f)$$

Both these inequalities are tight; the first for the OR and AND functions, and the second for the $\textsc{Tribes}_{n,m}$ and $\textsc{Tribes}^d_{n,m}$ functions. (For $\textsc{Tribes}^d_{n,m}$, $\mathrm{C}_0(\textsc{Tribes}^d_{n,m}) = m$, $\mathrm{C}_1(\textsc{Tribes}^d_{n,m}) = n$ and $\mathrm{Depth}(\textsc{Tribes}^d_{n,m}) = nm$, see e.g. Exercise 14.1 in [10].)

Since Rank $\leq$ Depth, the same upper bound also holds for Rank as well. But it is far from tight for the $\textsc{Tribes}_{n,m}$ function. In fact, the upper bound can be improved in general. Adapting the construction given in the proof of Proposition 5.1 slightly, we show the following.

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

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

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

Proof. The inequality holds trivially for constant functions since for such functions, Rank $=$ $\mathrm{C}_0 = \mathrm{C}_1 = 0$. So assume f is not constant. The proof is by induction on $\mathrm{C}_1(f)$.

Base Case: $\mathrm{C}_1(f) = 1$. 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 $\mathrm{Rank}(g) \leq (\mathrm{C}_0(g) - 1)(\mathrm{C}_1(g) - 1) + 1$ is true for all g with $\mathrm{C}_1(g) \leq k$. Let f satisfy $\mathrm{C}_1(f) = k + 1$. Pick an $a \in f^{-1}(0)$ and a mimimum-size 0-certificate S for a. Without loss of generality, assume that $S = \{x_1, x_2, \ldots, x_\ell\}$ for some $\ell = |S| \leq \mathrm{C}_0(f)$. Now, take a complete decision tree T_0 of depth l on these l variables. Each of its leaves corresponds to the unique input $c = (c_1, c_2, ..., c_l) \in \{0,1\}^l$ reaching this leaf. At each such leaf, attach a minimal rank decision tree T_c for the subfunction $f_c \triangleq f(c_1, c_2, ..., c_l, x_{l+1}, ..., x_n)$. 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, $\mathrm{C}_0(f_c) \leq \mathrm{C}_0(f)$ since certificate size cannot increase by assigning some variables. Further, $\mathrm{C}_1(f_c) \leq \mathrm{C}_1(f) - 1$; this because of the well-known fact (see e.g. [10]) 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 $f_c \neq 0$, $\mathrm{Rank}(T_c) \leq (\mathrm{C}_0(f_c) - 1)(k-1) + 1 \leq (\mathrm{C}_0(f) - 1)(k-1) + 1$. Thus T is obtained from a rank-ℓ

tree T_0 (with $\ell \leq \mathrm{C}_0(f)$) by attaching a tree of rank 0 to at least one leaf, and attaching trees of rank at most $(\mathrm{C}_0(f)-1)(k-1)+1$ to all leaves. From Proposition 4.2, we conclude that $\mathrm{Rank}(f) \leq \mathrm{Rank}(T) \leq ((\mathrm{C}_0(f)-1)(k-1)+1)+(l-1) \leq (\mathrm{C}_0(f)-1)(\mathrm{C}_1(f)-1)+1$. ◀

From Theorem 4.1, we see that the lower bound on Depth in Proposition 5.1 does not hold for Rank; for $m > n$, $\mathrm{Rank}(\textsc{Tribes}^d_{n,m}) = n < m = C(\textsc{Tribes}^d_{n,m})$. However, $\min\{\mathrm{C}_0(\textsc{Tribes}^d_{n,m}), \mathrm{C}_1(\textsc{Tribes}^d_{n,m})\} = n = \mathrm{Rank}(\textsc{Tribes}^d_{n,m})$. Further, for all the functions listed in Table 1, $\mathrm{Rank}(f)$ is at least as large as $\min\{\mathrm{C}_0(f), \mathrm{C}_1(f)\}$. However, even this is not a lower bound in general.

