# Lower Bounds on the Classical Simulation of Quantum Circuits for Quantum Supremacy

by

Alexander M. Dalzell

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017



Author .....
Department of Physics
May 12, 2017

Certified by .....
Aram W. Harrow
Assistant Professor
Thesis Supervisor

Accepted by .....
Professor Nergis Malvalvala
Senior Thesis Coordinator, Department of Physics

# Lower Bounds on the Classical Simulation of Quantum Circuits for Quantum Supremacy

by

Alexander M. Dalzell

Submitted to the Department of Physics
on May 12, 2017, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Physics

## Abstract

Despite continued experimental progress, no task has yet been performed on quantum technology that could not also have been performed quickly on today's classical computers. One proposed path toward achieving this milestone, which is often referred to as *quantum supremacy*, is to perform specific types of quantum circuits for which it is guaranteed, under plausible complexity theoretic conjectures, that any classical approximate weak simulation algorithm for these circuits must take more than polynomial time. Instantaneous quantum (IQP) circuits and Quantum Approximate Optimization Algorithm (QAOA) circuits are examples of circuits with this guarantee under the assumption that the polynomial hierarchy (PH) does not collapse. However, these arguments do not communicate how large these quantum circuits must be built before simulating them is hard in practice. We show how a fine-grained version of this assumption involving the PH leads to a fine-grained lower bound on the simulation time for IQP and QAOA circuits. Using the lower bound, we conclude that IQP circuits must contain roughly 1700 qubits, and QAOA circuits must contain roughly 7100 qubits before their simulation would be guaranteed to be intractable on today's fastest supercomputers. Additionally, we apply the same logic to find an asymptotic lower bound on the classical weak simulation of Clifford + $T$ circuits with $n$ qubits, $m$ Clifford gates, and $t$ $T$ gates, concluding that any simulation with runtime of the form $\text{poly}(n, m)2^{\gamma t}$ must have $\gamma > 1/135 \approx 0.0074$. The best existing algorithm of this form has $\gamma \approx 0.228$.

Thesis Supervisor: Aram W. Harrow
Title: Assistant Professor

# Acknowledgments

This work has been the product of significant contributions from many different people. First and foremost, I gratefully acknowledge the indispensable help of my thesis supervisor Aram Harrow for proposing this project one year ago, for patiently helping me learn the complexity theory and quantum supremacy background I needed to make progress, and for assisting me to work through the details of the result when I got stuck.

Additionally, Rolando La Placa has been a key contributor to this project at every step, assisting with brainstorming, background research, and the calculations themselves. His insight and advice when I presented my ideas, confusions, and frustrations allowed this project to progress much more quickly than otherwise possible. In particular, I thank him for the suggestion to apply our method to find a lower bound not only for IQP and QAOA circuits but also for Clifford + $T$ circuits, leading to the results in Chapter 5. This portion was completed with the assistance of Ryuji Takagi in locating the optimal $T$ gate implementation of $CCZ$ and $CCCZ$ gates. Another important technical contribution belongs to Ashley Montanaro; our lower bounds are much simpler (and tighter) due to his important suggestion to replace the problem MAJ-SAT with MAJ-ZEROS in our analysis.

Finally, I am grateful to Dax Koh for his help this past semester fleshing out the final ideas that went into this thesis, including the details of the QAOA and Clifford + $T$ circuit constructions for the MAJ-ZEROS problem. Additionally, Dax's feedback on early drafts of this thesis has been invaluable for improving its clarity and correctness.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

The central motivation for quantum computation is the expectation that quantum computers, when they are built, will provide an advantage over today's classical computers. However, nearly two decades after the first quantum algorithms were implemented experimentally [11, 26], no clear experimental demonstration of such an advantage has occurred; every computation performed thus far on quantum hardware has been reproducible by classical computers in a reasonable amount of time.

Theoretically speaking, there are many reasons to believe that this advantage will exist. Feynman [19] first argued that quantum computers would be advantageous for simulating quantum systems since classical simulations would in general require resources that grow exponentially with the size of the system — $2^n$ complex numbers are generally needed to fully describe an $n$-qubit quantum state. Later, Shor famously gave a quantum algorithm to factor integers in polynomial time [33], a feat that no known classical algorithm can match. Other algorithms likewise purport to eclipse the abilities of classical computers, but in all these cases, the existence of the quantum advantage has not been proven, but rather rests on conjectures that range in their degree of credibility.[1]

[1]Provable quantum advantages exist when the resource being measured is something other than the algorithm's runtime, such as oracle queries (e.g., Grover's search algorithm [22]) or circuit depth (see, e.g., [8]).

Thus, demonstrating the practical advantage of quantum computers — achieving *quantum supremacy* [31] — involves both theoretical and experimental elements. Quantum supremacy requires that a task be performed experimentally on a quantum computer that is classically intractable using known methods, but also whose classical intractability is entailed more generally by widely-believed theoretical conjectures. In many instances, suggested quantum supremacy tasks have no direct practical use, but achieving quantum supremacy is still important both for practical and theoretical reasons. Practically, it is an important milestone on the path to producing quantum devices that do perform useful computations not possible on a classical computer. Theoretically, it is important for proving that quantum mechanics endows physical systems with fundamentally greater computational power than they are classically thought to possess, contradicting the extended Church-Turing thesis, which claims that our notion of classical computation fully captures what is possible on physical hardware.

Recent experimental advances make achieving quantum supremacy a reasonable goal for quantum computing in the next decade, but further theoretical advancement is needed to understand which tasks should be chosen for implementation on near-term quantum hardware as well as to determine definitively when quantum supremacy has been achieved. Existing quantum supremacy arguments conclude that certain tasks will be hard for classical computers in a coarse asymptotic sense, but leave open the important practical question: how large must our quantum devices be before it is *actually* hard to reproduce their results on our current best classical technology? For some tasks, actually realizing the quantum advantage necessitates much larger quantum devices than what is expected to be achievable in the near future. For example, Shor's algorithm for factoring is a possible avenue toward quantum supremacy. It is a generally believed conjecture that no polynomial-time classical algorithm for factoring exists, so there is theoretical basis for a quantum advantage. However, practically speaking, classical computers are able to factor numbers with hundreds of bits [28], meaning a quantum supremacy demonstration using Shor's algorithm would require at least as many qubits, and likely many times more due to the need for qubit-

consuming error-correcting codes and fault-tolerant computation [2]. In comparison, current proposed quantum technology aims to operate on only tens of qubits [6], so demonstrating quantum supremacy this way is not likely to be feasible in the near future.

Other quantum supremacy arguments [2, 9, 18] assert that certain classes of quantum circuits cannot be classically simulated in polynomial time, but make no statement about how large the simulation time actually is. Thus, it is hard to assess if these tasks are, like factoring integers, difficult for classical computers in theory but not in practice, at least compared to the abilities of current quantum technology.

Our work begins to fill this void. Building on existing quantum supremacy arguments, we impose a more specific conjecture and arrive at a more specific conclusion about classical hardness: where previous arguments show that (under certain believable assumptions) "any classical simulation of certain quantum circuits must take more than polynomial time," we show that (under more refined but still believable assumptions) "any classical simulation of certain quantum circuits must take more than $s(q)$ time" where $s$ is an explicit exponential function in the number of qubits $q$. Thus, we find lower bounds on the classical simulation time for quantum-supremacy-related circuit simulation tasks, which allow us to determine how many qubits such a circuit must contain before simulating it classically is guaranteed to be intractable. This information is vital to determine when quantum supremacy has actually been achieved, and in the meantime allows us to determine which quantum supremacy proposals generate the most direct path to success.

## 1.2 Previous work

### 1.2.1 Quantum supremacy by quantum circuit simulation

Several quantum supremacy tasks have been proposed. As mentioned, known algorithms with exponential quantum speedups, such as Shor's algorithm [33], are one obvious avenue toward quantum supremacy, but, in the case of Shor's algorithm,

there are both practical and theoretical drawbacks. Practically, factoring integers on a quantum computer likely requires the full power of universal quantum computing — the ability to perform arbitrary fault-tolerant quantum gates — making the task more challenging experimentally. Moreover, theoretically, while it is a generally accepted conjecture that factoring integers cannot be done efficiently on a classical computer, the failure of this conjecture would not entail a revolutionary change in the way computer scientists understand computation [2]. As we will see, the conjecture underpinning many of the other quantum supremacy proposals has a significantly stronger theoretical basis.

To circumvent these difficulties, other tasks have been proposed that are more natural for quantum systems to perform and that do not require the full power of universal fault-tolerant quantum computation. Building on the idea that quantum computers should be difficult to simulate classically, it is more natural to choose a task that is explicitly related to simulating quantum systems. When we perform a quantum algorithm in the form of a quantum circuit, we obtain a random outcome drawn from some probability distribution. Sampling from this probability distribution is easy for a quantum computer, almost by definition, but if we believe simulating quantum circuits classically is hard, then doing the same with a classical computer should be difficult. As we will see, the difficulty of performing this task has a solid theoretical basis. Thus, we might be able to use the task of sampling from the probability distributions associated with quantum circuits to demonstrate quantum supremacy.

In certain instances, we can restrict these quantum circuits and still have strong guarantees about the hardness of classical simulation. The advantage here is that these restricted classes of quantum circuits might be easier to implement experimentally than general quantum circuits. Unlike factoring, sampling from these probability distributions often has no direct utility; the reason to perform these tasks experimentally lies purely in the fact that doing so would demonstrate quantum supremacy. In hindsight, this sort of sampling task makes sense: if our goal is to showcase the power of quantum computing, and not necessarily to do something useful, it will be easiest

to use tasks which are natural for quantum computers, like sampling from quantum circuit distributions.

It is important to note that these results still hold for *approximate* sampling, when the approximation is understood in the multiplicative sense. Additive approximate sampling requires additional assumptions and is not considered in our work.

The conjectures used to underpin these results are expressed in the language of complexity theory, the topic of Chapter 3. Complexity theorists have categorized problems into different complexity classes and proved relationships between those classes. For example, it is widely believed, but unproven, that the classes P and NP are not equal. Conjecturing that $\mathrm{P} \neq \mathrm{NP}$ allows other statements to be proven, including that no classical algorithm can efficiently compute the probabilities in the probability distribution associated with an arbitrary quantum circuit [18].

A stronger, but related conjecture states that the polynomial hierarchy (PH) does not collapse. The PH is a class made up of an infinite number of levels, and to say that the PH does not collapse means that each of the levels is not equal to the level above it. The non-collapse of the polynomial hierarchy is also widely believed, and implies that no classical algorithm can produce samples from the probability distribution associated with an arbitrary quantum circuit [9]. The non-collapse of the PH in this instance can be thought of intuitively as the statement that exactly counting solutions to certain computational problems is hard even when given the power to approximately count the number of solutions easily. It is important to emphasize that this conjecture is purely classical and bears little relation at first glance to quantum computation.

### 1.2.2 Specific quantum supremacy proposals

Recent work has used the basic reasoning discussed in the previous subsection to present proposals for quantum supremacy involving restricted classes of quantum circuits.

(1) Bremner, Josza, and Shepherd [9] considered instantaneous quantum (IQP) circuits, where all of the internal quantum gates commute, and showed how the non-collapse of the PH implies that the probability distribution of such circuits cannot be sampled from efficiently on a classical computer. This also holds when the sampling is only required to be approximate in the multiplicative sense.

(2) Aaronson and Arkhipov [2] showed how similar statements hold for linear optical systems, calling the associated computational task BosonSampling.

(3) Farhi and Harrow [18] proved the same result for multiplicative approximate simulation of Quantum Approximate Optimization Algorithm (QAOA) circuits, which have the advantage that they might have practical application in solving optimization problems.

(4) The Quantum-AI group at Google has suggested (and plans to experimentally implement) [6] a quantum supremacy experiment using random circuits on a 2D lattice of roughly 50 qubits. The theoretical basis for classical hardness lies in the conjecture QUATH [3], which is related but not the same as the argument involving the polynomial hierarchy.

For IQP circuits [10] and BosonSampling [2], it can also be shown that approximate simulation is hard in the additive sense, if additional conjectures are imposed. Additive approximations are generally advantageous because they might be more easily implemented experimentally; however, our work does not consider the additive case.

In Chapter 4, we consider IQP and QAOA circuits and show how a more specific lower bound on the classical simulation time can be derived if a stronger fine-grained assumption is used. It would likely be possible to extend our basic framework to the other quantum supremacy proposals as well.

### 1.2.3 Classical simulation algorithms

The theoretical arguments behind quantum supremacy aim to prove what classical algorithms cannot do. It is also interesting and important to understand what they can do. What kinds of circuits can be simulated efficiently, and how inefficient are the best general simulation algorithms?

It is productive to consider the Clifford + $T$ framework for this discussion. Clifford circuits are quantum circuits containing only Hadamard ($H$), phase ($S$), or CNOT ($CX$) gates. The Gottesman-Knill theorem states that these circuits can be simulated efficiently on a classical computer [20]. On the other hand, if we allow a circuit to use Clifford gates as well as $\pi/8$-gates ($T$), then we can perform any quantum computation we wish. Therefore, if simulating general quantum circuits is difficult, the difficulty in a sense must originate from the $T$ gates. Work has been done to find fast algorithms to simulate Clifford + $T$ gates. The fastest known algorithm is due to Bravyi and Gosset [7], which simulates a circuit on $n$ qubits with $t$ $T$ and $T^\dagger$ gates and $m$ total gates in time $\text{poly}(n, m)2^{0.228t}$. Thus, the scaling is exponential, but the exponentiality has been isolated to the total number of $T$ gates.

## 1.3 Summary of results

Chapter 2 and Chapter 3 introduce the necessary background for quantum circuits and complexity theory to understand our results. Chapter 4 and Chapter 5 contain our original results.

The main result of Chapter 4 is expressed by Corollary 4.3.7.1. We assume Conjecture 4.3.1, which claims an explicit exponential separation between the third level of the PH (but generalized to allow for greater than polynomial time) and the complexity class PP. We make arguments that this conjecture is believable and motivate it using an oracle separation. This is a fine-grained version of the conjecture that the PH does not collapse.

We show how this allows us to prove that if a classical algorithm can approximately weakly simulate IQP or QAOA circuits (up to multiplicative factor $\epsilon$) with $a$ gates

over $b$ qubits in time $a \cdot s(b)$ for some simulation-time function $s$, then $s$ must satisfy

$$c(g \cdot nL \cdot s(q)) > \frac{2^{n/5}}{92} - \frac{n}{8} \tag{1.1}$$

where $g$, $q$, and $L$ are given by

$$g = 4n + 13; \;\; q = n + 4; \;\; L = 50 \log(3n)/(2 - 7\epsilon)^2 \tag{1.2}$$

in the case of IQP circuits and

$$g = 29n + 48; \;\; q = 4n + 6; \;\; L = 50 \log(3n)/(2 - 7\epsilon)^2 \tag{1.3}$$

in the case of QAOA circuits and the function $c$ is given by

$$c(x) = \log(kx)(kx)\Big(4(kx)^2 + 6(kx) + 9\Big) + \log(kx)^2 \tag{1.4}$$

where $k = \max(16, 4 \log(8x)/3)$.

