The Oracle Separation of BQP and PH: A Recent Advancement in Quantum Complexity Theory

A CS254 STUDENT

Classical computing enables us to encode computation as transitions between concrete memory states according to a set of rules. Quantum computing harvests the power of quantum mechanics to allow us to compute with superpositions (i.e. a linear combination) of memory states. We gain a degree of parallelism that we can exploit for problems exhibiting a certain structure e.g. factoring numbers. This paper presents a brief overview of the theoretical underpinnings of quantum computing and quantum complexity theory without assuming any background in quantum physics or quantum computing. We will then focus on a recent breakthrough in the field: the oracle separation of the complexity classes BQP (bounded quantum polynomial time) and PH (polynomial hierarchy). Understanding this result will give us a deeper appreciation for the new problems we are able to solve efficiently on quantum computers but not on their classical counterparts.

Additional Key Words and Phrases: Quantum Computing, BQP, PH, Complexity Theory, Oracles

ACM Reference Format:
A CS254 Student. 2019. The Oracle Separation of BQP and PH: A Recent Advancement in Quantum Complexity Theory. 1, 1 (March 2019), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Most modern computers harvest the power of transistors as a cheap and scalable way to represent memory and operations on memory. Logical gates (e.g. AND, OR, NOT, NAND, NOR, XOR) enable us to perform operations on these memory states. Turing machines and Boolean circuits capture this kind of classical computation well since we deal only with concrete memory states.

Memory states in a quantum computer take on a more probabilistic nature. Instead of a bit being 0 or 1, they can be somewhere in between – collapsing to one of these states with a certain probability dependent on the physical system that represents the bit. Think of the classic Schrodinger's Cat thought experiment – until you open the box, the cat is both dead and alive (each with a probability of 50%). Similarly, we can place quantum bits, known as **qubits** in a state of superposition where the qubit can **collapse** to a 0 or a 1, each with a controllable probability, once measured by detectors. These qubits can be represented by various physical systems and phenomena e.g. (polarization?) photos, electron spins, etc.

Since our fundamental unit of representation, the qubit, has this strange probabilistic caveat, we must seek a different kind of theoretical representation. Although there do exist quantum analogues of Turing Machines [7], we will not discuss them here. Instead, we will present some fundamental quantum gates, which can be composed together to describe the full space of quantum programs. In the following sections, I will describe how qubits are represented as vectors in $\mathbb{C}^2$, how computational states are represented as tensor products of qubits (and more compactly, in Dirac bra-ket notation), and how gates are represented as unitary matrices. Before discussing the main result of the Raz Tal paper, I will also provide an example of two problems that can be solved exponentially faster on a quantum computer as compared to a classical computer: the Deutsch Oracle Problem and Simon's problem (which was a precursor to Shor's famous algorithm for factoring). Note that the following is not a comprehensive review of quantum computing, although hopefully it will provide the reader with guidance in being able to look up further resources as necessary.

Author's address: A CS254 Student.

XXXX-XXXX/2019/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 QUBITS

As column vectors, the qubits for 0 and 1 are represented as follows:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}, |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

The $|\cdot\rangle$ is called ket notation. We can describe states of multiple qubits using ket notation. The computational state of multiple qubits is their tensor product:

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, |3\rangle = |11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

As mentioned before, the components of these vectors can take on complex values. In this representation, the sums of the squares of the components of a vector describing a qubit evaluates to 1. We can write a qubit in the form $a_0 |0\rangle + a_1 |1\rangle$, we have that the probability the qubit collapses to $|0\rangle$ when measured is $|a_0|^2$. Similarly, the probability the qubit collapses to $|1\rangle$ when measured is $|a_1|^2$.

Often, two kets may be written together to implicitly denote their tensor product i.e. $|0\rangle |1\rangle = |0\rangle \otimes |1\rangle = |01\rangle$.