▶ **Lemma 5.3.** *$\min\{C_0(f), C_1(f)\}$ is not a lower bound on Rank(f); for the symmetric function $f = \textsc{Maj}_n \vee \textsc{Parity}_n$, when $n > 4$, Rank$(f) < \min\{C_0(f), C_1(f)\}$.*

Proof. Let f be the function $\textsc{Maj}_n \vee \textsc{Parity}_n$, for $n > 4$. Then $f(0^n) = 0$ and $\mathrm{C}_0(f, 0^n) = n$, and $f(10^{n-1}) = 1$ and $\mathrm{C}_1(f, 10^{n-1}) = n$. Also, f is symmetric, with $\mathrm{Gap}(f) = n/2$, so by Proposition 2.6, $\mathrm{Rank}(f) = n/2$. ◀

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

▶ **Lemma 5.4.** *Average certificate complexity is neither a upper bound nor a lower bound on the rank of a function; there exist functions f and g, such that Rank$(f) < C_{avg}(f)$ and $C_{avg}(g) <$ Rank(g).*

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

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

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

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

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

▶ **Lemma 5.5.** *For every Boolean function f, $C_{\min}(f) \leq$ Rank(f). This is tight for $\textsc{Or}_n$.*

Proof. Let T be a rank-optimal decision tree for f. Since the variables queried in any root-to-leaf path in T form a 0 or 1-certificate for f, we know that depth of each leaf in T must be at least $\mathrm{C}_{\min}(f)$. By Proposition 2.3(3), $\mathrm{Rank}(f) = \mathrm{Rank}(T) \geq \mathrm{C}_{\min}(f)$. ◀

Lemma 5.2 and Lemma 5.5 give these bounds sandwiching $\mathrm{Rank}(f)$:

▶ **Theorem 5.6.** *$C_{\min}(f) \leq$ Rank$(f) \leq (C_0(f)-1)(C_1(f)-1)+1 \leq (C(f)-1)^2+1$.*

As mentioned in Proposition 2.6, for symmetric functions the rank is completely characterised in terms of Gap of f. How does Gap relate to certificate complexity for such functions? It turns out that certificate complexity is characterized not by Gap but by $\mathrm{Gap}_{\min}$. Using this relation, the upper bound on $\mathrm{Rank}(f)$ from Lemma 5.2 can be improved for symmetric functions to $\mathrm{C}(f)$.

▶ **Lemma 5.7.** *For every symmetric Boolean function f on n variables, $C(f) = n - Gap_{\min}(f)$ and $n - C(f) + 1 \leq Rank(f) \leq C(f)$. Both the inequalities on rank are tight for* $\textsc{Maj}_{2k+1}$.

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

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

   Thus any certificate for x must have at least $a+(n-b)$ positions; hence $C(f, x) \geq n-(b-a)$.

Since the argument above works for any interval $[a, b]$ where f is constant, we conclude that $\mathrm{C}(f) = n - \mathrm{Gap}_{\min}(f)$.

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

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

As seen from Table 1, these bounds on Rank are tight for $\textsc{Maj}_{2k+1}$. ◀

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

▶ **Lemma 5.8.** *Certificate Complexity does not always bound Rank from above; for the function $f = \textsc{Maj}_{2k+1} \circ \textsc{Maj}_{2k+1}$, $C(f) < Rank(f)$.*

The proof is deferred to Section 6, where we develop techniques to bound the rank of composed functions. We also give, in Section 8, a proof based on the Prover-Delayer game characterisation from Theorem 3.1.

6 Rank of 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** ([13]). *For Boolean functions f, g, $Depth(f \circ g) = Depth(f)Depth(g)$.*

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

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

Proof. Let $f = \textsc{And}_n$ and $g = \textsc{Or}_n$. Then $\mathrm{Rank}(f) = \mathrm{Rank}(g) = 1$. But $\mathrm{Rank}(f \circ g) = \mathrm{Rank}(\textsc{Tribes}^d_n) = n$, 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 $\mathrm{Depth}(T_f)\mathrm{Rank}(T_g) = \mathrm{Depth}_w(f, [r, r, \ldots, r])$, where $r = \mathrm{Rank}(T_g)$. (This is the construction used in the proofs of Lemmas 4.3 and 4.6, where further properties of the PARITY function are used to show that the resulting tree's rank is even smaller than $\mathrm{Depth}(f)\mathrm{Rank}(g)$.) In fact, we show below (Theorem 6.3) that this holds more generally, when different functions are used in the composition. While this is a relatively straightforward generalisation here, it is necessary to consider such compositions for the lower bound we establish further on in this section.