The reasoning that leads to this result can be explained intuitively as a connection between simulation of quantum circuits and a purely classical algorithm. If a simulation algorithm were to exist that breaks the bound in Eq. (1.1), then we could use this simulation as a subroutine to solve a purely classical computational problem using fewer resources than we expect to be possible, breaking a believable conjecture.

The algebra in these expressions might obscure the key elements of this result. Notice that $c(x)$ scales approximately like $x^3$ (ignoring the $\log(x)$ factors), and $g$, $q$, and $L$ are polynomial in $n$, so Eq. (1.1) gives an implicit exponential lower bound on the function $s(q)$. If we look solely at the exponential dependence of this lower bound, we can see it roughly reads $s(b) > 2^{b/15}$. For a specific value of $b$, we can numerically solve for the lower bound on $s(b)$ exactly, corresponding to the minimum amount of time it must take to simulate each additional gate of the circuit with a classical algorithm.

Let's assume we have a classical computer that makes $10^{17}$ operations per second, in line with state-of-the-art classical supercomputers. Using the lower bound on $s(b)$, we can determine the number of qubits $b$ such that simulating each additional gate takes at least a day. For IQP circuits this calculation yields roughly 1700 qubits and for QAOA circuits it yields roughly 7100 qubits. Since we're dealing with a lower bound, tighter lower bounds could reduce the number of qubits we calculate, but our calculation provides an important preliminary piece of information about generally how large our quantum systems must be to be used for a quantum supremacy demonstration.

In Chapter 5, we make a similar argument to arrive at a lower bound on classical simulation algorithms for Clifford + $T$ circuits in Corollary 5.2.3.3. We assume the simulation time depends on $n$, $m$, and $t$ like $\text{poly}(n, m)2^{\gamma t}$ as in Bravyi and Gosset's algorithm [7], and, under the same fine-grained complexity-theoretic conjecture as before, we conclude that $\gamma > 1/135$. This is not tight compared to the best known algorithm where $\gamma = 0.228$, but it is a preliminary lower bound that limits how much these simulation algorithms can be improved.

# Chapter 2

# Quantum circuits

The extended Church-Turing thesis states that any physically reasonable implementation of an "algorithm" can be simulated efficiently on a Turing machine. The importance of this statement is that it defines a precise mathematical model — the Turing machine — that fully captures classical computation. Within the model, we can rigorously prove which tasks can and cannot be performed, and classify different tasks based on how much of a certain computational resource (e.g., time) they require. This is the topic of complexity theory and will be discussed in more detail in Chapter 3.

For quantum computation, we need a model to replace the Turing machine that fully captures what is physically possible before we can make any mathematical progress in understanding the resources different tasks require on a quantum computer. Several models have been analyzed, including the quantum circuit model [30, 14], adiabatic quantum computation [17], the quantum Turing machine [13], and measurement-based quantum computation [21]. All of these models have been shown to be equivalent in the sense that if one model can solve a task $\mathcal{T}$ efficiently, all of the other models can also solve $\mathcal{T}$ efficiently. Since these are thought to be physically implementable models for an "algorithm," albeit a *quantum* algorithm,[1] finding a computational task which can be performed efficiently in one of the mentioned

[1] After all, we do believe the physical world obeys the rules of quantum mechanics.

quantum models but not in the Turing machine model would refute the extended Church-Turing thesis. It is widely believed, yet unproven, that such a task exists.

In this work, we analyze the quantum circuit model, which is one of the most widely-used models for quantum computation. In short, the quantum circuit model defines a quantum algorithm as a sequence of *quantum gates* acting on a set of quantum bits, called *qubits*, which store the quantum data. Thus, the model makes a strong analogy to a classical circuit, where data is stored in classical bits which are acted upon by logic gates to perform the computation.

## 2.1 General quantum circuits

First, we will define the most general version of the quantum circuit model. The fundamental building block of a quantum circuit is a *qubit*. A qubit is simply any two-level quantum system, the state of which lies in the Hilbert space spanned by the states $|0\rangle$ and $|1\rangle$. Whereas a classical bit takes a value of either 0 or 1, a quantum bit will generally exist in a superposition of the two states: $\alpha|0\rangle + \beta|1\rangle$ for complex numbers $\alpha$ and $\beta$ such that $|\alpha|^2 + |\beta|^2 = 1$. When we have $n$ qubits, the state of the system is a superposition $\sum_{x=0}^{2^n-1} \alpha_x |x\rangle$ over the $2^n$ basis states corresponding to the $2^n$ possible $n$-bit strings. We will require in general that the state of a circuit on $n$ qubits begin in the $|0\rangle^{\otimes n}$ state.

The next element of a quantum circuit is the sequence of *quantum gates* that act on the qubits. Quantum gates are similar to classical gates like AND, OR, and NOT gates in that they mutate the data stored in certain bits and cause interaction between data on different bits, but there are several notable differences. To preserve the normalization of the state and respect the linearity of quantum mechanics, quantum gates must be represented by unitary matrices acting on the vector of amplitudes $\alpha_x$ corresponding to the state of the system. Thus, quantum gates must be reversible, meaning irreversible classical gates such as AND and OR do not carry over naturally into quantum computation. In general, all unitary gates are valid for quantum computation, and thus performing a certain task on a quantum computer is reduced to

finding a sequence of gates which creates the desired output. Typically, though, algorithms are restricted to using gates that act non-trivially on only a constant number of the qubits (that is, a number that does not depend on $n$), since these are most straightforward to implement experimentally.

The final element of a quantum circuit is the measurement, which produces the output of the circuit. For our purposes, we will always consider a measurement in the computational basis, meaning that if we measure $m$ qubits, we receive an outcome representing a bit string on those $m$ qubits, and the system collapses onto the corresponding projection of the Hilbert space. If the state of the system prior to measurement is $|\psi\rangle$, the probability of observing outcome $x$ will be the quantity $\langle\psi| P_x |\psi\rangle$ where $P_x$ is the projection operator onto the subspace corresponding to the measured $m$ bits taking the outcome $x$. In Figure 2-1 we see a drawing of an example of a quantum circuit.

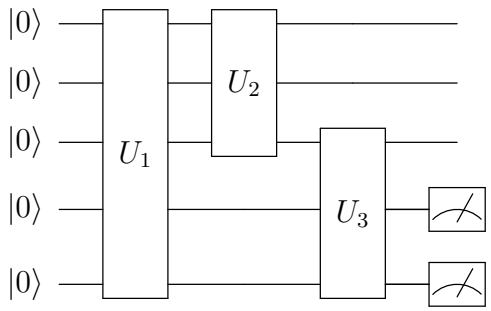


Figure 2-1: Example of a general quantum circuit with three gates and five qubits, where the last two qubits are measured. The gates $U_1$, $U_2$, and $U_3$ are unitary operations acting on a subset of the five qubits.

Let us pause to define a few notable gates. Important single-qubit gates include the Pauli gates

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.1}$$

In addition to the Pauli gates, we define the Hadamard gate $H$ and the phase gate $S$, as follows.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} ; \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{2.2}$$

The most basic interacting two-qubit gate is the CNOT gate, which flips the second qubit only when the first qubit is in the $|1\rangle$ state.

$$\text{CNOT} \equiv CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.3}$$

Adding a subscript to a gate specifies which qubit it acts on. For the CNOT gate $CX_{ij}$ indicates qubit $i$ is the control and qubit $j$ is the target of the bit flip. The idea of a controlled operation extends more generally. For a unitary $U$ acting on $m$ qubits, the controlled-$U$ operation $CU$ acts on $m+1$ qubits, applying $U$ to the final $m$ qubits only for states whose first qubit is $|1\rangle$.

The set $\{H_i, S_i, CX_{ij}\}$ for all qubits $i, j$ generates the *Clifford group* of unitary operations, which will be discussed later.

The sequence of gates we choose to implement will depend on the input to the task we are solving. If a quantum circuit has $g$ total gates, we will require that the description of the circuit — the number of qubits $n$, the sequence of gates, and the qubit or qubits to be measured — be computable classically in time which scales like a polynomial in $g$. Without this condition, we could solve hard classical problems with efficient quantum circuits by first using exponential classical time to compute the answer, and then hardwiring the answer into a polynomial-sized quantum circuit.

For each quantum circuit, there corresponds a probability distribution over the possible measurement outcomes, and each time we perform the quantum circuit, we obtain one sample from this probability distribution.

As defined, implementing a quantum circuit requires the ability to perform any unitary gate. In the next subsection, we will see that in practice it is not necessary

to have this power; we can approximate any unitary gate with gates from a limited, simple set.

## 2.2 Universal gate sets and compilation

For classical circuits, there exist universal gate sets such that any Boolean function can be computed using only gates drawn from the set. For example, the NAND gate by itself forms a universal gate set, since any Boolean function can be computed using only NAND gates. In quantum computing the idea is similar: a *universal gate set* is a set of gates such that any unitary operation can be approximated to arbitrary accuracy using only gates from the set. Thus, instead of needing to be able to implement any unitary operation, we only need to be able to implement gates from the universal set to be able to perform the computation arbitrarily accurately. This is important, since it would be an experimental nightmare if quantum computation really required the ability to perform every possible unitary gate. In experiment, it is especially hard to perform gates in which many qubits interact. Luckily, interacting gates can be built out of single qubit gates, and the simple $CX_{ij}$ gate introduced in the previous section.

**Theorem 2.2.1.** *All single-qubit gates along with CNOT gates between any two qubits form a universal gate set.*

**Theorem 2.2.2.** *Let*

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{2.4}$$

*Then $\{H, T\}$ is a universal set for all single-qubit unitary gates.*

*Proof.* The proofs of Theorem 2.2.1 and Theorem 2.2.2 appear in *Quantum Computation and Quantum Information* by Nielsen and Chuang [30]. □

Thus, the set

$$\{H_i, T_i, CX_{ij} \text{ for all } i \text{ and pairs } (i, j)\}$$

forms a universal gate set for all unitary gates.

The process of turning a unitary gate into a string of gates taken from a universal gate set is sometimes referred to as *compilation*. The existence of universal gate sets means compilation is possible, but it does not mean that the compiling can be done using a short string of gates. Naively, we expect the length of the approximating string to scale like $\text{poly}(1/\epsilon)$, where $\epsilon$ is the error in the approximation. Luckily, we can do exponentially better. The Solovay-Kitaev algorithm [12] shows how a unitary gate can be compiled into a string of only $\text{poly}(\log(1/\epsilon))$ gates from the universal set.

## 2.3 Restricted models of quantum circuits

In the previous sections, we discussed general quantum circuits. In this section, we will discuss specific classes of quantum circuits which have been restricted in some way. These restrictions have a debilitating effect on the power of the model — none of the models we mention are believed to be equivalent to the general quantum circuit model. Yet, each of the models is believed to retain some advantages over classical computation in the sense that classical computers cannot simulate them efficiently. They are interesting for several reasons. One reason is philosophical: assessing the power of models after we impose certain restrictions allows us to probe the boundary between quantum and classical and identify the key features that give quantum computation its power. A second reason is practical: it has proved difficult to experimentally create universal quantum computers which can implement general circuits. Perhaps it would be more feasible to implement specific subclasses of circuits on a real experimental device. Indeed, experiment is one motivation for the definitions of some of the models to follow.

### 2.3.1 Instantaneous quantum circuits (IQP)

The first restricted model we will consider is that of instantaneous quantum computations [32, 9]. Instantaneous quantum circuits with a polynomial number of gates and qubits, a notion we will make more precise in Chapter 3, will generate the In-

stantaneous Quantum Polytime (IQP) complexity class, so sometimes we will refer to the circuits themselves as IQP circuits. There are multiple ways to define this model; we will do so such that it fits well into the general circuit model we introduced above.

An instantaneous quantum circuit is a circuit where a Hadamard gate is applied to each qubit at the beginning and end of the computation, but the rest of the gates are diagonal gates. Each qubit begins in the $|0\rangle$ state but is immediately sent to the $|+\rangle = H|0\rangle = 1/\sqrt{2}|0\rangle + 1/\sqrt{2}|1\rangle$ state under the Hadamard operation, and each qubit is measured at the end of the computation in the computational basis. Thus, it would be equivalent to say that all the gates are diagonal, but each qubit begins in the $|+\rangle$ state and is measured in the $\{|+\rangle = H|0\rangle, |-\rangle = H|1\rangle\}$ basis. All of the internal diagonal gates commute, and therefore can be implemented in any order. In order for the model to be physically reasonable, we might need to impose a further restriction that each diagonal gate involve only a small constant number of qubits.

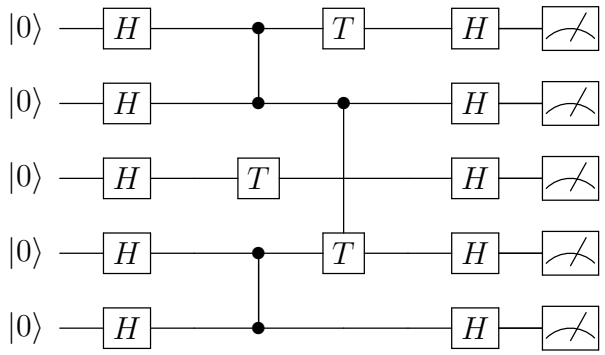


Figure 2-2: Example of an instantaneous quantum circuit. Each qubit must begin and end with a Hadamard gate, and all internal gates must be diagonal in the $Z$ basis. The vertical lines indicate controlled operations. Thus, in our example, the second qubit first undergoes a controlled-Z operation (CZ) with the first qubit, applying a minus sign only if both bits are in the $|1\rangle$ state, then acts as the control on a controlled-$T$ operation with the fourth qubit: the $T$ gate is applied only when the second qubit is in the $|1\rangle$ state.

We call these instantaneous circuits because, if we were to implement them by applying a constant Hamiltonian, the fact that the gates commute would allow us to perform all of the gates simultaneously simply by adding the separate Hamiltonians. This observation also makes it clear why we might want to restrict ourselves to local

gates; local gates correspond to local Hamiltonians which are simpler to implement experimentally.

### 2.3.2 Quantum approximate optimization algorithm (QAOA) circuits

The second model we will define, quantum approximate optimization algorithm (QAOA) circuits [16, 18], has some similarities with instantaneous quantum circuits. In fact, in a sense, QAOA can be thought of as multiple rounds of instantaneous operations.

A QAOA circuit is a circuit of the following form. Let the circuit consist of $n$ qubits, which begin in the $|0\rangle$ state but are immediately hit with a Hadamard gate, as in the instantaneous quantum circuit model. An integer $p$, and angles $\gamma_i, \beta_i$ for $i = 1, 2, \ldots, p$ are chosen. A diagonal Hamiltonian $C$ is specified such that $C = \sum_\alpha C_\alpha$ where each $C_\alpha$ is a constraint on a small subset of the bits, meaning for any bit string $z$, either $C_\alpha |z\rangle = 0$ or $C_\alpha |z\rangle = |z\rangle$ and $C_\alpha$ can be expressed as a function of some small constant number of the Pauli operators $Z_j$. We define the Hamiltonian $B = \sum_{j=1}^{n} X_j$, and $U(H, \theta) = \exp(-iH\theta)$. The remainder of the circuit consists of applying $U(C, \gamma_1)$, $U(B, \beta_1)$, $U(C, \gamma_2)$, $U(B, \beta_2)$, etc. for a total of $2p$ gates. Finally the qubits are measured in the computational basis.