3 QUANTUM GATES

We often represent Quantum gates as matrices since they can be viewed as linear transformations acting upon qubit states, which can be viewed as vectors. On a more technical note, a matrix U that represents a quantum gate must be unitary and have the property $U^\dagger U = I$, where $U^\dagger$ denotes the complex conjugate transpose of U. This detail has to do with preserving the nice mathematical properties of qubits we discussed above as well as the physical nature of qubits. In the following sections, we will discuss several common quantum gates.

3.1 NOT

Suppose we wish to define a linear operator that maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$ i.e. performs negations on qubits. We could express this transformation as the following matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

We denote this matrix as X because it is also known as the Pauli-X gate. Using our definitions above, one can verify that indeed $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. If we wish to perform a negation over two qubits, we can apply the tensor product of the operators: $(X \otimes X)|00\rangle = |11\rangle$. For an operator Q, we denote $Q \otimes Q \otimes \cdots \otimes Q$ (n times) as $Q^{\otimes n}$.

Thus far, we have demonstrated operators on concrete qubits. The real power from this abstraction comes from the fact that we can apply any of these quantum operators over superpositions of states i.e. qubits that can probabilistically be $|0\rangle$ or $|1\rangle$ depending on their underlying wave functions. For instance, we can imagine negating a qubit in the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ to obtain $\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle)$, by linearity.

3.2 CNOT

Building off the idea of a NOT gate, presented in the previous section, we can obtain a **controlled-not (CNOT)** gate. The CNOT gate takes in two qubits as input and negates the second qubit if the first qubit is set to $|1\rangle$. It has the following matrix representation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

3.3 Toffoli

Further extending the idea of a CNOT gate, we can obtain the **Toffoli gate**, which acts on three qubits as follows:

$$|c_1\rangle |c_2\rangle |t\rangle \rightarrow |t \oplus (c_1 \cdot c_2)\rangle$$

That is, the third qubit is flipped if both the first and second qubits are set to $|1\rangle$.

3.4 Hadamard

In order to place a qubit in a state of superposition, we apply the Hadamard gate H to it. As a matrix, it is defined as follows:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Note that $H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and that $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Applying the Hadamard gate again brings a qubit out of superposition: $H\frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}(H|0\rangle + H|1\rangle) = \frac{1}{2}(|0\rangle + |1\rangle + |0\rangle - |1\rangle) = \frac{1}{2}(2|0\rangle) = |0\rangle$.

Note that we must include the factor of $\frac{1}{\sqrt{2}}$ to ensure that we encode unit vectors since the squares of the components correspond to probabilities and these probabilities must sum to 1.

We can extend the Hadmard operator to n qubits as follows: the Hadamard of $|x_1\rangle |x_2\rangle \ldots |x_n\rangle$ is $\frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$. Where $(\cdot)$ denotes bitwise dot-product $\mod 2$. This formula follows from the expansion of the tensor product $\frac{1}{\sqrt{2^n}}(|0\rangle + (-1)^{x_1}|1\rangle)(|0\rangle + (-1)^{x_2}|1\rangle) \cdots (|0\rangle + (-1)^{x_n}|1\rangle)$, which follows from the fact that we can consider a Hadamard applied to a single qubit x as $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$. Note that we must multiply by the $\frac{1}{\sqrt{2^n}}$ factor for normalization purposes (recall that the squares of components correspond to probabilities and the probabilities of all possible states must add to 1).

3.5 Oracle Queries

Often times, when discussing complexity results in quantum computing, we consider machines that have access to an oracle that can compute some function $f : \{0,1\}^n \rightarrow \{0,1\}$. There are two common methods of querying this oracle f in a quantum circuit. The **CNOT-query** maps qubits as follows:

$$|x, w\rangle |x, w \oplus f(x)\rangle$$

. The **phase-query** flips around the input vector if $f(x) = 1$:

$$|x\rangle \rightarrow \left|(-1)^{f(x)}x\right\rangle$$

4 BQP

Quantum gates give us a foundation with which we can build quantum circuits. A language is in BQP if it can be computed by a polynomial size quantum circuit that errs with a probability of at most $\frac{1}{3}$. This is analogous to the definition of BPP. Indeed, we can show that $BPP \subseteq BQP$ [7] since quantum algorithms can use superposition states as coin-flips. BQP essentially captures the problems we view as feasible for quantum computers. It is natural to wonder whether quantum computers can efficiently solve certain problems that are infeasible for classical computers. The next two sections describe quantum algorithms that exponentially outperform their best known classical counter-parts. One caveat to note, however, is that these problems deal with oracles (so does the Raz and Tal result) and do not provide general separations between quantum and classical computation.

