Electronic Colloquium on Computational Complexity, Report No. 98 (2015)

Separations in Query Complexity Based on Pointer Functions

Andris Ambainis[1] Kaspars Balodis[1] Aleksandrs Belovs[1] Troy Lee[2]
Miklos Santha[3] Juris Smotrovs[1]

Abstract

In 1986, Saks and Wigderson conjectured that the largest separation between deterministic and zero-error randomized query complexity for a total boolean function is given by the function f on $n = 2^k$ bits defined by a complete binary tree of NAND gates of depth k, which achieves $R_0(f) = O(D(f)^{0.7537\cdots})$. We show this is false by giving an example of a total boolean function f on n bits whose deterministic query complexity is $\Omega(n/\log(n))$ while its zero-error randomized query complexity is $\widetilde{O}(\sqrt{n})$. This shows that the relations $D(f) \leq R_0(f)^2$ and $D(f) \leq 2R_1(f)^2$ are optimal, up to poly-logarithmic factors. We further show that the quantum query complexity of the same function is $\widetilde{O}(n^{1/4})$, giving the first example of a total function with a super-quadratic gap between its quantum and deterministic query complexities.

Variations of this function give new separations between several other query complexity measures, including: the first super-linear separation between bounded-error and zero-error randomized complexity, larger gaps between exact quantum query complexity and deterministic/randomized query complexities, and a 4th power separation between approximate degree and bounded-error randomized complexity.

All of these examples are variants of a function recently introduced by Göös, Pitassi, and Watson which they used to separate the unambiguous 1-certificate complexity from deterministic query complexity and to resolve the famous Clique versus Independent Set problem in communication complexity.

[1]Faculty of Computing, University of Latvia, (andris.ambainis@lu.lv, kbalodis@gmail.com, stiboh@gmail.com, juris.smotrovs@lu.lv).

[2]School of Physical and Mathematical Sciences, Nanyang Technological University and Centre for Quantum Technologies, Singapore (troyjlee@gmail.com).

[3]LIAFA, Univ. Paris 7, CNRS, 75205 Paris, France; and Centre for Quantum Technologies, National University of Singapore, Singapore 117543 (miklos.santha@liafa.univ-paris-diderot.fr).

ISSN 1433-8092

1 Introduction

Query complexity has been very useful for understanding the power of different computational models. In the standard version of the query model, we want to compute a boolean function $f\colon \{0,1\}^n \to \{0,1\}$ on an initially unknown input $x \in \{0,1\}^n$ that can only be accessed by asking queries of the form $x_i =?$. The advantage of query complexity is that we can often prove tight lower bounds and have provable separations between different computational models. This is in contrast to the Turing machine world where lower bounds and separations between complexity classes often have to rely on unproven assumptions. At the same time, the model of query complexity is simple and captures the essence of quite a few natural computational processes.

We use $D(f), R(f)$ and $Q(f)$ to denote the minimum number of queries in deterministic, randomized and quantum query algorithms[1] that compute f. It is easy to see that $Q(f) \leq R(f) \leq D(f)$ for any function f. For partial functions (that is, functions whose domain is a strict subset of $\{0,1\}^n$), huge separations are known between all these measures. For example, a randomized algorithm can tell if an n-bit boolean string has 0 ones or at least $n/2$ ones with a constant number of queries, while any deterministic algorithm requires $\Omega(n)$ queries to do this. Similarly, Aaronson and Ambainis [1] recently constructed a partial boolean function f on n variables that can be evaluated using one quantum query but requires $\Omega(\sqrt{n})$ queries for randomized algorithms.

The situation is quite different for total functions.[2] Here it is known that $D(f), R(f)$, and $Q(f)$ are all polynomially related. In fact, $D(f) = O(R(f))^3$ [13] and $D(f) = O(Q(f)^6)$ [3]. A popular variant of randomized algorithms is the zero-error (Las Vegas) model in which a randomized algorithm always has to output the correct answer, but the number of queries after which it stops can depend on the algorithm's coin flips. The complexity $R_0(f)$ is defined as the expected number of queries, over the randomness of the algorithm, for the worst case input x. A tighter relation $D(f) \leq R_0(f)^2$ is known for Las Vegas algorithms (this was independently observed by several authors [11, 4, 18]). Nisan has even shown $D(f) \leq 2R_1(f)^2$ [13], where $R_1(f)$ is the one-sided error randomized complexity of f.

While it has been widely conjectured that these relations are not tight, little progress has been made in the past 20 years on improving these upper bounds or exhibiting functions with separations approaching them. Between $D(f)$ and $R_0(f)$, the best separation known for a total function is the function NAND^k on $n = 2^k$ variables defined by a complete binary NAND tree of depth k. This function satisfies $R_0(\mathrm{NAND}^k) = O(D(\mathrm{NAND}^k)^{0.7537\cdots})$ [17]. Saks and Wigderson showed that this upper bound is optimal for NAND^k, and conjectured that this is the largest gap possible between $R_0(f)$ and $D(f)$ [15]. This function also provides the largest known gap between $R(f)$ and $D(f)$, and satisfies $R(\mathrm{NAND}^k) = \Omega(R_0(\mathrm{NAND}^k))$ [16]. This situation points to the broader fact that, as far as we are aware, no super-linear gap is known between $R(f)$ and $R_0(f)$ for a total function f. Between $Q(f)$ and $R(f)$, the largest known separation is quadratic, given by the OR function on n bits, which satisfies $Q(f) = O(\sqrt{n})$ [10] and $R(f) = \Omega(n)$.

1.1 Our results

We improve the best known separations between all of these measures. In particular, we show that

- There is a function f with $R_0(f) = \widetilde{O}(D(f)^{1/2})$, where the $\widetilde{O}$ notation hides poly-

[1] By default, we use $R(f)$ and $Q(f)$ to refer to bounded-error algorithms (i.e., algorithms that compute $f(x)$ correctly on every input x with probability at least 9/10).

[2] In the rest of the paper we will exclusively talk about total functions. Hence, we sometimes drop this qualification.

logarithmic factors. This refutes the nearly 30 year old conjecture of Saks and Wigderson [15], and shows that the upper bounds $D(f) \leq R_0(f)^2$ and $D(f) \leq 2R_1(f)^2$ are tight, up to poly-logarithmic factors.

- There is a function f with $R(f) = \widetilde{O}(R_0(f)^{2/3})$. This is the first example of a super-linear separation between these measures. In fact, our example also gives $R_1(f) = \widetilde{O}(R_0(f)^{2/3})$.
- There is a function f with $Q(f) = \widetilde{O}(D(f)^{1/4})$. This is the first improvement in nearly 20 years to the quadratic separation given by Grover's search algorithm [10].
- Let $Q_E(f)$ be the exact quantum query complexity, the minimal number of queries needed by a quantum algorithm that stops after a fixed number of steps and outputs $f(x)$ with probability 1. We exhibit functions f_1, f_2 for which $Q_E(f_1) = \widetilde{O}(R_0(f_1)^{3/5})$ and $Q_E(f_2) = \widetilde{O}(R(f_2)^{2/3})$. This improves the best known separation of Ambainis from 2011 [2] giving an f for which $Q(f) = O(R(f)^{0.867\ldots})$. Prior to the work of Ambainis, no super-linear separation was known, the largest known separation being a factor of 2, attained for the PARITY function [8].

A full list of our results are given in the following table.

	lower bound for all f		previous separation		this paper	function	result
$R_0(f)$	$\Omega(D(f)^{1/2})$	[11, 4, 18]	$O(D(f)^{.753\ldots})$	[17]	$\widetilde{O}(D(f)^{1/2})$	$f_{2n,n}$	Corollary 7
$Q(f)$	$\Omega(D(f)^{1/6})$	[3]	$O(D(f)^{1/2})$	[10]	$\widetilde{O}(D(f)^{1/4})$	$f_{2n,n}$	Corollary 7
$Q(f)$	$\Omega(R_0(f)^{1/6})$	[3]	$O(R_0(f)^{1/2})$	[10]	$\widetilde{O}(R_0(f)^{1/3})$	h_{n,n,n^2}	Corollary 14
$R_1(f)$	$\Omega(R_0(f)^{1/2})$	[13]	$O(R_0(f))$		$\widetilde{O}(R_0(f)^{2/3})$	h_{n,n,n^2}	Corollary 14
$Q_E(f)$	$\Omega(R_0(f)^{1/3})$	[12]	$O(R_0(f)^{0.86\ldots})$	[2]	$\widetilde{O}(R_0(f)^{3/5})$	$h_{\sqrt{n},n,n^{3/2}}$	Corollary 16
$Q_E(f)$	$\Omega(R(f)^{1/3})$	[12]	$O(R(f)^{0.86\ldots})$	[2]	$\widetilde{O}(R(f)^{2/3})$	h_{1,n,n^2}	Corollary 17
$\widetilde{\deg}(f)$	$\Omega(D(f)^{1/6})$	[3]	$O(R(f)^{1/2})$	[14]	$\widetilde{O}(R(f)^{1/4})$	h_{1,n,n^2}	Corollary 19

Other separations can be obtained from this table using relations between complexities in Figure 2.