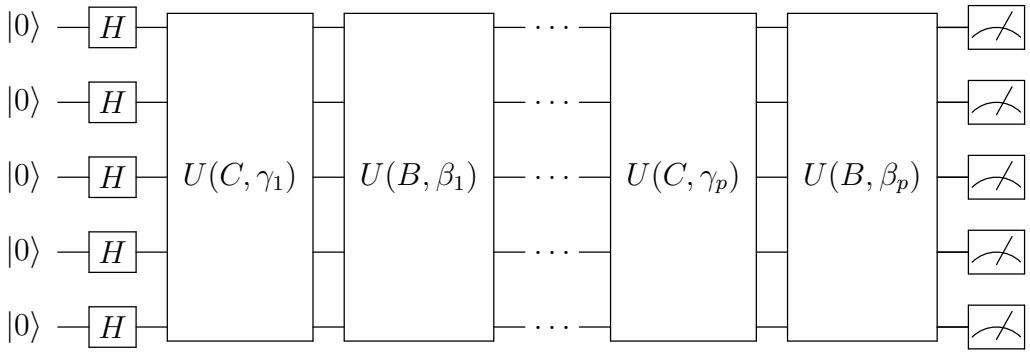


Figure 2-3: Example of a QAOA circuit. Each qubit begins with a Hadamard gate, and then $2p$ gates are performed alternating between applying Hamiltonian $C$ and applying Hamiltonian $B$.

Since $U(C, \gamma_j) = \prod_\alpha U(C_\alpha, \gamma_j)$, the gate $U(C, \gamma_j)$ can be performed as a sequence of commuting gates that perform the unitaries associated with the constraints $C_\alpha$. Thus each $U(C, \gamma_j)$ could form the internal portion of an instantaneous quantum circuit.

Since the operator $C$ is a sum of many constraints, it takes the form of a common type of computational problem, called a constraint satisfaction problem. For all bit strings $z$, $C|z\rangle = \lambda_z |z\rangle$, and a common and important problem asks us to find the maximum value of $\lambda_z$. There is evidence that QAOA circuits might be able to approximate this optimum value of $\lambda_z$ more efficiently than classical algorithms when $p > 1$ [16], so in comparison to instantaneous circuits, QAOA circuits might have more practical value.

## 2.4 Classical simulation of quantum circuits

### 2.4.1 Feynman's intuition

Why should a computer based on quantum mechanics be better than one based on classical physics? Because the world is fundamentally quantum mechanical! If you want to simulate quantum systems on a computer, it's natural to think you'll have more success if your computer is quantum. This was Richard Feynman's original intuition that helped birth the field of quantum computation in 1982 [19].

Using the quantum circuit model introduced above, we can understand this intuition more explicitly. The dimension of the Hilbert space for $n$ qubits is $2^n$, meaning we need an exponential number of complex amplitudes to describe the quantum state. On the other hand, describing a classical $n$-bit state, by definition, requires just $n$ binary numbers.

Thus, simulating a quantum circuit on a classical computer appears that it should require exponential time in general. The naive simulation algorithm operates by storing the $2^n$ amplitudes and updating them after each gate is applied.

However, how can we be sure that there isn't a clever way to perform this simulation efficiently? Three and a half decades after Feynman formed this intuition, we still can't prove that no clever algorithm exists, but we do have formal evidence based on plausible conjectures that this is the case, which will be discussed in Chapter 3.

### 2.4.2 Strong vs. weak simulation

The naive simulation algorithm we mentioned is an example of a *strong* simulation of a quantum circuit — it allows us to compute the exact probabilities that each measurement outcome occurs to arbitrary accuracy. But in a sense, a quantum computer can't even perform this calculation efficiently! In a quantum circuit, the exact value of the amplitudes are stored in the state, but they aren't accessible via measurement. Rather, when we measure the system, we obtain a sample from the probability distribution associated with the measurement. Using sampling alone, it would take exponential time to compute the exact outcome probabilities. Thus, if our goal is to simulate a quantum computer, we should not necessarily require that our algorithm satisfy such a strong notion of simulation.

In contrast, a system that can sample from the probability distribution associated with a quantum circuit is said to be *weakly* simulating the circuit. In this sense, quantum circuits naturally weakly simulate themselves, but they don't naturally strongly simulate themselves. Even under this weaker notion of simulation, we still have strong evidence that no classical algorithm can simulate general quantum circuits efficiently [18]. Additionally, we mention that a strong simulation with the ability to calculate both the joint probabilities and the marginal probabilities of the outcome probability distribution implies the existence of a weak simulation to sample from that probability distribution [37].

### 2.4.3 Approximate simulation

We can further notice that when our quantum circuit is restricted to using only $H$, $T$, and $CX$ gates, it cannot exactly perform any general quantum circuit; it can

only perform these circuits approximately. So a quantum computer using only these gates cannot obtain samples from the probability distribution associated with general quantum circuits, it can only obtain approximate samples. We conclude that when we wish to simulate quantum circuits with classical systems, what we should really wish for is the ability to approximately weakly simulate them.

There are two precise notions of approximate weak simulation. Suppose the true probability of measuring outcome $x$ with quantum circuit $\mathcal{C}$ over $n$ qubits is $P(x)$. And suppose our weak simulation algorithm produces the outcome $x$ with probability $Q(x)$. The approximate simulation is multiplicative with parameter $\epsilon$ if

$$|P(x) - Q(x)| \leq \epsilon P(x) \tag{2.5}$$

for all $x$. The approximate simulation is additive with parameter $\epsilon$ if

$$\sum_{x=0}^{2^n-1} |P(x) - Q(x)| \leq \epsilon \tag{2.6}$$

An $\epsilon$-multiplicative approximation is necessarily also an $\epsilon$-additive approximation since summing Eq. (2.5) over all strings $x$ yields Eq. (2.6). However, the converse is not true: an $\epsilon$-additive approximation is not necessarily also an $\epsilon$-multiplicative approximation.

### 2.4.4 Clifford circuits and the Gottesman-Knill theorem

Now, with a better understanding of the varying notions of simulation, are there methods to simulate quantum circuits that go beyond the naive algorithm? For certain classes of circuits, there are. Notably, a circuit that is composed entirely of gates from the Clifford group can be simulated efficiently classically, a fact known as the Gottesman-Knill theorem [20]. The Clifford group is generated by the set of $H$ and $S$ gates on each qubit along with the set of $CX$ gates between any two qubits. The Pauli gates are contained in the Clifford group (e.g., $Z = S^2$). In short, Clifford circuits can be simulated efficiently because the quantum state always remains in

a portion of Hilbert space which can be represented efficiently with classical states. The simulation algorithm works by keeping track of the state's *stabilizer*, the set of independent Pauli operators for which the state is an eigenvector. When a Clifford gate is applied to the circuit, the stabilizer changes but the rules for updating the stabilizer are simple and the update can be made efficiently. This is an example of strong simulation, because the exact probabilities of each measurement outcome can be calculated from the stabilizer at the end of the algorithm.

### 2.4.5 General simulation algorithms

In general, there is no known efficient algorithm for strongly or weakly simulating quantum circuits; however, there are algorithms that do better than the naive algorithm. One framework by which to search for algorithms is to consider Clifford + $T$ circuits. Clifford + $T$ circuits are universal for quantum computation, but the Gottesman-Knill theorem suggests that the Clifford part of the circuit should be easy to simulate, so the $T$ gates must be the source of the difficulty in simulation. If we have a circuit on $n$ qubits with $m$ total gates but only $t$ $T$ gates, we might expect algorithms to exist which simulate the circuit in time $\text{poly}(n, m)2^{\gamma t}$ where the exponential dependence lies entirely in the number of $t$ gates. Indeed, Bravyi and Gosset [7] gave an algorithm for which $\gamma \approx 0.228$.

# Chapter 3

# Complexity theory

## 3.1 The basics

Complexity theory is the field of mathematics and computer science that aims to classify computational tasks by their difficulty. Difficulty is measured by the amount of some computational resource, also known as the *complexity*, required to solve the computational problem. Examples of complexity are time complexity and space complexity: assuming a certain model for a computer such as a Turing machine, we can count the minimum number of time steps or the minimum number of bits of memory required by the model to solve a certain problem. Assuming our model of computation is physically reasonable, problems with larger complexity will require more physical resources to solve on real computers. Thus, complexity theory has actual practical uses in addition to allowing us to understand fundamental mathematical facts.

The following sections establish conventions and give detail to some of the concepts in complexity theory needed to understand our new results outlined in the following chapters, and the final section of this chapter outlines known quantum supremacy results using these concepts. For a more complete introduction to complexity theory, we refer the reader to books by Sipser [35] and by Du and Ko [15].

### 3.1.1 Computational problems

It is important not to let the heuristic nature of the previous discussion obscure the fact that complexity theory is a completely rigorous mathematical subject. For example, the notion of a computational problem has a precise mathematical definition: A computational problem $\mathcal{P}$ is a mapping from inputs $w = x_1 \dots x_n$ to outputs $y_1 \dots y_m = \mathcal{P}(w)$, where $w$ and $\mathcal{P}(w)$ are taken to be bit strings of length $n$ and $m$, respectively. One example of a problem might be computing the shortest distance between two places within a city. The input is a description of the city as a graph, with vertices representing intersections and edges representing the lengths between those intersections. Additionally, the two vertices corresponding to the points in question must be specified in the input. The output is the binary representation of the minimum distance. Problems where $m = 1$ and the output is just a single bit are called *decision problems*, and can be expressed as questions with yes or no answers. For example, our input might be an integer, and the problem might be: is the integer prime? An algorithm for decision problems is said to "accept" if it outputs 1, and "reject" if it outputs 0.

Given a problem, complexity theorists look for algorithms which can solve the problem by computing the output given the input. Finding an explicit algorithm establishes an upper bound on the complexity of the problem, but without a lower bound, it is impossible to know whether or not there is another algorithm requiring less resources that has not yet been found.

### 3.1.2 The complexity classes P and NP

When assessing complexity, complexity theorists typically care only about the asymptotic relationship between the complexity and the length of the input $n$. So, for time complexity, we classify problems into those with algorithms whose number of time steps depends on $n$ like $O(n)$, $O(n^2)$, $O(2^n)$, etc.

In fact, they often go a step further and group all problems with polynomial-time algorithms together. For example, the complexity class P is the set of decision

problems with deterministic classical algorithms whose time complexity can be constrained by a polynomial in $n$. When we say an algorithm is "efficient," we mean that it is a polynomial-time algorithm.

The class NP is the set of decision problems whose answers can be verified in polynomial time with the help of a *certificate*. More precisely, a problem $\mathcal{P}$ is in NP if there exists a deterministic classical polynomial-time algorithm $M$ which takes $w$, the input to $\mathcal{P}$, along with some bit string "certificate" $c$ as input and has the following properties:

(1) if $\mathcal{P}(w) = 1$ then there exists some $c$ for which $M(w, c) = 1$.

(2) if $\mathcal{P}(w) = 0$ then $M(w, c) = 0$ for all $c$.

Graph three-coloring, the question of whether the vertices of a certain graph can be assigned one of three colors such that no two adjacent vertices have the same color, is an example of an NP problem: if the answer is yes, it can be easily verified with the help of a certificate encoding the three-colored solution to the graph. However, if the certificate encoding the solution is not known ahead of time, it is not clear how it can be found without iterating through the exponential number of possible solutions to see if any of them are successful three-colorings. Whether P = NP, that is, if all problems whose solutions can be verified in polynomial time can also be solved in polynomial time, is a major open question in complexity theory, although it is widely believed that P $\neq$ NP.

A problem is called NP-hard if being able to solve it in polynomial time would imply that every NP problem can be solved in polynomial time, and hence P = NP. If a problem is both NP-hard and a member of NP, then it is called NP-complete. Three coloring is an example of an NP-complete problem.

## 3.2 Counting problems

Problems in NP ask whether or not there exists a certificate which can be used to verify the answer to the problem in polynomial time. In contrast, *counting problems* in

the class #P can be thought of as asking how many such certificates exist. Formally, we define #P by saying a problem $\mathcal{P}$ (on input $w$) is in #P if there exists a polynomial-time algorithm $M$ which takes $w$ as input, along with a certificate $c$, such that

$$|\{c : M(w, c) = 1\}| = \mathcal{P}(w). \tag{3.1}$$

So, for example, the problem of whether or not a three coloring for a graph exists is in NP, while the problem of how many three colorings there are is in #P. Evidently, #P is more powerful than NP: if we know how many certificates there are, we can easily determine whether or not that number is non-zero. Like problems in NP, problems in #P are generally believed not to have a polynomial-time solution; we can always count the number of certificates by brute force iteration, but since there are an exponential number of possible certificates, doing so would take exponential time.

In parallel to the definition of NP-hard and NP-complete problems, #P-hard problems are those for which polynomial time solutions would imply FP = #P, where FP is the set of problems with possibly multiple bits of output that can be solved in polynomial time (the analogue of P for problems with multiple bits of output). A problem is #P-complete if it is #P-hard and also in #P.

## 3.3 Probabilistic computation

So far this chapter, we have only discussed deterministic computation. However, quantum mechanics, and by extension quantum computers, are inherently probabilistic. So, before we move from classical to quantum computation, it makes sense to consider classical randomized computation. A randomized (or probabilistic) algorithm is an algorithm which at various points has the ability to make random transitions between states. We can model this with a deterministic algorithm that takes as an additional input a string of random bits $r$ of length equal to the maximum run time of the algorithm. At each time step, the machine can read at most one of these random bits. So a decision problem $\mathcal{P}$ has a probabilistic algorithm if there

exists a deterministic algorithm $M$ acting on the input to the problem $w$ along with a random string of bits $r$, such that, if $\mathcal{P}(w) = 1$, then $\Pr_r(M(w, r) = 1) > 1/2$ and if $\mathcal{P}(w) = 0$, then $\Pr_r(M(w, r) = 1) \leq 1/2$, where $\Pr_r$ indicates the probability taken over all possible bit strings $r$. The set of problems with probabilistic algorithms that use a polynomial number of time steps is called PP.

However, PP isn't a very practical model, because, if an algorithm $M$ for problem $\mathcal{P}$ accepts input $w$ with probability $1/2 + \epsilon$ for some exponentially small number $\epsilon$, we will have to run $M$ on $w$ an exponential number of times before we can be highly confident that $\mathcal{P}(w) = 1$.

Therefore, it is more reasonable to consider *bounded-error* probabilistic algorithms, where we require that if $\mathcal{P}(w) = 1$, $\Pr_r(M(w, r) = 1) \geq 2/3$ and if $\mathcal{P}(w) = 0$, then $\Pr_r(M(w, r) = 1) \leq 1/3$.[1] Given a bounded-error probabilistic algorithm for $\mathcal{P}$, we can become highly confident of $\mathcal{P}(w)$ using only a polynomial number of iterations of running $M$ on $w$. The class of problems with polynomial-time bounded-error probabilistic algorithms is called BPP.