5 A SIMPLE QUANTUM ALGORITHM

Consider the following scenario: you are given an oracle function $f : \{0,1\} \rightarrow \{0,1\}$ and wish to determine if $f(0) = f(1)$ or $f(0) \neq f(1)$. That is, we wish to determine $f(0) \oplus f(1)$. A classical computer would make two queries to f. A quantum computer only needs to make one query to f since we can pass a superposition of $|0\rangle$ and $|1\rangle$ to f. We can apply another quantum gate to the output of the oracle to bring the result out of superposition. Observe the following circuit diagram:

We can generalize this problem to n bits: given a function $f : \{0,1\}^n \rightarrow \{0,1\}$, determine if the function outputs 0 and 1 half the time or outputs the same value for all inputs. We must query at least $2^{n-1} + 1$ queries to f to determine this in a classical setting. With a

generalized version of the approach presented above, we can solve this with one query to f. Thus, the quantum algorithm provides an exponential speed-up as compared to its classical counterpart.

A quantum algorithm for the single qubit case goes as follows: prepare a single qubit register to contain $|0\rangle$. After applying a Hadamard gate, the register contains $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$. We apply the phase-query to this superposition to obtain $\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}}$ – this is by the linearity of the phase-query operator (all of these operators must be linear).

Now we apply a Hadamard to post-process. If we have that $f(0) \oplus f(1) = 0$ then we end up applying a Hadamard to $\frac{\pm(|0\rangle + |1\rangle)}{\sqrt{2}}$, which collapses to $\pm|0\rangle$ when measured. If we have that $f(0) \oplus f(1) = 1$ then we end up applying a Hadamard to $\frac{\pm(|0\rangle - |1\rangle)}{\sqrt{2}}$, which collapses to $\pm|1\rangle$ when measured. Thus, our quantum algorithm can determine, in one query, the value of $f(0) \oplus f(1)$. A classical algorithm would have to query the oracle twice to obtain the XOR of these values.

6 SIMON'S ALGORITHM

Simon's algorithm provides an efficient quantum solution to the following problem: given a function $f : \{0,1\}^n \to \{0,1\}^n$ (as a black box oracle) that is guaranteed to have the property that $f(x) = f(y)$ if and only if $x = y \oplus s$ for some fixed s, determine the value of s. A classical algorithm, even a randomized one, wouldn't be able to solve this problem much faster than by randomly guessing pairs of inputs, requiring exponential time.

Simon's quantum algorithm for solving this problem is illustrated in the circuit diagram below.

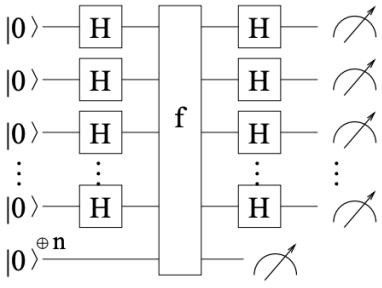

Fig. 1. From Aaronson's lecture notes – link here

We will query f using the CNOT-query defined before i.e. for two registers $|x\rangle|w\rangle$, the query gives $|x\rangle|w \oplus f(x)\rangle$. In the above circuit, the top n qubits (which we can think of as the first register) correspond to x and the bottom n qubits (which we can think of as the second register) correspond to w.

Before we feed the two registers into f, we apply the Hadamard operation to the first register to obtain $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0\rangle^{\otimes n}$.