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

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

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

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

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

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

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

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

The really interesting question, however, is whether we can show a good lower bound for the rank of a composed function. This will help us understand how good is the upper bound in Theorem 6.3. To begin with, note that for non-constant Boolean functions f, g, both f and g are sub-functions of $f \circ g$. Hence Proposition 2.4 implies the following.

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

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

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

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

$$\begin{aligned} Rank(f \circ (g_1, g_2, ..., g_n)) &\geq Depth_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 \\ &\geq Depth_w(f, [r_1, r_2, ..., r_n]) - (n - 1). \end{aligned}$$

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 $\text{Depth}_w(f, [r_1, \ldots, r_n]) \leq \text{Depth}_w(T, [r_1, \ldots, r_n]) \leq \text{Depth}_w(T, [r_1 - 1, \ldots, r_n - 1]) + n = \text{Depth}_w(f, [r_1 - 1, \ldots, r_n - 1]) + n$, 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 $f \circ (g_1, g_2, ..., g_n)$. For $i \in [n]$, let m_i be the arity of g_i. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m_i}$ the ith block of variables of h; g_i is evaluated on this block.

In the base case, $n = 1$. Since f is non-constant, f can either be x or $\neg x$; accordingly, h is either g_1 or $\neg g_1$. So $D_w(f, [r_1 - 1]) = r_1 - 1$ and $\text{Rank}(h) = \text{Rank}(g_1) = r_1$, 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 $D_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) + 1 = D_w(f, [0, 0, ..., 0]) + 1 = 1$. Since all r_i's are 1, each g_i's is either $x_{i,1}$ or $\neg x_{i,1}$, Thus h is the same as f upto renaming of the literals. Hence $\text{Rank}(h) = \text{Rank}(f) \geq 1$.

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

- Case 1: Both g_1^0 and g_1^1 are constant functions. Since g_1 is non-constant, $g_1^0 \neq g_1^1$, and $r_1 = \text{Rank}(g_1) = 1$. Assume that $g_1^0 = 0$ and $g_1^1 = 1$; the argument for the other case is identical. For $b \in \{0, 1\}$, let f_b be the function $f(b, x_2, \ldots, x_n)$; then $h_b = f_b \circ (g_2, \ldots, g_n)$. View f_b as functions on $n - 1$ variables.
  - Case 1a: Both f_0 and f_1 are constant functions. Then f is either x_1 or $\neg x_1$, so $\text{Depth}_w(f, [r_1 - 1, r_2 - 1, ..., r_n - 1]) = \text{Depth}_w(f, [0, r_2 - 1, ..., r_n - 1]) = 0$. Also, in this case, h is either $x_{1,1}$ or $\neg x_{1,1}$, so $\text{Rank}(h) = 1$. 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 $w_2, \ldots, w_n$, $D_w(f, [0, w_2, ..., w_n]) \leq D_w(f_1, [w_2, ..., w_n])$: we can obtain a decision tree for f witnessing this by first querying x_1, making the $x_1 = 0$ child a leaf labeled f_0, and attaching the optimal tree for f_1 on the $x_1 = 1$ branch. Second, note that since f_1 and all g_i are non-constant, so is h_1. Now

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

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

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

- Case 2: One of g_1^0 and g_1^1 is a constant function; assume without loss of generality that g_1^0 be constant. In this case, we can conclude that $\mathrm{Rank}(g_1) = \mathrm{Rank}(g_1^1)$: $\mathrm{Rank}(g_1^1) \leq \mathrm{Rank}(g_1)$ by Proposition 2.4, and $\mathrm{Rank}(g_1) \leq \mathrm{Rank}(g_1^1)$ as witnessed by a decision tree for g_1 that queries $x_{1,1}$ first, sets the $x_{1,1} = 0$ branch to a leaf labeled g_1^0, and attaches an optimal tree for g_1^1 on the other branch. Now