However, PP will still be a useful class to consider due to its connection to counting problems. In fact, PP is in a sense the decision version of #P. Given a problem in PP and the polynomial-time machine $M$ which solves it probabilistically, we can interpret the random strings $r$ as possible certificates for the problem instance. If $\mathcal{P}(w) = 1$ then strictly more than half the certificates cause $M$ to accept, and if $\mathcal{P}(w) = 0$, at most half the certificates cause $M$ to accept. Comparing this to #P algorithms, which tell us exactly how many certificates cause $M$ to accept, we can see how PP can be understood as a counting class in which we only care about whether or not the number is more than half the total. Thus, problems like determining whether a majority of the possible three-colorings of a graph are natural PP problems.

[1] Any constant greater than $1/2$ would be equivalent, up to polynomial-time overhead.

## 3.4 The polynomial hierarchy

A fruitful way of understanding the relationship between the classes we've mentioned is through the polynomial hierarchy (PH), a generalization of the class NP.

The class NP can be expressed as containing any problem $\mathcal{P}$ for which there exists a polynomial-time Turing machine $M$ acting on the input $w$ and a certificate $c_1$ such that given $w$

$$\exists c_1 \; M(w, c_1) = 1 \text{ iff } \mathcal{P}(w) = 1. \tag{3.2}$$

We can make a slightly more complex class along these lines: Let $\Sigma_2$P contain any problem for which there exists a polynomial-time Turing machine $M$ acting on input $w$ and certificates $c_1$ and $c_2$ such that given $w$:

$$\exists c_1 \forall c_2 \; M(w, c_1, c_2) = 1 \text{ iff } \mathcal{P}(w) = 1 \tag{3.3}$$

By disregarding $c_2$ altogether we see that $\Sigma_2$P contains NP, but $\Sigma_2$P might also contain problems which are not in NP.

We can more generally form $\Sigma_n$P by changing Eq. (3.3) to

$$\exists c_1 \forall c_2 \ldots Q_n c_n \; M(w, c_1, \ldots, c_n) = 1 \text{ iff } \mathcal{P}(w) = 1 \tag{3.4}$$

where $Q_n$ is $\forall$ if $n$ is even and $\exists$ if $n$ is odd.

These form the levels of the polynomial hierarchy: PH $= \cup_n \; \Sigma_n$P. The zeroth level is P, and the first level is NP. In the same sense that it is widely believed that P $\neq$ NP, it is also widely believed that $\Sigma_n$P $\neq \Sigma_{n+1}$P. If it were to be shown that $\Sigma_n$P $= \Sigma_{n+1}$P, this would imply that $\Sigma_n$P $= \Sigma_m$P for all $m \geq n$, the so-called "collapse" of the PH to the $n^{\text{th}}$ level. If the Polynomial hierarchy doesn't collapse, we say it is "infinite."

This framework is not exclusive to polynomial-time Turing machines. We can also define the class $\Sigma_j$TIME$(f(m))$ as those problems with algorithms satisfying Eq. (3.4) where $M$ is allowed to use $f(m)$ time steps on an input $w$ of size $m$.

An equivalent way of formulating the PH is through the concept of an *oracle*. An oracle algorithm $M^{\mathcal{P}}$, is an algorithm (in whatever model of our choosing) that has the ability to compute the answer to the problem $\mathcal{P}$ in a single time step. So, if $\mathcal{P}$ is the three-coloring problem, then any NP problem can be solved with a polynomial number of time steps using a $\mathcal{P}$ oracle: first we reduce the NP problem to an instance of three-coloring in polynomial time, then we solve it using the oracle.

We can also define new complexity classes relative to a certain oracle. The class $\mathrm{P}^{\mathcal{P}}$ is the set of all problems which can be solved with a deterministic polynomial-time $\mathcal{P}$-oracle algorithm. The class $\Sigma_n\mathrm{P}^{\mathcal{P}}$ is the class defined by Eq. (3.4) except that the algorithm $M$ is replaced by the oracle-algorithm $M^{\mathcal{P}}$. If A is a complexity class, then $\mathrm{P}^{\mathrm{A}}$ and $\Sigma_n\mathrm{P}^{\mathrm{A}}$ are defined by taking the union of $\mathrm{P}^{\mathcal{P}}$ and $\Sigma_n\mathrm{P}^{\mathcal{P}}$, respectively, for all $\mathcal{P}$ in A. Using this framework, it can be shown that $\Sigma_2\mathrm{P} = \mathrm{NP}^{\mathrm{NP}}$, and more generally $\Sigma_{n+1}\mathrm{P} = \Sigma_n\mathrm{P}^{\mathrm{NP}}$. These observations clarify why we expect the PH to be infinite: since we believe some NP problems require exponential time, giving any polynomial-time class the ability to solve NP problems in a single time step should allow it to efficiently solve some problems that previously required exponential time, meaning that ascending a level of the hierarchy allows us to solve new problems efficiently.

Using oracles, we can relate the PH, a class of decision problems, to counting problems in #P via Toda's theorem [15]:

$$\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{PP}} \tag{3.5}$$

This tell us that in a sense, the ability to solve counting problems is more powerful than the entire PH!

## 3.5 Quantum computation and postselection

We can also use complexity theory to describe quantum computation. We let BQP be the class of problems for which there is a bounded-error polynomial-time quantum algorithm. Using the quantum circuit model discussed in Chapter 2, we interpret a polynomial-time quantum algorithm for a problem $\mathcal{P}$ with input $w$ to be a quantum

circuit whose number of qubits and number of gates is polynomial in $|w|$, and whose description can be computed in classical polynomial time. Thus, BQP is the quantum analogue of BPP.

The fact that classical computation can be efficiently simulated on a quantum computer means that BPP $\subseteq$ BQP. It has also been shown that BQP $\subseteq$ PP [4], but otherwise, the exact placement of BQP within classical complexity classes is an open question.

Progress can be made if we consider a slightly modified model of quantum computation called postselected quantum computation. A postselected quantum circuit is one where, at the end of the algorithm, some qubits are measured and kept as outputs of the circuit, as before, but others are *postselected*. That is, they are measured but their measurement outcome is forced to take a certain value. In real life, we cannot force a measurement outcome to take a certain value, so this is not a reasonable physical model. In fact, postselection would be an extremely powerful ability. We can see how it is easy to solve all NP problems using postselection: we put the computer into a superposition over all possible certificates $c$, compute $M(w, c)$ into an ancilla qubit while maintaining superposition, and then postselect that the ancilla qubit is in the $|1\rangle$ state. The rest of the qubits will tell us which certificate $c$ causes $M(w, c) = 1$, if there is one at all.

The power of postselection extends further than NP, however. Letting PostBQP be the class of problems solvable up to bounded-error with polynomial postselected quantum circuits, Aaronson showed that PostBQP = PP [1]. Since PP is the decision version of the exact counting class #P, the intuition for this statement is that postselected quantum computation roughly has the power of exact counting.

Postselection is not exclusively a quantum power. We can also discuss classical postselected algorithms, which we model by considering probabilistic algorithms $M$ that take $w$ as input along with a random bit string $r$, except now $M$ has three options as output. It can accept, reject, or cancel the computation altogether. A problem $\mathcal{P}$ is a member of PostBPP if there is such an $M$ for which the probability that $M(w, r) = \mathcal{P}(w)$ given that $M(w, r)$ does not cancel is at least $2/3$. Thus, as

in the quantum case, NP is contained in PostBPP since postselection allows us to randomly guess a certificate and just cancel whenever we pick one that doesn't cause the algorithm to accept.

In contrast with PostBQP, PostBPP is known to be contained in the third level of the PH.[2] This can be shown by observing that we can solve a PostBPP by looking at how many random strings $r$ cause $M$ to accept or reject. We do not need to know the exact number of strings $r$, but only a multiplicative approximation since we are guaranteed that either the number that cause $M$ to accept is at least two times the number that cause $M$ to reject, or that the opposite is true. Approximate counting the number of accepting and rejecting strings in this fashion can be achieved using an oracle in $\Sigma_2$P [36, 23], meaning PostBPP $\subseteq \Sigma_3$P. The intuition for this statement is that PostBPP has roughly the same power as approximate counting.

Toda's theorem gives us formal evidence that PostBQP is more powerful than PostBPP, since if PostBPP = PostBQP then PP would lie in the PH causing its collapse, which we believe is not the case.

Intuitively, we understand this as the idea that exact counting is more difficult than approximate counting: even if we could approximately count certificates to an NP problem in polynomial time, it would still be exponentially difficult to count them exactly. This statement of course remains unproven, but it is a plausible conjecture under our current understanding of complexity theory.

## 3.6 Hardness of simulation for quantum circuits

### 3.6.1 Hardness for general quantum circuits

We can use the language of complexity theory to speak precisely about the difficulty of simulating quantum circuits. If there were a classical algorithm to strongly simulate quantum circuits in polynomial time, then it can be shown that P = NP,[3] but as

[2]The class PostBPP is equivalent to the class $\text{BPP}_{\text{PATH}}$, which is defined and proved to lie in the third level in Ref. [23].

[3]In fact, strong simulation is known to be #P-hard, so a polynomial-time classical simulation algorithm would imply FP = #P, entailing P = NP [18]

we discussed in Section 2.4.2, strong simulation isn't necessarily the most reasonable benchmark for simulation.

On the other hand, a classical polynomial-time algorithm to weakly simulate quantum circuits would not entail P = NP (as far as we know), but it would still imply that the PH collapses, albeit to a higher level. The reasoning for this requires postselection. Given a classical polynomial-time weak simulation algorithm, any problem solvable by a polynomial-sized quantum circuit would have a polynomial-time classical algorithm meaning BQP would equal BPP. The logic would still hold if we move to postselected quantum circuits and postselected classical algorithms: weak polynomial-time simulation of quantum postselected circuits implies PostBQP = PostBPP. However, in this case, since PostBQP = PP and PostBPP $\subseteq \Sigma_3$P, Toda's theorem tells us that PH $\subseteq$ P$^{\text{PP}}$ = P$^{\Sigma_3\text{P}} \subseteq \Sigma_4$P, so the entire PH is contained in the fourth level, meaning it collapses.[4]

If we can *approximately* weakly simulate general quantum circuits in classical polynomial time, then we can reach the same conclusion, but only if the approximation is multiplicative. Suppose a quantum algorithm solves problem $\mathcal{P}$ up to bounded error. Letting $Q(w)$ be the probability the quantum circuit outputs 1 on input $w$, we know that $Q(w) \geq 2/3$ if $\mathcal{P}(w) = 1$ and $Q(w) \leq 1/3$ if $\mathcal{P}(w) = 0$. If we have an approximate weak simulation algorithm with multiplicative parameter $1/5$, then we have the ability to produce the probability distribution $P(w)$ where $4/5 \leq P(w)/Q(w) \leq 6/5$. Hence, if $\mathcal{P}(w) = 1$ then $P(w) \geq 8/15$ and if $\mathcal{P}(w) = 0$ then $P(w) \leq 6/15$. By repeating the algorithm a constant number of times and taking the majority output, we can boost these probabilities above $2/3$ and below $1/3$ in the two cases, yielding a bounded-error classical algorithm. This would imply PostBQP = PostBPP and the collapse of the PH, as before.

In the strong, weak, and approximate (multiplicative) weak cases, we conclude that polynomial-time simulation implies the collapse of the PH. So, if we start with the believable conjecture that the PH is infinite, we conclude that there is no

[4] A more refined argument [23] puts PostBPP in P$^{\Sigma_2\text{P}} \subseteq \Sigma_3$P, and since P$^{\text{P}^{\Sigma_2\text{P}}}$ = P$^{\Sigma_2\text{P}}$, Toda's theorem (along with efficient classical weak simulation) actually implies a collapse of the PH to the third level, not the fourth.

polynomial-time algorithm to weakly or strongly simulate general quantum circuits. In other words, quantum circuits are supreme to classical algorithms.

### 3.6.2 Hardness for restricted classes of quantum circuits

Since restricted quantum circuits are themselves quantum circuits, a method for simulating a general quantum circuit is also a method for simulating the restricted class. However, it is possible that there are ways to simulate these restricted classes much more efficiently than general circuits, as is true for Clifford circuits via the Gottesman-Knill theorem (see Section 2.4.4). We now discuss how for QAOA and instantaneous circuits, this is not the case; in both cases, approximate weak simulation still implies the collapse of the PH.

Let QAOA refer to the set of problems that can be solved by polynomial-sized QAOA circuits, and let IQP be the same set for instantaneous quantum circuits. It is not believed that IQP = BQP or that QAOA = BQP due to the restrictions imposed on the QAOA and instantaneous models. However, we can show that PostQAOA = PostIQP = PostBQP, and hence if QAOA or instantaneous quantum circuits were exactly or approximately weakly simulable, then PostBPP = PostBQP and the PH collapses, as before. To show that PostQAOA = PostIQP = PostBQP, we must prove that any postselected quantum circuit can be simulated using a postselected QAOA or instantaneous quantum circuit. The proofs of these statements are expressed by the following theorems.

**Theorem 3.6.1.** PostIQP = PostBQP

*Proof.* This theorem was first proved in [9], which we reproduce here. The fact that PostIQP $\subseteq$ PostBQP follows from the fact that instantaneous circuits are a subclass of general quantum circuits, so we only must show the opposite inclusion. If a problem $\mathcal{P}$ is in PostBQP, then, fixing input $w$, there is a postselected quantum circuit $\mathcal{C}$ for which $\Pr(\mathcal{C}(w) = \mathcal{P}(w)) \geq 2/3$. We can approximate this circuit to arbitrary accuracy using solely $H$, $T$, and $CZ$ gates (since $CX_{ij} = H_j CZ_{ij} H_j$, we can replace $CX$ with $CZ$), so we can assume in the first place that $\mathcal{C}$ contains only gates

from this set. We can add two $H$ gates to the beginning and end of the circuit on each qubit since $H^2 = I$. Thus, our circuit is nearly in the form of an instantaneous quantum circuit; each qubit begins and ends with a Hadamard gate, and all of the internal $CZ$ and $T$ gates are diagonal. The only problem is the internal $H$ gates, which are not diagonal. To remedy this, for each internal $H$ gate we will insert the gadget from Figure 3-1 which adds one ancilla qubit and uses postselection to simulate the effect of the $H$ gate.

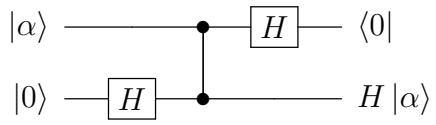


Figure 3-1: Gadget to simulate the effect of an internal $H$ gate within the postselected instantaneous circuit model. A new ancilla qubit is introduced into the $H|0\rangle$ state, a $CZ$ operation is applied between the two qubits, then the first qubit is measured and postselected into the outcome $|0\rangle$ (denoted by the symbol $\langle 0|$ at the end of the line). The resulting circuit fits into the instantaneous model and the second qubit is left in the state which results from applying $H$ to the original state of the first qubit.