1.2 Göös-Pitassi-Watson function

All of our separations are based on an amazing function recently introduced by Göös, Pitassi, and Watson [9] to resolve the deterministic communication complexity of the Clique vs. Independent set problem, thus solving a long-standing open problem in communication complexity. They solved this problem by first solving a coresponding question in the query complexity model and then showing a general "lifting theorem" that lifts the hardness of a function in the deterministic query model to the hardness of a derived function in the model of deterministic communication complexity. In the query complexity model, their goal was to exhibit a total boolean function f that has large deterministic query complexity and small unambiguous 1-certificate complexity.[3]

The starting point of their construction is the boolean function $f\colon \{0,1\}^M \to \{0,1\}$ with the input variables $x_{i,j}$ arranged in a rectangular grid $M = [n] \times [m]$. The value of the function is 1 if and only if there exists a unique all-1 column. The deterministic complexity of this function is $\Omega(nm)$, since it is hard to distinguish an input with precisely one zero in each column from the input in which one of the zeros is flipped to one. It is also easy to construct a 1-certificate

[3] A subcube is the set of strings consistent with a partial assignment $x_{i_1} = b_1, \ldots, x_{i_s} = b_s$. Its length is s, the number of assigned variables. The unambiguous 1-certificate complexity is the smallest s such that $f^{-1}(1)$ can be partitioned into subcubes of length s (whose corresponding partial assignments are consequently 1-certificates of f). See Section 2 for full definitions.

of length $n + m - 1$: Take the all-1 column and one zero from each of the remaining columns. This certificate is not always unique, however, as there can be multiple zeros in a column and any of them can be chosen in a certificate. Indeed, it is impossible to *partition* the set of all positive inputs into subcubes of small length.

Göös *et al.* added a surprisingly simple ingredient that solves this problem: pointers to cells in M. One can specify which zero to take from each column by requiring that there is a path of pointers that starts in the all-1 column and visits exactly one zero in all other columns, see Figure 1. Thus, the set of positive inputs breaks apart into a disjoint union of subcubes of small length. Since the pointers provide great flexibility in the positioning of zeros, this function is still hard for a deterministic algorithm.

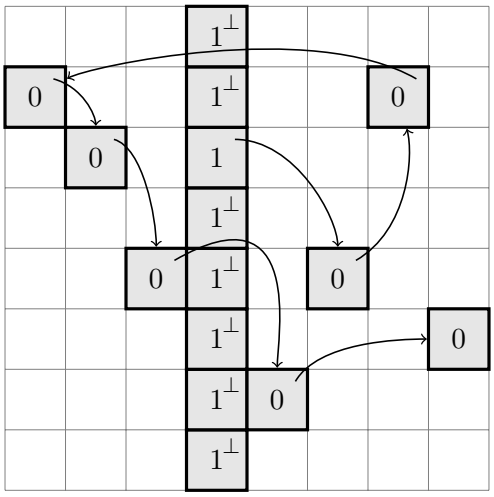

Figure 1: An example of a 1-certificate for the Göös-Pitassi-Watson function. The center of a cell $x_{i,j}$ shows $\mathrm{val}(x_{i,j})$ and the top right corner shows $\mathrm{point}(x_{i,j})$

Formally, the definition of the Göös-Pitassi-Watson function is as follows. Let n and m be positive integers, and $M = [n] \times [m]$ be a grid with n rows and m columns. Let $\widetilde{M} = M \cup \{\perp\}$. Elements in $\widetilde{M}$ are considered as pointers to the cells of M, where $\perp$ stands for the null pointer.

The function $g_{n,m} \colon (\{0,1\} \times \widetilde{M})^M \to \{0,1\}$ is defined as follows. We think of each tuple $v = (b, p) \in \{0,1\} \times \widetilde{M}$ in the following way. The element $b \in \{0,1\}$ of the pair is the *value* and the second element $p \in \widetilde{M}$ is the *pointer*. We will use the notation $\mathrm{val}(v) = b$ and $\mathrm{point}(v) = p$.

Although $g_{n,m}$ is not a boolean function, it can be converted into an associated boolean function by encoding the elements of the input alphabet $\Sigma = \{0,1\} \times \widetilde{M}$ using $\lceil \log |\Sigma| \rceil$ bits.

An input $(x_{i,j})_{(i,j)\in M}$ evaluates to 1 if and only if the following three conditions are satisfied (see Figure 1 for an illustration):

1. There is exactly one column b such that $\mathrm{val}(x_{i,b}) = 1$ for all $i \in [n]$. We call this the *marked column*.
2. In the marked column, there exists a unique cell a such that $x_a \neq (1, \perp)$. We call a the *special element*.
3. For the special element a, by following the pointers inductively defined as $p_1 = \mathrm{point}(x_a)$ and $p_{s+1} = \mathrm{point}(x_{p_s})$ for $s = 1, \ldots, m-2$ we visit every column except the marked column, and $\mathrm{val}(x_{p_s}) = 0$ for each $s = 1, \ldots, m-1$.

For each positive input x, the all-1 column satisfying items (1) and (2) of the definition, and the path following the pointers given by item (3) give a unique minimal 1-certificate of x. Thus,

the unambiguous 1-certificate complexity of this function is $n+m-1$. Göös *et al.* showed that this function has deterministic query complexity mn, giving a quadratic separation between the two when $n = m$.

1.3 Our technique and pointer functions

As described in the previous section, Göös *et al.* showed how pointers can make certificates unambiguous without substantially increasing their size. This technique turns out to be quite powerful for other applications as well. In fact, all results in this paper stem from a systematic application of this technique, in combination with several new ideas.

Making partial functions total One nice application of pointers is that they can essentially turn a partial function into a total one. This is beneficial as it is easy to prove separations for a partial function. Let us describe our separation between Las Vegas and Monte Carlo query complexities as an indicative example.

It is easy to provide a separation between R_0 and R for partial functions. Indeed, let $1 \leq k \leq m$ be integers, and consider a partial function f on m variables defined as follows. For $x \in \{0,1\}^m$, the value of $f(x)$ is 1 if the Hamming weight $|x| = k$, and $f(x) = 0$ if $|x| = 0$. Otherwise, the function is not defined. The Monte Carlo complexity of this function is $O(m/k)$, but its Las Vegas complexity is $m - k + 1$, since it takes that many queries to reject the all-0 string.

How can we obtain a total function with the same property that there are either exactly 0 or k marked elements? First, we can use pointers, and define an element j as marked if its value is 1 and it is contained in a cycle of exactly k marked elements. Thus, when one marked element is found, it only takes $k-1$ queries to find the remaining marked elements. However, this is not enough, as we have to show that all the remaining elements are zeros.

For this, we use the same idea as in the Göös-Pitassi-Watson function. Take the grid M, and define a column as marked if it belongs to a cycle of exactly k all-1 columns and has a path of zeros starting in it and going through all the remaining columns. Thus, indeed, each input contains either 0 or k marked columns. Moreover, testing that a column is all-1 takes n queries and with high probability we will find one after $O(m/k)$ trials, that is, in $O(nm/k)$ total queries. After an all-1 column is found, it takes $kn+m$ queries to verify that it is marked. We show that the Las Vegas complexity of this function is $\Omega(nm)$ using essentially the same idea as for the function f. Setting $k = n$ and $m = n^2$ gives a n^2 vs n^3 separation between the Monte Carlo and Las Vegas query complexities.

Our separations involving exact quantum query complexity follow the same idea.

Back pointers Using the Göös-Pitassi-Watson function $g_{n,m}$, we can already give a larger separation between R_0 and D than previously known. However, we do not know if this function can realize an optimal separation between these measures. A key new ingredient we use to achieve a nearly optimal separation is back pointers, which provide a hint that simplifies the job for the randomized algorithm, but does not help a deterministic algorithm.

A back pointer points to a column in $[m]$. To each zero on the path of condition (3), we add a pointer back to the marked (all-1) column. Let Z be the set of zeros in some column and $B(Z)$ be the set of columns pointed to by the back pointers in Z. If the value of function is 1, $B(Z)$ must contain the marked column.

We consider a randomized algorithm that estimates the number of zeros in each column of $B(Z)$ by sampling. If we find a zero in every column of $B(Z)$, then we can reject the input. On the other hand, we can tune the sampling so that if no zero is found in a column $c \in B(Z)$,

then with high probability c has at most $|Z|/2$ many zeros. We then move to this column c and repeat the process. Even if c is not marked, we have made progress by halving the number of zeros, and in a logarithmic number of repetitions we either find the marked column or reject.

Use of a balanced tree Replacing the path of pointers to unmarked columns with a balanced tree is used for extending the Las Vegas and deterministic lower bounds to Monte Carlo complexity. The problem with the original Göös-Pitassi-Watson function is that a randomized algorithm, when it finds a zero on the path in condition (3), can follow the path starting from that vertex. On average, it can thus eliminate half of the columns.

We fix this problem by replacing the path in condition (3) with a balanced binary tree. The expected size of a subtree rooted in a node of the tree is logarithmic. Thus, even if the algorithm finds a zero, it does not learn much, and we are able to prove an $\Omega(nm/\log m)$ lower bound.

Our second application of this technique is for quantum algorithms. After the algorithm finds a marked column, it should check the zeros given by the path from condition (3). The last element of the path can be only accessed in m queries. However, if we arrange the zeros in a binary tree, each zero can be accessed in a logarithmic number of queries, hence, they can be tested in $\widetilde{O}(\sqrt{m})$ queries using Grover's search.

In principle, both of the above problems can be solved by adding direct pointers from the special element (a in condition (2)) to a zero in each non-marked column (that is, using an $(m-1)$-ary tree of depth 1 instead of a binary tree of depth $O(\log m)$). The problem with this solution is that the size of the alphabet becomes exponential, rendering this construction useless for boolean functions.