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

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

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

$$\begin{aligned}
\mathrm{Rank}(h_b) &= \mathrm{Rank}(f \circ (g_1^b, g_2, \ldots, g_n)) \\
&\geq \mathrm{Depth}_w(f, [r_1^b - 1, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{by induction on } M \\
&= \mathrm{Depth}_w(f, [r_1 - 2, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{since } r_1 - 1 = r_1^b. \\
\text{Hence } \mathrm{Rank}(h) &\geq 1 + \min_b \mathrm{Rank}(h_b) \\
&\geq \mathrm{Depth}_w(f, [r_1 - 2, r_2 - 1, \ldots, r_n - 1]) + 2 && \text{derivation above} \\
&\geq \mathrm{Depth}_w(f, [r_1 - 1, r_2 - 1, \ldots, r_n - 1]) + 1 && \text{by Fact 2.1}
\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

Using Theorem 6.6, we can now complete the proof of Lemma 5.8.

Proof. (of Lemma 5.8) Consider the composed function $f = \textsc{Maj}_{2k+1} \circ \textsc{Maj}_{2k+1}$. Note that from the lower bound in Theorem 6.6, and the entries in Table 1, $\mathrm{Rank}(\textsc{Maj}_{2k+1} \circ \textsc{Maj}_{2k+1}) \geq (2k+1)k + 1$. On the other hand, it is straightforward to verify that $\mathrm{C}(f) = (k+1)^2$. Thus for $k > 1$, $\mathrm{Rank}(f) > \mathrm{C}(f)$. ◀

7 Tightness of Rank and Size relation

In Proposition 2.5, we saw a relation between rank and size. The relationship is essentially tight. The function $f = \textsc{Parity}_n$ witnesses the tightness of both the inequalities. Since $\mathrm{Rank}(\textsc{Parity}) = n$, Proposition 2.5 tells us that $\log \mathrm{DTSize}(\textsc{Parity})$ lies in the range $[n, n \log e]$, and we know that $\log \mathrm{DTSize}(\textsc{Parity}) = n$.

For the $\textsc{Tribes}_n$ function, which has $N = n^2$ variables, we know from Theorem 4.1 that $\mathrm{Rank}(\textsc{Tribes}_n) = n$. Thus Proposition 2.5 tells us that $\log \mathrm{DTSize}(\textsc{Tribes}_n)$ lies in the range $[n, n \log(en)]$. (See also Exercise 14.9 [10] for a direct argument showing $n \leq \log \mathrm{DTSize}(\textsc{Tribes}_n)$). But that still leaves a $(\log(en))$-factor gap between the two quantities. We show that the true value is closer to the upper end. To do this, we establish a stronger size lower bound for decision trees computing $\textsc{Tribes}_n^d$.

▶ **Lemma 7.1.** *For every $n, m \geq 1$, every decision tree for $\textsc{Tribes}_{n,m}^d$ has at least m^n 1-leaves and n 0-leaves.*

Proof. Recall that $\textsc{Tribes}^d_{n,m} = \bigwedge_{i\in[n]} \bigvee_{j\in[m]} x_{i,j}$. We call $x_{i,1}, x_{i,2}, \ldots, x_{i,m}$ the ith block of variables. We consider two special kinds of input assignments: 1-inputs of minimum Hamming weight, call this set S_1, and 0-inputs of maximum Hamming weight, call this set S_0. Each $a \in S_1$ has exactly one 1 in each block; hence $|S_1| = m^n$. Each $b \in S_0$ has exactly m zeroes, all in a single block; hence $|S_0| = n$. We show that in any decision tree T for $\textsc{Tribes}^d_{n,m}$, 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 $\textsc{Tribes}^d_{n,m}(a') = 0$. 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 $\textsc{Tribes}^d_{n,m}(a'') = 1$. 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 2.5 is also essentially tight for the $\textsc{Tribes}^d_n$ function.

The size lower bound from Lemma 7.1 can also be obtained by specifying a good Delayer strategy in the asymmetric Prover-Delayer game and invoking Proposition 3.4.; see Section 8.