It is easy to verify that this circuit accomplishes the stated goal. If all such internal $H$ gates are replaced with one of these gadgets, we have an instantaneous polynomial-size quantum circuit which has the same output as the original circuit $\mathcal{C}$, and hence, any PostBQP problem $\mathcal{P}$ can be solved in PostIQP. □

**Theorem 3.6.2.** PostQAOA = PostBQP

*Proof.* The proof is similar to the proof of Theorem 3.6.1, only using a different gadget to replace internal $H$ gates. The gadget also requires one ancilla qubit. This construction is found in [18]. □

The new results we present in this work, which are contained in Chapters 4 and 5, focus on working through these previous quantum supremacy results while paying attention to the run time of these algorithms in a fine-grained sense. That is, no longer will we consider two algorithms equivalent if they differ only by a polynomial, or even by just a constant factor. In our work, we keep track of the exact form of the run time expressions at each step. We combine this with a fine-grained assumption regarding

the hardness of PP in comparison to the PH, whereas previous arguments simply claim the infinitude of the PH with no specificity as to what super-polynomial function actually separates neighboring levels. At the end, we arrive at lower bounds for simulation that are more specific than simply "simulation takes more than polynomial-time."

# Chapter 4

# Lower bounds for simulation of IQP and QAOA circuits

## 4.1 Outline of lower-bounds argument

Previous quantum supremacy arguments begin with the assumption that the polynomial hierarchy is infinite, and arrive at the conclusion that any approximate weak simulation algorithm must take more than polynomial time. Our argument follows the same steps, but begins with a fine-grained assumption about the polynomial hierarchy and arrives at a fine-grained lower bound on the classical simulation.

We begin by describing a postselected IQP or QAOA circuit for solving a PP-complete problem, which we know is possible since PP = PostBQP = PostIQP = PostQAOA. The PP-complete problem we will use is called MAJ-ZEROS: given a degree-3 polynomial $f$ in $n$ variables over the field $\mathbb{F}_2$, is

$$\text{gap}(f) \equiv |\{z : f(z) = 0\}| - |\{z : f(z) = 1\}| \tag{4.1}$$

greater than zero? In other words, is the number of zeros more than half the $2^n$ possible bit string inputs to the polynomial? This problem has a very natural solution in PostIQP and a somewhat natural solution in PostQAOA.

Next, we suppose there exists a classical algorithm to weakly simulate any IQP or QAOA circuit in time $g \cdot s(q)$ where $q$ is the number of qubits in the circuit and $g$ is the number of gates, leaving the function $s$ unspecified. Note that $s$ need not be a polynomial. This classical postselected algorithm can be used to simulate the PostQAOA and PostIQP circuit for MAJ-ZEROS, meaning that MAJ-ZEROS is in the class PostBPTIME$(g \cdot s(q))$.

Finally, since PostBPP $\subseteq \Sigma_3$P, we can turn our PostBPTIME algorithm into a $\Sigma_3$TIME algorithm, with runtime that is polynomially related to $g \cdot s(q)$. Thus, we have an implicit solution for the PP-complete problem MAJ-ZEROS in the third level of the TIME hierarchy (it cannot be called the polynomial hierarchy because the solution may require more than polynomial runtime). Since PP is essentially as difficult as exact counting, we expect a PP-complete problem to be difficult, even after ascending to the third level of the hierarchy, so we make the plausible conjecture that (roughly) MAJ-ZEROS $\notin \Sigma_3$TIME$(2^{n/5})$. The exact function we use is not exactly $2^{n/5}$, but asymptotically similar, and is given further motivation using an oracle separation, as we will explain. If this conjecture is true, then the runtime of our $\Sigma_3$TIME algorithm for MAJ-ZEROS, expressed in terms of the simulation function $s$, must be greater than $2^{n/5}$, giving us a lower bound for the simulation time function $s$.

In other words, we show that if this lower bound on the classical simulation time $s(b)$ for a circuit with $b$ qubits were broken, then we could use the simulation algorithm as a subroutine to make a purely classical $\Sigma_3$TIME algorithm for the problem MAJ-ZEROS running in less time than we conjecture must be required.

## 4.2 The problem MAJ-ZEROS

The problem MAJ-ZEROS works within the field $\mathbb{F}_2$, that is, the integers mod 2. The input to the problem is a polynomial in $n$ variables with degree at most 3 and no constant term. Since the only non-zero element in $\mathbb{F}_2$ is 1, every term in the polynomial has coefficient 1. One example could be $f(z) = z_1 + z_2 + z_1z_2 + z_1z_2z_3$.

Evaluating $f$ for a given string $z$ to determine whether $f(z) = 0$ or $f(z) = 1$ can be done efficiently, but since there are an exponential number of possible strings $z$, it naively takes exponential time to count the number of strings $z$ for which $f(z) = 0$, or equivalently, to compute the function $\text{gap}(f)$ given by Eq. (4.1). To turn this into a decision problem, the question posed by MAJ-ZEROS is whether or not $\text{gap}(f) > 0$.

The problem MAJ-ZEROS is a natural problem to work with because there is an elegant correspondence between degree-3 polynomials and IQP circuits involving $Z$, $CZ$, and $CCZ$ gates [29]. In particular, if we label qubit $i$ with the variable $z_i$, then we can form a circuit $\mathcal{C}_f$ from the polynomial $f$ as follows: if the term $z_i$ appears in $f$, we perform the gate $Z_i$; if the term $z_i z_j$ appears, we perform the gate $CZ_{ij}$; and if the term $z_i z_j z_k$ appears, we perform the gate $CCZ_{ijk}$ within the diagonal portion of the IQP circuit. For example, for the polynomial $f(z) = z_1 + z_2 + z_1 z_2 + z_1 z_2 z_3$, the circuit $\mathcal{C}_f$ is shown in Figure 4-1.

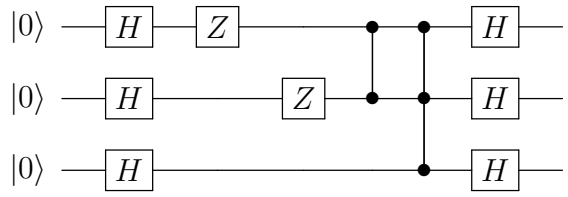


Figure 4-1: IQP circuit $\mathcal{C}_f$ corresponding to the degree-3 polynomial $f(z) = z_1 + z_2 + z_1 z_2 + z_1 z_2 z_3$. The unitary $U_f$ implemented by the circuit has the property that $\langle 0| U_f |0\rangle = \text{gap}(f)/2^n$ where in this case $n = 3$.

Suppose the circuit $\mathcal{C}_f$ performs the unitary $U_f$. Then the crucial property of this correspondence is that $\langle 0| U_f |0\rangle = \frac{\text{gap}(f)}{2^n}$, where $|0\rangle$ is shorthand for the starting $|0\rangle^{\otimes n}$ state. This is easily seen by noting that the initial set of $H$ gates generates the equal superposition state $|B\rangle = \sum_{x=0}^{2^n-1} |x\rangle / \sqrt{2^n}$, so $\langle 0| U_f |0\rangle = \langle B| U_f' |B\rangle$ where $U_f'$ is the diagonal unitary implemented by the internal portion of $\mathcal{C}_f$. Since $U_f'$ applies a $(-1)$ phase to states $|x\rangle$ for which $f(x) = 1$, $\langle 0| U_f |0\rangle = \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \langle y|x\rangle / 2^n = \sum_{x=0}^{2^n-1} (-1)^{f(x)} / 2^n = \text{gap}(f)/2^n$. Thus, $\text{gap}(f)$ can be computed by calculating the amplitude of the $|0\rangle$ state produced by the circuit.

MAJ-ZEROS is in the class PP: given $f$, we simply choose a random bit string $z$ and output the result $\text{NOT}(f(z))$, which can be computed in polynomial time. If

$\text{gap}(f) > 0$, this algorithm outputs 1 more than half the time, and if $\text{gap}(f) \leq 0$, it outputs 0 at least half the time, as required by a PP algorithm.

**Theorem 4.2.1.** *MAJ-ZEROS is* PP*-complete.*

*Proof.* Montanaro [29] gave a quantum-circuit proof that computing $\text{gap}(f)$ exactly for degree-3 polynomials over $\mathbb{F}_2$ is #P-complete by reduction from the problem of computing $\text{gap}(g)$ for an arbitrary boolean function $g$, which is #P-complete by definition.

Using the same proof, one can show that MAJ-ZEROS is PP-complete by reduction from the problem of determining whether $\text{gap}(g) > 0$ for an arbitrary boolean function $g$, which is PP-complete by definition. □

**Theorem 4.2.2.** *MAJ-ZEROS restricted to input polynomials for which each variable appears in at most 3 terms is* PP*-complete.*

*Proof.* Montanaro [29] showed that given an arbitrary degree-3 polynomial $f$, by introducing new variables, we can form a degree-3 polynomial $f'$ for which each variable appears in at most 3 terms and such that $\text{gap}(f) = \text{gap}(f')$. Thus, MAJ-ZEROS can be reduced to MAJ-ZEROS on inputs with the desired restriction, proving the theorem. □

Therefore, in the following sections, we will assume that all inputs to MAJ-ZEROS are such that each variable appears in at most 3 terms.

## 4.3 Derivation of lower bounds

### 4.3.1 PostIQP circuit for solving MAJ-ZEROS

Since MAJ-ZEROS is in PP and PP = PostBQP = PostIQP, we expect there to exist a PostIQP algorithm for MAJ-ZEROS. We outline one such algorithm in this section.

**Theorem 4.3.1.** *For a degree-3 polynomial $f$ on $n$ variables where each variable appears in at most three terms, there exists a family of postselected instantaneous quantum circuits $\{\mathcal{Q}_i\}_{i=0}^{n-1}$, each acting on $q = n + 4$ qubits and at most $g = 4n + 13$ gates, such that taking $n \cdot L$ samples from $\mathcal{Q}_i$ for different values of $i$ allows us to solve MAJ-ZEROS for the input $f$ with error probability $n\exp(-2L/25)$. Thus, we require only $O(n\log(n))$ samples to achieve arbitrarily small error probability.*

Our wording implies that the post-processing of the circuit samples is done classically, but of course we could do this instead by making many copies of the circuit and turning the entire algorithm into one large PostIQP circuit for the problem MAJ-ZEROS. The way we stated the algorithm, however, will make more sense once we classically simulate the circuit since, when simulation time is exponential in the circuit size, it is easier to simulate a smaller circuit many times than a larger circuit a single time.

*Proof.* We build $\mathcal{Q}_i$ out of two separate components. The first component will be the circuit $\mathcal{C}'_f$ which does not depend on $i$. In the previous section, we outlined the procedure for producing an IQP circuit $\mathcal{C}_f$ with $n$ qubits implementing the unitary $U_f$ given a degree-3 polynomial $f$ on $n$ variables. This unitary has the property that $\langle 0| U_f |0\rangle = \text{gap}(f)/2^n$. We write $U_f = H^{\otimes n} V_f H^{\otimes n}$, where $V_f$ is the unitary corresponding to the internal diagonal portion of the IQP circuit. $V_f$ consists solely of $Z$, $CZ$, and $CCZ$ gates. We construct a new circuit $\mathcal{C}'_f$ that implements unitary $U'_f$ by adding one ancilla qubit and using it as an additional control for every internal gate that appears in $V_f$. Thus, $Z$ gates in $U_f$ become $CZ$ gates in $U'_f$ controlled by the ancilla, $CZ$ gates become $CCZ$ gates, and $CCZ$ gates become $CCCZ$ gates. Since the ancilla controls every internal operation, we can write $U'_f = (H^{\otimes n} \otimes I) C V_f H^{\otimes (n+1)}$, where $CV_f$ denotes that the entire $V_f$ operation is controlled by the ancilla qubit. We temporarily omit the final $H$ gate on the ancilla qubit, because we will continue performing gates on it before it is measured. Then, we postselect on the original $n$ qubits being in the $|0\rangle$ state. An example of the circuit $\mathcal{C}'_f$ for the function $f(z) = z_1 + z_2 + z_1 z_2 + z_1 z_2 z_3$ is shown in Figure 4-2.

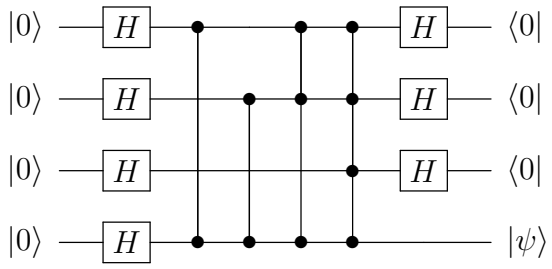


Figure 4-2: The circuit $\mathcal{C}'_f$ corresponding to the degree-3 polynomial $f(z) = z_1 + z_2 + z_1z_2 + z_1z_2z_3$, to be compared to the circuit $\mathcal{C}_f$ from Figure 4-1. By controlling each diagonal operation from $\mathcal{C}_f$ with an ancilla qubit, and postselecting on the first $n = 3$ qubits being $|0\rangle$ (denoted by symbol $\langle 0|$), the state $|\psi\rangle$ given by Eq. (4.2) is produced.

The ancilla qubit is left in the state $|\psi\rangle$ where, up to normalization,

$$|\psi\rangle = |0\rangle + \frac{\mathrm{gap}(f)}{2^n}|1\rangle \equiv |0\rangle + r\,|1\rangle \tag{4.2}$$

where $r = \mathrm{gap}(f)/2^n$.

If we draw $|\psi\rangle$ as a vector in the real plane $\mathbb{R}^2$ with $|0\rangle$ on the $x$-axis and $|1\rangle$ on the $y$-axis, then solving MAJ-ZEROS is equivalent to determining whether $|\psi\rangle$ lies in the upper half-plane.

Without postselection, this would be exponentially difficult since two vectors on opposite halves of the plane could be exponentially close. However, postselection allows us to, in a sense, pry exponentially close vectors apart in polynomial time. This procedure will form the second part of the circuit $\mathcal{Q}_i$, and we describe it as follows.

We introduce another ancilla qubit and create the state $\alpha\,|0\rangle - \beta\,|1\rangle$ for arbitrary $\alpha \geq 0$, $\beta = \sqrt{1-\alpha^2}$. We will show later how this can be done within the PostIQP framework. We use this ancilla to perform a CNOT gate targeted at the ancilla containing the state $|\psi\rangle$, which is then postselected into $|0\rangle$. This leaves the second ancilla in the state $\alpha\,|0\rangle - \beta r\,|1\rangle$, up to normalization. To perform the CNOT gate within the PostIQP framework, we write it as $H_a CZ_{ba} H_a$. The construction of state $\alpha\,|0\rangle - \beta\,|1\rangle$ is made apparent by writing it as $Z_{-\pi/2}X_\theta\,|0\rangle = Z_{-\pi/2}HZ_\theta H\,|0\rangle$, where

$\theta = 2\arccos(\alpha)$, $X_\phi = \exp(-i\phi X/2)$, and $Z_\phi = \exp(-i\phi Z/2)$. The circuit diagram depicting this construction is given in Figure 4-3.

