Improved Quantum Query Upper Bounds Based on Classical Decision Trees

Arjan Cornelissen
Institute for Logic, Language, and Computation, University of Amsterdam, The Netherlands
QuSoft, Amsterdam, The Netherlands

Nikhil S. Mande
QuSoft, Amsterdam, The Netherlands
CWI, Amsterdam, The Netherlands

Subhasree Patro
QuSoft, Amsterdam, The Netherlands
CWI Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Abstract

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 strictly 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.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Query Complexity, Decision Trees, Decision Tree Rank

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.15

Related Version *Full Version*: `https://arxiv.org/abs/2203.02968` [12]

Funding *Nikhil S. Mande*: 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, ended in December 2021).
Subhasree Patro: Mainly supported by the Robert Bosch Stiftung, also additionally supported by NWO Gravitation grants NETWORKS and QSC, and through QuantERA ERA-NET Cofund project QuantAlgo (project number 680-91-034, ended in December 2021).

Acknowledgements We thank Ronald de Wolf for useful comments.

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 $\Theta(\sqrt{n})$) [16, 21, 23]. 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) $2n+1$ 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 9). We obtain our quantum query upper bounds by constructing explicit *span programs* and *dual adversary solutions* by exploiting 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 [20]. 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 [30], and it is known that span programs characterize bounded-error quantum query complexity of Boolean functions up to a constant factor [29, 22]. Span programs have been used to design quantum algorithms for various graph problems such as st-connectivity [9], cycle detection and bipartiteness testing [3, 11], graph connectivity [18], and has been also used for problems such as formula evaluation [30, 28, 19]. Recently, Beigi and Taghavi [4] 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 [5], 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 was developed by Høyer, Lee and Špalek [17], 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 [29, 22]. 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. Quantum query algorithms have been developed by explicitly constructing dual adversary solutions, for example for the well-studied k-distinctness problem [7], S-isomorphism and hidden subgroup problems [8], gapped group testing [2], etc.

1.3 Related Works

Lin and Lin [23] showed how a classical algorithm computing a function $f : D_f \to \mathcal{R}$ with $D_f \subseteq \{0,1\}^n$ and $\mathcal{R}$ as an arbitrarily large finite set, 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 $f \subseteq \{0,1\} \times \mathcal{R}$ 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]

▶ **Theorem 1** (Lin and Lin [23, 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 $x_{p_1}, \dots, x_{p_{\tilde{T}(x)}}$ be the query results of $\mathcal{A}$ on an apriori 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 $b_1, \dots, b_{t-1} \in \{0,1\}^{t-1}$ 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,*

$$\forall x \in \{0,1\}^n, \quad \sum_{t=1}^{\tilde{T}(x)} \left| \mathcal{G}(x_1, \dots, x_{p_{t-1}}) - x_{p_t} \right| \leq G.$$

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 [5] gave an alternate proof of Theorem 1 using the framework of *non-binary span programs* introduced by them in [4]. 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), which is $O(\sqrt{TG})$, where T denotes the depth and G the *guessing complexity* (see Definition 11) of the underlying decision tree, respectively.

[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.

[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 [21].

In a follow-up work [6], conditional on some constraints, Beigi, Taghavi and Tajdini implement the span-program-based algorithm of [5] in a time-efficient manner. In another work [32], Taghavi uses non-binary span programs to give a tight quantum query algorithm for the oracle identification problem, simplifying an earlier algorithm due to Kothari [21].

1.4 Our Contributions

The *rank* of a decision tree is a combinatorial measure introduced by Ehrenfeucht and Haussler [14] in the context of learning theory, formally defined as follows.

▶ **Definition 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 3** (Rank of a Boolean function). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. 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* $1/3$.

We observe here that the guessing complexity of a deterministic decision tree equals its rank (see Claim 12). This answers a question of Lin and Lin [23, 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 exponentially 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 example, [13] 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 has a clean equivalent formulation as follows.

▶ **Theorem 4.** *Let $\mathcal{T}$ be a decision tree computing a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ with depth T. Then* $\mathsf{Q}(f) = O(\sqrt{\mathsf{rank}(\mathcal{T})T})$.