Choice of separating functions We have outlined above three ingredients that can be added to the original Göös-Pitassi-Watson function: increasing the number of marked columns, using back pointers, and identifying unmarked columns by a tree of pointers. These ingredients can be added in various combinations to produce different effects. In order to reduce the number of functions introduced, in this paper we stick to two variations: a function $f_{n,m}$ which incorporates back pointers and the use of a balanced tree, and a function $h_{k,n,m}$ which has k marked columns and uses a balanced tree but has no back pointers. This does not mean that a given separation cannot be proven with a different combination of ingredients. For example, as outlined above, the R_0 vs D separation can be proven for the original Göös-Pitassi-Watson function by equipping each node of the path with a back pointer to the marked column.

2 Preliminaries

We let $[n] = \{1, 2, \ldots, n\}$. We use $f(n) = \widetilde{O}(g(n))$ to mean that there exists constants c, k and an integer N such that $|f(n)| \leq c|g(n)| \log^k(n)$ for all $n > N$.

In the remaining part of this section, we define the notion of query complexity for various models of computation. For more detail on this topic, the reader may refer to the survey [7]. Relations between various models are depicted in Figure 2.

Deterministic query complexity Let Σ be a finite set. A decision tree T on n variables and the input alphabet Σ is a rooted tree, where

- internal nodes are labeled by elements of $[n]$;
- every internal node v has degree $|\Sigma|$ and there is a bijection between the edges from v to its children and the elements of Σ;

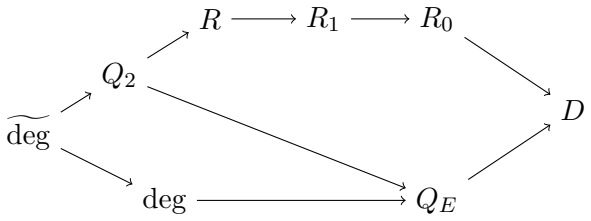

Figure 2: Relations between various complexities. An arrow means that complexity on the left is at most the complexity on the right.

- leaves are labeled from $\{0, 1\}$.

The output of the decision tree T on input $x \in \Sigma^n$, denoted $T(x)$, is determined as follows. Start at the root. If this is a leaf, then output its label. Otherwise, if the label of the root is $i \in [n]$, then follow the edge labeled by x_i (this is called a *query*) and recursively evaluate the corresponding subtree. We say that T computes the function $f\colon \Sigma^n \to \{0, 1\}$ if $T(x) = f(x)$ on every input x. The cost of T on input x, denoted $C(T, x)$, is the number of internal nodes visited by T on x. The deterministic query complexity $D(f)$ of f is the minimum over all decision trees T computing f of the maximum over all x of $C(T, x)$.

Randomized query complexity We follow the definitions for randomized query complexity given in [19, 13]. A randomized decision tree T_μ is defined by a probability distribution μ over deterministic decision trees. On input x, a randomized decision tree first selects a deterministic decision tree T according to μ, and then outputs $T(x)$. The expected cost of T_μ on input x is the expectation of $C(T, x)$ when T is picked according to μ. The worst-case expected cost of T_μ is the maximum over inputs x of the expected cost of T_μ on input x.

There are three models of randomized decision trees that differ in the definition of "computing" a function f.

- Zero-error (Las Vegas): It is required that the algorithm gives the correct output with probability 1 for every input x, that is, every deterministic decision tree T in the support of μ computes f.
- One-sided error: It is required that negative inputs are rejected with probability 1, and positive inputs are accepted with probability at least $1/2$.
- Two-sided error (Monte Carlo): It is required that the algorithm gives the correct output with probability at least $9/10$ for every input x.

The error probability in the one-sided and two-sided cases can be reduced to ε by repeating the algorithm $O(\log \frac{1}{\varepsilon})$ times.

We define randomized query complexities $R_0(f), R_1(f)$, and $R(f)$ as the minimum worst-case expected cost of a randomized decision tree to compute f in the zero, one-sided, and bounded-error sense, respectively.

Distributional query complexity A common way to show lower bounds on randomized complexity, and the way we will do it in this paper, is to consider distributional complexity [19]. The cost of a deterministic decision tree T with respect to a distribution ν, denoted $C(T, \nu)$, is $\mathbb{E}_{x \leftarrow \nu}[C(T, x)]$. The decision tree T computes a function f with distributional error at most δ if $\Pr_{x \leftarrow \nu}[T(x) = f(x)] \geq 1 - \delta$. Finally, the δ-error distributional complexity of T with respect

to ν, denoted $\Delta_{\delta,\nu}(f)$, is the minimum of $C(T,\nu)$ over all T that compute f with distributional error at most δ.

Yao has shown the following:

Theorem 1 (Yao [19]). *For any distribution ν and a function f, $R_0(f) \geq \Delta_{0,\nu}(f)$ and $R(f) \geq \frac{1}{2}\Delta_{2/10,\nu}(f)$.*

In general, Yao shows that the δ-error randomized complexity of a function f is at least $\frac{1}{2}\Delta_{2\delta,\nu}(f)$, for any distribution ν. We obtain the constant $\frac{2}{10}$ on the right hand side as we have defined $R(f)$ for algorithms that err with probability at most $\frac{1}{10}$.

Quantum query complexity The main novelty in a quantum query algorithm is that queries can be made in superposition. For this exposition we assume $\Sigma = [|\Sigma|]$ (this identification can be made in an arbitrary way). The memory of a quantum query algorithm contains two registers, the query register H_Q which holds two integers $j \in [n]$ and $p \in \Sigma$ and the workspace H_W which holds an arbitrary value. A query on input x is encoded as a unitary operation O_x in the following way. On input x and an arbitrary basis state $|j,p\rangle|w\rangle \in H_Q \otimes H_W$,

$$O_x|j,p\rangle|w\rangle = |j, p + x_j \bmod |\Sigma|\rangle|w\rangle \quad .$$

A quantum query algorithm begins in the initial state $|0,0\rangle|0\rangle$ and on input x proceeds by interleaving arbitrary unitary operations independent of x and the operations O_x. The cost of the algorithm is the number of applications of O_x. The outcome of the algorithm is determined by a two-outcome measurement, specified by a complete set of projectors $\{\Pi_0, \Pi_1\}$. If $|\Psi_x\rangle$ is the final state of the algorithm on input x, the probability that the algorithm outputs 1 is $\|\Pi_1|\Psi_x\rangle\|^2$. The exact quantum query complexity of the function f, denoted $Q_E(f)$, is the minimum cost of a quantum query algorithm that outputs $f(x)$ with probability 1 for every input x. The bounded-error quantum query complexity of the function f, denoted $Q(f)$, is the minimum cost of a quantum query algorithm that outputs $f(x)$ with probability at least $9/10$ for every input x.

We will describe our quantum algorithms as classical algorithms which use the following well-known quantum algorithms as subroutines. Let O_x be a quantum oracle encoding a string $x \in \{0,1\}^n$.

- Grover's search [10, 5]: Assume it is known that $|x| \geq t$. There is a quantum algorithm using $O(\sqrt{n/t})$ queries to O_x that finds an i such that $x_i = 1$ with probability at least $9/10$.

- Exact Grover's search [5]. Assume it is known that $|x| = t$. There is a quantum algorithm using $O(\sqrt{n/t})$ queries to O_x that finds an i such that $x_i = 1$ with certainty.

- Approximate counting [6]: Let $t = |x|$. There is a quantum algorithm making $O(\sqrt{n})$ queries to O_x that outputs a number $\tilde{t}$ satisfying $|\tilde{t} - t| \leq \frac{t}{10}$ with probability at least $9/10$.

- Amplitude amplification [5]: Assume a quantum algorithm $\mathcal{A}$ prepares a state $|\psi\rangle = \alpha_0|0\rangle|\psi_0\rangle + \alpha_1|1\rangle|\psi_1\rangle$, where ψ, ψ_0 and ψ_1 are unit vectors, and α_0 and α_1 are real numbers. Thus, the success probability of $\mathcal{A}$, i.e., probability of obtaining 1 in the first register after measuring $|\psi\rangle$, is α_1^2.

  Assume a lower bound p is known on α_1^2. There exists a quantum algorithm that makes $O(1/\sqrt{p})$ calls to $\mathcal{A}$, and either fails, or generates the state $|1\rangle|\psi_1\rangle$. The success probability of the algorithm is at least $9/10$.

In all of these quantum subroutines, the error probability can be reduced to ε by repeating the algorithm $O(\log \frac{1}{\varepsilon})$ times.

Polynomial degree Every boolean function $f\colon \{0,1\}^n \to \{0,1\}$ has a unique expansion as a multilinear polynomial $p = \sum_{S\subseteq[n]} \alpha_s \prod_{i\in S} x_i$. The *degree* of f, denoted $\deg(f)$, is the size of a largest monomial x_S in p with nonzero coefficient α_S. The *approximate degree* of f, denoted $\widetilde{\deg}(f)$, is

$$\widetilde{\deg}(f) = \min\left\{\deg(g) \;\middle|\; |g(x) - f(x)| \le \tfrac{1}{10} \text{ for all } x \in \{0,1\}^n\right\} \ .$$

For any quantum algorithm that uses T queries to the quantum oracle O_x, its acceptance probability is a polynomial of degree at most $2T$ [3]. Therefore, $\deg(f) \le 2Q_E(f)$ and $\widetilde{\deg}(f) \le 2Q_2(f)$.