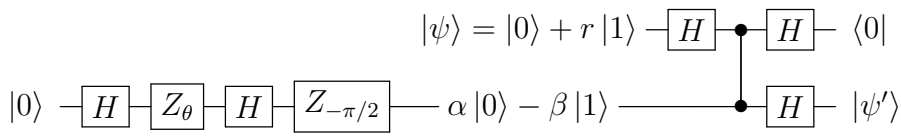


Figure 4-3: The circuit that, given $|\psi\rangle$, creates the state $|\psi'\rangle = (\alpha - \beta r)\,|0\rangle + (\alpha + \beta r)\,|1\rangle$, up to normalization, for arbitrary $\alpha$ and $\beta$. The angle $\theta$ is $2\arccos(\alpha)$. The gate $Z_\phi$ denotes the gate $\exp(-i\phi Z/2)$. The state $|\psi\rangle$ is prepared by the circuit $\mathcal{C}'_f$ as exemplified by Figure 4-2. To make this into a PostIQP circuit, we must replace the internal $H$ gates with the gadget from Figure 3-1, as described in the proof of Theorem 3.6.1.

Replacing the two internal $H$ gates for this construction with the postselected IQP gadgets in Figure 3-1 will incur two extra ancilla qubits.

If we let $\beta/\alpha = \eta^i$ where $\eta = 1 + \sqrt{2} \approx 2.41$ for some integer $0 \leq i \leq n-1$, then this circuit leaves the second qubit in the state, up to normalization,

$$|\psi'\rangle = (1 - \eta^i r)\,|0\rangle + (1 + \eta^i r)\,|1\rangle\,. \tag{4.3}$$

Crucially, if $r \leq 0$, then when we measure in the computational basis, the chance of measuring the outcome 1 is always at most $1/2$.

On the other hand, if $r > 0$, we claim that there exists some $i$ so that we measure the outcome 1 with probability greater than $9/10$. The proof is as follows. In this case, since $\mathrm{gap}(f)$ is an even integer, we write $r = x/\eta^{n-1}$ where $1 < x \leq \eta^{n-1}$. Since, $\eta^{k-1} + \eta^k = \eta^{k+1} - \eta^k$ for all $k$, if $x$ lies between $\eta^k$ and $\eta^{k+1}$, it is either less than $\eta^k + \eta^{k-1}$ or greater than $\eta^{k+1} - \eta^k$. Therefore, there must exist $j$ (equal to either $k$ or $k+1$) between 0 and $n-1$ such that $x = \eta^j + x'$ where $|x'| \leq \eta^{j-1}$. We choose $i = n - j - 1$. Then our state pre-measurement is (up to normalization) $-x'/\eta^j\,|0\rangle + (2 + x'/\eta^j)\,|1\rangle$. Since $|x'/\eta^j| \leq 1/\eta$, the chances of measuring 0 are at most $1/(10 + 4\sqrt{2}) \approx 0.064$. Thus, the chances of measuring 1 are greater than $9/10$.

We let the circuit $\mathcal{Q}_i$ be given by the composition of circuit $\mathcal{C}'_f$ and the circuit in Figure 4-3 where we set $\beta/\alpha = \eta^i$. By taking enough samples from these $\mathcal{Q}_i$, we can

determine if an $i$ exists for which the chances of measuring 1 is greater than $9/10$, and consequently whether $\text{gap}(f) > 0$. We do this as follows.

For each of the $n$ possible values of $i$, we repeat the circuit $\mathcal{Q}_i$ $L$ times. If for some value of $i$, more than $0.7L$ outcomes are 1, then it is likely that $\text{gap}(f) > 0$ and the algorithm outputs 1 on the input $f$. If none of the values of $i$ achieve this threshold, then the algorithm rejects the input $f$.

Now we analyze the probability of error. Suppose in reality $r \leq 0$. For a given value of $i$, the probability of getting a 1 outcome is smaller than $1/2$. The Hoeffding inequality tells us that the sum of $L$ repetitions of an experiment, each giving outcome 1 with probability $p$, will be greater than $(p + \delta)L$ with probability smaller than $\exp(-2\delta^2 L)$. In our case, $p = 0.5$, and $\delta = 0.2$. We can take a union bound over each of the $n$ possible values of $i$ to conclude that the probability that the algorithm fails is at most $n \exp(-2L/25)$.

Now suppose in reality $r > 0$. Then there is some value of $i$ where the probability of getting a 1 is at least 0.9, so the algorithm can fail only if we register less than $0.7L$ 1 outcomes for this value of $i$, which happens with probability at most $\exp(-2L/25)$.

We conclude that the algorithm's failure probability is bounded by $n \exp(-2L/25)$. We can choose $L$ large enough to make this quantity arbitrarily small, showing that this is indeed a bounded-error algorithm.

The circuit acts on the original $n$ qubits, the controlling ancilla qubit, the ancilla qubit prepared into the state $\alpha |0\rangle - \beta |1\rangle$, and the two auxiliary qubits introduced to perform internal Hadamard gates with postselection. This gives a total of $q = n + 4$ qubits. Since the polynomial $f$ can be assumed to be such that each variable appears in at most 3 terms, there can be at most $2n$ terms, since there are only $n$ possible single-variable terms, so the rest must have at least two variables. Each term requires one gate. Additionally, we require 5 internal diagonal gates as shown in Figure 4-3. This gives a total of $2n + 5$ internal gates and $2n + 8$ Hadamards, one applied to each qubit at the beginning and end of the algorithm, for a total of $4n + 13$ gates. □

**Theorem 4.3.2.** *There exists a family of circuits $\{\mathcal{Q}_i\}_{i=0}^{n-1}$ as in Theorem 4.3.1 such that with $n \cdot L$ approximate samples up to multiplicative parameter $\epsilon < 2/7$, the problem*

*MAJ-ZEROS can be solved with chance of failure* $n\exp(-(2-7\epsilon)^2 L/50)$*. Thus, even in the case of approximate sampling, only* $O(n\log(n))$ *samples are needed to achieve arbitrarily small error probability.*

*Proof.* First we note that when $\epsilon = 0$, we arrive at Theorem 4.3.1. Using the same circuit, if $r \leq 0$, the sample is guaranteed to be 0 with probability $(1+\epsilon)/2$ for all values of $i$. If $r > 0$, there is guaranteed to be a value of $i$ for which the output will be 1 at least $9(1-\epsilon)/10$. The difference between these two probabilities is $(4-14\epsilon)/10$, so if we set our threshold number of successes to be their average times $L$, that is $(7-2\epsilon)L/10$, then we can use the Hoeffding inequality with $\delta = (2-7\epsilon)/10$, proving the theorem. □

### 4.3.2 PostQAOA circuit for solving MAJ-ZEROS

Now, we give the analogous statement for QAOA circuits.

**Theorem 4.3.3.** *For a degree-3 polynomial* $f$ *on* $n$ *variables where each variable appears in at most three terms, there is a family of postselected QAOA quantum circuits* $\{\mathcal{Q}_i\}_{i=0}^{n-1}$*, acting on at most* $q = 4n+6$ *qubits and requiring at most* $g = 29n+48$ *constraints, such that taking* $n \cdot L$ *samples from* $\mathcal{Q}_i$ *for different values of* $i$ *allows us to solve MAJ-ZEROS for the input* $f$ *with error probability* $n\exp(-2L/25)$*. Thus, we require only* $O(n\log(n))$ *samples to achieve arbitrarily small error probability.*

*Proof.* The proof follows the same process as the proof of Theorem 4.3.1. First, we show how the circuit $\mathcal{C}'_f$ defined in the proof of Theorem 4.3.1 can be converted to a QAOA circuit. The original column of $H$ gates is already part of the QAOA framework. We implement the diagonal portion of $\mathcal{C}'_f$ by letting the angle of rotation $\gamma = \pi/4$ and constructing a Hamiltonian $C$ which is the sum of constraints such that $\exp(-i\gamma C)$ implements this part of the circuit. Since $CZ = \exp(-i\pi\left|11\right\rangle\left\langle 11\right|) = \exp(-i\gamma 4\left|11\right\rangle\left\langle 11\right|)$, for each $CZ$ gate we add four copies of the constraint which is satisfied only if both bits are 1. Likewise, $CCZ = \exp(-i\gamma 4\left|111\right\rangle\left\langle 111\right|)$ and $CCCZ = \exp(-i\gamma 4\left|1111\right\rangle\left\langle 1111\right|)$, so for each of these types of gates we add constraints which are satisfied only if all the bits involved are 1. By setting $C$ to be the sum of all these

constraints, we can see that $\exp(-i\gamma C)\,|z\rangle = (-1)^{f(z)}\,|z\rangle$. Thus, we've translated the diagonal portion of $\mathcal{C}'_f$ to the QAOA framework, and all that remains is the final column of $H$ gates. This does not translate directly into a QAOA circuit; we must use postselection. Each $H$ gate can be implemented by introducing an ancilla qubit and eight new constraints, and postselecting on the original qubit; this is described in [18].

This formula shows how the state $|\psi\rangle = |0\rangle + r\,|1\rangle$ defined in Eq. (4.2) can be created. To determine if $r$ is greater than zero, we must use postselection with a scheme similar to that from the proof of Theorem 4.3.1, except we won't be able to create the state $\alpha\,|0\rangle - \beta\,|1\rangle$ as before since rotations by arbitrary angles is not a capability of QAOA as it is for instantaneous circuits. Instead, to create the state where $\beta/\alpha = \eta^i$, we will use $i$ ancilla qubits and $i$ total $T$ gates as illustrated in the circuit in Figure 4-4.

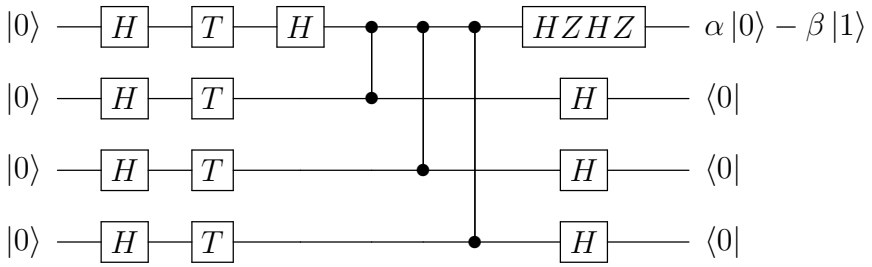


Figure 4-4: Circuit that produces the state $\alpha\,|0\rangle - \beta\,|1\rangle$ for $\beta/\alpha = \eta^i$ and $i = 4$. The final $i-1$ qubits are postselected into the $|0\rangle$ state. This circuit is implementable within the QAOA framework.

To understand the construction in Figure 4-4, first note that the state $HTH\,|0\rangle = \cos(\pi/8)\,|0\rangle + \sin(\pi/8)\,|1\rangle$. By inserting an $HH = I$ into the circuit on each of the final $i-1$ qubits before any of the $CZ$ gates, it's clear how the circuit is equivalent to one in which each qubit starts in the state $\cos(\pi/8)\,|0\rangle + \sin(\pi/8)\,|1\rangle$, and then CNOT gates are applied sequentially on each of the final $i-1$ qubits controlled by the first qubit. By postselecting these final $i-1$ qubits into the $|0\rangle^{\otimes(i-1)}$ state, we send the first qubit into the $\cos^i(\pi/8)\,|0\rangle + \sin^i(\pi/8)\,|1\rangle$ state. Finally, by hitting the first qubit with 2 $Z$ and 2 $H$ gates in the order $Z$, $H$, $Z$, then $H$ (which equals

$Z$ then $X$), we create the state $\sin^i(\pi/8)\,|0\rangle - \cos^i(\pi/8)\,|0\rangle = \alpha\,|0\rangle - \beta\,|1\rangle$ where $\beta/\alpha = \cot(\pi/8)^i = \eta^i$, as $\cot(\pi/8) = 1 + \sqrt{2}$.

The circuit in Figure 4-4 can be implemented in the QAOA framework with $\gamma = \pi/4$ simply by adding more constraints to the Hamiltonian $C$ and performing the $H$ gates by introducing an ancilla qubit, eight more constraints, and then postselecting. We have already seen how $CZ$ gates are implemented. The $T$ and $Z$ gates are implementable since $T = \exp(-i\gamma\,|1\rangle\langle 1|)$ and $Z = \exp(-i\gamma 4\,|1\rangle\langle 1|)$.

Thus, we replace the portion of the circuit in Figure 4-3 that creates $\alpha\,|0\rangle - \beta\,|1\rangle$ with the circuit in Figure 4-4, and the rest of the proof proceeds identically to that of Theorem 4.3.1.

Finally, we count the total number of qubits and constraints needed to implement this algorithm. For qubits, we have the $n+1$ qubits from the circuit $\mathcal{C}'_f$, plus $n$ qubits which arise to perform the final column of $H$ gates on the first $n$ qubits. We also need $i$ ancillas to create the state $\alpha\,|0\rangle - \beta\,|1\rangle$ and another $i+2$ ancillas to perform the $i+2$ $H$ gates within that construction (not including the ones at the very beginning, which are built into QAOA). Finally, there are 3 internal $H$ gates needed to create the state $|\psi'\rangle$ from $|\psi\rangle$ and $\alpha\,|0\rangle - \beta\,|1\rangle$, as in Figure 4-3. This gives a total of $2n+2i+6$ qubits. Since $i \leq n$, this gives at most $4n+6$ qubits.

The number of constraints we need for the $\mathcal{C}'_f$ portion of the overall circuit is 4 for each of the at most $2n$ terms in the polynomial $f$ plus 8 for each of the $H$ gates among the first $n$ qubits in $\mathcal{C}'_f$, for a total of $16n$ constraints. The preparation of $\alpha\,|0\rangle - \beta\,|1\rangle$ costs $i$ constraints for the $i$ $T$ gates, $4(i-1)$ constraints for the $i-1$ $CZ$ gates, $8(i+2)$ constraints for the $i+2$ $H$ gates, plus an additional 8 constraints for the 2 $Z$ gates, for a total of $13i+20$ constraints. Finally, creating $|\psi'\rangle$ from $|\psi\rangle$ and $\alpha\,|0\rangle - \beta\,|1\rangle$ costs 24 constraints for the 3 $H$ gates and 4 constraints for the $CZ$ gate, giving us a grand total of at most $29n+48$ constraints after imposing $i \leq n$. This proves the theorem as stated. □

**Theorem 4.3.4.** *There exists a family of postselected* QAOA *circuits* $\{\mathcal{Q}_i\}_{i=0}^{n-1}$ *as in Theorem 4.3.3 such that with* $n \cdot L$ *approximate samples up to multiplicative parameter* $\epsilon < 2/7$*, the problem MAJ-ZEROS can be solved with chance of failure* $n\exp(-(2-$

$7\epsilon)^2 L/50)$. *Thus, even in the case of approximate sampling, only* $O(\log(n))$ *samples are needed to achieve arbitrarily small error probability.*

*Proof.* The proof is the same as Theorem 4.3.2. □

### 4.3.3 Moving from PostBPTIME to $\Sigma_3$TIME

We have constructed PostIQP and PostQAOA circuits that, using only a polynomial number of samples, allow us to solve the PP-complete problem MAJ-ZEROS up to bounded error. With the ability to postselect, these samples could be produced trivially using a machine with the capability to perform IQP or QAOA circuits. However, we can also produce these samples using a postselected classical computer with the ability to weakly simulate QAOA or IQP circuits. Theorem 4.3.2 tells us that this is still true even if the classical computer can only approximately weakly simulate the circuits up to some multiplicative factor $\epsilon$, as long as $\epsilon < 2/7$.

