Improved Quantum Query Upper Bounds Based on Classical Decision Trees

Arjan Cornelissen[1], Nikhil S. Mande[2], and Subhasree Patro[3]

[1]Simons Institute for the Theory of Computing, University of California, Berkeley, United States of America
[2]University of Liverpool, United Kingdom
[3]Eindhoven University of Technology, Netherlands

We consider the following question in query complexity: Given a classical query algorithm in the form of a decision tree, when does there exist a quantum query algorithm with a speed-up (i.e., that makes fewer queries) over the classical one? We provide a general construction based on the structure of the underlying decision tree, and prove that this can give us an up-to-quadratic quantum speed-up in the number of queries. In particular, our results give a bounded-error quantum query algorithm of cost $O(\sqrt{s})$ to compute a Boolean function (more generally, a relation) that can be computed by a classical (even randomized) decision tree of size s. This recovers an $O(\sqrt{n})$ algorithm for the Search problem, for example.

Lin and Lin [Theory of Computing'16] and Beigi and Taghavi [Quantum'20] showed results of a similar flavor. Their upper bounds are in terms of a quantity which we call the "guessing complexity" of a decision tree. We identify that the guessing complexity of a decision tree equals its *rank*, a notion introduced by Ehrenfeucht and Haussler [Information and Computation'89] in the context of learning theory. This answers a question posed by Lin and Lin, who asked whether the guessing complexity of a decision tree is related to any measure studied in classical complexity theory. We also show a polynomial separation between rank and its natural randomized analog for the complete binary AND-OR tree.

Beigi and Taghavi constructed span programs and dual adversary solutions for Boolean functions given classical decision trees computing them and an assignment of non-negative weights to edges of the tree. We explore the effect of changing these weights on the resulting span program complexity and objective value of the dual adversary bound, and capture the best possible weighting scheme by an optimization program. We exhibit a solution to this program and argue its optimality from first principles. We also exhibit decision trees for which our bounds are asymptotically stronger than those of Lin and Lin, and Beigi and Taghavi. This answers a question of Beigi and Taghavi, who asked whether different weighting schemes in their construction could yield better upper bounds.

Arjan Cornelissen: ajcornelissen@outlook.com
Nikhil S. Mande: mande@liverpool.ac.uk
Subhasree Patro: patrofied@gmail.com

An extended abstract of this work appeared in the proceedings of FSTTCS 2022 [CMP22].

Contents

1 Introduction

In this paper, we address the following question: given a classical algorithm performing a task, along with a description of the algorithm, when can one turn it into a quantum algorithm and obtain a speed-up in the process? Of specific interest to us in this paper is the case when the classical algorithm can be efficiently represented by a decision tree. For instance, consider the problem of identifying the first marked item in a list, say x, of n items, each of which may or may not be marked. The input is initially unknown, and one has access to it via an *oracle*. On being queried $i \in [n]$, the oracle returns whether or not x_i is marked. Moreover, queries can be made in superposition. The goal is to minimize the number of queries in the worst case and output the correct answer with high probability for every possible input list. This problem is closely related to the Search problem, and is known to admit a quadratic quantum speed-up (its classical query complexity is $\Theta(n)$ and quantum query complexity is $\Theta(\sqrt{n})$) [Gro96, DH96, Kot14, LL16]. It is easy to construct a classical decision tree (in fact, a decision list) of depth n and size (which is the number of nodes in the tree) $2n+1$ that solves the above-mentioned problem. In view of the quadratic speed-up that quantum query algorithms can achieve for this problem, this raises the following natural question: Given a classical decision tree of size s that

computes a function, is there a bounded-error quantum query algorithm of cost $O(\sqrt{s})$ that solves the same function? Among other results, we answer this in the affirmative (see Corollary 1.7). We obtain our quantum query upper bounds by constructing explicit *span programs* and *dual adversary solutions* by exploiting the structure of the initial classical decision tree. In the discussions below, let $\mathsf{Q}_\varepsilon(f)$ be the ε-error quantum query complexity of f. When $\varepsilon = 1/3$, we drop the subscript and call $\mathsf{Q}(f)$ the bounded-error quantum query complexity of f.

1.1 Span Programs

Span programs are a computational model introduced by Karchmer and Wigderson [KW93]. Roughly speaking, a span program defines a function depending on whether or not the "target vector" of an input is in the span of its associated "input vectors". Span programs were first used in the context of quantum query complexity by Reichardt and Špalek [RŠ12], and it is known that span programs characterize bounded-error quantum query complexity of Boolean functions up to a constant factor [Rei11, LMR+11]. Span programs have been used to design quantum algorithms for various graph problems such as *st*-connectivity [BR12], cycle detection and bipartiteness testing [Ari15, CMB18], graph connectivity [JJKP18], and has been also used for problems such as formula evaluation [RŠ12, Rei09, JK17]. Recently, Beigi and Taghavi [BT19] defined a variant of span programs (non-binary span programs with orthogonal inputs, abbreviated NBSPwOI) for showing upper bounds on the quantum query complexity of non-Boolean input/output functions $f : [\ell]^n \to [m]$. Moreover in a follow-up work [BT20], they also use non-binary span programs for showing upper bounds for a variety of graph problems, for example, the maximum bipartite matching problem.

1.2 Dual Adversary Bound

The general adversary bound for Boolean functions f (Equation (6)) was developed by Høyer, Lee and Špalek [HLŠ07], and they showed that this quantity gives a lower bound on the bounded-error quantum query complexity of f. It was eventually shown that this bound actually characterizes the bounded-error quantum query complexity of f up to a constant factor [Rei11, LMR+11]. Quantum query algorithms have been developed by explicitly constructing dual adversary solutions, for example for the well-studied k-distinctness problem [Bel12], S-isomorphism and hidden subgroup problems [Bel19], gapped group testing [ABRW16], etc. The general adversary bound can be expressed as a semidefinite program that admits a dual formulation. Thus, feasible solutions to the dual adversary program yield quantum query upper bounds.

1.3 Related Works

Lin and Lin [LL16] showed how a classical algorithm computing a function $f : D_f \to \mathcal{R}$ with $D_f \subseteq \{0,1\}^n$, equipped with an efficient 'guessing scheme', could be used to construct a faster quantum query algorithm computing f. Moreover, their results apply to the setting where $f \subseteq \{0,1\}^n \times \mathcal{R}$ is a relation. A deterministic algorithm computing a relation f takes an input $x \in \{0,1\}^n$ and outputs a value b such that $(x,b) \in f$. More precisely, they showed the following:[1]

[1]For ease of readability, we state the theorem for deterministic decision trees. Lin and Lin actually showed a stronger statement that holds even if one considers randomized decision trees and randomized guessing algorithms.

Theorem 1.1 (Lin and Lin [LL16, Theorem 5.4]). *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Let $\mathcal{A}$ be a deterministic algorithm computing f that makes at most T queries. Let $x_{p_1}, \ldots, x_{p_{\tilde{T}(x)}}$ be the query results of $\mathcal{A}$ on an a priori unknown input string x. For $x \in \{0,1\}^n$, let $\tilde{T}(x) \leq T$ denote the number of queries that $\mathcal{A}$ makes on input x. Suppose there is another deterministic algorithm $\mathcal{G}$ which takes as input $x_{p_1}, \ldots, x_{p_{t-1}} \in \{0,1\}^{t-1}$ for any $t \in [T]$, and outputs a guess for the next query result of $\mathcal{A}$. Assume that $\mathcal{G}$ makes at most G mistakes for all x. That is,*

$$\forall x \in \{0,1\}^n, \quad \sum_{t=1}^{\tilde{T}(x)} |\mathcal{G}(x_{p_1}, \ldots, x_{p_{t-1}}) - x_{p_t}| \leq G.$$

Then,

$$\mathsf{Q}(f) = O(\sqrt{TG}).$$

Lin and Lin provided two proofs of the above theorem, one of which involved constructing an explicit quantum query algorithm. This algorithm is iterative, and works as follows: In the i'th iteration, use a modified version of Grover's search algorithm to find the next mistake that $\mathcal{G}$ makes. Since the algorithm uses at most T queries and makes at most G mistakes on any input, the quantum query complexity of the final algorithm can be shown to be $O(\sqrt{TG})$ using the Cauchy-Schwarz inequality.[2]

Recently, Beigi and Taghavi [BT20] gave an alternate proof of Theorem 1.1 using the framework of *non-binary span programs* introduced by them in [BT19]. Using this they showed that a similar statement also holds for functions $f : [\ell]^n \to \mathcal{R}$ with non-binary inputs and outputs. A sketch of their proof is as follows: Given an assignment of real weights to the edges of the given decision tree computing f, they construct a dual adversary solution and span program. The complexity of the resultant query algorithm is a function of the weights assigned to the edges. They propose a particular weighting scheme based on an edge-coloring (guessing algorithm) of the given decision tree. For Boolean functions, the quantum query complexity upper bound they obtain matches the bound of Lin and Lin (Theorem 1.1), which is $O(\sqrt{TG})$, where T denotes the depth and G the *guessing complexity* (see Definition 3.2) of the underlying decision tree, respectively.

In a follow-up work [BTT22], conditional on some constraints, Beigi, Taghavi and Tajdini implement the span-program-based algorithm of [BT20] in a time-efficient manner. Recently, Taghavi [Tag22] used non-binary span programs to give a tight quantum query algorithm for the oracle identification problem, simplifying an earlier algorithm due to Kothari [Kot14].

1.4 Our Contributions

We observe here that the guessing complexity of a deterministic decision tree equals its *rank* (see Definition 2.2 and Claim 3.3), which is a combinatorial measure introduced by Ehrenfeucht and Haussler [EH89] in the context of learning theory. This answers a question of Lin and Lin [LL16, page 4], where the authors asked if G is related to any combinatorial measure studied in classical decision-tree complexity. The rank of a function (which is the minimum rank of a decision tree computing it) can be unboundedly smaller than the function's certificate complexity, sensitivity, block sensitivity, and even (exact or approximate) polynomial degree, as can be easily witnessed by the OR function. See, for

[2] We skip some crucial details here, such as the cost of the modified Grover's search algorithm, and an essential step that uses span programs to avoid a logarithmic overhead that error reduction would have incurred. Techniques that address both of the above-mentioned steps are due to Kothari [Kot14].

example, [DM21] for more relationships between rank and other combinatorial measures of Boolean functions. In view of the above-mentioned equivalence of G and decision tree rank, Theorem 1.1 has a clean equivalent formulation as follows.

Theorem 1.2. *Let $\mathcal{T}$ be a decision tree computing a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ with depth T and rank G. Then,*

$$\mathsf{Q}(f) = O(\sqrt{TG}).$$

We introduce a new measure, *randomized rank* of randomized decision trees (see Definition 2.2), which we denote by rrank and which is a natural probabilistic analog of rank. This exactly captures the notion of the randomized analog of the value of G in Theorem 1.1. It is easy to show with this definition that our proof of Theorem 1.2 also holds with respect to randomized decision trees and randomized rank. Thus we obtain the following easy-to-state reformulation of [LL16, Theorem 5.4].

Theorem 1.3. *Let $\mathcal{T}$ be a randomized decision tree computing a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ with depth T and randomized rank G. Then,*

$$\mathsf{Q}_{2/5}(f) = O(\sqrt{TG}).$$

Thus, up to a small loss in the success probability, upper bounds obtained from Theorem 1.3 are strictly stronger than those obtained from Theorem 1.2 for relations whose randomized rank is much smaller than their rank. Hence a natural question is if this can be the case.

It is easy to exhibit maximal separations between rank and randomized rank for partial functions. One such separation is witnessed by the Approximate-Majority function, which is the function that outputs 0 if the Hamming weight of an n-bit input is less than $n/3$ and outputs 1 if the Hamming weight of the input is more than $2n/3$. It is easy to show an $\Omega(n)$ lower bound on its rank and an $O(1)$ upper bound on its randomized rank (see Claim 3.7). Whether or not such a separation holds for total Boolean functions is not so clear. We show a polynomial separation for the complete binary AND-OR tree. This is the first of our main theorems.

Theorem 1.4. *For the complete binary AND-OR tree $F : \{0,1\}^n \to \{0,1\}$,*

$$\mathsf{rrank}(F) = O\left(\mathsf{rank}(F)^{\log \frac{1+\sqrt{33}}{4}}\right) \approx O\left(\mathsf{rank}(F)^{.753\ldots}\right).$$

To prove this theorem, we show a rank lower bound of $\Omega(n)$ using known connections between rank and Prover-Delayer games [PI00], and the randomized-rank upper bound immediately follows from an upper bound of $O\left(n^{\log \frac{1+\sqrt{33}}{4}}\right)$ on its randomized decision-tree complexity [SW86].

Beigi and Taghavi [BT20, Section 6] asked if one could improve their results by using different choices of weights in their constructions of span programs and dual adversary vectors. For decision trees that make queries to a bit string, we answer this question in the affirmative by providing a weighting scheme that improves upon their bounds. We exhibit a family of decision trees for which our bounds are strictly stronger than those of Beigi and Taghavi, and Lin and Lin (see Corollary 1.7 and the discussion below it). We argue that optimizing the dual adversary bound and the witness complexity of Beigi and Taghavi's span program with variable weights is a minimization program with constraints linear in the variables and inverses of the variables. We give a weighting scheme and moreover we show in Theorem 5.4 that our weighting scheme is optimal, thus subsuming Theorem 1.1.

Beigi and Taghavi's construction also works for decision trees that query a non-Boolean string. They adapt the weighting scheme from the Boolean case by associating one weight to a single query outcome, and the other weight to all the other query outcomes. One can adapt the weighting scheme we introduce in this work to the non-Boolean setting in a similar manner, however we do not prove optimality of the weighting scheme in this case. We leave a possibly tighter generalization to the non-Boolean setting to future work.

For a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ and a deterministic decision tree $\mathcal{T}$ computing it, let $\tilde{f}$ be the function that takes an n-bit string x as input, and outputs the leaf of $\mathcal{T}$ reached on input x. Let $P(\mathcal{T})$ denote the set of root-to-leaf paths in $\mathcal{T}$. For a path $P \in P(\mathcal{T})$, let $\overline{P}$ denote the set of edges that have exactly one vertex in common with P.

Definition 1.5 (Weight optimization program)**.** *For a decision tree $\mathcal{T}$, define its* weight optimization program *by the minimization problem with constraints outlined in Program 1. Let $\mathrm{OPT}_{\mathcal{T}}$ denote the optimum of this program.*

Variables	$\{W_e : e \text{ is an edge in } \mathcal{T}\}, \alpha, \beta$		
Minimize	$\sqrt{\alpha\beta}$		
s.t.	$\sum_{e \in \overline{P}} W_e$	$\leq \alpha,$	for all paths $P \in P(\mathcal{T})$
	$\sum_{e \in P} \frac{1}{W_e}$	$\leq \beta,$	for all paths $P \in P(\mathcal{T})$
	W_e	$> 0,$	for all edges e in $\mathcal{T}$
	α, β	$\geq 0.$	

Program 1: The *weight optimization program* capturing the weight assignment to edges of $\mathcal{T}$ that optimizes the witness complexity of the NBSPwOI and dual adversary vector constructions of Beigi and Taghavi (see Section 4).

The following is our second main theorem.

Theorem 1.6. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation and let $\mathcal{T}$ be a decision tree computing f. Then,*

$$\mathsf{Q}(\tilde{f}) = O(\mathrm{OPT}_{\mathcal{T}}).$$

Our results give us a new way to bound $\mathrm{OPT}_{\mathcal{T}}$ and thus $\mathsf{Q}(\tilde{f})$ from above. Combined with the earlier results from Beigi and Taghavi, this gives us the following corollary. For formal definitions of the measures below, see Section 2.

Corollary 1.7. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation and let $\mathcal{T}$ be a deterministic decision tree computing it, weighted with the canonical weight assignment as defined in Definition 5.1. Then, the quantum query complexity of $\tilde{f}$ (in fact $\mathrm{OPT}_{\mathcal{T}}$) satisfies the following two bounds:*

1. *The rank-depth bound:* $\mathsf{Q}(\tilde{f}) = O\left(\sqrt{\mathsf{rank}(\mathcal{T})\mathrm{depth}(\mathcal{T})}\right)$.
2. *The size bound:* $\mathsf{Q}(\tilde{f}) = O\left(\sqrt{\mathrm{DTSize}(\mathcal{T})}\right)$.

Using standard arguments to deal with the case when $\mathcal{T}$ is a randomized decision tree, this gives an upper bound on the bounded-error quantum query complexity of a function in terms of its randomized decision-tree size complexity, as well as in terms of its randomized rank and depth (see Corollary 5.5).

It was shown by Reichardt [Rei11] that the quantum query complexity of evaluating Boolean formulas of size s is $O(\sqrt{s})$. In particular, this implies the size bound in Corollary 1.7 for Boolean functions f since the formula size of a Boolean function is bounded

above by a constant times its decision-tree size (see Claim A.2). Not only does our bound also hold for *relations*, but the query algorithm we obtain is actually an algorithm for $\tilde{f}$ when the underlying tree is deterministic. While this yields trivial bounds for most relations (since almost all Boolean functions have super-polynomial decision-tree size complexity, for example, while $\mathsf{Q}(f)$ is at most n), it recovers the $O(\sqrt{n})$ bound for the search problem [Gro96], for example. We also exhibit a family of decision trees for which the size bound is strictly stronger than the rank-depth bound (see Figure 6).

1.5 Subsequent work

In a preliminary version of this paper [CMP22] we conjectured that rank and randomized rank are polynomially related for all total Boolean functions. Note that techniques used to prove the analogous polynomial equivalence for deterministic and randomized query complexities [Nis91] (also see [BW02]) cannot work since they use intermediate measures such as certificate complexity, sensitivity and block sensitivity, all of which are maximal for the OR function even though its rank is 1.