8 Proofs using Prover-Delayer Games

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

Prover strategy for $\textsc{Tribes}_{n,m}$, proving Lemma 4.3

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

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

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

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

Delayer strategy for $\textsc{Tribes}_{n,m}$, proving Lemma 4.5

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

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

Note that with this strategy, the Delayer ensures that the game ends with function value 1. (No row has all variables set to 0.) Observe that to certify a 1-input of the function, the Prover must query at least one variable in each row. Since $m \geq 2$, the Delayer gets to score at least one point per row, and thus has a score of at least n at the end of the game.

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

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

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

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

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

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

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

Prover strategy in terms of certificate complexity, proving Lemma 5.2

We give a Prover strategy which restricts the Delayer to $(\text{C}_0(f)-1)(\text{C}_1(f)-1)+1$ points. Let $\tilde{f}$ be the function obtained by assigning values to the variables queried so far. As long as $\text{C}_1(\tilde{f}) > 1$, Prover picks an $a \in \tilde{f}^{-1}(0)$ and its 0-certificate S, and queries all the variables in S one by one. If at any point the Delayer defers a decision to the Prover, the Prover chooses the value according to a. When $\text{C}_1(\tilde{f})$ becomes 1, the Prover picks an $a \in \tilde{f}^{-1}(1)$ and its 1-certificate $\{i\}$ and queries the variable x_i. If the Delayer defers the decision, the Prover chooses a_i.

The above strategy restricts the Delayer to $(\text{C}_0(f)-1)(\text{C}_1(f)-1)+1$ points; the proof is essentially same as Lemma 5.2.

Delayer strategy for $f = \text{MAJ}_{2k+1} \circ \text{MAJ}_{2k+1}$, proving Lemma 5.8

The following Delayer strategy always scores $(k+1)^2+k^2$ points, greater than $C(f) = (k+1)^2$.

At an intermediate stage of the game, say that a row is b-determined if the variables that are already set in this row already fix the value of MAJ_{2k+1} on this row to be b, and is determined if it is b-determined for some b. Let M_b be the number of b-determined rows. If the game has not yet ended, then $M_0 \leq k$ and $M_1 \leq k$.

On query $x_{i,j}$, let n_0, n_1 be the number of variables in row i already set to 0 and to 1 respectively. The Delayer defers the decision if

- row i is already determined, or
- $n_0 = n_1 < k$, or
- $n_0 = n_1 = k$ and $M_0 = M_1$.

Otherwise, if $n_0 \neq n_1$, then the Delayer chooses the value b where $n_b < n_{1-b}$. If $n_0 = n_1 = k$, then the Delayer chooses the value b where $M_b < M_{1-b}$.

This strategy ensures that at all stages until the game ends, $|M_0 - M_1| \leq 1$, and furthermore, in all rows that are not yet determined, $|n_0 - n_1| \leq 1$. Thus a row is determined only after all variables in the row are queried, and the Delayer gets a point for every other query, making a total of k points per determined row. Further, for $k+1$ rows, the Delayer also gets an additional point on the last queried variable. The game cannot conclude before all $2k+1$ rows are determined, so the Delayer scores at least $(k+1)^2 + k^2$ points.

Prover and Delayer strategies for composed functions, proving Theorem 6.6

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

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

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

Delayer strategy in asymmetric game in $\textsc{Tribes}^d_n$, proving Lemma 7.1

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

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

1. Case 1: The Prover concludes with function value 0. In this case, the Prover must have queried all variables in some row, say the i-th row, and chosen 0 for all of them. For the last variable queried in the i-th row, the Delayer would have responded with $(p_0, p_1) = (0, 1)$, and hence scored ∞ points in the round and the game.
2. Case 2: The Prover concludes with function value 1. In this case, the Prover must have set a variable to 1 in each row. We show that the Delayer scores at least $\log n$ points per row. Pick a row arbitrarily, and let k be the number of free variables in the row when the first variable in that row is set to 1. The Prover sets $n - k$ variables in this row to 0 before he sets the first variable to 1. For $b \in \{0,1\}$, let $p_{b,j}$ represents the p_b response of the Delayer when there are j free variables in the row. That is, $p_{0,j} = 1 - \frac{1}{j} = \frac{j-1}{j}$ and $p_{1,j} = \frac{1}{j}$. The contribution of this row to the overall score is at least

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

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

9 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.

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

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

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

5 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.

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

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

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

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

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

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

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

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

14 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.

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

16 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.

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

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

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il