We won't deal with specific simulation algorithms; rather, we will assume that there exists a classical weak simulation algorithm for IQP or QAOA circuits containing $a$ gates (or constraints, in the case of QAOA) on $b$ qubits that runs in time $s(a, b)$. However, it is reasonable to assume that the simulation time should be linear in the number of gates, so we simplify $s(a, b)$ to $a \cdot s(b)$.

**Theorem 4.3.5.** *Given the ability to classically approximately weakly simulate* IQP *or* QAOA *circuits with* $a$ *gates (or constraints) over* $b$ *qubits in time* $a \cdot s(b)$ *up to multiplicative parameter* $\epsilon$*, there is a postselected bounded-error algorithm for MAJ-ZEROS with runtime* $g \cdot nL \cdot s(q)$ *where*

$$\begin{aligned} g &= 4n + 13 \\ q &= n + 4 \\ L &= 50 \log(3n)/(2 - 7\epsilon)^2 \end{aligned} \tag{4.4}$$

*in the case of* IQP *circuits and*

$$\begin{aligned} g &= 29n + 48 \\ q &= 4n + 6 \\ L &= 50\log(3n)/(2-7\epsilon)^2 \end{aligned} \tag{4.5}$$

*in the case of* QAOA *circuits. In other words, MAJ-ZEROS* $\in$ PostBPTIME($g \cdot nL \cdot s(q)$).

*Proof.* We use the classical simulation algorithm to produce the $nL$ samples from circuits $\mathcal{Q}_i$ for different values of $i$ and post-process them as described in the proof of Theorem 4.3.1, giving a solution to MAJ-ZEROS which fails with probability at most $n\exp(-(2-7\epsilon)^2 L/50) = 1/3$. □

We could stop here and conjecture that MAJ-ZEROS $\notin$ PostBPTIME($h(n)$), that is, that there is no postselected bounded-error algorithm for MAJ-ZEROS running in less than $h(n)$ time, for some super-polynomial function $h$, such as $h(n) = 2^{n/5}$. This would generate a lower bound on the simulation time $s(b)$ for a circuit with $b$ qubits.

However, it's not clear how to judge for what function $h(n)$ this conjecture would be reasonable. If $h(n)$ were a polynomial, this conjecture would be believable since its falsity would imply MAJ-ZEROS $\in$ PostBPP $\subseteq$ PH, causing the collapse of the PH. On the other hand, we know of a classical $O(\text{poly}(n)2^n)$-time algorithm for MAJ-ZEROS which operates by iterating through all $2^n$ possible settings of $z$ and computing whether $f(z) = 0$ in order to compute gap($f$). In fact, this algorithm does not utilize postselection or randomness at all, so it is not believable that no better algorithm exists; the conjecture for $h(n) = 2^n$ is probably not true.

By changing our algorithm from PostBPTIME to $\Sigma_3$TIME we change the form of our conjecture to MAJ-ZEROS $\notin \Sigma_3\text{TIME}(h(n))$. Since the polynomial hierarchy has been studied much more deeply than the class PostBPP, it will be easier for us to understand for what $h(n)$ our conjecture is believable.

**Theorem 4.3.6.** *If a problem $\mathcal{P}$ has a postselected bounded-error algorithm running in time $h(n)$, it has a* $\Sigma_3$TIME *algorithm running in $c(h(n))$ where*

$$c(x) = \log(kx)(kx)\Big(4(kx)^2 + 6(kx) + 9\Big) + \log(kx)^2 \tag{4.6}$$

*and $k = \max(16, 4\log(8x)/3)$. In other words, $\mathcal{P} \in$ PostBPTIME$(h(n))$ implies $\mathcal{P} \in \Sigma_3\text{TIME}(c(h(n)))$.*

**Corollary 4.3.6.1.** *If it is possible to weakly simulate* IQP *or* QAOA *circuits as in Theorem 4.3.5, MAJ-ZEROS $\in \Sigma_3\text{TIME}(c(g \cdot nL \cdot s(q)))$ where $n$ is the number of variables in the input polynomial and $g$, $q$, and $L$ are given by Eqs. (4.4) for* IQP *circuits or Eqs. (4.5) for* QAOA *circuits.* □

The full proof of Theorem 4.3.6 is contained in Appendix A.1, but here we mention the main idea.

*Proof idea for Theorem 4.3.6.* Let the postselected probabilistic machine that solves $\mathcal{P}$ be denoted $M$, which acts on the length-$n$ input to the problem $w$ and a random string $r$ of length equal to the runtime $h(n)$ of $M$. Let $A(w)$, $R(w)$, and $C(w)$ be the sets of strings $r$ such that $M(w, r)$ accepts, rejects, and cancels, respectively. Then, the bounded-error condition reads: if $\mathcal{P}(w) = 1$ then $|A(w)| \geq 2|R(w)|$, and if $\mathcal{P}(w) = 0$ then $2|A(w)| \leq |R(w)|$. The idea behind the proof is to approximate $|A(w)|$ and $|R(w)|$ up to a multiplicative factor of $\sqrt{2}$, allowing us to determine which is larger and subsequently output the correct answer to $\mathcal{P}(w)$. It is a long-established fact that approximate counting in this sense can be done in the third level $\Sigma_3$TIME in time that is polynomial in the runtime of $M$, but the exact form of that polynomial is rarely computed [36]. In the appendix, we carry through this calculation and explicitly describe the approximate counting algorithm. The algorithm tests whether there exists a set of $j$ linear hash functions $\{H_i\}_{i=1}^{j}$ that compress the random inputs $r$ down to $j$ bits such that at least one of the hash functions causes no collisions for various values of $j$. The maximum $j$ for which this is possible communicates information about $|A(w)|$ and $|R(w)|$, and testing whether such a set exists can be performed efficiently using a $\Sigma_2$P oracle, putting the algorithm in $\Sigma_3$P.

### 4.3.4 Conclusion and result

Corollary 4.3.6.1 gives us an implicit algorithm for solving MAJ-ZEROS in the third level of the TIME hierarchy. If we conjecture that MAJ-ZEROS $\notin \Sigma_3\text{TIME}(h(n))$ for some function $h(n)$, this will imply a lower bound on the simulation time $s(b)$ for a circuit over $b$ qubits.

Since a polynomial third-level algorithm for MAJ-ZEROS would cause the collapse of the polynomial hierarchy, a polynomial $h(n)$ would lead to a believable conjecture, but also only a polynomial lower bound on the simulation time. On the other hand, as we mentioned in the discussion about conjectures relating to PostBPTIME, there is a naive algorithm for MAJ-ZEROS that runs in time $O(\text{poly}(n)2^n)$ that doesn't use the added power of being in the third level, so $h(n) = 2^n$ is probably not a believable conjecture.

However, since we have moved the discussion to the TIME hierarchy, we have a much better framework from which to discuss such conjectures. A similar discussion has been had with regard to the separation between classes P and NP. Not only is it widely believed that NP-complete problems like three-coloring are not in P, it is also generally believed that there exists a constant $c$ such that these problems are not in the class $\text{TIME}(2^{cn})$, that is, some problems in NP require exponential time. This statement is called the *exponential time hypothesis* (ETH). The *strong exponential time hypothesis* (SETH) goes further, stating that some problems in NP are not in $\text{TIME}(2^{cn})$ for any $c < 1$, that is, they require $2^n$ time.

Recall that P is the zeroth level of the polynomial hierarchy and NP is the first level. Our discussion instead centers around a separation between the third level and the class PP, which in a certain sense is more powerful than the entire PH since $\text{PH} \subseteq \text{P}^{\text{PP}}$.[1] Since the relationship between P and NP resembles the relationship between adjacent levels of the PH, the work surrounding ETH and SETH gives some

[1]The relation between PP itself and the PH is complicated, and largely unknown. We know that PP contains NP. However, there is evidence, in work by Beigel et al. [5] separating $\text{P}^{\text{NP}}$ from PP with respect to an oracle, that PP does not contain any level of the PH above NP.

force to the notion that we should likewise believe conjectures claiming exponential separation between PP and the third level of the hierarchy.

We augment this argument with the following fine-grained oracle separation between PP and $\Sigma_3$TIME. While an oracle separation is not a proof that two classes are not equal, it does show that any proof purporting to claim that they are equal must be non-relativizing; many of the most effective proof techniques in complexity theory are necessarily relativizing.

**Theorem 4.3.7.** *There is an oracle $A$ such that* $\mathrm{PP}^A \not\subset \Sigma_3\mathrm{TIME}^A(\frac{2^{n/5}}{92} - \frac{n}{8})$.

The proof, which utilizes known lower bounds on the size of classical circuits that compute the majority function, is left for Appendix $A.2$.

Inspired by this oracle separation, we make the following conjecture.

**Conjecture 4.3.1.** *MAJ-ZEROS* $\notin \Sigma_3\mathrm{TIME}(h(n))$ *where* $h(n) = \frac{2^{n/5}}{92} - \frac{n}{8}$.

Conjecture 4.3.1 combined with Corollary 4.3.6.1 implies that $c(g{\cdot}nL{\cdot}s(q)) > h(n)$, an implicit lower bound on the function $s$, as stated in the following corollary.

**Corollary 4.3.7.1.** *Assume conjecture 4.3.1. If there exists a classical algorithm that can approximately simulate* IQP *or* QAOA *circuits with $a$ gates (or constraints) over $b$ qubits up to multiplicative parameter $\epsilon$ in time $a \cdot s(b)$, the function $s$ must satisfy*

$$c(g \cdot nL \cdot s(q)) > \frac{2^{n/5}}{92} - \frac{n}{8} \tag{4.7}$$

*where $g$, $q$, and $L$ are given by Eqs. (4.4) for* IQP *circuits, and by Eqs. (4.5) for* QAOA *circuits.* □

Corollary 4.3.7.1 is the main result of this chapter. The form of the algebra makes it difficult to express the lower bound succinctly in a form $s(b) > t(b)$ for some function $t$, but we can numerically solve for and plot the resulting bound. For $\epsilon = 0$, this is shown in Figure 4-5.

The fastest supercomputers today can perform at $10^{17}$ FLOPs (floating-point operations per second).[2]f We can numerically determine from the plot in Figure 4-5

[2]A list of the fastest supercomputers is maintained at `https://www.top500.org/statistics/list/`

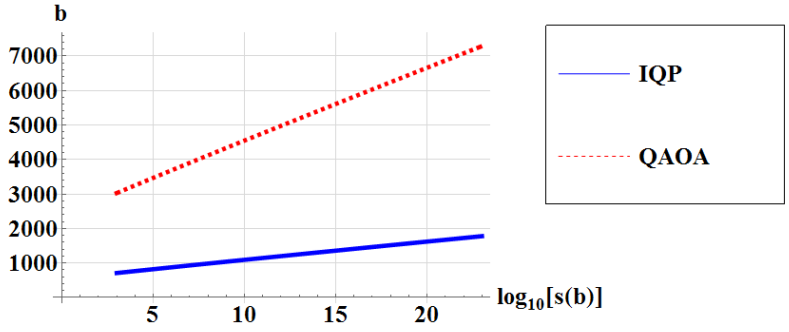


Figure 4-5: Numerically calculated lower bound on the function $s(b)$ when $\epsilon = 0$ giving the time per gate necessary to simulate an IQP or QAOA circuit over $b$ qubits.

the number of qubits $b$ such that the lower bound on $s(b)$ is equal to the maximum number of operations today's supercomputers can perform in one day. For IQP circuits, it is roughly 1700, and for QAOA circuits, it is roughly 7100. Since the total simulation time is $a \cdot s(b)$ where $a$ is the number of gates (or constraints), in these scenarios, it would take the most powerful supercomputer at least an entire day *for each gate* to simulate IQP or QAOA circuits with this many qubits. We believe this is a good measure of intractability.

Letting $\epsilon$ be positive has little effect on these numbers. Even when $\epsilon$ is as large as 0.1, the corresponding numbers increase by only 20 qubits for IQP circuits, bringing the total closer to 1750 qubits, and by only 80 qubits for QAOA circuits, bringing the total to closer to about 7150 qubits. This makes sense since positive $\epsilon$ leads to a polynomial amount of overhead, and since the bound on $s(b)$ is exponential, this polynomial overhead should only cost an additional small logarithmic number of qubits.

# Chapter 5

# Lower bounds for simulation of Clifford + $T$ circuits

## 5.1 Outline of lower-bounds argument

In the previous chapter, we showed how imposing a fine-grained complexity theoretic assumption leads to lower bounds on the simulation time for instantaneous and QAOA circuits. In this chapter, we present a nearly identical argument for lower bounds on the simulation time of Clifford + $T$ circuits. The main difference, therefore, is that the circuit we build to solve the MAJ-ZEROS problem will be a Clifford + $T$ circuit, and the fundamental resource we are counting is the number of $T$ gates, not the number of qubits. Whereas before our simulation time was $a \cdot s(b)$ for a circuit with $a$ gates and $b$ qubits, now our simulation time will be $\text{poly}(a, b) \cdot s(t)$ where $t$ is the number of $T$ gates. Throughout this section, both $T$ and $T^\dagger$ gates will be included whenever we count the number of $T$ gates in a circuit.

## 5.2 Derivation of lower bounds

### 5.2.1 Clifford + $T$ circuit for solving MAJ-ZEROS

**Theorem 5.2.1.** *For a degree-3 polynomial $f$ on $n$ variables where each variable appears in at most three terms, there exists a family of postselected quantum circuits $\{\mathcal{Q}_i\}_{i=0}^{n-1}$ made up solely of Clifford gates and $T$ gates, each acting on $q = 4n + 3$ qubits and containing at most $t = 9n$ $T$ gates as well as $g = 34n + 6$ Clifford gates, such that taking $n \cdot L$ samples from $\mathcal{Q}_i$ for different values of $i$ allows us to solve MAJ-ZEROS for the input $f$ with error probability $n\exp(-2L/25)$. Thus, we require only $O(n\log(n))$ samples to achieve arbitrarily small error probability.*

*Proof.* We use the same basic circuit as in our proofs of Theorem 4.3.1 and Theorem 4.3.3. The creation of the state $|\psi\rangle$ by circuit $\mathcal{C}'_f$ involves $H$, $CZ$, $CCZ$, and $CCCZ$ gates. The $H$ and $CZ$ gates are in the Clifford + $T$ gate set. The $CCZ$ gates can be decomposed into Clifford + $T$ using only four $T$ gates using the construction from [25], which also describes how this circuit can be embedded within itself to perform a $CCCZ$ gate with eight $T$ gates. We reproduce this decomposition in Figure 5-1. In [24], it is shown that four $T$ gates is optimal.