Certificate complexity A *partial assignment* in Σ^n is a string in $a \in (\Sigma \cup \{\star\})^n$. The length of a partial assignment is the number of non-star values. A string $x \in \Sigma^n$ is *consistent* with an assignment a if $x_i = a_i$ whenever $a_i \neq \star$. Every partial assignment defines a *subcube*, which is the set of all strings consistent with that assignment. For every subcube there is a unique partial assignment that defines it, and we define the length of a subcube as the length of this assignment.

For $b \in \{0,1\}$, a *b-certificate* for a function $f\colon \Sigma^n \to \{0,1\}$ is a partial assignment such that the value of f is b for all inputs in the associated subcube. The *b-certificate complexity* of f is the smallest number k such that the set $f^{-1}(b)$ can be written as a union of subcubes of length at most k. The *unambiguous b-certificate complexity* of f is the smallest number k such that the set $f^{-1}(b)$ can be written as a disjoint union of subcubes of length at most k.

Booleanizing a function While we define functions over a nonboolean alphabet Σ, it is more typical in query complexity to discuss boolean functions. Fix a surjection $b\colon \{0,1\}^{\lceil \log|\Sigma| \rceil} \to \Sigma$. For a function $f\colon \Sigma^n \to \{0,1\}$, we define the associated boolean function $\tilde{f}\colon \{0,1\}^{n\lceil \log|\Sigma| \rceil} \to \{0,1\}$ by $\tilde{f}(x) = f(b(x))$. A lower bound on f in the model where a query returns an element of Σ will also apply to $\tilde{f}$ in the model where a query returns a boolean value. Also, if f can be computed with t queries then we can convert this into an algorithm for computing $\tilde{f}$ with $t\lceil \log|\Sigma| \rceil$ queries by querying all the bits of the desired element. We will state our theorems for nonboolean functions where a query returns an element of Σ and the alphabet size $|\Sigma|$ will always be polynomial in the input length. By the remarks above, such separations can be converted into separations for the associated boolean function with a logarithmic loss.

3 Separations against deterministic complexity

Let n, m, M, $\widetilde{M}$ be as in the definition of the Göös-Pitassi-Watson function. Let also $\widetilde{C} = [m] \cup \{\perp\}$ be the set of pointers to the columns of M. The input alphabet of our function is $\Sigma = \{0,1\} \times \widetilde{M} \times \widetilde{M} \times \widetilde{C}$. For $v \in \Sigma$, we call the elements of the quadruple the *value*, the *left pointer*, the *right pointer* and the *back pointer* of v, respectively. We use notation $\mathrm{val}(v)$, $\mathrm{lpoint}(v)$, $\mathrm{rpoint}(v)$, and $\mathrm{bpoint}(v)$ for them in this order.

Let T be a fixed balanced oriented binary tree with m leaves and $m-1$ internal vertices. For instance, we can make the following canonical choice. If $m = 2^k$ is a power of two, we use the completely balanced binary tree on m leaves as depicted in Figure 3 on the left. Each leaf is at distance k from the root. Otherwise, assume $2^k < m < 2^{k+1}$. Take the completely balanced tree on 2^k leaves, and add a pair of children to each of its $m - 2^k$ leftmost leaves. An example is in Figure 3 on the right.

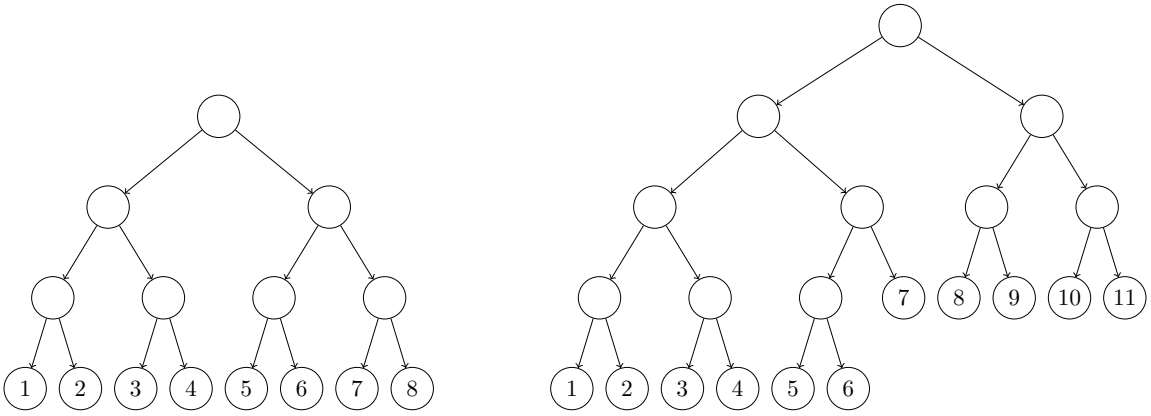

Figure 3: A completely balanced tree on 8 leaves, and a balanced tree on 11 leaves.

We have the following labels in T. The outgoing arcs from each node are labeled by 'left' and 'right'. The leaves of the tree are labeled by the elements of $[m]$ from left to right, with each label used exactly once. For each leaf $j \in [m]$ of the tree, the path from the root to the leaf defines a sequence of 'left' and 'right' of length $O(\log m)$, which we denote $T(j)$.

The function $f_{n,m} \colon \Sigma^M \to \{0,1\}$ is defined as follows. For an input $x = (x_{i,j})$, we have $f_{n,m}(x) = 1$ if and only if the following conditions are satisfied (for an illustration refer to Figure 4):

1. There is exactly one column $b \in [m]$ such that $\mathrm{val}(x_{i,b}) = 1$ for all $i \in [n]$. We refer to it as the *marked column*.
2. In the marked column, there exists a unique cell a such that $x_a \neq (1, \perp, \perp, \perp)$. We call a the *special element*.
3. For each non-marked column $j \in [m] \setminus \{b\}$, let ℓ_j be the end of the path which starts at the special element a and follows the pointers lpoint and rpoint as specified by the sequence $T(j)$. We require that ℓ_j exists (no pointer on the path is $\perp$), ℓ_j is in the jth column, $\mathrm{val}\big(x_{\ell_j}\big) = 0$ and $\mathrm{bpoint}\big(x_{\ell_j}\big) = b$.

Theorem 2. *If $n \geq 2m$ and m is sufficiently large, the deterministic query complexity $D(f_{n,m}) = \Omega(nm)$.*

Proof. We describe an adversary strategy that ensures that the value of the function is undetermined after $nm/2$ many queries, provided $m \geq 4$.

The adversary maintains a set B of columns that can potentially be marked. Initially, $B = [m]$, and will shrink as queries are made. For each $b \in B$, the adversary maintains a set of cells $R_b \subseteq M$ as eventual leaves if b is set to be the marked column. Initially, $R_b = \emptyset$. The sets R_b will grow as the algorithm advances.

The response of the adversary to a query (i, j) is given by the following mutually exclusive cases.

- Case 1: If $(i, j) \in R_b$ for some $b \in B$, then reply with the quadruple $(0, \perp, \perp, b)$.
- Case 2: If $j \in B$ then reply with $(1, \perp, \perp, \perp)$. If less than m unqueried cells are left in column j, then add one unqueried cell from column j to each of the sets R_b for $b \in B \setminus \{j\}$. When this is done, update $R_j \leftarrow \emptyset$ and $B \leftarrow B \setminus \{j\}$.

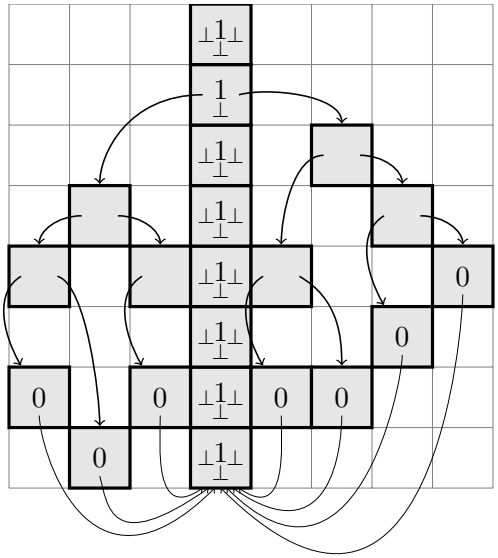

Figure 4: An example of a 1-certificate for the function $f_{8,8}$. The tree T is like in Figure 3 on the left. The center of a cell $x_{i,j}$ shows $\mathrm{val}(x_{i,j})$, the bottom of the cell shows $\mathrm{bpoint}(x_{i,j})$ and the bottom left and right sides show $\mathrm{lpoint}(x_{i,j})$ and $\mathrm{rpoint}(x_{i,j})$, respectively. Values and pointers that are not shown can be chosen arbitrarily.

- Case 3: If neither case 1 nor case 2 holds, reply with $(0, \bot, \bot, \bot)$.

Claim 3. *If $B \neq \emptyset$ and there are at least $n + 2m$ unqueried cells, then the function value is undetermined.*

Proof. If $B \neq \emptyset$, then no all-one column has been constructed, thus by answering all remaining queries with value 0, the adversary can make the function evaluate to 0.

To see the function value can also be set to 1, let $b \in B$. Add to R_b one unqueried cell from each column $j \in B, j \neq b$. Note that R_b now contains one cell from each column $j \neq b$. Assign the quadruple $(0, \bot, \bot, b)$ to each unqueried cell in R_b, and assign the quadruple $(1, \bot, \bot, \bot)$ to every unqueried cell in column b except for one special cell a. Using the cell a as the root construct a tree of pointers isomorphic to T using as internal vertices some of the remaining unqueried cells, and such that the jth leaf is the element of R_b in the jth column. Finally, assign the quadruple $(1, \bot, \bot, \bot)$ to every other cell.