It is not hard to see that, up to a logarithmic factor in the input size, (randomized) rank is equivalent to logarithm of (randomized) decision tree size complexity. Using this equivalence, it was subsequently shown [CDM+23] that up to a polylogarithmic factor in the input size our conjecture is true. That is, rank and randomized rank are polynomially related (up to a polylogarithmic factor in the input size) for all total Boolean functions.

As a consequence, the quantum query upper bounds we obtain in Theorem 1.3 can only be polynomially better than the upper bound obtained in Theorem 1.2.

An alternative proof for Theorem 1.6 also appears in [Cor25, Section 5.4].

1.6 Organization

We provide necessary preliminaries in Section 2. In Section 3, we discuss the rank of decision trees, and prove that it is equal to the guessing complexity. We also prove Theorem 1.4, which is a polynomial separation between rank and randomized rank for the complete binary AND-OR tree, in Section 3. In Section 4 we discuss Beigi and Taghavi's construction of a span program and a dual adversary solution for a relation f by assigning weights to edges of a classical decision tree computing f, and we show that the best possible weighting scheme is captured in Program 1, proving Theorem 1.6. In Section 5, we exhibit a solution to Program 1 and argue its optimality from first principles. In Appendix A we prove an upper bound on formula size of a Boolean function in terms of its decision-tree size. In Appendix B we exhibit another interesting, albeit sub-optimal, solution to Program 1.

2 Preliminaries

All logarithms in this paper are taken base 2. For a bit $b \in \{0,1\}$, let $\overline{b}$ denote the bit $1-b$. For a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$, define the domain of f to be $D_f := \{x \in \{0,1\}^n : \exists b \in \mathcal{R}$ such that $(x,b) \in f\}$. For a vector v, let $\|v\|$ denote its ℓ_2-norm. For a matrix M, let $\|M\|$ denote its spectral norm. For matrices M and N of the same dimensions, let $M \circ N$ denote their entry-wise (Hadamard) product. Let $\delta_{a,b}$ be the function that outputs 1 when $a = b$ and 0 otherwise. We use $[T]$ to denote the set $\{1, \ldots, T\}$ where $T \in \mathbb{Z}^+$. A *restriction* is a partial assignment of variables $x_1, \ldots, x_n$ to values in $\{0,1\}$. Formally, a restriction is a function $\rho : [n] \to \{0,1,\perp\}$, where $\rho(i) = \perp$ indicates that the value of x_i is left free. For a function $f : \{0,1\}^n \to \{0,1\}$ and a restriction $\rho : [n] \to \{0,1,\perp\}$, the restricted function $f|_\rho$ is defined on the free variables in the natural way.

2.1 Decision Trees

A decision tree computing a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ is a binary tree with leaf nodes labeled in $\mathcal{R}$, each internal node is labeled by a variable x_i and has two outgoing edges, labeled 0 and 1. On input $x \in D_f$, the tree's computation proceeds by computing x_i as indicated by the node's label and following the edge indicated by the value of the computed variable. The output value at the leaf, say $b \in \mathcal{R}$, must be such that $(x, b) \in f$. Given a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ and a deterministic decision tree $\mathcal{T}$ computing it, define the function $\tilde{f}$ that takes an input $x \in \{0,1\}^n$ and outputs the leaf of $\mathcal{T}$ reached on input x. Let $V(\mathcal{T})$ denote the set of vertices in $\mathcal{T}$, and, we use $V_I(\mathcal{T})$ to denote the set of internal nodes of $\mathcal{T}$. We use the notation $J(v)$ to denote the variable queried at vertex v for an internal node v. For a vertex $v \in V_I(\mathcal{T})$ and $q \in \{0,1\}$, let $N(v,q)$ denote the vertex that is the child of v along the edge that has label q. For neighbors v, w, let $e(v,w)$ denote the edge between them. For an input $x \in \{0,1\}^n$, let P_x denote the unique path in $\mathcal{T}$ from its root to $\tilde{f}(x)$. For a path P in $\mathcal{T}$, we say an edge e *deviates from* P if exactly one vertex of e is in P. For a path P in $\mathcal{T}$, define $\overline{P} := \{e : e \text{ deviates from } P\}$. We let $P(\mathcal{T})$ denote the set of all paths from the root to a leaf in $\mathcal{T}$. We assume that decision trees computing relations f contain no extraneous leaves, i.e., for all leaves there is an input $x \in D_f$ that reaches that leaf. We also assume that for every path P in $\mathcal{T}$ and index $i \in [n]$, the variable x_i is queried at most once on P.

The decision tree complexity (also called *deterministic query complexity*) of f, denoted $\mathsf{D}(f)$, is defined as follows.

$$\mathsf{D}(f) := \min_{\mathcal{T}:\mathcal{T} \text{ is a DT computing } f} \text{depth}(\mathcal{T}).$$

Note that a deterministic decision tree in fact computes a function, since each input reaches exactly one leaf on the computation path of the tree. A randomized decision tree is a distribution over deterministic decision trees. We say a randomized decision tree computes a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ with error $1/3$ if for all $x \in \{0,1\}^n$, the probability of outputting a $b \in \mathcal{R}$ such that $(x,b) \in f$ is at least $2/3$. The depth of a randomized decision tree is the maximum depth of a deterministic decision tree in its support. Define the randomized decision tree complexity (also called *randomized query complexity*) of f as follows.

$$\mathsf{R}(f) := \min_{\substack{\mathcal{T}:\mathcal{T} \text{ is a randomized DT} \\ \text{that computes } f \text{ up to error } 1/3}} \text{depth}(\mathcal{T}).$$

Another measure of interest to us in this work is the *decision-tree size complexity* of f.

Definition 2.1 (Decision-tree size complexity)**.** *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Define the* decision-tree size complexity *of f, which we denote by* $\text{DTSize}(f)$*, as*

$$\text{DTSize}(f) := \min_{\mathcal{T}:\mathcal{T} \text{ computes } f} \text{DTSize}(\mathcal{T}),$$

where $\text{DTSize}(\mathcal{T})$ *denotes the number of nodes of $\mathcal{T}$. Analogously, the* randomized decision-tree size complexity *of f is defined to be*

$$\text{RDTSize}(f) := \min_{\substack{\mathcal{T}:\mathcal{T} \text{ is a randomized DT} \\ \text{that computes } f \text{ up to error } 1/3}} \text{RDTSize}(\mathcal{T}),$$

where $\text{RDTSize}(\mathcal{T})$ *denotes the maximum number of nodes of a decision tree in the support of $\mathcal{T}$.*

It is easy to observe that the number of nodes in a deterministic decision tree equals one less than twice the number of leaves in the tree.

We require the notion of rank of a decision tree introduced by Ehrenfeucht and Haussler [EH89]. We also require a natural randomized variant of the same, which to the best of our knowledge, has not been studied in the literature.

Definition 2.2 (Decision tree rank and randomized rank)**.** *Let $\mathcal{T}$ be a binary decision tree. Define the rank of $\mathcal{T}$ recursively as follows: For a leaf node a, define $\mathsf{rank}(a) = 0$. For an internal node u with children v, w, define*

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

Define $\mathsf{rank}(\mathcal{T})$ to be the rank of the root of $\mathcal{T}$. Define the randomized rank *of a randomized decision tree to be the maximum rank of a deterministic decision tree in its support.*

Definition 2.3 (Rank of a relation)**.** *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Define the rank of f, which we denote by $\mathsf{rank}(f)$, by*

$$\mathsf{rank}(f) = \min_{\mathcal{T}:\mathcal{T} \text{ computes } f} \mathsf{rank}(\mathcal{T}).$$

Analogously define the randomized rank of f, which we denote by $\mathsf{rrank}(f)$, to be the minimum randomized rank of a randomized decision tree that computes f to error $1/3$.

2.2 Quantum Query Complexity

We refer the reader to [NC16, Wol19] for the basics of quantum computing. A quantum query algorithm $\mathcal{A}$ for a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ begins in a fixed an initial state $|\psi_0\rangle$, applies a sequence of unitaries $U_0, O_x, U_1, O_x, \cdots, U_T$, and performs a measurement. Here, the initial state $|\psi_0\rangle$ and the unitaries $U_0, U_1, \ldots, U_T$ are independent of the input. The unitary O_x represents the "query" operation, and maps $|i\rangle|b\rangle$ to $|i\rangle|b + x_i \mod 2\rangle$ for all $i \in [n]$ and maps $|i\rangle$ to $|i\rangle$ whenever $i = 0$. We say that $\mathcal{A}$ is an *ε-error algorithm* computing f if for all x in the domain of f, the probability of outputting $b \in \mathcal{R}$ such that $(x, b) \in f$ is at least $1 - \varepsilon$. The *ε-error quantum query complexity* of f, denoted by $\mathsf{Q}_\varepsilon(f)$, is the least number of queries required for a quantum query algorithm to compute f with error ε. When the subscript ε is dropped we assume $\varepsilon = 1/3$; the *bounded-error query complexity* of f is $\mathsf{Q}(f)$.

2.3 Span Programs

The model of span programs of interest to us is that of "non-binary span programs with orthogonal inputs", abbreviated NBSPwOI. This model was introduced by Beigi and Taghavi [BT19]. A NBSPwOI $(P, w, \overline{w})$ evaluating a function $f : D_f \to [m]$ with $D_f \subseteq [\ell]^n$ consists of the following:

- A finite dimensional vector space V equipped with an inner product,
- for every $i \in [m]$, a target vector $|t_i\rangle \in V$,
- for every $j \in [n]$ and every $q \in [\ell]$, an input set $I_{j,q} \subseteq V$.

Let I be the union of all input sets:

$$I = \bigcup_{j\in[n]} \bigcup_{q\in[\ell]} I_{j,q}. \tag{1}$$

For every $x \in D_f$ we define the set of *available vectors for* x to be

$$I(x) = \bigcup_{j\in[n]} I_{j,x_j}. \tag{2}$$

We say that the set of vectors $I \setminus I(x)$ is *unavailable for* x. The set P comprises all of the above components. Let A be the $(d \times |I|)$-dimensional matrix consisting of all input vectors as its columns with $d = \dim(V)$. We say $(P, w, \overline{w})$ evaluates f if for every $x \in D_f$,

$$|t_\alpha\rangle \in \operatorname{span}(I(x)) \iff \alpha = f(x).$$

Moreover, there should be two witnesses indicating this: a positive witness $|w_x\rangle \in \mathbb{C}^{|I|}$ and a negative witness $|\overline{w}_x\rangle \in V$ satisfying the following conditions:

- The coordinates of $|w_x\rangle$ associated to unavailable vectors are zero.
- $A|w_x\rangle = |t_\alpha\rangle$.
- For all $|v\rangle \in I(x)$,
$$\langle v|\overline{w}_x\rangle = 0.$$
- $\forall \beta \neq \alpha$ we have $\langle t_\beta|\overline{w}_x\rangle = 1$.

Let w and $\overline{w}$ denote the collections of positive and negative witnesses, respectively. The positive and negative witness sizes of $(P, w, \overline{w})$ are defined to be

$$\operatorname{wsize}^+(P, w, \overline{w}) := \max_{x\in D_f} \||w_x\rangle\|^2, \tag{3}$$

$$\operatorname{wsize}^-(P, w, \overline{w}) := \max_{x\in D_f} \left\|A^\dagger|\overline{w}_x\rangle\right\|^2. \tag{4}$$

The *witness size* of $(P, w, \overline{w})$ is defined as

$$\operatorname{wsize}(P, w, \overline{w}) := \sqrt{\operatorname{wsize}^-(P, w, \overline{w}) \cdot \operatorname{wsize}^+(P, w, \overline{w})}. \tag{5}$$

Beigi and Taghavi [BT19, Theorem 2] showed that for any NBSPwOI $(P, w, \overline{w})$ evaluating the function f, its complexity $\operatorname{wsize}(P, w, \overline{w})$ is an upper bound on $\mathsf{Q}(f)$.

Theorem 2.4 ([BT19])**.** *Let $f : D_f \to [m]$ be a function with $D_f \subseteq [\ell]^n$, and let $(P, w, \overline{w})$ be a NBSPwOI computing f. Then,*

$$\mathsf{Q}(f) = O(\operatorname{wsize}(P, w, \overline{w})).$$

2.4 Dual Adversary Bound

Let $f : D_f \to [m]$ be a function with $D_f \subseteq [\ell]^n$. Let Γ be a Hermitian matrix with rows and columns labeled by elements of D_f. The matrix Γ is called an *adversary matrix* for f if $\Gamma[x, y] = 0$ whenever $f(x) = f(y)$. Høyer, Lee and Špalek [HLŠ07] defined the quantity

$$\operatorname{Adv}^\pm(f) = \max_{\Gamma\neq 0} \frac{\|\Gamma\|}{\max_i \|\Gamma \circ D_i\|}, \tag{6}$$

where D_i is a $\{0,1\}$-valued matrix with $D_i[x,y] = 1 - \delta_{x_i,y_i}$. They showed that $\mathsf{Q}(f) = \Omega(\mathrm{Adv}^{\pm}(f))$. Reichardt [Rei11] subsequently proved that $\mathsf{Q}(f) = \Theta(\mathrm{Adv}^{\pm}(f))$ for Boolean functions $f : \{0,1\}^n \to \{0,1\}$. This result was later generalized by Lee et al. [LMR+11, Theorem 1.1] who showed

$$\mathsf{Q}(f) = \Theta(\mathrm{Adv}^{\pm}(f)) \tag{7}$$

for non-Boolean functions as well. They first observed that the quantity $\mathrm{Adv}^{\pm}(f)$ in Equation (6) can be viewed as a feasible solution to a semidefinite program (SDP),

$$\mathrm{Adv}^{\pm}(f) = \max\{\|\Gamma\| : \forall i \in [n], \|\Gamma \circ D_i\| \leq 1\}, \tag{8}$$

where the maximization is over all real adversary matrices for f, i.e., symmetric matrices Γ such that $\Gamma[x,y] = 0$ whenever $f(x) = f(y)$. While the primal solution of SDP in Equation (8) gives a lower bound on the quantum query complexity of f, they argued that a solution to the dual of this SDP gives upper bounds on $\mathsf{Q}(f)$. Based on duality of SDPs, they first showed ([LMR+11, Lemma A.1]) that the dual SDP is of the following form.

Variables	$\{\|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{\|w_{xj}\rangle : x \in D_f, j \in [n]\}, d$	
Minimize	$\max_{x \in D_f} \max\left\{\sum_{j=1}^n \||u_{xj}\rangle\|^2, \sum_{j=1}^n \||w_{xj}\rangle\|^2\right\}$	
s.t.	$\sum_{j \in [n]: x_j \neq y_j} \langle u_{xj}\|w_{yj}\rangle = 1 - \delta_{f(x),f(y)}$	$\forall x, y \in D_f$
	$\|u_{xj}\rangle, \|w_{xj}\rangle \in \mathbb{C}^d$	for all $x \in D_f$

Program 2: Dual SDP for f

They then showed that a feasible solution to the SDP in Program 2 yields an upper bound on the quantum query complexity of f ([LMR+11, Theorem 4.1]).

Theorem 2.5 ([LMR+11])**.** *Let $f : D_f \to [m]$ be a function with $D_f \subseteq [\ell]^n$, let C denote the optimal value of Program 2 for f. Then*

$$\mathsf{Q}(f) = O(C). \tag{9}$$

3 Decision Tree Rank

In this section, we first rephrase Theorem 1.1 in terms of a measure of decision trees which we term 'guessing complexity'. This reformulation was essentially done by Beigi and Taghavi [BT20, Section 3]. We then show that the guessing complexity of a decision tree equals its rank, proving Theorem 1.2. Finally, we show a polynomial separation between rank and randomized rank for the complete binary AND-OR tree.

3.1 Guessing Complexity and Rank

Definition 3.1 (G-coloring [BT20, Definition 1])**.** *A G-coloring of a decision tree $\mathcal{T}$ is a coloring of its edges by two colors black and red, in such a way that any vertex of $\mathcal{T}$ has at most one outgoing edge with black color.*

Definition 3.2 (Guessing Complexity)**.** *Let $\mathcal{T}$ be a decision tree. Let $P(\mathcal{T})$ denote the set of root-to-leaf paths in $\mathcal{T}$. Define the* guessing complexity *of $\mathcal{T}$, which we denote by $G(\mathcal{T})$, by*

$$G(\mathcal{T}) = \min_{\text{G-colorings of } \mathcal{T}} \max_{P \in P(\mathcal{T})} \text{number of red edges on } P.$$

Claim 3.3. *Let $\mathcal{T}$ be a decision tree. Then,*

$$G(\mathcal{T}) = \mathsf{rank}(\mathcal{T}).$$

Proof. Let v, v_L and v_R be the root of $\mathcal{T}$, and the left and right children of v, respectively. Let $\mathcal{T}_L$ and $\mathcal{T}_R$ denote the subtrees of $\mathcal{T}$ rooted at v_L and v_R, respectively.

Consider a G-coloring of $\mathcal{T}$. This naturally induces a G-coloring of $\mathcal{T}_L$ and $\mathcal{T}_R$. We consider two cases:

- $G(\mathcal{T}_L) = G(\mathcal{T}_R) = k$, say. One of the edges (v, v_L) or (v, v_R) must be colored red. Assume without loss of generality that (v, v_L) is the red edge. Since we assumed $G(\mathcal{T}_L) = k$, $\mathcal{T}_L$ contains a path with at least k red edges under the G-coloring induced from the given G-coloring of $\mathcal{T}$. But this induces a path in $\mathcal{T}$ with $k+1$ red edges, and hence $G(\mathcal{T}) \geq G(\mathcal{T}_L) + 1$.
- If $G(\mathcal{T}_L) \neq G(\mathcal{T}_R)$, we have $G(\mathcal{T}) \geq \max\{G(\mathcal{T}_L), G(\mathcal{T}_R)\}$, witnessed by the G-colorings induced on $\mathcal{T}_L$ and $\mathcal{T}_R$ by the G-coloring of $\mathcal{T}$.