Randomized rank, which we denote by rrank, is a natural probabilistic analog of rank. This exactly captures the notion of the randomized analog of the value of G in Theorem 1. It is easy to show with this definition that our proof of Theorem 4 also holds with respect to randomized decision trees and randomized rank: sample a decision tree from the support of $\mathcal{T}$ according to its underlying distribution, and run a $(1/10)$-error[3] quantum query algorithm of cost $O(\sqrt{\mathsf{rank}(\mathcal{T})T})$ from Theorem 4 on the resultant tree. The success probability is at least $(9/10) \cdot (2/3) = 3/5$. Thus we obtain the following easy-to-state reformulation of [23, Theorem 5.4].

[3] The success probability of the algorithm in Theorem 4 can be boosted by repeating the algorithm a large constant number of times and then outputting the majority of the outcomes.

▶ **Theorem 5.** *Let $\mathcal{T}$ be a randomized decision tree computing a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ with depth T. Then $\mathsf{Q}_{2/5}(f) = O(\sqrt{\mathsf{rrank}(\mathcal{T})T})$.*

Thus, up to a small loss in the success probability, upper bounds obtained from Theorem 5 are strictly stronger than those obtained from Theorem 4 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. 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 6.** *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 [27], 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 [31].

We conjecture 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 [26] (also see [10]) 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.

Beigi and Taghavi [5, 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. We answer this question in the affirmative by providing a weighting scheme that improves upon their bounds. By a careful analysis, we argue that the 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.

▶ **Definition 7** (Weight optimization program)**.** *For a decision tree $\mathcal{T}$, define its* weight optimization program *by the minimization problem with constraints outlined in Program 1 (see Appendix A.1 for relevant definitions). Let $\mathrm{OPT}_{\mathcal{T}}$ denote the optimum of this program.*

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 3).

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	$> 0.$	

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. The following is our second main theorem.

▶ **Theorem 8.** *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation and let $\mathcal{T}$ be a decision tree computing f. Then $\mathsf{Q}(\tilde{f}) = O(\mathrm{OPT}_{\mathcal{T}})$.*

We give a recursively-defined weighting scheme that achieves equality for all the constraints in Program 1 and argue from first principles that our solution is optimal (see Theorem 26), thus subsuming Theorem 1. Combined with the earlier results from Beigi and Taghavi and using our recursive expression for $\mathrm{OPT}_{\mathcal{T}}$, this gives us the following corollary, giving a new way to bound $\mathrm{OPT}_{\mathcal{T}}$ and thus $\mathsf{Q}(\tilde{f})$ from above.

▶ **Corollary 9.** *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 23. 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: $\mathsf{Q}(\tilde{f}) = O\left(\sqrt{\mathrm{DTSize}(\mathcal{T})}\right)$ where $\mathrm{DTSize}(\mathcal{T})$ denotes the number of nodes in $\mathcal{T}$.*

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 27).

It was shown by Reichardt [29] 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 9 for Boolean functions f since the formula size of a Boolean function is bounded above by a constant times its decision-tree size (see the full version of this paper [12, 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 [16], for example. We also exhibit a family of decision trees for which the size bound is strictly stronger than the rank-depth bound of Lin and Lin, and Beigi and Taghavi (see Figure 3).

1.5 Organization

In Section 2, we discuss the rank of decision trees, and prove that it is equal to guessing complexity. Furthermore, we prove a separation between rank and randomized rank for the complete binary AND-OR tree. In Appendix B we discuss how to construct 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 capture the weighting scheme in an optimization program. In Section 3 we prove Theorem 8. Finally, in Section 4, we exhibit a solution to Program 1 and argue its optimality from first principles. We refer the reader to the appendices for preliminaries and missing proofs.

2 Decision Tree Rank

In this section, we first rephrase Theorem 1 in terms of a measure of decision trees which we term “guessing complexity”. This reformulation was essentially done by Beigi and Taghavi [5, Section 3]. We then show that the guessing complexity of a decision tree equals its rank, proving Theorem 4. Finally, we show a polynomial separation between rank and randomized rank for the complete binary AND-OR tree.