To carry out the construction above, we need m unqueried cells outside of column b and the set R_b to place the internal vertices of the tree. Since there are at most n unqueried cells in column b and m unqueried cells in R_b, it suffices to have $n + 2m$ unqueried cells to do this. □

As it takes at least $m(n - m)$ queries to eliminate all elements of B and at least $n(m - 2)$ many queries to make the number of unqueried cells less than $n + 2m$, we obtain an $nm/2$ lower bound, when $m \geq 4$. □

Theorem 4. *The Las Vegas randomized complexity $R_0(f_n) = \widetilde{O}(n + m)$.*

Proof. For the description, see Algorithm 1. With each iteration of the loop in step 2, k gets reduced by half until it becomes zero, hence, after $O(\log n)$ iterations of the loop, the algorithm terminates.

Algorithm 1 A Las Vegas randomized algorithm for the function $f_{n,m}$

VerifyColumn(j) tests whether the column j is marked

1. If column j does not satisfy condition (2) of the definition of $f_{n,m}$, then reject. Otherwise, let a be the special element.
2. Following the left and right pointers from a and querying the elements along the way, check that the tree rooted at a satisfies condition (3) of the definition of $f_{n,m}$. If it does, accept. Otherwise, reject.

TestColumn(c, k) always returns 'True' if column c has no zeros. If it has more than $k/2$ zeros, returns 'False' with probability $\geq 1 - 1/(nm)^2$. Returns anything in the intermediate cases.

1. Query $O(\frac{n}{k}\log(nm))$ random elements from column c. If no zero was found, return 'True'. Otherwise, return 'False'.

Main procedure of the algorithm

1. Let j be an arbitrary column in $[m]$, and $k \leftarrow n$.
2. Repeat the following actions:
   (a) Query all the elements of column j. If all of them have value 1, VerifyColumn(j).
   (b) If column j contains more than k zeros, then query all the elements of M and output the value of the function.
   (c) Else, let C be the set of nonnull back pointers stored in the zero elements of column j. For each $c \in C$, TestColumn(c, k). If 'False' is obtained for all the columns, reject. Otherwise, let j be any column with outcome 'True'.
   (d) If $k = 0$, reject. Otherwise, let $k \leftarrow \lfloor k/2 \rfloor$, and repeat the loop.

Let us check the correctness of the algorithm. The algorithm only accepts from the procedure VerifyColumn which verifies the existence of a 1-certificate. Thus, the algorithm never accepts a negative input.

To see the algorithm always accepts a positive input, let the input x be positive with marked column b. Consider one iteration of the loop in step 2. If $j = b$, then the algorithm accepts in VerifyColumn(j) on step 2(a). Now assume $j \neq b$. Then, column j contains a zero with a back pointer to b, hence, the algorithm does not reject on step 2(c). The algorithm also does not reject on step 2(d) since, when $k = 0$, the condition in 2(b) applies.

Let us now estimate the expected number of queries made by the algorithm. Condition in step 2(b) is obviously not satisfied on the first iteration of the loop. On a specific later iteration, the probability this condition is satisfied is at most $1/(nm)^2$ by our definition of TestColumn(c, k). Since the loop is repeated $O(\log n)$ times, the contribution of step 2(b) to the complexity of the algorithm is $o(1)$.

If step 2(b) is not invoked, we have the following complexity estimates. VerifyColumn uses $O(m)$ queries. Step 1 uses n queries. Since $|C| \leq k$ on step 2(c), the number of queries in this step is $\widetilde{O}(n)$. Since there is only a logarithmic number of iterations of the loop, the total number of queries is $\widetilde{O}(n + m)$. □

Theorem 5. *The quantum query complexity* $Q(f_{n,m}) = \widetilde{O}(\sqrt{n} + \sqrt{m})$.

Proof. The algorithm is a quantum analogue of Algorithm 1, and it is described in Algorithm 2.

Algorithm 2 A quantum algorithm for the function $f_{n,m}$

VerifyColumn(j) tests whether the column j is marked

1. Use Grover's search to find an element a in column j with nonnull left or right pointer. If no element found, reject. If $\mathrm{val}(x_a) = 0$, reject.
2. Use Grover's search to verify that all elements in column j except a are equal to $(1, \perp, \perp, \perp)$. If not, reject.
3. Use Grover's search to check that condition (3) of the definition of $f_{n,m}$ is satisfied. If it is, accept. Otherwise, reject.

FindGoodBackPointer(j, k) if column j contains $\approx k$ zeros, finds a back pointer to a column containing $\leq k/2$ zeros. If there is one, finds it with probability at least $1/4k$.

1. Use Grover's search to find a zero v out of $\approx k$ contained in column j.
2. If $\mathrm{bpoint}(x_v) = \perp$, return 'False'. Otherwise $c \leftarrow \mathrm{bpoint}(x_v)$.
3. Return 'True' if column c has no zeros, and return 'False' if it has more than $k/2$ zeros. Return anything in the intermediate cases. This can be done using Grover's search.

Main procedure of the algorithm

1. Let j be an arbitrary column in $[m]$.
2. Repeat the following actions. If the loop does not finish after $10 \log n$ iterations, reject.
   (a) Use quantum counting to estimate the number of zeros in column j with relative accuracy $1/10$. Let k be the estimate. If $k = 0$, VerifyColumn(j).
   (b) Execute quantum amplitude amplification on the FindGoodBackPointer(j, k) subroutine amplifying for the output 'True' of the subroutine and assuming its success probability is at least $1/4k$. Let c be the corresponding value of the subroutine after amplification.
   (c) Set $j \leftarrow c$. Repeat the loop.

We assume that every elementary subroutine of the algorithm is repeated sufficiently to reduce its error probability to at most $1/(nm)^2$. This requires a logarithmic number of repetitions, which can be absorbed into the $\widetilde{O}$ factor. Since the algorithm makes less than $O(n+m)$ queries, we may further assume that all the elementary subroutines are performed perfectly.

The analysis is similar to Theorem 4. Again, the algorithm only accepts from VerifyColumn, which is called at most once. The three steps of VerifyColumn correspond to the three conditions defining a 1-input. Any negative input violates one of these conditions, and thus will fail one of these tests with high probability.

Now suppose we have a positive input x with marked column b. In this case, each non-marked column contains a zero with a back pointer to the marked column b. We want to argue that the algorithm accepts x with high probability. The following is the cornerstone of the analysis.

Claim 6. *If the input x is positive and column j contains at most $2k$ zeros, then step 2(b) of the algorithm finds a column c containing at most $k/2$ zeros with high probability.*

Proof. We first claim that FindGoodBackPointer(j, k) returns 'True' with probability at least $1/4k$. Indeed, we assumed that the probability Grover's search on step 1 fails is negligible.

Thus, with high probability, before execution of step 2, v is chosen uniformly at random from the at most $2k$ zeros in column j. One of these zeros contains a back pointer to the marked column b. If it is chosen, step 3 returns 'True' with certainty, which proves our first claim.

Thus, amplitude amplification in step 2(b) of the main procedure will generate the 'True'-portion of the final state of the FindGoodBackPointer subroutine with high probability. Again, since we assume that the error probability of Grover's search on step 3 of FindGoodBackPointer is negligible, we may assume this portion of the state only contains columns c with at most $k/2$ zeros. □

Consider the loop in step 2. We may assume quantum counting is correct in step 2(a). If $j = b$, then VerifyColumn(j) is called on step 2(a), and the algorithm accepts with high probability. So, consider the case $j \neq b$. Column j contains a zero, hence, with high probability, VerifyColumn is not executed on step 2(a). Thus, by Claim 6, the number of zeros in column j gets reduced by a factor of 1.1/2. Therefore, after less than $10 \log n$ iterations, the number of zeros in column j becomes zero, which means $j = b$, and the algorithm accepts with high probability.

Let us now estimate the complexity of the algorithm. Checking steps (1) and (2) of VerifyColumn takes $\widetilde{O}(\sqrt{n})$ many queries. For step (3) of VerifyColumn we need to check that the tree of pointers rooted from x_a satisfies condition (3) from the definition of $f_{n,m}$. That a single path from the root to a leaf is correct can be checked with $O(\log m)$ (classical) queries. There are m many paths, thus Grover's search can look for a violation in the correctness of a path with $\widetilde{O}(\sqrt{m})$ many queries. Overall, VerifyColumn takes $\widetilde{O}(\sqrt{n} + \sqrt{m})$ queries.

The complexity of FindGoodBackPointer(j, k) is $\widetilde{O}(\sqrt{n/k})$ using Grover's search. Thus, the complexity of step 2(b) is $\widetilde{O}(\sqrt{n})$. The complexity of step 2(a) is $\widetilde{O}(\sqrt{n})$ by quantum counting.