In the other direction, we construct an optimal G-coloring of $\mathcal{T}$ given optimal G-colorings of $\mathcal{T}_L$ and $\mathcal{T}_R$. The edges of $\mathcal{T}_L$ and $\mathcal{T}_R$ in $\mathcal{T}$ are colored exactly as they are in the given optimal G-colorings of them. It remains to assign colors to the two remaining edges (v, v_L) and (v, v_R). We again have two cases:

- $G(\mathcal{T}_L) = G(\mathcal{T}_R) = k$, say. Arbitrarily color one of the edges (v, v_L) and (v, v_R) (say, (v, v_L)) red, and color the other edge black. The maximum number of red edges on a path has increased by 1. Thus, $G(\mathcal{T}) \leq G(\mathcal{T}_L) + 1$.
- $G(\mathcal{T}_L) > G(\mathcal{T}_R)$, say (the other case follows a similar argument). Color the edge (v, v_L) black and (v, v_R) red. Thus, the maximum number of red edges on a path in $\mathcal{T}$ equals $\max\{G(\mathcal{T}_L), G(\mathcal{T}_R) + 1\} = G(\mathcal{T}_L)$.

Thus, we have

$$G(\mathcal{T}) = \begin{cases} \max\{G(\mathcal{T}_L), G(\mathcal{T}_R)\} & \text{if } G(\mathcal{T}_L) \neq G(\mathcal{T}_R) \\ G(\mathcal{T}_L) + 1 & \text{if } G(\mathcal{T}_L) = G(\mathcal{T}_R), \end{cases}$$

The measure $\mathsf{rank}(\mathcal{T})$ is defined exactly as the above (Definition 2.2), proving the claim. □

The guessing algorithm $\mathcal{G}$ in Theorem 1.1 corresponds to a natural G-coloring of $\mathcal{T}$ of cost G: for each internal vertex, color the guessed edge black and the other edge red. Thus, Theorem 1.2 immediately follows from Claim 3.3 and Theorem 1.1.

Proof of Theorem 1.3. A quantum query algorithm for f is as follows: sample a decision tree from the support of $\mathcal{T}$ according to its underlying distribution, and run a 9/10-error quantum query algorithm of cost $O(\sqrt{TG})$ from Theorem 1.2 on the resultant tree.[3] The success probability of this algorithm is at least $(9/10) \cdot (2/3) = 3/5$ for all inputs $x \in D_f$. □

[3]This is possible since the deterministic decision trees in the support of $\mathcal{T}$ compute *functions*, which admit efficient error reduction with a constant overhead in query complexity by standard techniques (run the algorithm from Theorem 1.2 a large constant many times and return the majority output).

The rank of a decision tree essentially captures the largest depth of a complete binary subtree of the original tree. Thus, the rank of a tree is bounded from above by the logarithm of the size of the tree.

Observation 3.4 ([EH89, Lemma 1])**.** *Let $\mathcal{T}$ be a deterministic decision tree of size s. Then,*

$$\mathsf{rank}(\mathcal{T}) \leq \log(s+1) - 1.$$

Along with this observation and the simple observation that the depth of a decision tree is at most its size, Theorem 1.3 yields the following statement.

Theorem 3.5. *Let $\mathcal{T}$ be a randomized decision tree of size s that computes a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$. Then,*

$$\mathsf{Q}_{2/5}(f) = O(\sqrt{s \log s}).$$

Note here that it suffices to prove that $\mathsf{Q}(\tilde{f}) = O(\sqrt{s \log s})$ where s is the size of a *deterministic* decision tree computing f, since standard techniques and error reduction yield the required bound for randomized trees. In Appendix B we show an explicit NBSPwOI and dual adversary solution witnessing the same bound. In Sections 4 and 5 we show an explicit NBSPwOI and dual adversary solution witnessing a stronger bound without the logarithmic factor. We choose to still give the weaker bound in Appendix B as the weighting scheme seems considerably different from that in Section 5, and the weights are also efficiently computable.

We note here the equivalence of the rank of a Boolean function and the value of an associated Prover-Delayer game introduced by Pudlák and Impagliazzo [PI00]. We use this equivalence in the next part of this section to show that the rank of the complete binary AND-OR tree is polynomially larger than its randomized rank.

The game is played between two players: the Prover and the Delayer, who construct a partial assignment, say $\rho \in \{0,1,\perp\}^n$, in rounds. To begin with the assignment is empty, i.e., $\rho = \perp^n$. In a round, the Prover queries an index $i \in [n]$ for which the value x_i is not set in ρ (i.e., $\rho_i = \perp$). The Delayer either answers $x_i = 0$ or $x_i = 1$, or defers the choice to the Prover. In the latter case, the Delayer scores a point. The game ends when the Prover knows the value of the function, i.e., when $f|_\rho$ is a constant function. The value of the game, $\mathsf{val}(f)$, is the maximum number of points the Delayer can score regardless of the Prover's strategy. The following result is implicit in [PI00] (also see [DM21, Theorem 3.1] for an explicit statement and proof).

Claim 3.6. *Let $f : \{0,1\}^n \to \{0,1\}$ be a (possibly partial) Boolean function. Then,*

$$\mathsf{rank}(f) = \mathsf{val}(f).$$

3.2 A Separation Between Rank and Randomized Rank

We first note that there can be maximal separations between rank and randomized rank if we consider partial functions. This is witnessed by the well-studied Approximate-Majority function, for example.

Claim 3.7. *Let $f : \mathcal{D} \subset \{0,1\}^n \to \{0,1\}$ be a partial function defined as follows:*

$$f(x) = \begin{cases} 0 & |x| \leq n/3 \\ 1 & |x| \geq 2n/3. \end{cases}$$

Then, $\mathsf{rank}(f) = \Theta(n)$ *and* $\mathsf{rrank}(f) = \Theta(1)$.

Proof. Clearly $\mathsf{rank}(f) = O(n)$. For the lower bound, we use the equivalence from Claim 3.6. A valid Delayer strategy is as follows: allow the Prover to choose input values for their first $n/3$ queries. It is easy to see that no matter what values the Prover chooses, the function can never be restricted to become a constant after these $n/3$ queries. Thus, $\mathsf{val}(f) \geq n/3$ (and this can be easily seen to be tight). The randomized rank upper bound follows from the easy fact that $\mathsf{R}(f) = O(1)$ and $\mathsf{rrank}(f) \leq \mathsf{R}(f)$ for all f. □

When we restrict f to be a total function, it is no longer clear whether or not randomized rank can be significantly smaller than rank. In view of the example above, one might be tempted to consider functions that witness maximal separations between deterministic and randomized query complexity. The current state-of-the-art separation of $\mathsf{D}(f) = \Omega(n)$ vs. $\mathsf{R}(f) = \tilde{O}(\sqrt{n})$ is witnessed by variants of 'pointer jumping' functions [GPW18, ABB+17, MRS18]. One might hope to use a similar argument as in the proof of Claim 3.7 to show that $\mathsf{rank}(f) = \Omega(n)$ (and a randomized rank upper bound of $\tilde{O}(\sqrt{n})$ immediately follows from the randomized query upper bound). However, it is not hard to show that the rank of these functions is actually $\tilde{O}(\sqrt{n})$, rendering this approach useless for these variants of pointer jumping functions. Nevertheless, we are able to use such an approach to show a separation between rank and randomized rank for another function whose deterministic and randomized query complexities are polynomially separated. After an initial version of this paper was made public, a subsequent work [CDM+23] used the equivalence between (randomized) rank and logarithm of (randomized) decision tree size to show a polynomial equivalence (up to a polylogarithmic factor in the input size) between rank and randomized rank for all Boolean functions. They also observed that a deterministic-randomized query complexity separation can be lifted to the same deterministic-randomized rank separation for a related function [CDM+23, Appendix C]. Thus, the pointer jumping functions [GPW18, ABB+17, MRS18] can be used to construct a function witnessing a quadratic separation between rank and randomized rank. Finding the best possible (polynomial) separation between rank and randomized rank remains an interesting open question.

In the remainder of this section, let $F : \{0,1\}^n \to \{0,1\}$ be defined as the function evaluated by a complete $(\log n)$-depth binary tree as described below. Assume $\log n$ to be an even integer, and the top node of this tree to be an OR gate. Nodes in subsequent layers alternate between AND's and OR's, and nodes at the bottom layer contain variables. We call this the complete AND-OR tree of depth $\log n$. See Figure 1 for a depiction of the complete depth-4 AND-OR tree.

It is easy to see via an adversarial argument that $\mathsf{D}(F) = n$. Saks and Wigderson [SW86] showed that $\mathsf{R}(F) = \Theta(n^{\log \frac{1+\sqrt{33}}{4}}) \approx \Theta(n^{0.753\ldots})$.

Theorem 3.8 ([SW86, Theorem 1.5])**.** *Let $F : \{0,1\}^n \to \{0,1\}$ be the complete AND-OR tree of depth $\log n$. Then*

$$\mathsf{D}(F) = n, \qquad \mathsf{R}(F) = \Theta\left(n^{\log \frac{1+\sqrt{33}}{4}}\right) \approx \Theta(n^{0.753\ldots}).$$

We show that the rank of F equals $(n+2)/3$.

Theorem 3.9. *Let $F : \{0,1\}^n \to \{0,1\}$ be the complete AND-OR tree of depth $\log n$. Then*

$$\mathsf{rank}(F) = \frac{n+2}{3}.$$

This proof uses the characterization of rank by the value of the Prover-Delayer game in Claim 3.6. For the lower bound, we exhibit a strategy of the Delayer that scores $(n+2)/3$

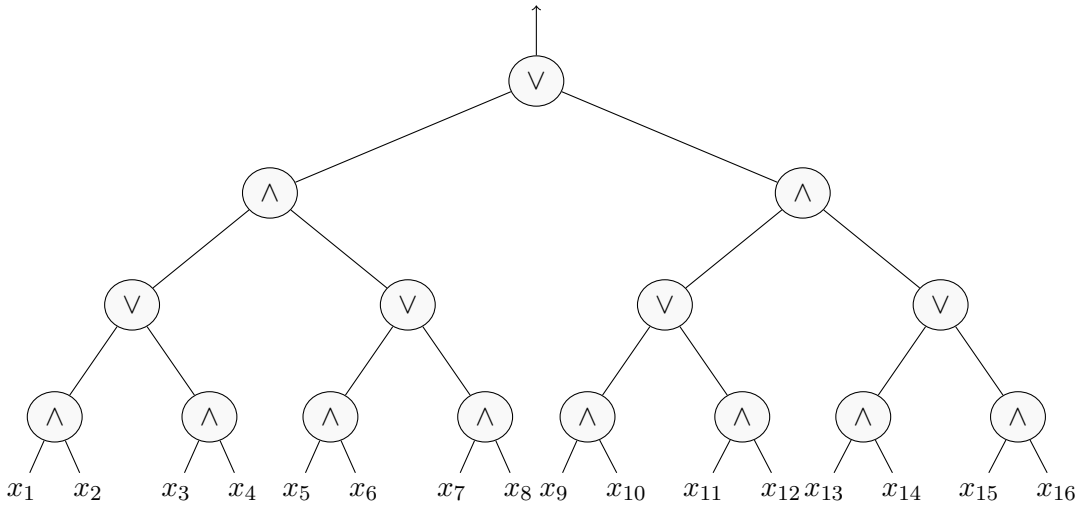


Figure 1: Complete AND-OR tree of depth 4

points regardless of the strategy of the Prover. For the upper bound, we show a strategy of the Prover which ensures that the Delayer can score at most $(n+2)/3$ points.

Before we exhibit these strategies, we describe how partial assignments to the AND-OR tree can be captured by modifications of the tree itself. To that end, we formalize required properties of AND-OR trees in the following definition.

Definition 3.10. *Let $n \in \mathbb{N}$, and let $\mathcal{T}$ be a tree, whose internal nodes are labeled by either $\wedge$ or $\vee$, and whose leaves are labeled by indices in $[n]$, such that no two leaves are labeled by the same index. Then, we refer to $\mathcal{T}$ as an* AND-OR tree*. Edges are directed from the root to leaves, and if $\mathcal{T}$ contains no nodes, we say that it is empty. When $\mathcal{T}$ is non-empty, then the corresponding Boolean function it computes is denoted by $f_{\mathcal{T}} : \{0,1\}^S \to \{0,1\}$, where $S \subseteq [n]$ is the set of index labels that correspond to the leaves present in $\mathcal{T}$. If $\mathcal{T}$ is empty, we say that it computes a constant function, and we have to additionally supply the outcome $b \in \{0,1\}$. We often identify $\mathcal{T}$ with $f_{\mathcal{T}}$.*

We define some notation to describe the nodes within $\mathcal{T}$:

1. *We denote the root node of $\mathcal{T}$ by $\mathsf{root}(\mathcal{T})$.*
2. *For a leaf node v, we denote the variable it is querying by $\mathsf{index}(v) \in [n]$.*
3. *For a node v, define $\mathsf{label}(v) \in \{\wedge, \vee, \mathsf{leaf}\}$ to be the type of node.*
4. *For a non-root node v, define $\mathsf{parent}(v)$ to be the node in the tree that is the parent of v.*
5. *For an internal node v, define $\mathsf{child}(v)$ to be the set*
$$\mathsf{child}(v) \coloneqq \{w \in \mathcal{T} : v = \mathsf{parent}(w)\}.$$
6. *For a non-root node v, we define its* proper parent*, $\mathsf{pparent}(v)$, recursively as*
$$\mathsf{pparent}(v) = \begin{cases} \mathsf{undefined}, & \text{if } v = \mathsf{root}(\mathcal{T}), \\ \mathsf{parent}(v), & \text{if } |\mathsf{child}(\mathsf{parent}(v))| \geq 2, \\ \mathsf{pparent}(\mathsf{parent}(v)), & \text{otherwise.} \end{cases}$$

We say that $\mathcal{T}$ is in reduced form *if all internal nodes have at least two children, and none of these are labeled by the same gate as the parent.*

By fixing a subset of the input variables as per a partial assignment $\rho \in \{0,1,\perp\}^n$, any AND-OR tree can be simplified to a reduced tree $\mathcal{T}|_\rho$, i.e., leaves can be removed, and in many cases gates can be removed as well. When we fix a bit in our partial assignment, then the resulting simplification process can be split into two steps, which we refer to as the update and the contract steps, respectively. We formally define the corresponding routines in Algorithms 1 and 2.

Algorithm 1 update subroutine

Input: $\mathcal{T}, i \in [n], b \in \{0,1\}$.

if $\exists$ leaf ℓ in $\mathcal{T}$ such that $\mathsf{index}(\ell) = i$ **then**
  if ℓ is the only leaf in $\mathcal{T}$ **then**
    $\mathcal{T} \leftarrow$ the empty tree computing b.
  else if ($b = 1$ and $\mathsf{label}(\mathsf{parent}(\ell)) = \wedge$) or ($b = 0$ and $\mathsf{label}(\mathsf{parent}(\ell)) = \vee$) **then**
    Remove the leaf ℓ and the edge above it from $\mathcal{T}$.
  else if $\mathsf{parent}(\ell) = \mathsf{root}(\mathcal{T})$ **then**
    $\mathcal{T} \leftarrow$ the empty tree computing b.
  else
    Remove the edge above $\mathsf{parent}(\ell)$, and the subtree rooted at $\mathsf{parent}(\ell)$.

Output: $\mathcal{T}$.

Algorithm 2 contract subroutine

Input: $\mathcal{T}$

while there is a node v in $\mathcal{T}$ with $|\mathsf{child}(v)| = 1$ **do**
  if v is not $\mathsf{root}(\mathcal{T})$ **then**
    $\mathsf{parent}(\mathsf{child}(v)) \leftarrow \mathsf{parent}(v)$
  Remove v from $\mathcal{T}$.
while there are vertices $a, a' \in \mathcal{T}$ with $a' = \mathsf{parent}(a)$ and $\mathsf{label}(a) = \mathsf{label}(a')$ **do**
  for $w \in \mathsf{child}(a)$ **do**
    $\mathsf{parent}(w) \leftarrow a'$
  Remove a from $\mathcal{T}$.

Output: $\mathcal{T}$.

We now formally prove that the update procedure modifies the AND-OR tree in such a way that the Boolean function computed by the updated tree indeed computes the function computed by the original tree after updating the partial assignment accordingly. This is the objective of the following claim.

Claim 3.11. *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $\rho \in \{0,1,\perp\}^n$ be a partial assignment. Let $\mathcal{T}$ be an AND-OR tree in reduced form, computing $f|_\rho$. Let $i \in [n]$ be such that $\rho_i = \perp$, and let $b \in \{0,1\}$. Then, the AND-OR tree $\mathcal{T}' = \mathsf{update}(\mathcal{T}, i, b)$ computes the Boolean function $f|_{\rho'}$, where $\rho'_i = b$, and $\rho'_j = \rho_j$ for $j \neq i$.*

Proof. We consider multiple cases.

- If there does not exist a leaf ℓ in $\mathcal{T}$ such that $\mathsf{index}(\ell) = i$, then $f|_\rho$ does not depend on the i'th input bit. Therefore, $f|_\rho = f|_{\rho'}$. Hence $\mathcal{T}'$, which is the same as $\mathcal{T}$, computes $f_{\rho'}$.

- Suppose such a leaf does exist and let ℓ be the leaf in $\mathcal{T}$ such that $\mathsf{index}(\ell) = i$. If ℓ is the only leaf present in $\mathcal{T}$, then $f|_\rho(x) = x_i$, which means that after setting $\rho'_i = b$, the function value is determined, i.e., $f|_{\rho'} = b$, which corresponds to the tree $\mathcal{T}'$ being empty.