2.1 Guessing Complexity and Rank

▶ **Definition 10** (G-coloring [5, 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 11** (Guessing Complexity). *Let $\mathcal{T}$ be a decision tree and 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 $G(\mathcal{T}) = \min_{\text{G-colorings of } \mathcal{T}} \max_{P \in P(\mathcal{T})}$* number of red edges on *P.*

▷ Claim 12. 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), proving the claim. ◁

The guessing algorithm $\mathcal{G}$ in Theorem 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 4 immediately follows from Claim 12 and Theorem 1.

Proof of Theorem 5. An 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 from Theorem 4 on the resultant tree.[4] The cost of this algorithm is $O(\sqrt{\mathsf{rrank}(\mathcal{T})T})$ and the success probability is at least $(9/10) \cdot (2/3) = 3/5$ for all inputs $x \in D_f$. ◀

[4] 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 4 a large constant many times and return the majority output).

The rank of a decision tree essentially captures the largest depth of a 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 13** ([14, 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 5 yields the following statement.

▶ **Theorem 14.** *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 the full version of this paper [12, Appendix B], we show an explicit NBSPwOI and dual adversary solution witnessing the same bound. In Sections 3 and 4 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 [12, Appendix B] as the weighting scheme seems considerably different from that in Section 4, 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 [27]. 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 [27] (also see [13, Theorem 3.1] for an explicit statement and proof).

▷ Claim 15. Let $f : \{0,1\}^n \to \{0,1\}$ be a (possibly partial) Boolean function. Then $\mathsf{rank}(f) = \mathsf{val}(f)$.

2.2 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, i.e., functions defined only on a subset of all possible inputs. This is witnessed by the well-studied Approximate-Majority function, for example.

▷ Claim 16. Let $f : \{0,1\}^n \to \{0,1\}$ 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 $\mathsf{rrank}(f) = \Theta(1)$.

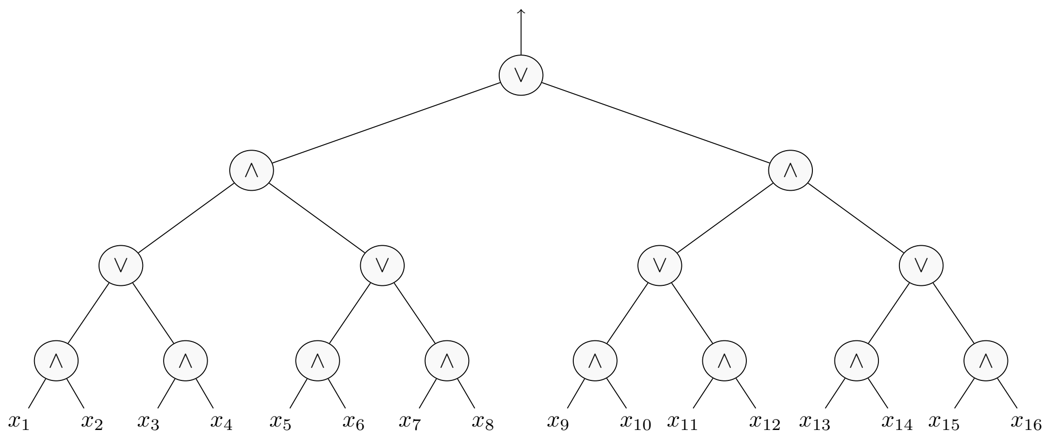


Figure 1 Complete AND-OR tree of depth 4.

Proof. Clearly $\mathsf{rank}(f) = O(n)$. For the lower bound, we use the equivalence from Claim 15. 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 [15, 1, 24]. One might hope to use a similar argument as in the proof of Claim 16 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 easy 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.

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 [31] showed that $\mathsf{R}(F) = \Theta(n^{\log \frac{1+\sqrt{33}}{4}}) \approx \Theta(n^{0.753\ldots})$.