As we run the main loop at most $O(\log n)$ many times, the total complexity of the algorithm is $\widetilde{O}(\sqrt{n} + \sqrt{m})$. □

Corollary 7. *There is a total boolean function f with $R_0(f) = \widetilde{O}(D(f)^{1/2})$ and $Q(f) = \widetilde{O}(D(f)^{1/4})$.*

Proof. We first obtain these separations for a non-boolean function. Take $f_{n,m}$ with $n = 2m$. Then the zero-error randomized query complexity is $\widetilde{O}(n)$ by Theorem 4, the quantum query complexity is $\widetilde{O}(\sqrt{n})$ by Theorem 5, and the deterministic query complexity is $\Omega(n^2)$ by Theorem 2. Since the size of the alphabet Σ is polynomial, this also gives the separations for the associated boolean function $\tilde{f}_{2m,m}$. □

4 Separations against randomized complexity

In this section we define a modification of the function used in the last section. Let n, m, M, $\widetilde{M}$ and T be as in the previous section, and let $k : 1 \leq k < m$ be an integer. The new function $h_{k,n,m} \colon \Sigma^M \to \{0,1\}$ is defined as follows. The input alphabet is $\Sigma = \{0,1\} \times \widetilde{M} \times \widetilde{M} \times \widetilde{M}$. For $v \in \Sigma$, we call the elements of the quadruple the *value*, the *left pointer*, the *right pointer* and the *internal pointer* of $x_{i,j}$, respectively. We use notation $\mathrm{val}(v)$, $\mathrm{lpoint}(v)$, $\mathrm{rpoint}(v)$, and $\mathrm{ipoint}(v)$ for them in this order.

For an input $x = (x_{i,j})$, we have $h_{k,n,m}(x) = 1$ if an only if the following conditions are satisfied (for an illustration refer to Figure 5):

1. There are exactly k columns $b_1, \ldots, b_k$ such that $\mathrm{val}(x_{i,b_s}) = 1$ for all $i \in [n]$ and each $s \in [k]$. We refer to these as the *marked columns*.

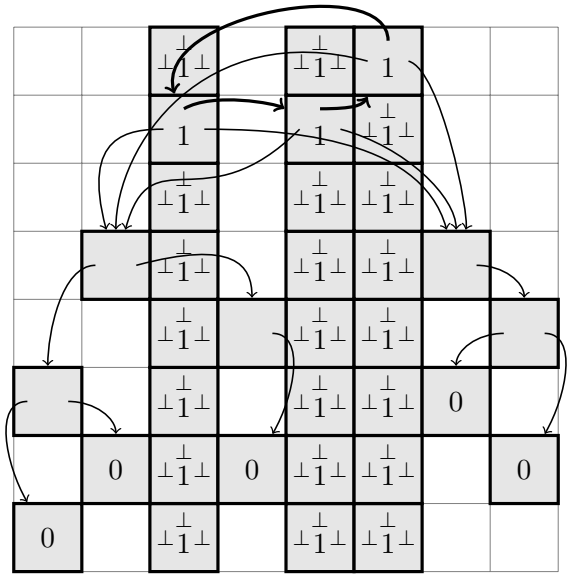

Figure 5: An example of a 1-certificate for the $h_{3,8,8}$ function. The tree T is like in Figure 3 on the left. The center of a cell $x_{i,j}$ shows $\mathrm{val}(x_{i,j})$, the top of the cell shows $\mathrm{ipoint}(x_{i,j})$ and the left and right sides show $\mathrm{lpoint}(x_{i,j})$ and $\mathrm{rpoint}(x_{i,j})$, respectively. Values and pointers that are not shown can be chosen arbitrarily.

2. Each marked column b_s contains a unique cell a_s such that $x_{a_s} \neq (1, \bot, \bot, \bot)$. We call a_s a *special element*.
3. We have $\mathrm{ipoint}(x_{a_s}) = a_{s+1}$ for all $s \in [k-1]$, and $\mathrm{ipoint}(x_{a_k}) = a_1$. Also, $\mathrm{lpoint}(x_{a_s}) = \mathrm{lpoint}(x_{a_t})$ and $\mathrm{rpoint}(x_{a_s}) = \mathrm{rpoint}(x_{a_t})$ for all $s, t \in [k]$.
4. For each non-marked column $j \in [m] \setminus \{b_1, \ldots, b_k\}$, let ℓ_j be the end of the path which starts at a special element a_s (whose choice is irrelevant) and follows the pointers lpoint and rpoint as specified by the sequence $T(j)$. We require that ℓ_j exists (no pointer on the path is $\bot$), ℓ_j is in the jth column, and $\mathrm{val}(x_{\ell_j}) = 0$.

Theorem 8. *If n and m are sufficiently large, the Las Vegas randomized query complexity $R_0(h_{k,n,m}) = \Omega(nm)$ for any $k < m/2$.*

Proof. We construct a hard probability distribution on negative inputs such that any Las Vegas randomized algorithm has to make $\Omega(nm)$ queries in expectation to reject an input sampled from it. Hence, for any Las Vegas algorithm, there exists a negative input, which requires an expected $\Omega(nm)$ number of queries to reject, proving $R_0(h_{k,n,m}) = \Omega(nm)$.

Each input $x = (x_{i,j})$ in the hard distribution is specified by a function $\ell_x \colon [m] \to [n]$. The function specifies the positions of the leaves of the tree T in a possible positive instance. The definition of x is as follows

$$x_{i,j} = \begin{cases} (0, \bot, \bot, \bot), & \text{if } i = \ell_x(j); \\ (1, \bot, \bot, \bot), & \text{otherwise.} \end{cases}$$

The hard distribution is formed in this way from the uniform distribution on all functions ℓ_x. Thus, all pointers are null pointers, and each column contains exactly one zero element in a random position.

Claim 9. *Any Las Vegas algorithm can reject an input x from the hard distribution only if*

- *it has found $m - k + 1$ zeros; or*
- *it has queried more than $n(m - k) - 2m$ elements.*

Proof. Assume these conditions are not met. Then, we can construct a positive input y that is consistent with the answers to the queries obtained by the algorithm so far.

Indeed, choose a set $B = \{b_1, \dots, b_k\}$ of columns where no zero was found. Define $a_s = \big(\ell_x(b_s), b_s\big)$ for each $s \in [k]$. These elements have not been queried yet. Define $\mathrm{val}(y_{a_s}) = 1$ for all s, as well as $\mathrm{ipoint}(y_{a_s}) = a_{s+1}$ for all $s \in [k-1]$, and $\mathrm{ipoint}(y_{a_k}) = a_1$.

Remove from the tree T the leaves with labels in B and the root. Let the resulting graph be T'. For each $j \notin B$, put the leaf of T' with label j into $(\ell_x(j), j)$, set its value to 0 and all pointers to $\perp$. Put the remaining nodes of T' into the still unqueried cells of M preserving the structure of the graph. Set their value to 0 and their internal pointers to $\perp$. Let u and v be the cells where the left and the right child of the root of T went. For each $s \in [k]$, set $\mathrm{lpoint}(y_{a_s}) = u$ and $\mathrm{rpoint}(y_{a_s}) = v$. Set all the remaining cells to $(1, \perp, \perp, \perp)$. The resulting input is positive, and consistent with the answers to the queries obtained by the algorithm. □

By Theorem 1 it suffices to show that any deterministic algorithm $\mathcal{D}$ makes an expected $\Omega(nm)$ number of queries to find $m - k$ zeros in an input from the hard distribution. Let us prove this formally.

Consider a node S of the decision tree $\mathcal{D}$. Call a column $j \in [m]$ *compromised* in S if either a zero was found in it, or more than $n/2$ of its elements were queried. For an input x, let $A_t(x)$ be the number of compromised columns on input x after t queries. Similarly, let $B_t(x)$ be the number of queries made outside the compromised columns. Let us define

$$I_t(x) = A_t(x) + \frac{2}{n} B_t(x).$$

Note that $A_t(x)$ can only increase as t increases, whereas $B_t(x)$ can increase or decrease.

Claim 10. *For a non-negative integer t, we have*

$$\mathbb{E}_x\big[I_{t+1}(x)\big] - \mathbb{E}_x\big[I_t(x)\big] \le \frac{4}{n}, \tag{1}$$

where the expectation is over the inputs in the hard distribution.

Proof. Fix t. We say two inputs x and y are equivalent if they get to the same vertex of the decision tree after t queries. We prove that (1) holds with the expectation taken over each of the equivalency classes. Fix an equivalence class, let x be an input in the class, and (i, j) be the variable queried by $\mathcal{D}$ on the $(t+1)$st query on the input x. Note that (i, j), $A_t(x)$ and $B_t(x)$ do not depend on the choice of x.

Consider the following cases, where each case excludes the preceding ones. All expectations and probabilities are over the uniform choice of an input in the equivalence class.

- The jth column is compromised. Then $I_{t+1}(x) = I_t(x)$, and we are done.
- After the cell (i, j) is queried, more than half of the cells in the jth column have been queried. Then, $A_t(x)$ increases by 1, and $B_t(x)$ drops by $\lfloor n/2 \rfloor$. Hence, $\mathbb{E}_x\big[I_{t+1}(x)\big] \le \mathbb{E}_x\big[I_t(x)\big] + 1/n$.
- Consider the remaining case. We have $\Pr_x[i = \ell_x(j)] \le 2/n$. If $i = \ell_x(j)$, then $A_t(x)$ grows by 1, and $B_t(x)$ can only decrease. If $i \ne \ell_x(j)$, then $A_t(x)$ does not change, and $B_t(x)$ grows by 1. Thus, $\mathbb{E}_x\big[I_{t+1}(x)\big] - \mathbb{E}_x\big[I_t(x)\big] \le \frac{2}{n} + \frac{2}{n} = \frac{4}{n}$. □

Let $t = \lfloor n(m-k)/8 \rfloor$. Clearly, $\mathbb{E}_x[I_0(x)] = 0$ for all x. Claim 10 implies $\mathbb{E}_x[I_t(x)] \leq (m-k)/2$. By Markov's inequality, $\Pr_x\big[I_t(x) \geq m-k\big] \leq 1/2$. By Claim 9, the probability $\mathcal{D}$ has not rejected x after $t' = \min\{t,\ n(m-k)-2m\}$ queries is at least $1/2$. Hence, $R_0(h_{k,n,m}) \geq t'/2 = \Omega(nm)$. □

Theorem 11. *If n and m are sufficiently large, the randomized query complexity $R(h_{1,n,m}) = \Omega\big(\frac{nm}{\log m}\big)$.*

Note that Theorem 11 only considers the case $k = 1$. It is proven in a similar fashion to Theorem 8 using the additional fact that the expected size of a subtree rooted in a node of a balanced tree is logarithmic. The proof is given in Section 5.