- If ℓ is a child of an $\wedge$-node, say v, and $b = 1$, then the output of v is completely determined by the other edges going into it. Since we assumed $\mathcal{T}$ to be in reduced form, at least one other such edge must exist. Therefore, the node ℓ can be removed, and the new tree $\mathcal{T}'$ computes $f|_{\rho'}$. The same argument holds when v is an $\vee$-node and $b = 0$.

- Now suppose that $\mathsf{parent}(\ell)$ is the root of the tree, it is labeled by $\wedge$, and $b = 0$. In that case, we the $\wedge$ evaluated to 0, and therefore the function value is completely determined. Thus, $f|_{\rho'} = 0$, and we output the empty tree computing 0. The same argument applies when $\mathsf{parent}(\ell)$ is the root of the tree, it is labeled by $\vee$, and $b = 1$.

- Finally, if ℓ is a child of an $\wedge$-node that is not the root of $\mathcal{T}$, say v, and $b = 0$, then the output of v is always 0, and the other inputs to v are now redundant. Therefore, we can remove v and the subtree rooted at it. The same holds for the case when v is an $\vee$-node and $b = 1$.

This completes the proof. □

Note that the tree that update outputs might not be in reduced form. Next, we show that the contract routine takes as input an AND-OR tree, converts it into reduced form, and does not change the function it computes.

Claim 3.12. *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $\rho \in \{0,1,\perp\}^n$ be a partial assignment. Let $\mathcal{T}$ be an AND-OR tree computing $f|_\rho$. Then $\mathcal{T}' = \mathsf{contract}(\mathcal{T})$ is an AND-OR tree in reduced form that computes $f|_\rho$.*

Proof. We analyze the two `while` loops in Algorithm 2.

- In the loop beginning at Line 1, we are selecting internal nodes that have exactly one child. Since these internal nodes are either $\wedge$- or $\vee$-gates, they act as the identity function, which means that they are redundant. Therefore, they can be removed without altering the function that is being computed, and the their child can be directly connected to their parent, if the node itself is not the root node to begin with. This is precisely what is done in Lines 2-4. After this first `while` block, we ensure that all internal nodes have at least two children, and the function computed by the tree has not changed. This part of the algorithm is described in Figure 2.

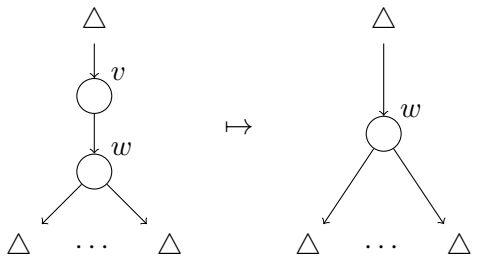


Figure 2: Graphical depiction of the first step in the contract procedure where $|\mathsf{child}(v)| = 1$

- In the loop beginning at Line 5, we select pairs of neighboring vertices (say v and $w = \mathsf{parent}(v)$) that have the same label. In this case, v and w can be contracted without changing the function being computed (due to the associativity of $\wedge$ and $\vee$). Concretely, this means that the algorithm takes the children of v, and adds them as children to w (Line 6-7), after which v is removed (Line 8). This can only increase the number of children of w, which means that its number of children must still be at least two. Moreover, this process is repeated until there are no more neighboring vertices that have the same label. Therefore, the resulting tree $\mathcal{T}'$ is in reduced form. This step is displayed in Figure 3.

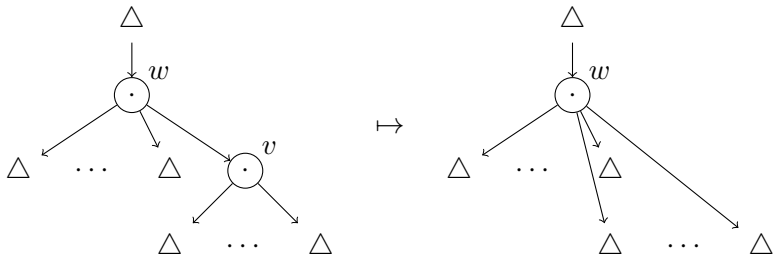


Figure 3: Graphical depiction of the second step in the contract procedure.

This completes the proof. □

Now that we have described two routines to modify an AND-OR tree $\mathcal{T}$, we can describe the specific strategies for the Delayer and Prover, that witness lower and upper bounds, respectively, on the rank of the complete binary AND-OR tree.

3.2.1 Delayer's Strategy (Lower Bound)

The Delayer starts by constructing the complete binary AND-OR tree $\mathcal{T}$ on n bits. Next, whenever the Prover asks variable i, Algorithm 3 describes the procedure that the Delayer follows. The output of this algorithm is a new AND-OR tree $\mathcal{T}$, which is being used as input for Algorithm 3 upon the next query of the Prover.

Algorithm 3 Delayer strategy for a reduced AND-OR tree $\mathcal{T}$

Input: $\mathcal{T}$, i

$\ell \leftarrow$ leaf in $\mathcal{T}$ such that $\mathsf{index}(\ell) = i$.
if ℓ is the only leaf in $\mathcal{T}$ **then**
    $b \leftarrow$ Prover's choice.
else if $\mathsf{label}(\mathsf{parent}(\ell)) = \vee$ **then**
    $b \leftarrow 0$.
else
    $v \leftarrow \mathsf{parent}(\ell)$.
    if $|\mathsf{child}(v)| > 2$ **then**
        $b \leftarrow 1$.
    else
        if $v = \mathsf{root}(\mathcal{T})$ **then**
            $b \leftarrow 1$.
        else
            $\{m\} \leftarrow \mathsf{child}(v) \setminus \{\ell\}$.
            if $\exists x \in \mathsf{child}(m) : \mathsf{label}(x) = \wedge$ **then**
                $b \leftarrow 1$.
            else
                $u \leftarrow \mathsf{parent}(v)$.
                if $\forall x \in \mathsf{child}(u) \setminus \{v\}$, $\mathsf{label}(x) = \mathrm{leaf}$ **then**
                    $b \leftarrow 1$.
                else
                    $b \leftarrow$ Prover's choice.

Output: $\mathsf{contract}(\mathsf{update}(\mathcal{T}, i, b))$.

Next, we argue that the Delayer always earns exactly $(n+2)/3$ points with this strategy. To that end, we introduce the following progress measure.

Definition 3.13. *Let $\mathcal{T}$ be an AND-OR tree (not necessarily in reduced form). We define the following cost function recursively on the nodes of $\mathcal{T}$, as*

$$c(v) = \begin{cases} 0, & \text{if } \mathsf{label}(v) = \mathsf{leaf}, \\ c(w), & \text{if } \mathsf{child}(v) = \{w\}, \\ 1, & \text{if } \mathsf{label}(v) = \wedge, |\mathsf{child}(v)| \geq 2, \\ \sum\limits_{w \in \mathsf{child}(v)} c(w), & \text{if } \mathsf{label}(v) = \vee. \end{cases}$$

We also define the set of marked nodes M, as

$$M = \{v \in \mathcal{T} : \mathsf{label}(v) = \vee \text{ and } |\mathsf{child}(v)| \geq 2 \\ \text{and } (\mathsf{pparent}(v) = \mathsf{undefined} \text{ or } \mathsf{label}(\mathsf{pparent}(v)) = \wedge)\}.$$

We now define a progress measure of $\mathcal{T}$, as

$$P(\mathcal{T}) = \begin{cases} 1 + \sum\limits_{v \in M} \max\{c(v) - 1, 0\}, & \text{if } \mathcal{T} \text{ is non-empty}, \\ 0, & \text{otherwise}. \end{cases}$$

We observe that $c(v)$ only recursively depends on $\{c(w) : w \in \mathsf{child}(v)\}$. Consequently, if we prove that the cost function is not altered on all children of v then $c(v)$ is also not altered. In particular, it is completely determined by the subtree rooted at v.

Now, we show that as long as the progress function is positive, the game does not end.

Claim 3.14. *If $P(\mathcal{T}) > 0$, then the function computed by $\mathcal{T}$ is not a constant function.*

Proof. If $P(\mathcal{T}) > 0$, then $\mathcal{T}$ is not empty. Thus, there is at least one leaf (variable) in the tree. Setting values of all leaves to 0 causes $\mathcal{T}$ to evaluate to 0, and setting values of all leaves to 1 causes $\mathcal{T}$ to evaluate to 1. □

Next, we show that the contract procedure does not affect the progress measure.

Claim 3.15. *The* contract *procedure in Algorithm 2 does not alter the value of $P(\mathcal{T})$.*

Proof. The first part of the contract subroutine, i.e., Lines 1 to 4, looks for an internal node v that has exactly one child w and contracts it into a new node v'. Let $\mathcal{T}$ be the AND-OR tree before the contraction, and let $\mathcal{T}'$ be the tree afterwards. Similarly, let c and c' be the cost functions defined on $\mathcal{T}$ and $\mathcal{T}'$, respectively, and let M and M' be their marked sets. Since the cost function evaluated at a node is completely determined by the subtree rooted at it, we observe that $c'(v') = c(w)$. Moreover, it is easy to verify that both when $\mathsf{label}(v) = \vee$ and when $\mathsf{label}(v) = \wedge$, we have that $c(v) = c(w) = c'(v')$. It follows that the cost function on the rest of the tree is not changed anywhere. Finally, $v \notin M$ since $|\mathsf{child}(v)| = 1$. Furthermore, $w \in M$ if and only if $v' \in M$, since this only depends on the proper parent of v and v' and their children, which are the same. Thus, $P(\mathcal{T}) = P(\mathcal{T}')$.

Next, we focus on the second part of the contract routine, which merges two nodes that have the same label. Since the first part of the algorithm has been completed, we can assume that all vertices have at least two children, which means that the proper parent of any internal vertex is its direct parent. Now, let $\mathcal{T}$, $\mathcal{T}'$, c, c', M and M' be as before, and let v and w be the parent and child nodes in $\mathcal{T}$, and let v' be the new node in $\mathcal{T}'$. We immediately observe that $w \notin M$, since it has the same label as its parent (which is its proper parent since all nodes have at least 2 children after the first phase of Algorithm 2), and $v \in M \Leftrightarrow v' \in M'$, since both have at least two children, and hence it only depends on their parents, which are the same.

- Suppose that v and w are both labeled by $\vee$. Then,

$$
\begin{aligned}
c'(v') &= \sum_{x \in \mathsf{child}(v')} c'(x) && \text{by definition of the cost function and } \mathsf{label}(v') = \vee \\
&= \sum_{x \in \mathsf{child}(v) \setminus \{w\}} c'(x) + \sum_{x \in \mathsf{child}(w)} c'(x) && \text{by Lines 6-7 of Algorithm 2} \\
&= \sum_{x \in \mathsf{child}(v) \setminus \{w\}} c(x) + \sum_{x \in \mathsf{child}(w)} c(x) \\
& && \text{since } c(x) \text{ only depends on } c(y) \text{ for } y \in \mathsf{child}(x) \\
&= c(w) + \sum_{x \in \mathsf{child}(v) \setminus \{w\}} c(x) \\
& && \text{by definition of the cost function, and since } \mathsf{label}(w) = \vee \\
&= \sum_{x \in \mathsf{child}(v)} c(x) \\
&= c(v),
\end{aligned}
$$

  where the last equality follows from the definition of the cost function in Definition 3.13, and since the label of v was assumed to be $\vee$. Since the cost function of any given node depends only on its immediate children it follows that the cost function is not changed in any other part of the tree.

- Similarly, if v and w are both labeled by $\wedge$, then since both v and w have more than 1 child, so does v', and hence $c(v) = c'(v') = 1$, and thus in this case the cost function is not changed anywhere else either.

Thus, the contract procedure leaves $P(\mathcal{T})$ invariant. □

Claim 3.16. *In Algorithm 3, if the score is increased by 1, then $P(\mathcal{T})$ is decreased by 1. At any other point in the algorithm, the value of $P(\mathcal{T})$ does not change. In other words,* score $+ P(\mathcal{T})$ *is constant throughout the Prover-Delayer game.*

Proof. We already know from Claim 3.15 that the contract procedure does not impact the progress measure $P(\mathcal{T})$. Thus, it remains to check that the update procedure decreases the progress measure by 1 if and only if the Delayer scores a point. To that end, let $\mathcal{T}$ be the tree at the start of Algorithm 3, and let $\mathcal{T}'$ be the tree after the update procedure is performed, but before the contract procedure is called, in the final line of Algorithm 3. Let c, c', M and M' be the cost functions and marked sets of $\mathcal{T}$ and $\mathcal{T}'$, respectively.

Below, we go through the same case analysis as presented in Algorithm 3. Also see Figure 4 for a graphical description of each case.

1. Suppose that $\mathcal{T}$ contains only one leaf, which the Prover queries. We have $P(\mathcal{T}) = 1$, the Delayer scores a point in Line 3 of the Algorithm 3, and the update routine makes the tree empty in Line 3 of Algorithm 1. Thus, we find that $\mathcal{T}'$ is empty, and hence $P(\mathcal{T}) - P(\mathcal{T}') = 1 - 0 = 1$.

2. Suppose that the parent of the leaf being queried, denoted by v as in Line 7 of Algorithm 3, is labeled by $\vee$. Then, the Delayer sets the corresponding variable to 0 in Line 5 of Algorithm 3, which means that Line 5 of Algorithm 1 removes ℓ from the tree. We find that $c(v) - c'(v) = c(\ell) = 0$, and hence $c|_{\mathcal{T}'} = c'$. Moreover, if v had more than 2 children in $\mathcal{T}$, then $M = M'$, and hence $P(\mathcal{T}) = P(\mathcal{T}')$. On the other hand, if v has exactly 2 children in $\mathcal{T}$, then v will have only one child in $\mathcal{T}'$ and hence $v \notin M'$. Let w be the other child of v in $\mathcal{T}$. Since $\mathcal{T}$ is in reduced form, w is either a leaf or its label is $\wedge$, wich implies that $w \notin M \cap M'$. Moreover, if w is an internal node, it must have at least 2 children, and hence any node in the subtree rooted at w will have a proper parent that is also contained in that subtree. Therefore, $M = M'$, and since the cost function evaluated at any given root only depends on the tree rooted at the given node, we also find that $c'(v) = c'(w) = c(w) = c(v)$. Thus, $P(\mathcal{T}) = P(\mathcal{T}')$, and hence the progress measure is not affected.

3. Next we analyze the case where the parent node v has label $\wedge$. Since $\mathcal{T}$ is in reduced form, v has at least 2 children, and hence $c(v) = 1$. Suppose that v has more than 2 children. Then, Line 9 of Algorithm 3 sets the corresponding variable to 1, which means that Line 5 of Algorithm 1 removes ℓ from the tree. Since the remaining number of children of v is still at least 2, we have that $c'(v) = c(v) = 1$, and so also on the remainder of the tree, the cost function is not changed. Similarly, $M = M'$, and hence the $P(\mathcal{T}') = P(\mathcal{T})$.

4. We next consider the case where v has exactly 2 children, and we let m be the other child of v, as in Line 14 of Algorithm 3. If v is the root of $\mathcal{T}$, then Line 12 in Algorithm 3 sets the variable associated to the queried leaf to 1, which means that Line 5 in Algorithm 1 removes the queried leaf from the tree. The cost function on all nodes in $\mathcal{T}$ apart from v will not be altered, and $v \notin M \cup M'$ and $m \in M \cap M'$. Thus, $P(\mathcal{T}) = P(\mathcal{T}')$, proving that the progress measure is left unaltered.

5. It remains to check the case where v is not the root of $\mathcal{T}$, in which case we can let $u = \mathsf{parent}(v)$. Since $\mathcal{T}$ is in reduced form, we have that the label of u must be different from the label of v, so we find that $\mathsf{label}(u) = \vee$. Suppose that m is an internal vertex and that at least one child of m is labeled by $\wedge$, then we have $c(m) \geq 1$, and similarly $c(u) \geq 1$. Line 16 of Algorithm 3 now sets the queried leaf to 1, which means that Line 5 in Algorithm 1 removes the queried leaf from the tree. Since v is now left with just one child, $c'(v) = c'(m) = c(m)$. This implies that $c'(u) - c(u) = c'(v) - c(v) = c(m) - 1$, and we also find that $u, m \in M$ and $u \in M'$, but $m \notin M'$. Thus,

$$\begin{aligned}\mathcal{P}(\mathcal{T}) - \mathcal{P}(\mathcal{T}') &= \max\{c(u) - 1, 0\} + \max\{c(m) - 1, 0\} - \max\{c'(u) - 1, 0\} \\ &= c(u) - 1 + c(m) - 1 - (c'(u) - 1) \\ &= c(u) - c'(u) + c(m) - 1 = 0,\end{aligned}$$

from which we observe that the progress measure is indeed not altered.

6. It remains to check the case where m does not have a child labeled by $\wedge$, which implies that it is a leaf itself or all its children must be leaves, and thus $c(m) = 0$. Suppose that all children of u, except for v, are leaves, which means that $c(u) = c(v) = 1$. Then, Line 20 of Algorithm 3 sets the variable corresponding to the queried leaf to 1, which means that Line 5 of Algorithm 1 removes it from the tree. Then, we find that $c'(u) = c'(v) = c'(m) = c(m) = 0$, and since $\mathcal{T}$ is in reduced form, the parent of u, if it exists, must be labeled by $\wedge$ and have multiple children, which means that the cost function is not altered anywhere outside of the subtree rooted at u. Finally, since

$$\max\{c(u) - 1, 0\} = \max\{c(m) - 1, 0\} = \max\{c'(u) - 1, 0\} = \max\{c'(m) - 1, 0\} = 0,$$

we conclude that none of the vertices in the subtree rooted at u makes any contribution to the progress measure, and hence it remains unaffected.