▶ **Theorem 17** ([31, 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 18.** *Let $F : \{0,1\}^n \to \{0,1\}$ be the complete AND-OR tree of depth $\log n$. Then*

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

For a proof we refer the reader to the full version [12, Proof of Theorem 3.9]. Since randomized rank is at most randomized decision tree complexity, F witnesses a separation between rank and randomized rank. This proves Theorem 6.

3 Proof of Theorem 8

In order to give a proof of Theorem 8, we require some useful properties of the non-binary span program with orthogonal inputs (NBSPwOI) and dual adversary solution constructed by Beigi and Taghavi [5]. For details of these constructions, we refer the reader to Appendix B.

The main result of Beigi and Taghavi [4] that we need is stated in the theorem below.

▶ **Theorem 19** ([4]). *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,*

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

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, in terms of the objective value of Program 1.

▶ **Theorem 20.** *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 $(P, w, \bar{w})$ with variable weights as in Appendix B,*

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

Similarly, it is known that solutions to the dual adversary program (Program 2) yield quantum query upper bounds.

▶ **Theorem 21** ([22]). *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*

$$Q(f) = O(C). \tag{1}$$

▶ **Theorem 22.** *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 2 with variable weights as in Appendix B, and let $\mathrm{OPT}_{\mathcal{T}}$ denote the optimal value of Program 1. Then $C = \mathrm{OPT}_{\mathcal{T}}$.*

We now prove Theorem 8, using these results.

Proof of Theorem 8. We give two proofs, one via span program witness complexity, and another via the dual adversary bound.

1. Consider the NBSPwOI $(P, w, \bar{w})$ for $\tilde{f}$ as constructed in Sections B.1 and B.2. Theorem 20 implies $\mathrm{wsize}(P, w, \bar{w}) \le \mathrm{OPT}_{\mathcal{T}}$. Theorem 19 implies $\mathsf{Q}(\tilde{f}) = O(\mathrm{OPT}_{\mathcal{T}})$.
2. Consider the dual adversary solution for $\tilde{f}$ as constructed in Appendix B.3. Theorems 21 and 22 imply $\mathsf{Q}(\tilde{f}) = O(C) = O(\mathrm{OPT}_{\mathcal{T}})$. ◀

4 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 decision tree with two colors, and then assigning weights depending on the color that the edge is colored by [5, Section 3]. They raise 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 7, 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 the full version of this paper [12, Appendix B].

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. More precisely, in the i'th iteration, we assign weights to all edges that have maximum distance i from a leaf.

First, we define the weight assignment, which we will refer to as the *canonical weight assignment*, in Definition 23. Then, we prove its optimality, resulting in Theorem 26. Finally, we prove Corollary 9, which gives some upper bounds on the optimal objective value, in terms of natural measures of the decision tree.

▶ **Definition 23** (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 24.** *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 $W_e = \sqrt{\beta_L/\alpha_L}(W_L)_e$ for all edges e in $\mathcal{T}_L$ and $W_{e'} = \sqrt{\beta_R/\alpha_R}(W_R)_{e'}$ for all edges e' in $\mathcal{T}_R$, and choose W_{e_L} and W_{e_R} as in Definition 23. Furthermore, let

$$\alpha := \max\left\{\mathrm{OPT}_{\mathcal{T}_L} + W_{e_R}, \mathrm{OPT}_{\mathcal{T}_R} + W_{e_L}\right\}, \qquad \beta := \max\left\{\mathrm{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}}, \mathrm{OPT}_{\mathcal{T}_R} + \frac{1}{W_{e_R}}\right\}. \tag{2}$$

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

$$\sum_{e\in P} \frac{1}{W_e} = \frac{1}{W_{e_L}} + \sqrt{\frac{\alpha_L}{\beta_L}} \sum_{e\in P\setminus\{e_L\}} \frac{1}{(W_L)_e} \leq \sqrt{\alpha_L\beta_L} + \frac{1}{W_{e_L}} = \mathrm{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}} \leq \beta,$$

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

$$\sum_{e\in \overline{P}} W_e = \sqrt{\frac{\beta_L}{\alpha_L}} \sum_{e\in \overline{P}\setminus\{e_R\}} (W_L)_e + W_{e_R} \leq \sqrt{\alpha_L\beta_L} + W_{e_R} = \mathrm{OPT}_{\mathcal{T}_L} + W_{e_R} \leq \alpha,$$