4.1 Randomized One-Sided and Quantum versus Las Vegas

Theorem 12. *The randomized query complexity with a one-sided error $R_1(h_{k,n,m}) = O(nm/k + kn + m)$.*

Proof. We search for a column consisting only of ones. In the positive case, there are k such columns, so we will find one after $O(m/k)$ trials with high probability. If we do not find such a column, we reject. Each trial takes n queries, so we spend $O(nm/k)$ queries on this step.

If we find a marked column j, we query all $x_{i,j}$ for $i \in [n]$ to check that it satisfies condition (2) of the definition of $h_{k,n,m}$. Let a be the corresponding special element. We follow the internal pointer from a to check that condition (3) is also satisfied and to find all the marked columns. We then check condition (2) on them as well. All this requires kn queries.

After that, we follow the left and right pointers from a, and check that condition (4) of the definition of $h_{k,n,m}$ is satisfied. This requires $O(m)$ queries.

We never accept a negative instance, so the algorithm has one-sided error. □

Theorem 13. *The quantum query complexity $Q(h_{k,n,m}) = \widetilde{O}(\sqrt{nm/k} + \sqrt{kn} + k + \sqrt{m})$.*

Proof. We search for a column consisting only of ones using Grover's search. Testing one column takes $O(\sqrt{n})$ queries. Also, in the positive case, there are k such columns, so we will find one after $O(\sqrt{nm/k})$ queries with high probability. If we do not find such a column, we reject.

In case we find a marked column j, we use Grover's search to check that it satisfies condition (2) of the definition of $h_{k,n,m}$. This requires $O(\sqrt{n})$ queries. Let a be the corresponding special element. We follow the internal pointer from a to find all the special elements and to check that condition (3) of the definition of $h_{k,n,m}$ is satisfied. This requires k queries. We use Grover's search to check that all the remaining elements of the marked columns are equal to $(1, \perp, \perp, \perp)$. This requires $O(\sqrt{kn})$ queries.

After that, we check condition (4) of the definition of $h_{k,n,m}$. Since there are less than m elements ℓ_j to check and each one can be tested in $O(\log m)$ queries, Grover's search can check this condition in $\widetilde{O}(\sqrt{m})$ queries. □

Corollary 14. *There is a total boolean function f with $R_1(f) = \widetilde{O}(R_0(f)^{2/3})$ and $Q(f) = \widetilde{O}(R_0(f)^{1/3})$.*

Proof. We first obtain these separations for a non-boolean function. Take $h_{k,n,m}$ with $k = n$ and $m = n^2$. Then the one-sided error randomized complexity is $O(n^2)$ by Theorem 12, the quantum complexity is $\widetilde{O}(n)$ by Theorem 13, and the Las Vegas randomized complexity is $\Omega(n^3)$ by Theorem 8. Since the size of the alphabet Σ is polynomial, we obtain the required separations for the associated boolean function. □

4.2 Exact Quantum versus Randomized

Theorem 15. *The exact quantum query complexity* $Q_E(h_{k,n,m}) = O(n\sqrt{m/k} + kn + m)$.

Proof. The algorithm is similar to Theorem 12. The only difference is that on the first step we use the exact version of Grover's search to find a column consisting only of ones. We assume there are exactly k such columns. This takes us $O(n\sqrt{m/k})$ queries, and in the positive case we find a marked column with certainty.

The remaining steps of the algorithm are like in the proof of Theorem 12. □

Corollary 16. *There is a total boolean function* f *with* $Q_E(f) = O(R_0(f)^{3/5})$.

Proof. Take $h_{k,n,m}$ with $k = \sqrt{n}$ and $m = n^{3/2}$. Then the exact quantum query complexity is $O(n^{3/2})$ by Theorem 15 and the Las Vegas randomized query complexity is $\Omega(n^{5/2})$ by Theorem 8. Since the size of the alphabet Σ is polynomial, this also gives the separation for the associated boolean function $\tilde{h}_{k,n,m}$. □

Corollary 17. *There exists a total boolean function* f *with* $Q_E(f) = \widetilde{O}(R(f)^{2/3})$.

Proof. Take $h_{1,n,m}$ with $m = n^2$. Then the exact quantum query complexity is $O(n^2)$ by Theorem 15 and the Monte Carlo randomized query complexity is $\widetilde{\Omega}(n^3)$ by Theorem 11. Since the size of the alphabet Σ is polynomial, this also gives the separation for the associated boolean function $\tilde{h}_{1,n,m}$. □

4.3 Approximate Polynomial Degree versus Monte Carlo

Theorem 18. *Let* $\tilde{h}_{1,n,m} \colon \{0,1\}^{nm\lceil \log |\Sigma| \rceil} \to \{0,1\}$ *be the boolean function associated to* $h_{1,n,m}$. *The approximate polynomial degree* $\widetilde{\deg}(\tilde{h}_{1,n,m}) = \widetilde{O}(\sqrt{n} + \sqrt{m})$

Proof. For $j \in [m]$, let $g_j \colon \{0,1\}^{nm\lceil \log |\Sigma| \rceil} \to \{0,1\}$ be defined as follows. The value $g_j(x)$ is 1 if $\tilde{h}_{1,n,m}(x) = 1$, and j is the marked column. Otherwise, $g_j(x) = 0$.

For each $j \in [m]$, the function $g_j(x)$ can be evaluated in $\widetilde{O}(\sqrt{n} + \sqrt{m})$ quantum queries in the same way as we test a column j (after we have found a candidate marked column) in the proof of Theorem 13. Repeating this quantum algorithm $O(\log m)$ times, we may assume that its error probability is at most $1/(10m)$. We then use the connection between quantum query algorithms and polynomial degree of [3] to construct a polynomial $p_j(x)$ of degree $2T$ (where T is the number of queries) that is equal to the acceptance probability of this algorithm. The polynomial $p_j(x)$ is of degree $\widetilde{O}(\sqrt{n} + \sqrt{m})$ and satisfies $0 \le p_j(x) \le 1/(10m)$ if $g_j(x) = 0$, and $1 - 1/(10m) \le p_j(x) \le 1$ otherwise.

We then define a polynomial $p(x) = \sum_j p_j(x)$. If $h_{1,n,m}(x) = 0$, then all $g_j(x) = 0$ and $0 \le p(x) \le 1/10$. Otherwise, there is unique $b \in [m]$ such that $g_b(x) = 1$, and all other $g_j(x) = 0$. In this case, $1 - 1/(10m) \le p(x) \le 11/10$. Thus, $p(x)$ is an approximating polynomial to $h_{1,n,m}$ and its degree is $\widetilde{O}(\sqrt{n} + \sqrt{m})$. □

Corollary 19. *There is a total boolean function* f *with* $\widetilde{\deg}(f) = \widetilde{O}(R(f)^{1/4})$.

Proof. Take $\tilde{h}_{1,n,m}$ from Theorem 18 with $m = n$. Then the approximate degree is $\widetilde{O}(\sqrt{n})$, and the Monte Carlo randomized query complexity is $\Omega(n^2)$ by Theorem 11. □

5 Proof of Theorem 11

Lemma 20. *Assume T is a balanced binary tree with m leaves, and at least a $1/4$ fraction of its nodes are marked. Let u be sampled from all the marked nodes of T uniformly at random. The expected size of the subtree rooted at u does not exceed $C_0 \log m$ for some constant C_0.*

Proof. Let us first consider the case when T is a complete balanced binary tree with $2^k - 1$ nodes and all its nodes are marked. Then, the expected size of the subtree is (where i is the height of the node u)

$$\sum_{i=1}^{k} \frac{2^{k-i}}{2^k - 1} \cdot (2^i - 1) \le k. \tag{2}$$

In the general case, T can be embedded into a complete balanced binary tree T' with $2^k - 1$ nodes, where $k = \lceil \log m \rceil + 1$. Mark in T' all the nodes marked in T. Again, an $\Omega(1)$ fraction of the nodes is marked. Hence, for each node u, a probability that u is sampled from the marked nodes of T' is at most a constant times its probability to be sampled from all the nodes of T'. Thus, the expected size of the subtree is at most a constant times the value in (2). □

By Theorem 1, it suffices to construct a hard distribution on inputs and show that any deterministic decision tree that computes $h_{1,n,m}$ with distributional error less than $2/10$ on the hard distribution makes an $\Omega(nm/\log m)$ expected number of queries. By Markov's inequality, it suffices to show that any deterministic decision tree that performs this task with error $3/8$ has *depth* $\Omega(nm/\log m)$. We now define the hard distribution.

Let T be the balanced binary tree from the definition of $h_{1,n,m}$. Denote by r the root of the tree, and by T^{N} the set of internal nodes of T. The latter has cardinality $m - 1$.