7. Thus, it remains to investigate the case where u has multiple children that are labeled by $\wedge$, which means that $c(u) \geq 2$. In that case, the Delayer scores a point in Line 22 of Algorithm 3. If the Prover chooses the value 1, then Line 5 of Algorithm 1 removes the leaf from the tree. Since $c(u) \geq 2$, and $c(v) = 1$, we find that $c'(u) \geq 1$. Since $\mathcal{T}$ is in reduced form, we find that the label of the parent of u, if it exists, must be $\wedge$, and it must have multiple children, from which we infer that the cost function remains unaffected on all of $\mathcal{T}$ except for the subtree rooted at u. Furthermore, we have $u \in M \cap M'$, and $\max\{c(m) - 1, 0\} = \max\{c'(m) - 1, 0\} = 0$, which means that

$$\begin{aligned}P(\mathcal{T}) - P(\mathcal{T}') &= \max\{c(u) - 1, 0\} - \max\{c'(u) - 1, 0\} \\ &= c(u) - c'(u) = c(v) - c'(v) = 1 - c'(m) = 1 - c(m) = 1,\end{aligned}$$

which indeed proves that the progress measure is decreased by 1.

8. On the other hand, if the Prover chooses 0, then Line 7 in Algorithm 1 removes the leaf from the tree, as well as the node v and the subtree rooted at it. Again, the cost function is not altered on all of $\mathcal{T}$, except for the subtree rooted at u. If u has more than 2 children in $\mathcal{T}$, then, we find that $u \in M$ and $u \in M'$, which implies

$$P(\mathcal{T}) - P(\mathcal{T}') = \max\{c(u) - 1, 0\} - \max\{c'(u) - 1, 0\} = c(u) - c'(u) = c(v) = 1,$$

and if u has exactly 2 children in $\mathcal{T}$, then $c(u) = 2$, $u \in M$ and $u \notin M'$, from which we find that

$$P(\mathcal{T}) - P(\mathcal{T}') = \max\{c(u) - 1, 0\} = 1.$$

Thus, indeed, the progress measure is decreased by 1 iff the Prover scores a point (which happened in Items 1, 7 and 8 above). This completes the proof. □

Now that we know that the Delayer's strategy ensures that $\mathsf{score} + P(\mathcal{T})$ remains constant throughout the Prover-Delayer game, we observe that the initial value of the progress measure tells us how many points will be scored throughout the game.

Claim 3.17. *At the start of the game, the progress measure is* $(n+2)/3$.

Proof. At the beginning of Algorithm 3 the tree $\mathcal{T}$ is a complete binary tree with every node $v \in \mathcal{T}$ with $\mathsf{label}(v) = \vee$ has exactly two children, say w_1 and w_2, with $\mathsf{label}(w_1) = \mathsf{label}(w_2) = \wedge$. Since w_1 and w_2 both have two children as well, we have $c_\wedge(w_1) = c_\wedge(w_2) = 1$, which implies that $\max\{c_\wedge(v) - 1, 0\} = 2$. Therefore, the progress measure is equal to 1 plus the number of $\vee$-labelled nodes, which can be easily seen to equal (with $4^k = n$),

$$1 + \sum_{j=1}^{k} \frac{n}{4^j} = 1 + \frac{n}{4} \sum_{j=0}^{k-1} \frac{1}{4^j} = 1 + \frac{n}{4} \frac{1 - \frac{1}{4^k}}{1 - \frac{1}{4}} = 1 + \frac{n}{4} \cdot \frac{1 - \frac{1}{n}}{\frac{3}{4}} = 1 + \frac{n-1}{3} = \frac{n+2}{3},$$

completing the proof. □

Claim 3.18. *The rank of* F *is at least* $(n+2)/3$.

Proof. The progress measure starts at $(n+2)/3$ (Claim 3.17), decreases by exactly 1 whenever a point is scored (Claim 3.16), and the game doesn't end until it is 0 (Claim 3.14). This proves that the Delayer can at least score $(n+2)/3$ points. Thus, by using Claim 3.6, we see that the rank is indeed bounded from below by $(n+2)/3$. □

3.2.2 Prover's Strategy (Upper Bound)

The Prover also starts by building the complete AND-OR tree $\mathcal{T}$ on n bits. Then, in every round of the Prover-Delayer game, the Prover asks the variable that is associated to an arbitrary leaf of the tree $\mathcal{T}$. Next, if the Delayer answers 0 or 1, the Prover updates the tree with the update procedure, Algorithm 1. If instead, the Delayer asks the Prover to choose, then the Prover always chooses 0, and updates the tree accordingly. Finally, the Prover puts the tree back into reduced form using the $\mathsf{contract}$ procedure. The algorithm is formally stated in Algorithm 4.

Algorithm 4 Prover's strategy

Input: $\mathcal{T}$

Query an arbitrary leaf of $\mathcal{T}$, associated with variable i.
$b \leftarrow$ answer from Delayer.
if $b =$ Prover's choice **then**
    Choose 0.
    $b \leftarrow 0$.

Output: $\mathsf{contract}(\mathsf{update}(\mathcal{T}, i, b))$.

We now argue that if the Prover uses the strategy outlined in Algorithm 4, the Delayer cannot score more than $(n+2)/3$ points. To that end, we introduce the following progress measure:

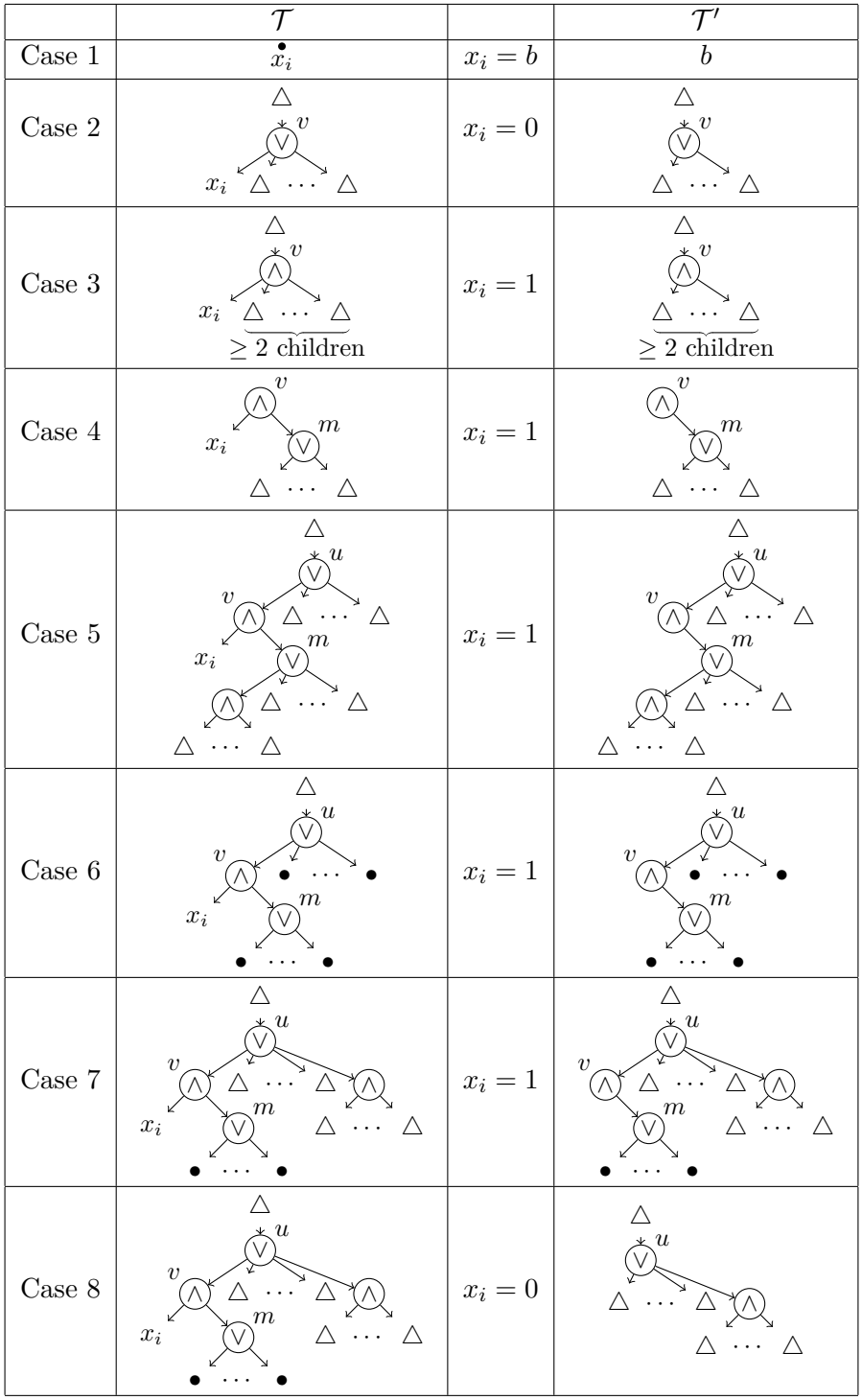


Figure 4: Graphical depiction of the cases considered in the proof of Claim 3.16. Black dots indicate leaves, and triangles depict arbitrary AND-OR trees. In cases 6 to 8, the subtree rooted at m can also be replaced by a single leaf.

Definition 3.19. *Let $\mathcal{T}$ be an AND-OR tree (not necessarily in reduced form). We define the following cost function recursively on the nodes of $\mathcal{T}$, as*

$$d(v) = \begin{cases} \sum_{w \in \mathsf{child}(v)} d(w), & \text{if } \mathsf{label}(v) = \vee, \\ d(w), & \text{if } \mathsf{child}(v) = \{w\}, \\ 1, & \text{otherwise.} \end{cases}$$

We also define the set of marked nodes M as before in Definition 3.13, as

$$M = \{v \in \mathcal{T} : \mathsf{label}(v) = \vee \text{ and } |\mathsf{child}(v)| \geq 2 \\ \text{and } (\mathsf{pparent}(v) = \mathsf{undefined} \text{ or } \mathsf{label}(\mathsf{pparent}(v)) = \wedge)\}.$$

We now define a new progress measure of $\mathcal{T}$, as

$$S(\mathcal{T}) = \begin{cases} 1 + \sum_{v \in M} \max\{d(v) - 1, 0\}, & \text{if } \mathcal{T} \text{ is non-empty,} \\ 0, & \text{otherwise.} \end{cases}$$

Note that the cost function introduced in Definition 3.19 only differs slightly from the cost function introduced in Definition 3.13, in that leaves have cost 1 in Definition 3.19, whereas they have cost 0 in Definition 3.19. Just as before, we observe that d evaluated at v only depends on the immediate children of v, and whether v belongs to M only depends on its parent.

Claim 3.20. *If the progress measure is 0, the Prover-Delayer game ends.*

Proof. If the progress measure is 0, then the tree is empty, and hence the function value must be completely determined. This implies that the game ends, completing the proof. □

Claim 3.21. *The* contract *procedure does not alter the value of $S(\mathcal{T})$.*

Proof. The first part of the contract procedure looks for a node v that has only one child w, and it contracts these into a new node v'. Let $\mathcal{T}$ be the tree before this contraction, and $\mathcal{T}'$ the resulting tree afterwards, and define the cost functions and marked sets d, d', M and M' accordingly. We have $d(v) = d(w)$, both when $\mathsf{label}(v) = \vee$ and $\mathsf{label}(v) = \wedge$, and we also have $d(v') = d(w)$, since $\mathsf{label}(v') = \mathsf{label}(w)$ and the cost function only depends on their immediate children. Next observe that $v \notin M$ since $|\mathsf{child}(v)| = 1$, and $\mathsf{pparent}(w) = \mathsf{pparent}(v')$. Thus, $w \in M \iff v' \in M'$. Since the cost of other nodes and the marked set remain unaffected in the remainder of the tree, we find that $S(\mathcal{T}') = S(\mathcal{T})$.

In the second part of the contract procedure, we find a node v and one of its children w having the same label, and we contract them into a new node v'. Again let $\mathcal{T}$ be the tree before this contraction, and $\mathcal{T}'$ the resulting tree afterwards. Define the cost functions d, d', M and M', accordingly.

- If $\mathsf{label}(v) = \mathsf{label}(w) = \vee$, then

$$\begin{aligned} d'(v') &= \sum_{x \in \mathsf{child}(v')} d'(x) && \text{by definition of the cost function and } \mathsf{label}(v') = \vee \\ &= \sum_{x \in \mathsf{child}(v) \setminus \{w\}} d'(x) + \sum_{x \in \mathsf{child}(w)} d'(x) && \text{by Lines 6-7 in Algorithm 2} \end{aligned}$$

$$= \sum_{x \in \mathsf{child}(v) \setminus \{w\}} d(x) + \sum_{x \in \mathsf{child}(w)} d(x)$$

since $d(x)$ depends only on $d(y)$ for $y \in \mathsf{child}(x)$

$$= \sum_{x \in \mathsf{child}(v) \setminus \{w\}} d(x) + d(w)$$

by definition of the cost function and $\mathsf{label}(w) = \vee$

$$= d(v),$$

where the final equality follows from the definition of the cost function and the fact that $\mathsf{label}(v) = \vee$.

- Similarly, if $\mathsf{label}(v) = \mathsf{label}(w) = \wedge$, then because v and w both have more than one child each, then so does v', and hence $d'(v') = d(v) = 1$.

Thus, both in the cases where the labels are $\wedge$ and $\vee$, the cost function on the rest of the tree is not altered. Moreover, $w \notin M$ because its parent is v (which is its proper parent since all nodes have at least 2 children after the first phase of Algorithm 2), which has the same label as w. Moreover $v \in M \Leftrightarrow v' \in M'$, because their proper parents are the same. Thus, we find that $S(\mathcal{T}') = S(\mathcal{T})$, completing the proof. □

Claim 3.22. *Every time the Delayer asks the Prover to choose, the progress measure decreases by at least* 1*.*

Proof. We already know from Claim 3.21 that the contract procedure does not impact the progress measure $S(\mathcal{T})$. Thus, it remains to check that the update procedure decreases the progress measure by at least 1, whenever the Delayer allows the Prover to choose. To that end, $\mathcal{T}$ be the tree at the moment the Delayer asks the Prover to choose. Since the Prover puts $\mathcal{T}$ back into reduced form after every round, we can assume it is in reduced form. According to the strategy outlined in Algorithm 4, the Prover chooses 0. Let $\mathcal{T}'$ be the result of the update procedure, called in the last line of Algorithm 4, and let d, d', M and M' be the associated cost functions and marked sets.

Let the queried leaf be denoted by ℓ. We consider three cases.

1. Suppose ℓ is the final remaining leaf in the tree. Then, Line 3 in the update procedure, i.e., Algorithm 1, will render the tree completely empty, and hence we find that $S(\mathcal{T}) - S(\mathcal{T}') = 1 - 0 = 1$. Thus, the progress measure indeed decreases by 1.

2. Let $v = \mathsf{parent}(\ell)$. Suppose $\mathsf{label}(v) = \vee$. Then, since the Prover always chooses 0 according to Line 5 in Algorithm 4, we find that Line 5 of the update procedure, i.e., Algorithm 1, removes ℓ from the tree. Thus, we find that $d(v) - d'(v) = d(\ell) = 1$, by Definition 3.19. Moreover, $v \in M$ by Definition 3.19 because $\mathsf{label}(v) = \vee$, $|\mathsf{child}(v)| \geq 2$, and v is either the root node, or $\mathsf{label}(\mathsf{pparent}(v)) = \mathsf{label}(\mathsf{parent}(v)) = \wedge$ and $|\mathsf{child}(\mathsf{parent}(v))| \geq 2$ as $\mathcal{T}$ is in reduced form. Moreover, $d(v) \geq 2$, because v has at least 2 children that are either leaves or $\wedge$-nodes, and similarly $d'(v) \geq 1$. Thus, $\max\{d(v) - 1, 0\} = d(v) - 1$ and $\max\{d'(v) - 1, 0\} = d'(v) - 1$. Finally, the cost of other nodes and the set of marked nodes remain unaffected in the other parts of the tree, we find that $S(\mathcal{T}) - S(\mathcal{T}') = d(v) - d'(v) = 1$. Hence, also in this case the progress measure is decreased by 1.

3. On the other hand, suppose $\mathsf{label}(v) = \wedge$. Since the Prover chooses 0, according to Line 5 of Algorithm 4, Line 7 of the update procedure, i.e., Algorithm 1, removes this node and the subtree rooted at v. If this node was the root, then the resulting

tree is empty, meaning that $S(\mathcal{T}) - S(\mathcal{T}') \geq 1 - 0 = 1$, and hence the progress measure is indeed decreased by at least 1. Otherwise, if v was not the root, then let $u = \mathsf{parent}(v)$. We find that $\mathsf{label}(u) = \vee$, because $\mathcal{T}$ was in reduced form. Furthermore, since $|\mathsf{child}(u)| \geq 2$ in $\mathcal{T}$, we find that $d(u) \geq 2$, and $d'(u) \geq 1$. Furthermore, $v \notin M$ because $\mathsf{label}(v) = \wedge$, and $u \in M$, because $\mathsf{label}(u) = \vee$, and the other conditions in the definition of M are satisfied too. Moreover, the cost function above the node u is not affected, and $M' \setminus \{u\} \subseteq M \setminus \{u\}$, where strict containment only happens whenever M contains nodes in the subtree rooted at v in $\mathcal{T}$.

There are now two possible cases:

- If $|\mathsf{child}(u)| > 2$ in $\mathcal{T}$, then $|\mathsf{child}(u)| \geq 2$ in $\mathcal{T}'$, and hence we find that $u \in M'$ as well. Therefore, the difference between the old and new progress measures can be expressed as $S(\mathcal{T}) - S(\mathcal{T}') \geq d(u) - d'(u) = d(v) = 1$. Thus, indeed, the progress measure decreases by at least 1.
- This leaves the case where $|\mathsf{child}(u)| = 2$ in $\mathcal{T}$, in which case we obtain that $d(u) = 2$ and $|\mathsf{child}(u)| = 1$ in $\mathcal{T}'$, and so $u \notin M'$. This implies that $S(\mathcal{T}) - S(\mathcal{T}') \geq \max\{d(u)-1, 0\} = 2-1 = 1$, and hence the progress measure decreases by at least 1 in this case as well.