Thus all constraints of Program 1 for $\mathcal{T}$ are satisfied with these values of α, β and the weighting scheme W. Hence,

$$\mathrm{OPT}_{\mathcal{T}} \leq \sqrt{\alpha\beta} = \sqrt{\max\left\{\mathrm{OPT}_{\mathcal{T}_L} + \frac{1}{W_{e_L}}, \mathrm{OPT}_{\mathcal{T}_R} + \frac{1}{W_{e_R}}\right\} \cdot \max\left\{\mathrm{OPT}_{\mathcal{T}_L} + W_{e_R}, \mathrm{OPT}_{\mathcal{T}_R} + W_{e_L}\right\}}. \quad (3)$$

Finally, observe from our choices of W_{e_L} and W_{e_R} from Definition 23 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 (3) implies

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

where the last equality follows from our choices of W_{e_L} and W_{e_R} from Definition 23. This completes the proof. ◀

We next show that this weight assignment is optimal.

▶ **Lemma 25.** *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 optimization program on $\mathcal{T}$, 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

$$\sqrt{\alpha_L\beta_L} \geq \mathrm{OPT}_{\mathcal{T}_L}, \qquad \sqrt{\alpha_R\beta_R} \geq \mathrm{OPT}_{\mathcal{T}_R}. \quad (4)$$

Furthermore, we have

$$\begin{aligned}\alpha &= \max_{P\in P(\mathcal{T})}\sum_{e\in\overline{P}} W_e = \max\left\{\max_{P\in P(\mathcal{T}_L)}\sum_{e\in\overline{P}} W_e + W_{e_R}, \max_{P\in P(\mathcal{T}_R)}\sum_{e\in\overline{P}} W_e + W_{e_L}\right\} \\ &= \max\left\{\alpha_L + W_{e_R}, \alpha_R + W_{e_L}\right\}, \qquad (5)\end{aligned}$$

and similarly

$$\begin{aligned}\beta &= \max_{P\in P(\mathcal{T})}\sum_{e\in P}\frac{1}{W_e} = \max\left\{\max_{P\in P(\mathcal{T}_L)}\sum_{e\in P}\frac{1}{W_e} + \frac{1}{W_{e_L}}, \max_{P\in P(\mathcal{T}_R)}\sum_{e\in P}\frac{1}{W_e} + \frac{1}{W_{e_R}}\right\} \\ &= \max\left\{\beta_L + \frac{1}{W_{e_L}}, \beta_R + \frac{1}{W_{e_R}}\right\}. \qquad (6)\end{aligned}$$

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

$$\mathrm{OPT}_{\mathcal{T}} = \sqrt{\alpha\beta} = \frac{\beta}{\gamma} = \gamma\alpha = \frac{1}{2}\left[\frac{\beta}{\gamma} + \gamma\alpha\right]. \qquad (7)$$

Now, let

$$\delta = \frac{1}{2} + \frac{\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R}}{2\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}}, \qquad (8)$$

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 (7)} \\ &= \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 (5) and (6)} \\ &\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) + \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) + \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 (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}{2\sqrt{(\mathrm{OPT}_{\mathcal{T}_L} - \mathrm{OPT}_{\mathcal{T}_R})^2 + 4}} + 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 (8)} \\ &= \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 26.** *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 $\mathrm{OPT}_{\mathcal{T}}$ follows from Lemma 24 and the lower bound follows from Lemma 25. ◀

Observe that Lemma 24 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 2. 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.