After we feed this state into f, we obtain the following state: $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$. We then measure the second register, which collapses its state and leaves the contents of the first register as $\frac{|x\rangle + |y\rangle}{\sqrt{2}}$ (for some particular value x). Note that when we say "collapse," we refer to the fact that the act of measuring a qubit forces it out of superposition into some concrete state. Since the first and second registers are interlinked, the first register will contain a value that lines up with some particular $f(x)$ in the second register. Thus, we must express the state of the first register as a superposition still, but between $|x\rangle$ and $|y\rangle$ such that $x \oplus y = s$ (since they map to the same output).

Finally, we apply a Hadamard operator to the first register to obtain $\frac{H^{\oplus n}|x\rangle + H^{\oplus n}|y\rangle}{\sqrt{2}}$. By our previous formula for the Hadamard operator applied to multiple qubits, we obtain the sum

$$\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x \cdot z} |z\rangle + \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{y \cdot z} |z\rangle\right)$$

Where, as before $(\cdot)$ denotes the element-wise dot product $\mod 2$. The above sum simplifies to $\frac{1}{\sqrt{2^{n+1}}} \sum_{z \in \{0,1\}^n} ((-1)^{x \cdot z} + (-1)^{y \cdot z})|z\rangle$.

We then measure the result of the first register, which collapses to a particular z. In order for us to be able to measure something on the register, we will have that $x \cdot z = y \cdot z$ (or else the coefficient of $|z\rangle$ would be 0). This expression expands out to $x \cdot z = x \cdot (z \oplus s) \cdot z = (x \cdot z) \oplus (s \cdot z)$, which is true if and only if $s \cdot z = 0$. Thus, one round of Simon's algorithm gives us a linear equation about s. If we repeat this process a sufficient (linear) number of times, with high probability, we will obtain enough distinct linear equations to determine exactly the value of s (solving the linear equations can be done by Gaussian elimination, which can be done in $O(n^3)$ time). Since a classical algorithm would require exponentially many queries, Simon's algorithm is exponentially faster.

7 MOTIVATING WORK AND BACKGROUND

Scott Aaronson originally introduces the Forrelation distribution to provide an oracle separation of the relation versions of BQP and PH i.e. versions of these classes that deal with search problems as opposed to decision problems.

7.1 The Forrelation Distribution

The Forrelation distribution as used by Raz and Tal is defined as follows. Sample $x_1, x_2, \ldots, x_N$ from a normal distribution with mean 0 and variance 1 (where $N = 2^n$ and n is the size of inputs to the oracle). Let these values $x_1, \ldots, x_N$ be the components of the vector X. Define Y to be the Hadamard operator applied to X i.e. $Y = H_N X$. We then multiply each of the values in X and Y by ε (which Raz and Tal define to be $\frac{1}{24 \ln N}$). Now we take the values in X and Y and probabilisitcally round them to -1 and 1. Define probabilistic rounding as follows: for each entry z_i in the vector, round the entry to 1 with a probability of $\frac{1 + \text{trnc}(z_i)}{2}$ and to -1 with a probability

of $\frac{1-\text{trnc}(z_i)}{2}$, where $\text{trnc}(a) = \min(1, \max(-1, a))$ compresses values to the range $[-1, 1]$. Note that the expected value of each z'_i in our final result is $\text{trnc}(z_i)$. The resulting (X, Y) is what Raz and Tal call the distribution $\mathcal{D}$.

7.2 A Note on Promises and Black Boxes

On a more technical note, we should mention that Raz and Tal's result separates the promise version of BQP and PH. A promise version of a complexity class is one in which we restrict our inputs to come from a particular subset of all possible inputs. Here, the promise we are given is that the black-box (that outputs either -1 or 1) must be a uniform distribution or the Forrelation distribution defined above. Note that this still gives us an oracle O such that $\text{BQP}^O \not\subseteq \text{PH}^O$. This is because we can roll the enforcement of the promise into the functionality of the oracle. Note also that an oracle separation of BQP and PH does not necessarily imply that $\text{BQP} \not\subseteq \text{PH}$, although it does serve as promising evidence that this is the case.