This completes the proof. □

Claim 3.23. *Initially, the progress measure $S(\mathcal{T})$ is $(n+2)/3$.*

Proof. Initially, every $\vee$-gate in the AND-OR tree is in M, because it is either the root node, or it has a parent node with 2 children labeled by $\wedge$. Moreover, the cost of every node labeled by $\vee$ is 2, because it has two children, both labeled by $\wedge$. Thus, we obtain that $S(\mathcal{T})$ is 1 plus the number of nodes labeled by $\vee$, which is $(n+2)/3$ (see the proof of Claim 3.17). □

Claim 3.24. *The rank of F is at most $(n+2)/3$.*

Proof. Since the progress measure starts at $(n+2)/3$ (Claim 3.23), with every point scored it decreases by at least 1 (Claim 3.22), and the game ends whenever the progress measure S is 0 (Claim 3.20), the Prover can ensure that the number of points scored by the Delayer is at most $(n+2)/3$. From Claim 3.6, we infer that $\mathsf{rank}(F) \leq (n+2)/3$, completing the proof. □

Proof of Theorem 3.9. Combining Claim 3.18 and Claim 3.24 completes the proof. □

Proof of Theorem 1.4. Since randomized rank is at most randomized decision tree complexity, Theorem 3.8 implies

$$\mathsf{rrank}(F) \leq \mathsf{R}(f) = \Theta\left(n^{\log \frac{1+\sqrt{33}}{4}}\right) \approx \Theta\left(n^{.753\ldots}\right).$$

Theorem 3.9 implies

$$\mathsf{rank}(F) = \frac{n+2}{3}.$$

This proves the theorem. □

4 Proof of Theorem 1.6

The main results of this section provide a characterization of the optimal witness complexity and objective value of the dual adversary bound, based on the weighting scheme in Beigi and Taghavi's construction of an NBSPwOI and a dual adversary solution for $\tilde{f}$ given a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ and a decision tree $\mathcal{T}$ computing it [BT20, Section 3] (recall from Section 2 that $\tilde{f}$ takes an input $x \in \{0,1\}^n$ and outputs the leaf of $\mathcal{T}$ reached on input x), in terms of the objective value of Program 1. We describe their construction in a modular fashion: we leave the choice of 'weights' of the vectors in the span program and dual adversary solution unfixed. We show that the witness complexity and dual adversary bounds thus obtained are captured by the objective value of Program 1. In the next section we demonstrate a choice of weights and prove its optimality. In Appendix B we exhibit another interesting choice of weights.

4.1 Span Program Construction

In order to define the NBSPwOI, we first assign strictly positive real weights W_e to all edges e in the decision tree. These weights play a crucial role in the witness complexity analysis.

The following is the NBSPwOI for $\tilde{f}$.

- The vector space is the span of $\{|v\rangle : v \in V(\mathcal{T})\}$.
- The input vectors are

$$I_{j,q} = \bigcup_{v \in V_I(\mathcal{T}) : J(v) = j} \left\{ \sqrt{W_{e(v,N(v,q))}} \left(|v\rangle - |N(v,q)\rangle \right) \right\}. \tag{10}$$

  That is, for all $j \in [n]$ and $q \in \{0,1\}$, the input vectors correspond to edges corresponding to answers of queries of the form $x_j = q$. In other words, for every vertex $v \in V_I(\mathcal{T})$, $e(v, N(v, x_{J(v)}))$ is always the unique available outgoing edge of v. Moreover, these vectors are weighted, and we leave these weights variable for now.

- Let r denote the root vertex of $\mathcal{T}$. For each leaf u of $\mathcal{T}$, the associated target vector is given by

$$|t_u\rangle = |r\rangle - |u\rangle.$$

We now give positive and negative witnesses for every $x \in D_f$, argue that the above span program evaluates $\tilde{f}$, and analyze the positive and negative witness complexities.

Note that, we use $v \in P_x$ to denote a vertex in the path P_x, and, we use $e \in P_x$ to denote an edge in the path P_x. For every $x \in D_f$, we can express the corresponding target vector by a telescoping sum of vectors that are all available to x, as

$$\begin{aligned} |t_{\tilde{f}(x)}\rangle = |r\rangle - |\tilde{f}(x)\rangle &= \sum_{v \in P_x \setminus \{\tilde{f}(x)\}} |v\rangle - |N(v, x_{J(v)})\rangle \\ &= \sum_{v \in P_x \setminus \{\tilde{f}(x)\}} \frac{1}{\sqrt{W_{e(v,N(v,x_{J(v)}))}}} \left(\sqrt{W_{e(v,N(v,x_{J(v)}))}} \left(|v\rangle - |N(v, x_{J(v)})\rangle \right) \right). \end{aligned}$$

Note from Equation (10) that all the vectors in the curly braces above are available for x. Any vector in $\mathbb{C}^I$ can be viewed as indexed by elements of I. Define $|w_x\rangle \in \mathbb{C}^{|I|}$ as follows: For a node $v \in P_x \setminus \left\{ \tilde{f}(x) \right\}$, assign the value $\frac{1}{\sqrt{W_{e(v,N(v,x_{J(v)}))}}}$ to the entry corresponding

to the vector $\sqrt{W_{e(v,N(v,x_{J(v)}))}}\left(|v\rangle - |N(v,x_{J(v)})\rangle\right)$, and set all other coefficients to 0. On the other hand, we let

$$|\overline{w}_x\rangle = \sum_{v\in P_x} |v\rangle.$$

For any vector $|v''\rangle$ in $I(x)$, because it is of the form

$$|v''\rangle = \sqrt{W_{e(v',N(v',x_{J(v')}))}}\left(|v'\rangle - |N(v',e(v',x_{J(v')}))\rangle\right)$$

for some $v' \in V_I(\mathcal{T})$, we have

$$\langle\overline{w}_x|v''\rangle = \sum_{v\in P_x} \sqrt{W_{e(v',N(v',x_{J(v')}))}}\left(\langle v|v'\rangle - \langle v|N(v',e(v',x_{J(v')}))\rangle\right) = 0.$$

For a leaf $u \neq \tilde{f}(x)$ of $\mathcal{T}$,

$$\langle t_u|\overline{w}_x\rangle = \sum_{v\in P_x}\langle r|v\rangle - \sum_{v\in P_x}\langle u|v\rangle = 1 - 0 = 1.$$

This implies that the NBSPwOI indeed computes $\tilde{f}$. For the positive and negative witness sizes, we have

$$\mathrm{wsize}^+(P,w,\overline{w}) = \max_{x\in D_f} \||w_x\rangle\|^2 = \max_{x\in D_f} \sum_{v\in P_x\setminus\{\tilde{f}(x)\}} \frac{1}{W_{e(v,N(v,x_{J(v)}))}} = \max_{P\in P(\mathcal{T})}\sum_{e\in P}\frac{1}{W_e},$$

and

$$\begin{aligned}\mathrm{wsize}^-(P,w,\overline{w}) &= \max_{x\in D_f}\left\|A^\dagger|\overline{w}_x\rangle\right\|^2 = \max_{x\in D_f}\left\|\sqrt{W_{e(v,N(v,x_{J(v)}))}}\left(\langle v| - \langle N(v,x_{J(v)})|\right)|\overline{w}_x\rangle\right\|^2 \\ &= \sum_{v\in P_x\setminus\{\tilde{f}(x)\}} W_{e(v,N(v,\overline{x_{J(v)}}))} = \max_{P\in P(\mathcal{T})}\sum_{e\in\overline{P}} W_e.\end{aligned}$$

Now, it remains to find the weight assignment W that minimizes the total complexity of the NBSPwOI, which is given by

$$\mathrm{wsize}(P,w,\overline{w}) = \sqrt{\mathrm{wsize}^-(P,w,\overline{w})\cdot\mathrm{wsize}^+(P,w,\overline{w})} = \sqrt{\max_{P\in P(\mathcal{T})}\sum_{e\in P}\frac{1}{W_e}\cdot\max_{P\in P(\mathcal{T})}\sum_{e\in\overline{P}} W_e}.$$

Thus, if we assign weights W to the edges of $\mathcal{T}$ as earlier in this section, and set $\alpha = \max_{P\in P(\mathcal{T})}\sum_{e\in\overline{P}} W_e$ and $\beta = \max_{P\in P(\mathcal{T})}\sum_{e\in P}\frac{1}{W_e}$, the construction from earlier in this section gives rise to an explicit NBSPwOI computing $\tilde{f}$ with witness complexity of $\sqrt{\alpha\beta}$. This is exactly captured by Program 1, giving us the following theorem.

Theorem 4.1. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation, let $\mathcal{T}$ be a decision tree computing f and let $\mathrm{OPT}_{\mathcal{T}}$ denote the optimal value of Program 1. Then, for the construction of $(P,w,\overline{w})$ with variable weights as earlier in this section,*

$$\mathrm{wsize}(P,w,\overline{w}) = \mathrm{OPT}_{\mathcal{T}}.$$

In the next subsection we show that a solution to Program 1 also gives a dual adversary solution with the same objective value.

4.2 Dual Adversary Solution

We give a refined analysis of Beigi and Taghavi's construction of a dual adversary solution for a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ given a deterministic decision tree $\mathcal{T}$ that computes f. Recall that for a deterministic tree $\mathcal{T}$ computing f, $\tilde{f}$ is the function that takes input $x \in D_f$ and outputs the leaf of $\mathcal{T}$ reached on input x. We show that a dual adversary solution for $\tilde{f}$ can also be obtained by different settings of weights as in the previous subsection, and obtain a corresponding dual adversary bound as the optimum value of Program 1.

We construct vectors $\{|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{|w_{xj}\rangle : x \in D_f, j \in [n]\}$ that are feasible solutions to Program 2 for $\tilde{f}$, which we recall below.

Variables	$\{\|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{\|w_{xj}\rangle : x \in D_f, j \in [n]\}, d$	
Minimize	$\max_{x \in D_f} \max\left\{\sum_{j=1}^n \||u_{xj}\rangle\|^2, \sum_{j=1}^n \||w_{xj}\rangle\|^2\right\}$	
s.t.	$\sum_{j \in [n]: x_j \neq y_j} \langle u_{xj}\|w_{yj}\rangle = 1 - \delta_{\tilde{f}(x),\tilde{f}(y)}$	$\forall x, y \in D_f$
	$\|u_{xj}\rangle, \|w_{xj}\rangle \in \mathbb{C}^d$	for all $x \in D_f$

Program 3: Dual SDP for $\tilde{f}$

Let $V_I(\mathcal{T})$ denote the set of internal nodes of $\mathcal{T}$. Consider the basis set $\{|v\rangle : v \in V_I(\mathcal{T})\}$. We construct the vectors $|u_{xj}\rangle$ and $|w_{xj}\rangle$ in the space $\mathbb{C}^{V_I(\mathcal{T})}$. Additionally, we use $V_j(\mathcal{T})$ to denote the set of vertices associated with query index j, i.e., $V_j(\mathcal{T}) = \{v \in V_I(\mathcal{T}) : J(v) = j\}$.

Define $|u_{xj}\rangle$ and $|w_{xj}\rangle$ as follows.

$$|u_{xj}\rangle = \begin{cases} \frac{1}{\sqrt{W_{e(v,N(v,x_j))}}}|v\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}), \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

and,

$$|w_{xj}\rangle = \begin{cases} \sqrt{W_{e(v,N(v,\overline{x_j}))}}|v\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}), \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

We claim that these vectors form a feasible solution to Program 3. Fix $x, y \in D_f$ with $\tilde{f}(x) \neq \tilde{f}(y)$. We now verify that the corresponding equality constraint in Program 3 is satisfied.

- There is a unique vertex in $\mathcal{T}$ where P_x and P_y deviate. Let $v \in V_I(\mathcal{T})$ denote this vertex, and let its associated query index be $J(v) = i$. We have $v \in P_x \cap P_y$ and $x_i \neq y_i$. In that case $\langle u_{xi}|w_{yi}\rangle = 1$ by the definitions of $|u_{xi}\rangle$ and $|w_{yi}\rangle$ from Equations (11) and (12).
- Consider an index $j \in [n] \setminus \{i\}$ such that $x_j \neq y_j$. Let v' and v'' be the vertices on P_x and P_y, respectively (if they exist), with $J(v') = J(v'') = j$. By the previous point, $v' \notin P_y$ and $v'' \notin P_x$. Thus, $\langle v'|v''\rangle = 0$ from Equations (11) and (12), which implies $\langle u_{xj}|w_{yj}\rangle = 0$.

Thus, we have for all $x, y \in D_f$ with $\tilde{f}(x) \neq \tilde{f}(y)$,

$$\sum_{j \in [n]: x_j \neq y_j} \langle u_{xj}|w_{yj}\rangle = 1 - \delta_{\tilde{f}(x),\tilde{f}(y)}.$$

In the case when $\tilde{f}(x) = \tilde{f}(y)$, the right hand side in the constraint evaluates to 0 and so does the left side, because the indices where x and y differ cannot be queried on their path

since x and y reach the same leaf in $\mathcal{T}$. Therefore, the set of vectors $\{|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{|w_{xj}\rangle : x \in D_f, j \in [n]\}$, and $d = |V_I(\mathcal{T})|$ form a feasible solution to Program 3 for $\tilde{f}$.

Theorem 4.2. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation, and let $\mathcal{T}$ be a decision tree computing it. Let C denote the optimal value of Program 3 with variable weights as earlier in this section, and let $\mathrm{OPT}_{\mathcal{T}}$ denote the optimal value of Program 1. Then $C = \mathrm{OPT}_{\mathcal{T}}$.*

Proof of Theorem 4.2. Let C denote the objective value of Program 3 with the settings of vectors $\{|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{|w_{xj}\rangle : x \in D_f, j \in [n]\}$ as defined in Equations (11) and (12), and $d = |V_I(\mathcal{T})|$.

We now argue that $C = \mathrm{OPT}_{\mathcal{T}}$, where $\mathrm{OPT}_{\mathcal{T}}$ denotes the optimal solution of Program 1. First note that for all $x \in D_f$,

$$\sum_{j=1}^{n} \||u_{xj}\rangle\|^2 = \sum_{e \in P_x} \frac{1}{W_e} \qquad \text{and} \qquad \sum_{j=1}^{n} \||w_{xj}\rangle\|^2 = \sum_{e \in \overline{P}_x} W_e.$$

Thus,

$$C = \min_{x \in D_f} \max \left\{ \sum_{j=1}^{n} \||u_{xj}\rangle\|^2, \sum_{j=1}^{n} \||w_{xj}\rangle\|^2 \right\} = \min_{x \in D_f} \max \left\{ \sum_{e \in P_x} \frac{1}{W_e}, \sum_{e \in \overline{P}_x} W_e \right\}.$$

Thus we can alternatively view C to be an optimal solution to the following optimization program.

Variables	$\{W_e : e \text{ is an edge in } \mathcal{T}\}, \alpha, \beta$		
Minimize	$\max\{\alpha, \beta\}$		
s.t.	$\sum_{e \in \overline{P}} W_e$	$\leq \alpha,$	for all paths $P \in P(\mathcal{T})$
	$\sum_{e \in P} \frac{1}{W_e}$	$\leq \beta,$	for all paths $P \in P(\mathcal{T})$
	W_e	$> 0,$	for all edges e in $\mathcal{T}$
	α, β	$\geq 0.$	

Program 4

We now show $C = \mathrm{OPT}_{\mathcal{T}}$. Let $\{W_e : e \text{ edge in } \mathcal{T}\}, \alpha, \beta$ be settings of variables in a feasible solution to Program 4, with objective value C. Clearly the same settings of variables also form a feasible solution to Program 1, since the constraints are the same. The corresponding objective value of Program 1 is $\sqrt{\alpha\beta} \leq \max\{\alpha, \beta\} = C$. Thus, $\mathrm{OPT}_{\mathcal{T}} \leq C$.

In the other direction, let $\{W_e : e \text{ edge in } \mathcal{T}\}, \alpha, \beta$ be settings of variables in a feasible solution to Program 1 with objective value $\mathrm{OPT}_{\mathcal{T}}$. Set

$$\begin{aligned} W'_e &= \sqrt{\beta/\alpha} \cdot W_e \qquad \text{for all edges } e \text{ in } \mathcal{T}, \\ \alpha' &= \sqrt{\alpha\beta}, \\ \beta' &= \sqrt{\alpha\beta}. \end{aligned}$$

It is easy to verify that this setting of variables is feasible for Program 4, and attains objective value $\max\{\alpha', \beta'\} = \sqrt{\alpha\beta} = \mathrm{OPT}_{\mathcal{T}}$. Thus, $C \leq \mathrm{OPT}_{\mathcal{T}}$, proving the theorem. □

We now prove Theorem 1.6, using results proved earlier in this section.

Proof of Theorem 1.6. We give two proofs, one via span program witness complexity, and another via the dual adversary bound.

1. Consider the NBSPwOI $(P, w, \overline{w})$ for $\tilde{f}$ as constructed in Section 4.1. Theorem 4.1 implies $\mathrm{wsize}(P, w, \overline{w}) \leq \mathrm{OPT}_{\mathcal{T}}$. Theorem 2.4 implies $\mathsf{Q}(\tilde{f}) = O(\mathrm{OPT}_{\mathcal{T}})$.

2. Consider the dual adversary solution for $\tilde{f}$ as constructed in Section 4.2. Theorems 2.5 and 4.2 imply $\mathsf{Q}(\tilde{f}) = O(C) = O(\mathrm{OPT}_{\mathcal{T}})$.