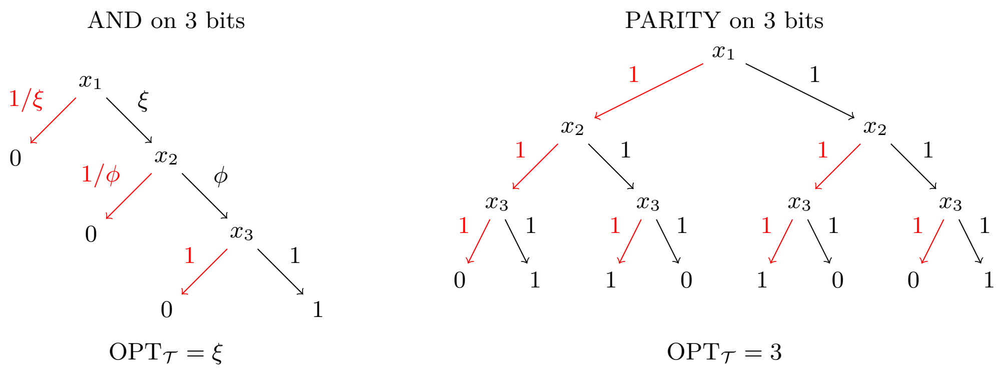


Figure 2 Examples of optimal weight assignments for two different decision trees. The red and black edges indicate the edges taken when the output of the query is 0 and 1, respectively, and the edge labels represent the weights. Left: Canonical weight assignment of the decision tree for the AND function 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 the PARITY on 3 bits, with optimal value 3.

Next, there are several 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. Corollary 9 exhibits two such bounds.

Proof of Corollary 9. Theorem 8 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 [5, Theorem 2], and the equality $G(\mathcal{T}) = \mathsf{rank}(\mathcal{T})$ from Claim 12.

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,

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

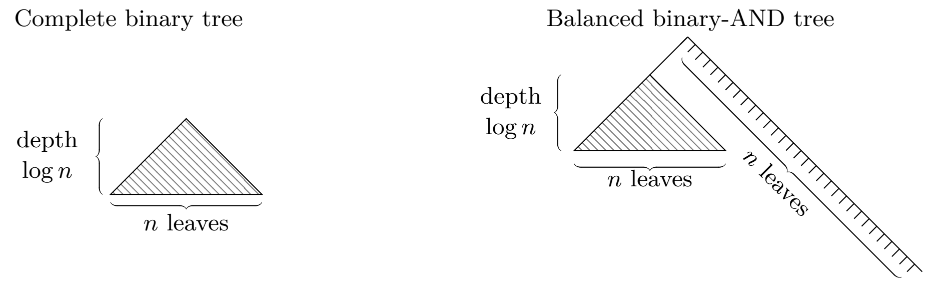
Then, by Theorem 26, the square of the optimal value of Program 1 for $\mathcal{T}$ equals

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

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

This completes the proof. ◀

Next, we note that the rank-depth and size bounds from Corollary 9 are incomparable, as witnessed by the examples displayed in Figure 3. In particular, our bounds are strictly stronger than those given by earlier works (Theorem 1) for the second tree in the figure.



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

Figure 3 Examples showing separations between the two bounds derived in Corollary 9. 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 9 when the initial tree is a randomized one.

▶ **Corollary 27.** *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. Then $\mathsf{Q}_{2/5}(f) = O\left(\sqrt{\mathsf{rrank}(\mathcal{T})T}\right)$.*

The proof of Corollary 27 follows along similar lines as the proof of Theorem 5 and we omit it.

References

1 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`.

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

3 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.

4 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.

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

6 Salman Beigi, Leila Taghavi, and Artin Tajdini. Time and query optimal quantum algorithms based on decision trees. *CoRR*, abs/2105.08309, 2021. arXiv:2105.08309.

7 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.

8 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.

9 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.

10 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.

11 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.

12 Arjan Cornelissen, Nikhil S. Mande, and Subhasree Patro. Improved quantum query upper bounds based on classical decision trees, 2022. doi:10.48550/ARXIV.2203.02968.

13 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.

14 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.

15 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.

16 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.

17 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.

18 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.

19 Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017.

20 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`.

21 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`.

22 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`.

23 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`.

24 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`.

25 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. URL: `https://www.cambridge.org/de/academic/subjects/physics/quantum-physics-quantum-information-and-quantum-computation/quantum-computation-and-quantum-information-10th-anniversary-edition?format=HB`.

26 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`.

27 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: `http://dl.acm.org/citation.cfm?id=338219.338244`.

28 Ben Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 544–551, 2009.

29 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`.