7.3 The Connection Between AC^0 and PH

A result from the 80s [3] shows that a PH machine with access to an oracle O can be simulated by a polynomial size constant depth (AC^0) circuit with access to the same oracle (it get access to the oracle's truth table). This connection has to do with the fact that we can replace existential quantifiers with $\wedge$ gates and universal quantifiers with $\vee$ gates. Thus, if we prove a lower bound for constant-depth circuits with access to some oracle O, we have proved a lower bound for any PH machine with access to O.

8 THE MAIN RESULT

At a high level, Raz and Tal show that a quantum algorithm can distinguish the Forrelation distribution from the uniform distribution with an advantage that can be efficiently boosted such that we can distinguish $\mathcal{D}$ from the uniform distribution $\mathcal{U}_{2N}$. Furthermore, they show that no AC^0 circuit can distinguish these two distributions with an advantage that can be efficiently boosted. As mentioned before, AC^0 circuits given access to an oracle can simulate PH-machines that are given access to an oracle. It follows that no PH-machine is able to distinguish the two distributions in this black-box model.

9 THE QUANTUM ALGORITHM

The fundamental step in the quantum algorithm Raz and Tal propose is derived from a circuit for solving the Forrelation problem, which was presented in a paper by Aaronson and Ambianis. The circuit involves $n+1$ qubits – the top qubit is a control qubit and the bottom n qubits serve as inputs to the oracles. The oracles are queried as followed for an input $|i\rangle |z\rangle$ to the oracle, we output $|0\rangle X(z) |z\rangle$ if i is $|0\rangle$. We output $|1\rangle Y(z) |z\rangle$ if i is $|1\rangle$ and then apply a Hadamard operator to the bottom n qubits. After the oracle query, we apply a second Hadamard to the control qubit and measure it. We accept if the measurement gives a $|0\rangle$.

Note that one branch (when the control qubit is $|1\rangle$) has an extra Hadamard applied to the bottom n qubits after the oracle query whereas the other doesn't. It actually turns out that, by symmetry, it doesn't matter where this extra Hadamard is applied so long as it is applied at the end of one branch and not the other (Scott Aaronson mentioned this in an email exchange I had with him – this follows from the fact that the Hadamard is its own inverse).

Here's a less formal explanation of the intuition behind this quantum algorithm: We can think of $Y = H_N X$ (where H_N is the Hadamard matrix). If we apply a Hadamard to Y we get $H_N H_N X = X$ and the control qubit becomes unentangled from the rest of the qubits, so applying a final Hadamard to the control qubit have us measure a $|0\rangle$ with much higher probability depending on how Forrelated X and Y are. If X and Y are not Forrelated with one another, then the control qubit will have an equal chance of collapsing to a $|0\rangle$ or $|1\rangle$ when measured.

The probability of this algorithm accepting (i.e. measuring $|0\rangle$ on the control qubit) is $\frac{1+\varphi(X,Y)}{2}$ where $\varphi(X, Y) = \frac{1}{N} \sum_{i\in[N], j\in[N]} X_i \cdot H_{i,j} \cdot Y_j$ [1].

Raz and Tal show that if X and Y are drawn from a uniform distribution, the expected value of $\varphi(X, Y)$ is 0. If X and Y are drawn from $\mathcal{D}$, the expected value of $\varphi(X, Y)$ is $\geq \frac{\varepsilon}{2}$. This means that the quantum algorithm can distinguish the uniform distribution from the Forrelated distribution with advantage at least $\frac{\varepsilon}{2} = \Omega(\frac{1}{\log N})$, which can be efficiently amplified to allow this algorithm to distinguish in polynomial time with probability $1 - 2^{-\text{polylog}(N)}$. Thus, the problem of distinguishing $\mathcal{U}_{2N}$ from $\mathcal{D}$ is in Promise-BQP.