An input $x = (x_{i,j})$ is defined by a quadruple $(v_x, \pi_x, \ell_x^{\mathrm{L}}, \ell_x^{\mathrm{N}})$, where

- $v_x \in \{0, 1\}$, it will be the value of the function $h_{1,n,m}$ on x;
- $\pi_x \colon T^{\mathrm{N}} \to [m]$ is an injection, it specifies to which columns the internal nodes of T will go; and
- $\ell_x^{\mathrm{L}} \colon [m] \to [n]$ and $\ell_x^{\mathrm{N}} \colon [m] \to [n]$ are functions satisfying $\ell_x^{\mathrm{L}}(j) \neq \ell_x^{\mathrm{N}}(j)$ for each $j \in [m]$. They specify the rows where the leaves and the internal nodes of the tree T land in the column j.

The definition is as follows. Remove the leaf with the label $\pi_x(r)$ from T. For each column $j \neq \pi_x(r)$, put the leaf j into the cell $\big(\ell_x^{\mathrm{L}}(j), j\big)$, set all its pointers to $\perp$ and its value to 0. Then, for each internal node u of the tree, put it at $\big(\ell_x^{\mathrm{N}}(\pi_x(u)), \pi_x(u)\big)$, set its left and right pointers so that the structure of the tree is preserved. If $u \neq r$, assign its internal pointer to $\perp$, and set its value to 0. Otherwise, if $u = r$, assign its internal pointer to itself, and set its value to v_x. Assign the quadruple $(1, \perp, \perp, \perp)$ to all other cells. It is easy to see that the value of the function $h_{1,n,m}$ on this input is v_x.

The hard distribution is defined as the uniform distribution over the quadruples $(v_x, \pi_x, \ell_x^{\mathrm{L}}, \ell_x^{\mathrm{N}})$ subject to the constraint $\ell_x^{\mathrm{L}}(j) \neq \ell_x^{\mathrm{N}}(j)$ for each $j \in [m]$.

Let $\mathcal{D}$ be a deterministic decision tree of depth

$$D = \frac{nm}{64 C_0 \log m}.$$

We will prove that $\mathcal{D}$ errs on x, sampled from the hard distribution, with probability at least $3/8$.

Let x be an input from the hard distribution, and consider the vertex S of $\mathcal{D}$ after t queries to x. We say that a column $j \in [m]$ and the corresponding tree element $\pi_x^{-1}(j)$ (if it exists) are *compromised* on the input x after t queries if at least one of the following three conditions is satisfied:

- one of the cells $(\ell_x^{\mathrm{L}}(j), j)$ and $(\ell_x^{\mathrm{N}}(j), j)$ has been queried;
- more than a half of the cells in the jth column have been queried; or
- in the tree T there exists an ancestor u of $\pi_x^{-1}(j)$ such that one of the above two conditions is satisfied for $\pi_x(u)$.

Let $A_t(x)$ denote the number of compromised columns, and $B_t(x)$ denote the number of cells queried outside the compromised columns, both after t queries. Consider the following quantity

$$I_t(x) = \min\left\{ A_t(x) + \frac{4C_0 \log m}{n} B_t(x),\ \ \frac{m}{2} \right\}.$$

Note that $A_t(x)$ can only increase as t increases, whereas $B_t(x)$ can increase or decrease.

Claim 21. *For a non-negative integer t, we have*

$$\mathbb{E}_x\big[I_{t+1}(x)\big] - \mathbb{E}_x\big[I_t(x)\big] \le \frac{8C_0 \log m}{n}, \tag{3}$$

where the expectation is over the inputs in the hard distribution.

We will prove Claim 21 a bit later. Now let us show how it implies the theorem. Clearly, $I_0(x) = 0$ for all x. Claim 21 implies that $\mathbb{E}_x[I_D(x)] \le m/8$. By Markov's inequality, the probability that $I_D(x) \ge m/2$ is at most $1/4$.

Let x be an input satisfying $I_D(x) < m/2$. Thus, x has less than $m/2$ compromised columns after D queries. In particular, the variable $a = \big(\ell_x^{\mathrm{N}}(\pi_x(r)), \pi_x(r)\big)$, corresponding to the root of T, has not been queried. Let y be the input given by $(1 - v_x, \pi_x, \ell_x^{\mathrm{L}}, \ell_x^{\mathrm{N}})$. They only differ in a, and $h_{1,n,m}(x) \ne h_{1,n,m}(y)$. Hence, the decision tree $\mathcal{D}$ errs on exactly one of them.

This means that $\mathcal{D}$ errs on x sampled from the hard distribution with probability at least $3/8$.

Proof of Claim 21. We divide the inputs of the hard distribution into equivalence classes, and prove that (3) holds with the expectation over each of the classes. We say that two inputs x and y are equivalent if the following three conditions hold:

- after t queries, the decision tree $\mathcal{D}$ gets to the same vertex on x and y;
- the set C of compromised columns is the same in x and y;
- for all $j \in C$, $\pi_x^{-1}(j) = \pi_y^{-1}(j)$.

Fix an equivalence class, let x be an input in the class, and (i, j) be the variable queried by $\mathcal{D}$ on the $(t+1)$st query on the input x. Note that (i, j), as well as $A_t(x)$ and $B_t(x)$ do not depend on the choice of x. Consider the following cases, where each case excludes the preceding ones. All expectations and probabilities are over the uniform choice of an input in the equivalence class.

- We have $I_t(x) = m/2$. Then, $I_{t+1}(x) \le m/2$, and we are done.
- The jth column is compromised. Then $I_{t+1}(x) = I_t(x)$, and we are done.

- After the cell (i, j) is queried, more than half of the cells in the jth column have been queried. As $I_t(x) < m/2$, less than half of the columns are compromised. By Lemma 20 with non-compromised nodes marked, the expected growth of $A_t(x)$ is at most $C_0 \log m$. On the other hand, the drop in $B_t(x)$ is at least $\lfloor n/2 \rfloor$. Hence, $\mathbb{E}_x\big[I_{t+1}(x)\big] \le \mathbb{E}_x\big[I_t(x)\big]$.
- Consider the remaining case. We have $\Pr_x\big[i \in \big\{\ell_x^{\mathrm{L}}(j), \ell_x^{\mathrm{N}}(j)\big\}\big] \le 4/n$.
  - If i is one of $\ell_x^{\mathrm{L}}(j)$ or $\ell_x^{\mathrm{N}}(j)$, then, as in the previous case, the expected growth of $A_t(x)$ is at most $C_0 \log m$, and $B_t(x)$ can only decrease.
  - If $i \neq a_x(j)$, then $A_t(x)$ does not change, and $B_t(x)$ grows by 1.

  Thus,
$$\mathbb{E}_x\big[I_{t+1}(x)\big] - \mathbb{E}_x\big[I_t(x)\big] \le \frac{4}{n} \cdot C_0 \log m + \frac{4C_0 \log m}{n} = \frac{8C_0 \log m}{n}. \qquad \square$$

Acknowledgments

This research is partially funded by the Singapore Ministry of Education and the National Research Foundation, also through NRF RF Award No. NRF-NRFF2013-13, and the Tier 3 Grant "Random numbers from quantum processes," MOE2012-T3-1-009. This research is also partially supported by the European Commission IST STREP project Quantum Algorithms (QALGO) 600700, by the ERC Advanced Grant MQC, Latvian State Research Programme NeXIT project No. 1 and by the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project).

Part of this work was done while A.B. was at Centre for Quantum Technologies, Singapore. A.B. thanks Miklos Santha for hospitality.

References

[1] S. Aaronson and A. Ambainis. Forrelation: A problem that optimally separates quantum from classical computing. In *Proc. of 47th ACM STOC*, pages 307–316, 2015. `arXiv:1411.5729`.

[2] A. Ambainis. Superlinear advantage for exact quantum algorithms. In *Proc. of 45th ACM STOC*, pages 891–900, 2013. `arXiv:1211.0721`.

[3] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Earlier: *FOCS'98*, `arXiv:quant-ph/9802049`.

[4] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *Proc. of 28th IEEE FOCS*, pages 118–126, 1987.

[5] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74, 2002. `arXiv:quant-ph/0005055`.

[6] G. Brassard, P. Høyer, and A. Tapp. Quantum counting. In *Proc. of 25th ICALP*, volume 1443 of *LNCS*, pages 820–831. Springer, 1998. `arXiv:quant-ph/9805082`.

[7] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288:21–43, 2002.

[8] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998. `arXiv:quant-ph/9708016`.

[9] M. Göös, T. Pitassi, and T. Watson. Deterministic communication vs. partition number. `ECCC:2015/50`, 2015.

[10] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. of 28th ACM STOC*, pages 212–219, 1996. `arXiv:quant-ph/9605043`.

[11] J. Hartmanis and L. A. Hemachandra. One-way functions, robustness, and non-isomorphism of NP-complete sets. In *Proceedings of 2nd Structure in Complexity Theory*, pages 160–173, 1987.

[12] G. Midrijānis. Exact quantum query complexity for total boolean functions. `arXiv:quant-ph/0403168`, 2004.

[13] N. Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. Earlier: *STOC'89*.

[14] N. Nisan and M. Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994. Earlier: *STOC'92*.

[15] M. Saks and A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. of 27th IEEE FOCS*, pages 29–38, 1986.

[16] M. Santha. On the Monte Carlo boolean decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 6(1):75–87, 1995.

[17] M. Snir. Lower bounds for probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[18] G. Tardos. Query complexity or why is it difficult to separate $\mathbf{NP}^A \cap \mathbf{coNP}^A$ from $\mathbf{P}^A$ by a random oracle. *Combinatorica*, 9:385–392, 1990.

[19] A. C. Yao. Probabilistic computations: toward a unified measure of complexity. In *Proc. of 18th IEEE FOCS*, pages 222–227, 1977.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de