□

5 An Optimal Weight Assignment

It now remains to investigate how we can assign weights to edges in a decision tree so as to optimize the objective value of Program 1. Beigi and Taghavi gave an explicit weighting scheme by coloring the edges of the decision tree with two colors, and then assigning weights depending on the color that the edge is colored by [BT20, Section 3]. They raised the question whether their scheme can be significantly improved upon. We answer this question affirmatively, by giving an *optimal* solution to the weight optimization program from Definition 1.5, and providing a constructive algorithm to compute the optimal weights in this section. We also give an alternative, albeit sub-optimal, assignment of weights in Appendix B.

First, we define the weight assignment, which we will refer to as the *canonical weight assignment*, in Definition 5.1. The construction we present is recursive, in the sense that we first assign weights to the edges connected to the leaves, and subsequently work our way up the tree until we reach the root node. Then, we prove its optimality, resulting in Theorem 5.4. Finally, we prove Corollary 1.7, which gives some upper bounds on the optimal objective value, in terms of natural measures of the decision tree.

Definition 5.1 (Canonical weight assignment)**.** *Let $\mathcal{T}$ be a non-trivial decision tree with root node r. Let L and R be the two children nodes of r, connected to r by the edges e_L and e_R, respectively. Let $\mathcal{T}_L$ and $\mathcal{T}_R$ be the subtrees of $\mathcal{T}$ rooted at L and R, respectively. Then, we assign weights to e_L and e_R, by setting*

$$W_{e_L} = \frac{\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2},$$
$$W_{e_R} = \frac{\mathrm{OPT}_{\mathcal{T}_R} - \mathrm{OPT}_{\mathcal{T}_L} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_R} - \mathrm{OPT}_{\mathcal{T}_L})^2 + 4}}{2}.$$

In order to define the weights W_{e_L} and W_{e_R}, we need to know the optimal values of Program 1 for the subtrees $\mathcal{T}_L$ and $\mathcal{T}_R$. We now proceed to show how one can compute these optimal values via a recurrence relation.

Lemma 5.2. *Let $\mathcal{T}$ be a non-trivial decision tree, and let L and R be the two children nodes of the root node. Let $\mathcal{T}_L$ and $\mathcal{T}_R$ be the subtrees of $\mathcal{T}$ rooted at L and R, respectively. Then,*

$$\mathrm{OPT}_{\mathcal{T}} \leq \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2}.$$

Proof. Suppose we have weight assignments W_L and W_R on the subtrees $\mathcal{T}_L$ and $\mathcal{T}_R$, and positive parameters α_L, α_R and β_L, β_R such that they form feasible solutions to the optimization programs for the left and right subtrees $\mathcal{T}_L$ and $\mathcal{T}_R$, and such that they attain the optimal values $\sqrt{\alpha_L \beta_L} = \mathrm{OPT}_{\mathcal{T}_L}$ and $\sqrt{\alpha_R \beta_R} = \mathrm{OPT}_{\mathcal{T}_R}$. Now, let the weighting scheme W on $\mathcal{T}$ be defined such that

$$W_e = \sqrt{\beta_L / \alpha_L}(W_L)_e \tag{13}$$

for all edges e in $\mathcal{T}_L$,

$$W_{e'} = \sqrt{\beta_R/\alpha_R}(W_R)_{e'} \tag{14}$$

for all edges e' in $\mathcal{T}_R$, and choose W_{e_L} and W_{e_R} as in Definition 5.1. Furthermore, let

$$\alpha := \max\left\{\mathrm{OPT}_{\mathcal{T}_L} + W_{e_R}, \mathrm{OPT}_{\mathcal{T}_R} + W_{e_L}\right\}, \tag{15}$$

$$\beta := \max\left\{\mathrm{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}}, \mathrm{OPT}_{\mathcal{T}_R} + \frac{1}{W_{e_R}}\right\}. \tag{16}$$

First observe that for a path $P \in P(\mathcal{T})$ whose first edge is e_L (e_R, respectively) must have all remaining edges in $\mathcal{T}_L$ ($\mathcal{T}_R$, respectively). For every path $P \in P(\mathcal{T})$ containing e_L (essentially the same calculation also shows the same upper bound for paths containing e_R), we have

$$\begin{aligned}
\sum_{e\in P}\frac{1}{W_e} &= \frac{1}{W_{e_L}} + \sqrt{\frac{\alpha_L}{\beta_L}}\sum_{e\in P\setminus\{e_L\}}\frac{1}{(W_L)_e} && \text{by Equation (13)}\\
&\le \frac{1}{W_{e_L}} + \sqrt{\alpha_L\beta_L}\\
&= \frac{1}{W_{e_L}} + \mathrm{OPT}_{\mathcal{T}_L} \le \beta,
\end{aligned}$$

where the last inequality and equality follow since (W_L, α_L, β_L) is a feasible solution to Program 1 for $\mathcal{T}_L$. For every path $P \in P(\mathcal{T})$ containing e_L (essentially the same calculation also shows the same upper bound for paths containing e_R), we have

$$\begin{aligned}
\sum_{e\in \overline{P}} W_e &= W_{e_R} + \sqrt{\frac{\beta_L}{\alpha_L}}\sum_{e\in \overline{P}\setminus\{e_R\}}(W_L)_e && \text{by Equation (14)}\\
&\le W_{e_R} + \sqrt{\alpha_L\beta_L}\\
&= W_{e_R} + \mathrm{OPT}_{\mathcal{T}_L} \le \alpha,
\end{aligned}$$

where the last inequality and equality follow since (W_L, α_L, β_L) is a feasible solution to Program 1 for $\mathcal{T}_L$. Thus all constraints of Program 1 for $\mathcal{T}$ are satisfied with these values of α, β and the weighting scheme W. Hence,

$$\begin{aligned}
&\mathrm{OPT}_{\mathcal{T}} \le \sqrt{\alpha\beta}\\
&= \sqrt{\max\left\{\mathrm{OPT}_{\mathcal{T}_L} + W_{e_R}, \mathrm{OPT}_{\mathcal{T}_R} + W_{e_L}\right\}\cdot\max\left\{\mathrm{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}}, \mathrm{OPT}_{\mathcal{T}_R} + \frac{1}{W_{e_R}}\right\}}.
\end{aligned} \tag{17}$$

Finally, observe from our choices of W_{e_L} and W_{e_R} from Definition 5.1 that

$$\begin{aligned}
\frac{1}{W_{e_L}} &= \frac{2}{\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}\\
&= \frac{2(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R} - \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4})}{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 - (\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 - 4}\\
&= \frac{\mathrm{OPT}_{\mathcal{T}_R} - \mathrm{OPT}_{\mathcal{T}_L} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2} = W_{e_R}.
\end{aligned}$$

Hence Equation (17) implies

$$\begin{aligned}\mathrm{OPT}_{\mathcal{T}} &\leq \max\left\{\mathrm{OPT}_{\mathcal{T}_L} + W_{e_R}, \mathrm{OPT}_{\mathcal{T}_R} + W_{e_L}\right\} \\ &= \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2},\end{aligned}$$

where the last equality follows from our choices of W_{e_L} and W_{e_R} from Definition 5.1. This completes the proof. □

We now show that this weight assignment is optimal.

Lemma 5.3. *Let $\mathcal{T}$ be a non-trivial decision tree, and let L and R be the two children nodes of the root node. Let $\mathcal{T}_L$ and $\mathcal{T}_R$ be the subtrees of $\mathcal{T}$ rooted at L and R, respectively. Then,*

$$\mathrm{OPT}_{\mathcal{T}} \geq \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2}.$$

Proof. Suppose we have a weight assignment W and variables $\alpha, \beta > 0$, such that W, α and β form a feasible solution to the weight optimization program for $\mathcal{T}$, and attain optimality. We define

$$\alpha_L = \max_{P \in P(\mathcal{T}_L)} \sum_{e \in \overline{P}} W_e, \qquad \beta_L = \max_{P \in P(\mathcal{T}_L)} \sum_{e \in P} \frac{1}{W_e},$$

$$\alpha_R = \max_{P \in P(\mathcal{T}_R)} \sum_{e \in \overline{P}} W_e, \qquad \beta_R = \max_{P \in P(\mathcal{T}_R)} \sum_{e \in P} \frac{1}{W_e}.$$

Observe that $W|_{\mathcal{T}_L}$, α_L and β_L give a feasible solution to Program 1 for $\mathcal{T}_L$. Similarly, $W|_{\mathcal{T}_R}$, α_R and β_R give a feasible solution to Program 1 for $\mathcal{T}_R$. This implies that

$$\sqrt{\alpha_L \beta_L} \geq \mathrm{OPT}_{\mathcal{T}_L}, \qquad \sqrt{\alpha_R \beta_R} \geq \mathrm{OPT}_{\mathcal{T}_R}. \tag{18}$$

Furthermore, we have

$$\begin{aligned}\alpha &= \max_{P \in P(\mathcal{T})} \sum_{e \in \overline{P}} W_e = \max\left\{\max_{P \in P(\mathcal{T}_L)} \sum_{e \in \overline{P}} W_e + W_{e_R}, \max_{P \in P(\mathcal{T}_R)} \sum_{e \in \overline{P}} W_e + W_{e_L}\right\} \\ &= \max\left\{\alpha_L + W_{e_R}, \alpha_R + W_{e_L}\right\},\end{aligned} \tag{19}$$

and similarly

$$\begin{aligned}\beta &= \max_{P \in P(\mathcal{T})} \sum_{e \in P} \frac{1}{W_e} = \max\left\{\max_{P \in P(\mathcal{T}_L)} \sum_{e \in P} \frac{1}{W_e} + \frac{1}{W_{e_L}}, \max_{P \in P(\mathcal{T}_R)} \sum_{e \in P} \frac{1}{W_e} + \frac{1}{W_{e_R}}\right\} \\ &= \max\left\{\beta_L + \frac{1}{W_{e_L}}, \beta_R + \frac{1}{W_{e_R}}\right\}.\end{aligned} \tag{20}$$

Finally, let $\gamma = \sqrt{\beta/\alpha}$, and observe that

$$\mathrm{OPT}_{\mathcal{T}} = \sqrt{\alpha\beta} = \frac{\beta}{\gamma} = \gamma\alpha = \frac{1}{2}\left[\frac{\beta}{\gamma} + \gamma\alpha\right]. \tag{21}$$

Now, let

$$\delta = \frac{1}{2} + \frac{\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R}}{2\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}, \tag{22}$$

and observe that $\delta \in [0,1]$. Thus,

$$
\begin{aligned}
\mathrm{OPT}_{\mathcal{T}} &= \frac{1}{2}\left[\frac{\beta}{\gamma} + \gamma\alpha\right] && \text{by Equation (21)} \\
&= \frac{1}{2}\left[\max\left\{\frac{\beta_L}{\gamma} + \frac{1}{\gamma W_{e_L}}, \frac{\beta_R}{\gamma} + \frac{1}{\gamma W_{e_R}}\right\} + \max\left\{\gamma\alpha_L + \gamma W_{e_R}, \gamma\alpha_R + \gamma W_{e_L}\right\}\right] \\
& && \text{by Equations (19) and (20)} \\
&\geq \frac{1}{2}\left[\delta\left(\frac{\beta_L}{\gamma} + \frac{1}{\gamma W_{e_L}}\right) + (1-\delta)\left(\frac{\beta_R}{\gamma} + \frac{1}{\gamma W_{e_R}}\right)\right. \\
&\qquad \left. + \delta\left(\gamma\alpha_L + \gamma W_{e_R}\right) + (1-\delta)\left(\gamma\alpha_R + \gamma W_{e_L}\right)\right] \\
& && \text{since } \max\{a,b\} \geq \delta a + (1-\delta)b \text{ as } \delta \in [0,1] \\
&= \frac{1}{2}\left[\delta\left(\frac{\beta_L}{\gamma} + \gamma\alpha_L\right) + (1-\delta)\left(\frac{\beta_R}{\gamma} + \gamma\alpha_R\right)\right. \\
&\qquad \left. + \left(\frac{\delta}{\gamma W_{e_L}} + (1-\delta)\gamma W_{e_L}\right) + \left(\delta\gamma W_{e_R} + \frac{1-\delta}{\gamma W_{e_R}}\right)\right] && \text{rearranging terms} \\
&\geq \delta\sqrt{\alpha_L\beta_L} + (1-\delta)\sqrt{\alpha_R\beta_R} + \sqrt{\delta(1-\delta)} + \sqrt{\delta(1-\delta)} \\
& && \text{since } (a+b)/2 \geq \sqrt{ab} \text{ for all } a,b \geq 0 \\
&\geq \delta\mathrm{OPT}_{\mathcal{T}_L} + (1-\delta)\mathrm{OPT}_{\mathcal{T}_R} + 2\sqrt{\delta(1-\delta)} && \text{by Equation (18)} \\
&= \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R}}{2} + \frac{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2}{2\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}} \\
&\qquad + 2\sqrt{\frac{1}{4} - \frac{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2}{4((\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4)}} \\
& && \text{plugging the value of } \delta \text{ from Equation (22)} \\
&= \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R}}{2} + \frac{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2}{2\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}} + \sqrt{\frac{4}{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}} \\
&= \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R}}{2} + \frac{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}{2\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}} \\
&= \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2},
\end{aligned}
$$

completing the proof. □

We can now combine both lemmas above into the following theorem, providing a recursive characterization of the optimal value of Program 1 for all decision trees.

Theorem 5.4. *Let $\mathcal{T}$ be a non-trivial decision tree, and let L and R be the two children nodes of the root node. Let $\mathcal{T}_L$ and $\mathcal{T}_R$ be the subtrees of $\mathcal{T}$ rooted at L and R, connected to the root node by edges e_L and e_R, respectively. Then,*

$$
\mathrm{OPT}_{\mathcal{T}} = \frac{\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R} + \sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{2}.
$$

Proof. The upper bound on follows from Lemma 5.2 and the lower bound follows from Lemma 5.3. □

Observe that Definition 5.1 can be used to recursively assign optimal weights to the edges of any given decision tree. We display this technique in two examples in Figure 5. Note that the objective value of Program 1 for a trivial decision tree (a single node) is 0, which provides the basis for our recursion.

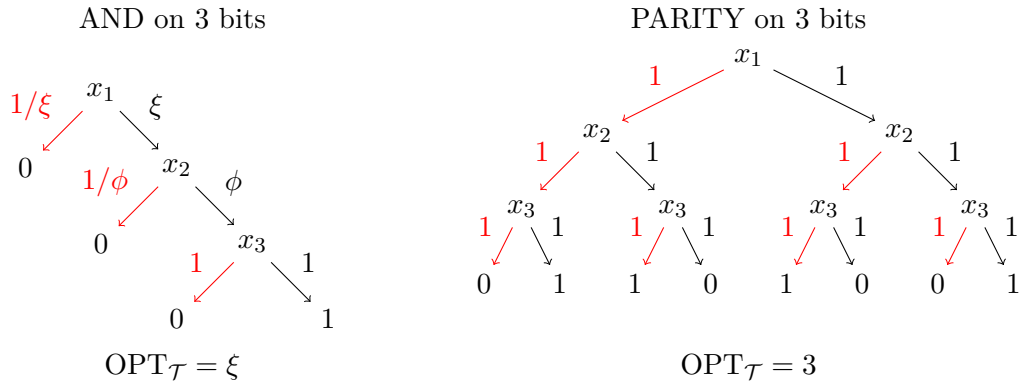


Figure 5: Examples of optimal weight assignments for two different decision trees. The red and black edges indicate the edges taken when the output of the query is 0 and 1, respectively, and the edge labels represent the weights. Left: Canonical weight assignment of the decision tree for the AND function on 3 bits, where $\phi = \frac{1+\sqrt{5}}{2}$, and $\xi = \frac{\phi+\sqrt{\phi+5}}{2}$. The objective value is ξ. Right: Canonical weight assignment of the decision tree for PARITY on 3 bits, with optimal value 3.

In Corollary 1.7 we exhibit two ways in which we can conveniently upper bound the optimum value of Program 1 in terms of well-studied measures of the underlying decision tree.

Proof of Corollary 1.7. Theorem 1.6 implies that it suffices to prove both of the required upper bounds on $\mathrm{OPT}_{\mathcal{T}}$ rather than $\mathsf{Q}(f)$. The rank-depth bound follows directly from the bound $\mathrm{OPT}_{\mathcal{T}} \leq 2\sqrt{G(\mathcal{T}) \cdot \mathrm{depth}(T)}$ derived in [BT20, Theorem 2], and the equality $G(\mathcal{T}) = \mathsf{rank}(\mathcal{T})$ from Claim 3.3.