30 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`.

31 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`.

32 Leila Taghavi. Simplified quantum algorithm for the oracle identification problem. *CoRR*, abs/2109.03902, 2021. `arXiv:2109.03902`.

33 Ronald de Wolf. Quantum computing: Lecture notes. *CoRR*, abs/1907.09415, 2019. `arXiv:1907.09415`.

A Preliminaries

All logarithms in this paper are taken base 2. For a bit $b \in \{0,1\}$, let $\neg b$ denote the bit $1-b$. Throughout this paper, $\mathcal{R}$ denotes an arbitrarily large but finite set. 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.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 $x \in \{0,1\}^n$, 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 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 $x \in D_f$, 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

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

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

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

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

▶ **Definition 28** (Decision-tree size complexity). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Define the* decision-tree size complexity *of f, which we denote by* $\mathrm{DTSize}(f)$*, as*

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

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

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

where $\mathrm{RDTSize}(\mathcal{T})$ *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.

A.2 Quantum Query Complexity

We refer the reader to [25, 33] 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 $|0\rangle$ to $|0\rangle$. 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 $\mathsf{Q}(f)$.

B Construction of Span Programs and Dual Adversary Solution of [5]

In this section, we describe 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 [5, Section 3] (recall from Appendix A that $\tilde{f}$ takes an input $x \in \{0,1\}^n$ and outputs the leaf of $\mathcal{T}$ reached on input x). 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. We exhibit another interesting choice of weights in the full version of this paper [12, Appendix B].

B.1 Span Program Construction

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 [4]. We refer the reader to the full version of this paper [12, Section 2.3] for basics.

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, presented in Appendix B.2.

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

- The vector space is determined by the orthonormal basis indexed by vertices of $\mathcal{T}$: $\{|v\rangle : v \in V(\mathcal{T})\}$.
- The input vectors are

$$I_{j,q} = \bigcup_{v \in V_I(\mathcal{T}) : J(v) = j} \left\{ \sqrt{W_{e(v,N(v,q))}} \, (|v\rangle - |N(v,q)\rangle) \right\}. \tag{9}$$

  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.

B.2 Witness Complexity Analysis

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

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

On the other hand, we let $|\bar{w}_x\rangle = \sum_{v \in P_x} |v\rangle$. For any vector in $|v'\rangle \in I(x)$, we have

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

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

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

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

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

and

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

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

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

Thus, if we assign of weights W to the edges of $\mathcal{T}$ as in Sections B.1 and B.2, 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 Theorem 20.

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

B.3 Dual Adversary Solution

We refer the reader to the full version of this paper [12, Section 2.4] for basics. Here in this section we give a simplified 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, which we recall below.

Program 2 Dual SDP for $\tilde{f}$. By replacing the $\delta_{\tilde{f}(x),\tilde{f}(y)}$ term with $\delta_{f(x),f(y)}$ in the constraints we instead get the dual SDP for f.

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

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.

$$|u_{xj}\rangle = \begin{cases} \frac{1}{\sqrt{W_{e(v,N(v,x_j))}}}|v\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}), \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

and,

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

We claim that these vectors form a feasible solution to Program 2. Fix $x, y \in D_f$ with $\tilde{f}(x) \neq \tilde{f}(y)$. We now verify that the corresponding equality constraint in Program 2 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 (10) and (11).
- 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 (10) and (11), 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 2 for $\tilde{f}$.

Proof of Theorem 22. Let C denote the objective value of Program 2 with the settings of $\{|u_{xj}\rangle : x \in D_f, j \in [n]\}$ and $\{|w_{xj}\rangle : x \in D_f, j \in [n]\}$ as defined in Equations (10) and (11), 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,

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

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

Program 3 Optimization program with C being its optimal solution.

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	$> 0.$	

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 3, 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

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

It is easy to verify that this setting of variables is feasible for Program 3, and attains objective value $\max\{\alpha', \beta'\} = \sqrt{\alpha\beta} = \mathrm{OPT}_{\mathcal{T}}$. Thus, $C \leq \mathrm{OPT}_{\mathcal{T}}$, proving the claim. ◀