10 ESTABLISHING THE CIRCUIT LOWER BOUND

Raz and Tal's proof of the classical lower bound comprises (as is often the case with lower bounds) the heftier section of their paper. We consider a function $f : \{\pm 1\}^{2N} \to \pm 1$, computable by an AC^0 circuit. Raz and Tal consider the multilinear extension of f i.e. expressing it in the form $f(x) = \sum_{S\subseteq[2N]} \hat{f}(S) \cdot \prod_{i\in S} x_i$. Note that in this form we have $f(\mathbf{0}) = \hat{f}(\emptyset) = E_{x\sim\mathcal{U}_{2N}}[f(x)]$ i.e. if we are given an input drawn uniformly at random, the expected value of the function evaluated at that input is what we get when we evaluate the function at $\mathbf{0}$ (the 0-vector). This makes intuitive sense since our inputs are defined over ± 1 and if these inputs are drawn uniformly at random, their expected value is 0. This is the same as the coefficient for the empty product of variables (i.e. 1), hence $\hat{f}(\emptyset)$.

The goal is to show that $|E_{z\sim\mathcal{D}}[f(x)] - E_{x\sim\mathcal{U}_{2N}}[f(x)]| \leq \frac{\text{polylog}(N)}{\sqrt{N}}$.

Let $\mathcal{G}'$ denote (X, Y) before they had the trnc function applied to them and before they were probabilistically rounded. For the purpose of their analysis (Raz and Tal also formally justify this), we can consider z being drawn from this from $\mathcal{G}'$, a standard multivariate Gaussian distribution.

10.1 Bounded L_1-norms to Pseudorandomness

Much of the analysis hinges on a result Tal showed in 2017: that AC^0 circuits are approximated by polynomials with bounded L_1-norms. In his 2017 paper, Avishay Tal shows the following bound for the sums of the absolute values of the coefficients of the multillinear

extension: for all $k \in \mathbb{N}$, $\sum_{S \subseteq [2N]:|S|=k} |\hat{f}(S)| \leq (c \cdot \log s)^{(d-1)k}$, where the circuit has s gates and depth $\leq d$. Raz and Tal later use this result to establish a bound for the expression $|E_{z \sim \mathcal{G}'}[f(P \circ z)] - f(\mathbf{0})|$ where $P \in [-p, p]^{2N}, p \leq \frac{1}{2}$ and $\circ$ denotes an element-wise product.

Raz and Tal further develop this bound and use it to analyze the expression $E[f(\text{trnc}(z^{\leq(t)})) - f(\mathbf{0})]$, where they define $z^{\leq(i)} = p \cdot (z^{(1)} + \ldots + z^{(t)})$, where each $z^{(i)} \sim \mathcal{G}'$. Note that each $z^{\leq(i)}$ is a Gaussian because it is a sum of independently and identically distributed Gaussians. Furthermore, if we set $t = N$ and $p = \frac{1}{\sqrt{t}}$, we have that $z^{\leq(t)} \sim \mathcal{G}'$. Thus $E[f(\text{trnc}(z^{\leq(t)}))]$ starts to look very close to $E_{x \sim \mathcal{U}_{2N}}[f(x)]$. Indeed $|E[f(\text{trnc}(z^{\leq(t)})) - f(\mathbf{0})]|$ is equivalent to $|E_{z \sim \mathcal{D}}[f(z)] - f(\mathbf{0})|$, which gives a bound on $|E_{z \sim \mathcal{D}}[f(x)] - E_{x \sim \mathcal{U}_{2N}}[f(x)]|$.

Raz and Tal show by induction that for $i = 0, \ldots, t-1$,
$|E[f(\text{trnc}(z^{\leq(i+1)}))] - E[f(\text{trnc}(z^{\leq(i+1)}))]| \leq 12 \cdot \epsilon^2 \cdot (c \cdot \log s)^{2(d-1)} \cdot N^{-1/2} + 12 \cdot N^{-2}$. We can aggregate these differences to establish $|E_{z \sim \mathcal{D}}[f(z)] - f(\mathbf{0})| \leq \sum_{i=0}^{t-1} |E[f(\text{trnc}(z^{\leq(i+1)}))] - E[f(\text{trnc}(z^{\leq(i+1)}))]|$, which we find is less than or equal to $32\varepsilon \cdot (c \cdot \log s)^{2(d-1)} \cdot N^{-1/2}$, which is $\frac{\text{polylog}(N)}{\sqrt{N}}$, proving the main lower bound.