For proving the size bound, we use induction to show that $\mathrm{OPT}_{\mathcal{T}} \leq \sqrt{2\mathrm{DTSize}(\mathcal{T})}$. First observe that it is true for the trivial decision tree. Next, suppose that it is true for the left and right subtrees $\mathcal{T}_L$ and $\mathcal{T}_R$ of $\mathcal{T}$. That is,

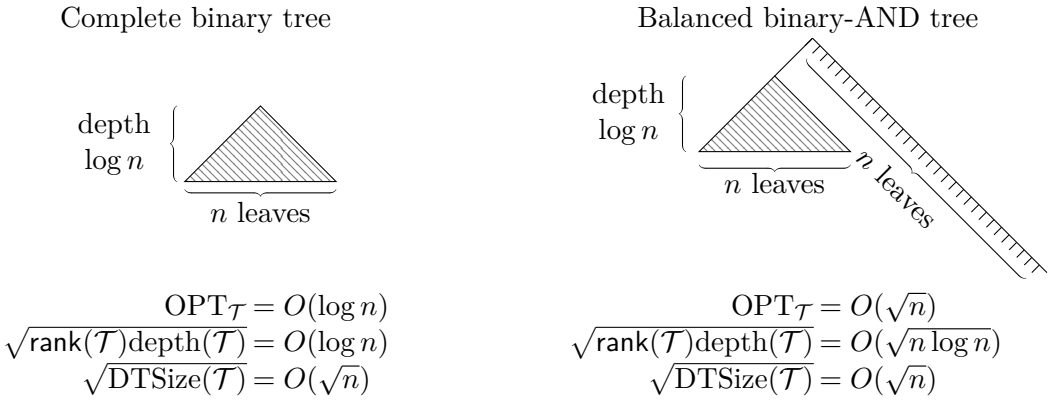
$$\mathrm{OPT}_{\mathcal{T}_L} \leq \sqrt{2\mathrm{DTSize}(\mathcal{T}_L)}, \qquad \text{and} \qquad \mathrm{OPT}_{\mathcal{T}_R} \leq \sqrt{2\mathrm{DTSize}(\mathcal{T}_R)}.$$

Then, by Theorem 5.4, the square of the optimal value of Program 1 for $\mathcal{T}$ equals

$$\begin{aligned}
\mathrm{OPT}_{\mathcal{T}}^2 &= \frac{(\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R})^2 + (\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}{4} \\
&\quad + \frac{2(\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R})\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{4} \\
&= \frac{2(\mathrm{OPT}_{\mathcal{T}_L}^2 + \mathrm{OPT}_{\mathcal{T}_R}^2) + 4}{4} + \frac{2(\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R})\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}{4} \\
&\leq \frac{2(\mathrm{OPT}_{\mathcal{T}_L}^2 + \mathrm{OPT}_{\mathcal{T}_R}^2) + 4 + (\mathrm{OPT}_{\mathcal{T}_L} + \mathrm{OPT}_{\mathcal{T}_R})^2 + (\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}{4} \\
&\qquad\qquad \text{since } 2ab \leq a^2 + b^2 \text{ for all } a, b \geq 0 \\
&= \mathrm{OPT}_{\mathcal{T}_L}^2 + \mathrm{OPT}_{\mathcal{T}_R}^2 + 2 \\
&\leq 2\mathrm{DTSize}(\mathcal{T}_L) + 2\mathrm{DTSize}(\mathcal{T}_R) + 2 = 2\mathrm{DTSize}(\mathcal{T}).
\end{aligned}$$

This completes the proof. □

Next, we note that the rank-depth and size bounds from Corollary 1.7 are incomparable, as witnessed by the examples displayed in Figure 6. In particular, our bounds are strictly stronger than those given by earlier works (Theorem 1.1) for the second tree in the figure.



$$\mathrm{OPT}_{\mathcal{T}} = O(\log n) \qquad \mathrm{OPT}_{\mathcal{T}} = O(\sqrt{n})$$
$$\sqrt{\mathsf{rank}(\mathcal{T})\mathrm{depth}(\mathcal{T})} = O(\log n) \qquad \sqrt{\mathsf{rank}(\mathcal{T})\mathrm{depth}(\mathcal{T})} = O(\sqrt{n \log n})$$
$$\sqrt{\mathrm{DTSize}(\mathcal{T})} = O(\sqrt{n}) \qquad \sqrt{\mathrm{DTSize}(\mathcal{T})} = O(\sqrt{n})$$

Figure 6: Examples showing separations between the two bounds derived in Corollary 1.7. The shaded regions represent complete binary trees. In the left example the rank-depth bound beats the size bound, whereas in the right example the opposite is true.

Finally, we note that we can obtain analogous quantum query upper bounds to those in Corollary 1.7 when the initial tree is a randomized one.

Corollary 5.5. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then,*

$$\mathsf{Q}_{2/5}(f) = O(\sqrt{\mathrm{RDTSize}(f)}).$$

Moreover, let $\mathcal{T}$ be a randomized decision tree computing f with depth T and randomized rank G. Then,

$$\mathsf{Q}_{2/5}(f) = O\left(\sqrt{TG}\right).$$

The proof of Corollary 5.5 follows along similar lines as the proof of Theorem 1.3, but we include it below for completeness.

Proof. Let $L = \mathrm{RDTSize}(f)$, and let $\mathcal{T}$ be a randomized decision tree witnessing this. For a decision tree $\mathcal{T}_\mu$ in the support of $\mathcal{T}$, define $f_\mu : \{0,1\}^n \to \mathcal{R}$ be the function computed by $\mathcal{T}_\mu$. Corollary 1.7 implies $\mathsf{Q}(\tilde{f_\mu}) = O\left(\sqrt{\mathrm{DTSize}(\mathcal{T}_\mu)}\right) = O\left(\sqrt{L}\right)$. By standard error-reduction techniques, $\mathsf{Q}_{9/10}(f_\mu) = O(\sqrt{L})$.

A quantum query algorithm for f is as follows: Sample $\mathcal{T}_\mu$ from the support of $\mathcal{T}$ according to its underlying distribution, and compute f_μ using the above-mentioned algorithm. The cost of this algorithm is $O(\sqrt{L})$ and the success probability is at least $(9/10) \cdot (2/3) = 3/5$ for all inputs $x \in D_f$, which proves the first query upper bound.

A similar argument shows the second query upper bound as well. □

Acknowledgements We thank Ronald de Wolf and the anonymous FSTTCS reviewers for helpful comments.

AC: This work was done while the author was at the Institute for Logic, Language, and Computation, University of Amsterdam and QuSoft. Supported by a Simons-CIQC postdoctoral fellowship through NSF QLCI Grant No. 2016245.

NM: This work was done while the author was at QuSoft and CWI, Amsterdam, and supported by the Dutch Research Council (NWO/OCW), as part of the Quantum Software

Consortium programme (project number 024.003.037), and through QuantERA ERA-NET Cofund project QuantAlgo (project number 680-91-034).

SP: This work was done while the author was at QuSoft and CWI, Amsterdam, and supported by Robert Bosch Stiftung and by NWO Gravitation grants NETWORKS and QSC, and EU grant QuantAlgo.

References

[ABB+17] Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *Journal of the ACM*, 64(5):32:1–32:24, 2017. Earlier version in STOC'16. doi:10.1145/3106234.

[ABRW16] Andris Ambainis, Aleksandrs Belovs, Oded Regev, and Ronald de Wolf. Efficient quantum algorithms for (gapped) group testing and junta testing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 903–922. SIAM, 2016. doi:10.1137/1.9781611974331.ch65.

[Ari15] Agnis Arins. Span-program-based quantum algorithms for graph bipartiteness and connectivity. In *Mathematical and Engineering Methods in Computer Science - 10th International Doctoral Workshop, MEMICS, Selected Papers*, volume 9548 of *Lecture Notes in Computer Science*, pages 35–41. Springer, 2015. doi:10.1007/978-3-319-29817-7_4.

[Bel12] Aleksandrs Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 207–216. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.18.

[Bel19] Aleksandrs Belovs. Quantum dual adversary for hidden subgroups and beyond. In *Proceedings of the 18th International Conference on Unconventional Computation and Natural Computation (UCNC)*, volume 11493 of *Lecture Notes in Computer Science*, pages 30–36. Springer, 2019. doi:10.1007/978-3-030-19311-9_4.

[BR12] Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, volume 7501 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 2012. doi:10.1007/978-3-642-33090-2_18.

[BT19] Salman Beigi and Leila Taghavi. Span program for non-binary functions. *Quantum Information and Computation*, 19(9&10):760–792, 2019. doi:10.26421/QIC19.9-10-2.

[BT20] Salman Beigi and Leila Taghavi. Quantum speedup based on classical decision trees. *Quantum*, 4:241, 2020. doi:10.22331/q-2020-03-02-241.

[BTT22] Salman Beigi, Leila Taghavi, and Artin Tajdini. Time-and query-optimal quantum algorithms based on decision trees. *ACM Transactions on Quantum Computing*, 3(4):1–31, 2022. doi:10.1145/3519269.

[BW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

[CDM+23] Arkadev Chattopadhyay, Yogesh Dahiya, Nikhil S. Mande, Jaikumar Radhakrishnan, and Swagato Sanyal. Randomized versus deterministic decision tree size. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 867–880. ACM, 2023. doi:10.1145/3564246.3585199.

[CMB18] Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Information and Computation*, 18(1&2):18–50, 2018. doi:10.26421/QIC18.1-2-2.

[CMP22] Arjan Cornelissen, Nikhil S. Mande, and Subhasree Patro. Improved quantum query upper bounds based on classical decision trees. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIcs*, pages 15:1–15:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSTTCS.2022.15.

[Cor25] Arjan Cornelissen. Quantum algorithms through graph composition, 2025. doi:10.48550/arXiv.2504.02115.

[DH96] Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996. URL: https://doi.org/10.48550/arXiv.quant-ph/9607014.

[DM21] Yogesh Dahiya and Meena Mahajan. On (simple) decision tree rank. In *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 213 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.15.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989. doi:10.1016/0890-5401(89)90001-1.

[GPW18] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM Journal on Computing*, 47(6):2435–2450, 2018. Earlier version in FOCS'15. doi:10.1137/16M1059369.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219. ACM, 1996. doi:10.1145/237814.237866.

[HLŠ07] Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 526–535. ACM, 2007. doi:10.1145/1250790.1250867.

[JJKP18] Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithms for connectivity and related problems. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.49.

[JK17] Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017. doi:10.22331/q-2017-08-17-26.

[Kot14] Robin Kothari. An optimal quantum algorithm for the oracle identification problem. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 25 of *LIPIcs*, pages 482–493. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.482.

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 102–111. IEEE Computer Society, 1993. doi:10.1109/SCT.1993.336536.

[LL16] Cedric Yen-Yu Lin and Han-Hsuan Lin. Upper bounds on quantum query complexity inspired by the Elitzur–Vaidman bomb tester. *Theory of Computing*, 12(1):1–35, 2016. doi:10.4086/toc.2016.v012a018.

[LMR+11] Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 344–353. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.75.

[MRS18] Sagnik Mukhopadhyay, Jaikumar Radhakrishnan, and Swagato Sanyal. Separation between deterministic and randomized query complexity. *SIAM Journal on Computing*, 47(4):1644–1666, 2018. Earlier versions in FSTTCS'15 and FSTTCS'16. doi:10.1137/17M1124115.

[NC16] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. doi:10.1017/CBO9780511976667.

[Nis91] Noam Nisan. CREW prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. Earlier version in STOC'89. doi:10.1137/0220062.

[PI00] 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. ACM/SIAM, 2000. URL: https://dl.acm.org/doi/10.5555/338219.338244.

[Rei09] Ben Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 544–551. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.55.

[Rei11] Ben Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–569. SIAM, 2011. doi:10.1137/1.9781611973082.44.

[RŠ12] Ben Reichardt and Robert Špalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8(1):291–319, 2012. Earlier version in STOC'08. doi:10.4086/toc.2012.v008a013.

[SW86] Michael E. Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 29–38. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.44.

[Tag22] Leila Taghavi. Simplified quantum algorithm for the oracle identification problem. *Quantum Mach. Intell.*, 4(2):1–7, 2022. doi:10.1007/s42484-022-00080-2.

[Wol19] Ronald de Wolf. Quantum computing: Lecture notes. *CoRR*, abs/1907.09415, 2019. arXiv:1907.09415, doi:10.48550/arXiv.1907.09415.

A Decision-Tree Size and Formula Size

We observe in this section that the formula size of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is at most a constant times the decision-tree size of f. While the proof is simple, we are unaware of such a statement appearing explicitly in the literature.

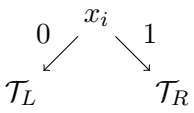
Definition A.1 (Boolean Formulas)**.** *A Boolean formula is a rooted binary tree, where internal nodes have in-degree 2 and are labeled with* AND *or* OR*, and leaf nodes are labeled by variables or their negations (x_i or $\overline{x_i}$). A Boolean formula computes a Boolean function in the natural way.*

The size of a Boolean formula is the number of nodes in it. Let $L(f)$ denote the minimum size of a Boolean formula computing f.

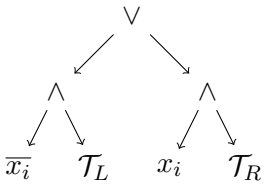
Claim A.2. *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then,*

$$L(f) \leq 5\mathrm{DTSize}(f).$$

Proof. Let $\mathcal{T}$ be a decision tree computing f. If $\mathcal{T}$ is a single leaf, say b, then the size-1 formula b computes f. Suppose $\mathcal{T}$ is of the form

$\mathcal{T} =$ 

The following formula can then easily be seen to compute f.



Recursively constructing trees for $\mathcal{T}_L$ and $\mathcal{T}_R$, we obtain the recurrence

$$L(f) \leq 5 + \mathrm{DTSize}(\mathcal{T}_L) + \mathrm{DTSize}(\mathcal{T}_R).$$

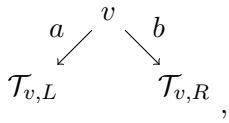
Using induction on the size of $\mathcal{T}$, solving this recurrence yields the claim. □

B Another Weight Assignment

In this section we give a different weight assignment to the weights in Program 1 to obtain an objective value of $O(\sqrt{\mathrm{DTSize}(\mathcal{T}) \log \mathrm{DTSize}(\mathcal{T})})$ for a deterministic decision tree $\mathcal{T}$. While this bound is weaker than the second bound in Corollary 1.7 by a logarithmic factor,

we choose to include it since we think the weight assignment may be of independent interest. Moreover, these weights are much easier to compute than those in Section 5 since the weights here have a neat closed-form expression, as opposed to the iterative construction in Section 5.

We define the weight assignment $\{W'_e : e \text{ is an edge in } \mathcal{T}\}$ as follows. Consider any vertex v of $\mathcal{T}$, and let $\mathcal{T}_v$, $\mathcal{T}_{v,L}$ and $\mathcal{T}_{v,R}$ denote the subtrees of $\mathcal{T}$ rooted at v, the left child of v and the right child of v, respectively. For the two outgoing edges of v, define the corresponding weights as in the following figure:



where

$$a = \frac{1}{\log \frac{\mathrm{DTSize}(\mathcal{T}_v)}{\mathrm{DTSize}(\mathcal{T}_{v,L})}}, \qquad b = \frac{1}{\log \frac{\mathrm{DTSize}(\mathcal{T}_v)}{\mathrm{DTSize}(\mathcal{T}_{v,R})}}. \tag{23}$$

For a leaf ℓ of $\mathcal{T}$, we have

$$\sum_{v \in P_\ell} \frac{1}{W'_{e(v,N(v))}} = \sum_{v \in P_\ell} \log \frac{\mathrm{DTSize}(\mathcal{T}_v)}{\mathrm{DTSize}(\mathcal{T}_{N(v)})} = \log \prod_{v \in P_\ell} \frac{\mathrm{DTSize}(\mathcal{T}_v)}{\mathrm{DTSize}(\mathcal{T}_{N(v)})} \tag{24}$$

where the summation excludes the leaf ℓ and $N(v)$ denotes the child of v that is on the path P_ℓ. The product is telescopic and equals $\log \mathrm{DTSize}(\mathcal{T})$. With the same notation and using $\overline{N(v)}$ to denote the child of v that is not on the path P_ℓ, we also have

$$\begin{aligned}
\sum_{e \in \overline{P_\ell}} W'_e &= \sum_{v \in P_\ell} \frac{1}{\log \frac{\mathrm{DTSize}(\mathcal{T}_v)}{\mathrm{DTSize}(\mathcal{T}_{\overline{N(v)}})}} = \sum_{v \in P_\ell} \frac{1}{\log \frac{1+\mathrm{DTSize}(\mathcal{T}_{N(v)})+\mathrm{DTSize}(\mathcal{T}_{\overline{N(v)}})}{\mathrm{DTSize}(\mathcal{T}_{\overline{N(v)}})}} \qquad (25)\\
&\le \sum_{v \in P_\ell} \frac{1}{\log\left(1 + \frac{\mathrm{DTSize}(\mathcal{T}_{N(v)})}{\mathrm{DTSize}(\mathcal{T}_{\overline{N(v)}})}\right)}\\
&\le \sum_{v \in P_\ell} \frac{1 + \frac{\mathrm{DTSize}(\mathcal{T}_{N(v)})}{\mathrm{DTSize}(\mathcal{T}_{\overline{N(v)}})}}{\frac{\mathrm{DTSize}(\mathcal{T}_{N(v)})}{\mathrm{DTSize}(\mathcal{T}_{\overline{N(v)}})}} \qquad \text{since } \tfrac{1}{\log(1+x)} \le \tfrac{1+x}{x} \text{ for all } x > 0\\
&\le \sum_{v \in P_\ell} \frac{\mathrm{DTSize}(\mathcal{T}_v)}{\mathrm{DTSize}(\mathcal{T}_{N(v)})}\\
&= \sum_{v \in P_\ell} 1 + \frac{\mathrm{DTSize}(\mathcal{T}_v) - \mathrm{DTSize}(\mathcal{T}_{N(v)})}{\mathrm{DTSize}(\mathcal{T}_{N(v)})}\\
&\le \sum_{v \in P_\ell} 1 + \mathrm{DTSize}(\mathcal{T}_v) - \mathrm{DTSize}(\mathcal{T}_{N(v)})\\
&= |P_\ell| + \mathrm{DTSize}(\mathcal{T}) - 1\\
&\le 2\mathrm{DTSize}(\mathcal{T}).
\end{aligned}$$

Thus, this setting of weights, along with the settings

$$\alpha = 2\mathrm{DTSize}(\mathcal{T}), \qquad \beta = \log \mathrm{DTSize}(\mathcal{T}),$$

gives an objective value of $O(\sqrt{\mathrm{DTSize}(\mathcal{T}) \log \mathrm{DTSize}(\mathcal{T})})$ Program 1 for $\mathcal{T}$.