Let's take a moment to reflect on the cleverness of this problem solving technique – by breaking up a Gaussian distribution into a sum of Gaussian distributions, Raz and Tal develop this bound. They take an incremental bound, apply it to the differences of partial sums, and aggregate these differences for their overall bound. This instance provides one of those unique scenarios in which we must take an "out-of-the-way" path to reach our intended destination.

Indeed, multiple layers of the proof exhibit this quality. First we must consider the multi-linear extension of the polynomial the circuit computes. Then we analyze the polynomial's coefficients and determine a bound for the sums of the absolute values of these coefficients. We use this bound to develop a bound for the differences in partial sums of Gaussian random variables. Through aggregating these differences, we develop a bound for the entire sum, which corresponds exactly to the original bound we were trying to analyze.

Another remarkable aspect of this paper is its uniquely collaborative nature. The authors collaborated with experts in quantum computing, e.g. Scott Aaronson, and built off of their results, e.g. Forrelation, to obtain the quantum part of their main result. They also built off of previous results in pseudorandom generators and Fourier analysis of AC^0 circuits to obtain their classical lower bound. This shows how much of research isn't about simply generating ideas on one's own, but rather rehashing the ideas of others, combining them, modifying them, and using them in ways that allow researchers to further push the boundaries of knowledge. In this way, research, even technical research, is a human and social practice.

This presentation of Raz and Tal's proof omits many technical details, but if you are interested in seeing these details, please look at the original paper linked in the references. Hopefully this brief summary can serve as a guide to be able to understand these details more deeply.

10.2 Intuitive Interpretations

In his blog post on this result [6], Scott Aaronson provides a very nice intuition that sums up the difference between the kinds of polynomials that AC^0 circuits can compute versus those computable by quantum algorithms. With regards to the bound on the L_1-norms, he says that AC^0 circuits lead to "thin" polynomials that must compute total Boolean functions while quantum algorithms lead to "thick" polynomials that "must be indecisive on many inputs." This fundamental difference between constant-depth circuits of polynomial size and quantum algorithms intuitively explains why quantum algorithms cannot be represented by AC^0 circuits.

11 IMPLICATIONS AND FUTURE DIRECTIONS

As mentioned before, one can view this result as evidence for BQP not being a subset of PH (in a general non-black box sense). Lance Fortnow mentioned on his blog that an interesting direction to consider would be seeing if it is possible to have P = NP = PH but P ≠ BQP [4]. This is asking: In a strange world in which the entire polynomial hierarchy collapses to P, will there be problems (in a general non-black-box sense) that will be easy for quantum computers but hard for classical computers? There is still much left to explore in the realm of quantum complexity and how it relates to the complexity of classical computation.

12 ACKNOWLEDGEMENTS

I would like to thank Li-Yang Tan, Avishay Tal, and Scott Aaronson for providing elaborate respones to my emails and helping me understand the papers I've discussed.

REFERENCES

[1] Raz, Tal 2018: https://eccc.weizmann.ac.il/report/2018/107/
[2] Aaronson, Ambianis 2015: https://www.scottaaronson.com/papers/for.pdf
[3] Furst, Saxe, Sipser 1984: https://wiki.epfl.ch/edicpublic/documents/Candidacy%20exam/Furst%20Saxe%20Sipser%20-%201984%20-%20Parity%20circuits%20and%20the%20polynomial-time%20hierarchy.pdf
[4] Lance Fortnow's Blog: https://blog.computationalcomplexity.org/2018/06/bqp-not-in-polynomial-time-hierarchy-in.html
[5] Tal 17: https://pdfs.semanticscholar.org/be3c/269fe212b0e1d8f20c3e54e2692ef58276df.pdf
[6] Aaronson's blog post: https://www.scottaaronson.com/blog/?p=3827.
[7] *Computational Complexity: A Modern Approach* by Boaz Borak and Sanjeev Arora