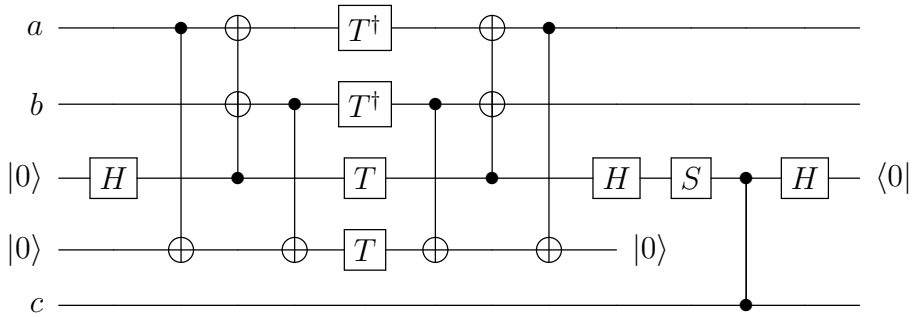


Figure 5-1: Decomposition of a $CCZ$ gate into postselected Clifford + $T$ gates, from [25], where we have used postselection instead of a corrective conditional gate. The circuit negates the input only if qubits $a$, $b$, and $c$ are all $|1\rangle$. It requires two ancilla qubits and uses four $T$ gates. $CCU$ gates for any gate $U$ can be constructed by replacing the $CZ$ gate in this circuit with a $CU$ gate, where the third qubit is the control and fifth qubit is the target. Therefore, we can implement a $CCCZ$ gate by replacing the $CZ$ gate with this circuit, embedding it inside of itself to create a circuit with eight $T$ gates. In [24], a lower bound of four $T$ gates is shown for such a decomposition, so the circuit is optimal.

In the worst case, there are $n$ terms in the polynomial $f$, each with three variables (recall that we can assume each variable appears at most three times). Thus, creating $|\psi\rangle$ requires $2n+1$ external $H$ gates, plus 8 $T$ gates, 16 CNOT gates, 6 $H$ gates, 2 $S$ gates, and a single $CZ$ gate for each of the $n$ terms. This gives an overall total of $8n$ $T$ gates, $16n$ CNOT gates, $8n+1$ $H$ gates, $2n$ $S$ gates, and $n$ $CZ$ gates.

To create the state $\alpha|0\rangle - \beta|1\rangle$ for $\beta/\alpha = \eta^i$, we use the construction in Figure 4-4. This requires $i$ $T$ gates, $2i+2$ $H$ gates, 2 $Z$ gates, and $i-1$ $CZ$ gates.

Finally, to create the state $|\psi'\rangle$ from $|\psi\rangle$ and $\alpha|0\rangle - \beta|1\rangle$, we need 3 $H$ gates and 1 $CZ$ gate, or by writing $H_j CZ_{ij} H_j = CX_{ij}$, just a single $H$ gate and a single CNOT gate.

If we impose $i \leq n$ we can find an upper bound on the number of each gate needed: $9n$ $T$ gates, $16n+1$ CNOT gates, $2n-1$ $CZ$ gates, $10n+4$ $H$ gates, $2n$ $S$ gates, and 2 $Z$ gates. If we require that $CZ$ be implemented with two $H$ gates and a CNOT, and that $Z$ be implemented as $S^2 = Z$, then the number of CNOT gates becomes $18n$, the number of $H$ gates becomes $14n+2$ and the number of $S$ gates becomes $2n+4$. Thus we have a total of $9n$ $T$ gates, and $34n+6$ other gates.

The number of qubits needed to create $|\psi\rangle$ is $n+1$ plus ancillas incurred in order to implement $CCZ$ and $CCCZ$ gates as in the construction in Figure 5-1. If we make the worst case assumption that every term has three variables, then each term requires four ancilla qubits. However, since the fourth qubit in Figure 5-1 ends up back in the $|0\rangle$ state, it can be reused to implement future $CCZ$ or $CCCZ$ gates. Since we have embedded the circuit within itself, there will be two reusable qubits, so we need a total of $2n$ non-reusable ancillas and 2 reusable ancillas to implement all $n$ terms.

To create $\alpha|0\rangle - \beta|1\rangle$ we need $i \leq n$ qubits, and creating $|\psi'\rangle$ from $|\psi\rangle$ and $\alpha|0\rangle - \beta|1\rangle$ requires no extra qubits. The total number of qubits is at most $4n+3$. $\square$

As before, the theorem extends to the approximate case simply by taking more samples.

**Theorem 5.2.2.** *There exists a family of quantum circuit* $\{\mathcal{Q}_i\}_{i=0}^{n-1}$ *as in Theorem 5.2.1 such that with* $n \cdot L$ *approximate samples up to multiplicative parameter* $\epsilon < 2/7$*, the problem MAJ-ZEROS can be solved with chance of failure* $n\exp(-(2-7\epsilon)^2 L/50)$*. Thus, even in the case of approximate sampling, only* $O(\log(n))$ *samples are needed to achieve arbitrarily small error probability.*

### 5.2.2 Conclusion and result

We assume the ability to weakly simulate a Clifford + $T$ circuit with $\tau$ $T$ gates, $a$ Clifford gates, and $b$ qubits up to multiplicative factor $\epsilon$ in time $\text{poly}(a, b, 1/\epsilon) \cdot s(\tau)$. This expression separates any exponential dependence solely to the number of $T$ gates, which is reasonable since Clifford circuits alone are simulable in polynomial time. Additionally, simulation algorithms have been found that have runtime expressed in this form, such as in [7], where $s(\tau) \approx 2^{0.228\tau}$.

Paralleling Theorem 4.3.5, we have the following.

**Theorem 5.2.3.** *Given the ability to classically approximately weakly sample Clifford + T circuits with* $\tau$ $T$ *gates,* $a$ *Clifford gates, and* $b$ *qubits in time* $poly(a, b, 1/\epsilon) \cdot s(\tau)$ *up to multiplicative parameter* $\epsilon$*, there is a postselected bounded-error algorithm for MAJ-ZEROS with runtime* $poly(g, q, 1/\epsilon) \cdot nL \cdot s(t)$ *where*

$$\begin{aligned} t &= 9n \\ g &= 34n + 8 \\ q &= 4n + 3 \\ L &= 50\log(3n)/(2-7\epsilon)^2 \end{aligned} \tag{5.1}$$

*In other words, MAJ-ZEROS* $\in$ PostBPTIME($poly(g, q, 1/\epsilon) \cdot nL \cdot s(t)$). □

Then, paralleling Corollary 4.3.6.1, we conclude from Theorem 4.3.6 the following.

**Corollary 5.2.3.1.** *If it is possible to weakly simulate Clifford + T circuits as in Theorem 5.2.3, MAJ-ZEROS* $\in \Sigma_3$TIME($c(poly(g, q, 1/\epsilon) \cdot nL \cdot s(t))$) *where* $n$ *is the*

*number of variables in the input polynomial, the function c is given by Eq. (4.6), and g, q, and L are given by Eqs. (5.1).* □

Thus, by imposing Conjecture 4.3.1, we arrive at the lower bound.

**Corollary 5.2.3.2.** *Assume Conjecture 4.3.1. If there exists a classical algorithm that can approximately simulate Clifford + T circuits with $\tau$ T gates, a Clifford gates and b qubits up to multiplicative parameter $\epsilon$ in time poly$(a, b, 1/\epsilon) \cdot s(t)$, then the function s must satisfy*

$$c(\text{poly}(g, q, 1/\epsilon) \cdot nL \cdot s(t)) > \frac{2^{n/5}}{92} - \frac{n}{8} \tag{5.2}$$

*where t, g, q, and L are given by Eqs. (5.1), and c is given by Eq. (4.6).* □

In Chapter 4 we used our lower bound to estimate the number of qubits needed for simulating the circuit to be classically hard. For Clifford + $T$ circuits, we will instead use the lower bound to state an asymptotic lower bound on the $t$-dependence of any Clifford + $T$ simulation algorithm.

**Corollary 5.2.3.3.** *Assume Conjecture 4.3.1. If a classical algorithm weakly simulates Clifford circuits with $\tau$ T gates, a Clifford gates, and b qubits up to multiplicative parameter $\epsilon$ in time poly$(a, b, 1/\epsilon) \cdot 2^{\gamma\tau}$, then $\gamma > 1/135 \approx 0.0074$.*

*Proof.* The simulation algorithm must satisfy Eq. (5.2) with $s(\tau) = 2^{\gamma\tau}$. The asymptotic dependence of the function $c$ is roughly $c(x) \approx x^3$ ignoring constants and logarithmic factors. This can be stated precisely by noting that for any constant $\delta > 0$, there exists a constant $k$ such that $c(x) < kx^{3+\delta}$ for sufficiently large $x$. By including $k$ within the function poly$(g, q, 1/\epsilon)$ and substituting $9n$ for $t$, we conclude that for sufficiently large $n$, the left-hand-side of Eq. (5.2) is less than poly$(n, 1/\epsilon) \cdot 2^{(3+\delta)\gamma(9n)}$. This must be greater than the right-hand-side which can be written as poly$(n) \cdot 2^{n/5}$. If $\gamma$ were less than or equal to $1/(3 \cdot 9 \cdot 5) = 1/135$, then for sufficiently large $n$, this inequality would be broken since for positive $\delta$, the left-hand-size would exhibit faster exponential growth. Thus, $\gamma > 1/135$. □

This is the main result of the chapter. Under the purely classical complexity theoretic assumption in Conjecture 4.3.1, we have concluded a lower bound on the exponential dependence of any multiplicative approximate simulation of Clifford + $T$ circuits. The best known classical algorithm of this form is by Bravyi and Gosset [7] and has $\gamma \approx 0.228$ as discussed in Section 2.4.5. While our lower bound is not extremely tight, we are unaware of any previously derived lower bounds for such algorithms.

The factor 135 can be simply understood as the product of three factors. A factor of 5 comes from the exponent in Conjecture 4.3.1, which we believe is reasonable due to the oracle separation in Theorem 4.3.7. A factor of 3 comes from the fact that the runtime gets roughly cubed when we convert from PostBPTIME to $\Sigma_3$TIME described by Theorem 4.3.6. Finally, a factor of 9 comes from the fact that there are $9n$ $T$ gates in the Clifford + $T$ circuit to solve MAJ-ZEROS from Theorem 5.2.1. Improving any of these three aspects of the result would directly improve the lower bound $\gamma > 1/135$.

In a sense, the main point can be thought of as a connection between Clifford + $T$ simulation and a $\Sigma_3$TIME algorithm for MAJ-ZEROS, the latter of which is purely classical. Even if Conjecture 4.3.1 were to fail in favor of a weaker conjecture, we would still have a (weaker) lower bound on Clifford + $T$ simulation. We can restate our result as the observation that if we had a Clifford + $T$ simulation algorithm with $T$-gate dependence $2^{\gamma\tau}$ such that $\gamma \leq 1/135$, we would also have a $\Sigma_3$TIME algorithm for MAJ-ZEROS that we expect not to exist.

# Chapter 6

# Conclusions and future directions

## 6.1 Bottlenecks and places for improvement

The novelty of our result lies in the fact that we are able to generate a fine-grained lower bound on the simulation time for IQP, QAOA, and Clifford + $T$ circuits, but there is no indication that these lower bounds are tight. Improvements to our analysis as well as completely different methods could generate an improved version of our results.

There are three main ways our analysis could be improved, corresponding to the three main steps in our result.

First, the lower bound on the runtime of any $\Sigma_3$TIME algorithm for MAJ-ZEROS expressed in conjecture 4.3.1 could be made tighter. Our lower bound scales roughly like $2^{n/5}$, but we could conjecture other functions $2^{kn}$ with $k > 1/5$. To support such a change, we would either need to find a tighter oracle separation, or some other means for motivating the new lower bound. The exponential dependence of our simulation lower bound is directly proportional to $k$. Improving $k$ by a certain factor $\alpha$ would decrease our qubits estimate in Section 4.3.4 by roughly $\alpha$, and increase our lower bound on $\gamma$ in Section 5.2.2 by exactly $\alpha$. However, since we know the Conjecture is not true when $k = 1$, we cannot hope to improve $k$ by more than a factor of 5.

Second, our choice of PP-complete problem and circuit construction to solve that problem with postselected IQP, QAOA, or Clifford + $T$ circuits could be improved.

For example, an earlier version of this work used the PP-complete satisfiability problem MAJ-SAT instead of MAJ-ZEROS, and the move to MAJ-ZEROS improved our results by nearly an order of magnitude. Could a wiser choice of problem yield an even better result? Since the number of ancilla qubits required to solve MAJ-ZEROS with IQP circuits does not grow with $n$, it seems to be close to the best we can do in this case. However, for QAOA circuits, $3n$ ancilla qubits are required to solve MAJ-ZEROS, leading to the factor of 4 difference between the expressions for $g$ in Eqs. (4.4) and Eqs. (4.5). This accounts for the roughly factor of 4 difference between our qubit estimates for IQP and QAOA in Section 4.3.4. Perhaps a more clever construction for solving MAJ-ZEROS or a better choice of problem would allow for a QAOA circuit needing only constant ancillas, bringing our result for QAOA closer to our result for IQP. Similarly, our Clifford + $T$ construction requires $9n$ $T$ gates, contributing a factor of 9 to the lower bound on $\gamma$. Optimizing this construction, or finding a different PP-complete problem with a more natural Clifford + $T$ construction could therefore improve our estimate.

Lastly, the reduction from PostBPTIME to $\Sigma_3$TIME encapsulated by Theorem 4.3.6 causes the runtime of the algorithm to increase according to the function $c$ from Eq. (4.6). Since $c(x)$ is roughly $x^3$ and the simulation time is exponential, this contributes roughly a factor of 3 to our analysis, both in terms of number of qubits for IQP and QAOA and for our lower bound on $\gamma$. Making improvements at this step is more likely to pose difficulty since the proof method for Theorem 4.3.6 involving hash functions seems to require significant overhead. Perhaps a different proof method, however, could improve this part of the analysis.

Ultimately, we see that there is some room for improvement on our result, but that, especially for IQP, we are unlikely to improve it considerably without significantly modifying our process.

## 6.2 Impact of result

We emphasize both a practical and theoretical way in which our results are impactful. Practically, our results in Chapter 4 give a rough estimate for how large IQP and QAOA circuits must be to guarantee that sampling their output distributions is classically intractable. In both cases, thousands of qubits are needed, which is considerably more than what is possible on current technology. While this estimate could be improved, we gain a rough idea of how many will be needed for future quantum supremacy demonstrations. As technology approaches this threshold, these estimates will be valuable for determining when quantum supremacy has actually been demonstrated. In the meantime, the differences between these estimates might help experimentalists decide which quantum supremacy avenues are most likely to be successful.

Theoretically, our results demonstrate a more fine-grained connection between the runtime of classical algorithms simulating quantum circuits and the runtime of classical third-level algorithms for the PP-complete problem MAJ-ZEROS. Previously, it had been shown that a polynomial simulation algorithm implies a polynomial $\Sigma_3$P algorithm for MAJ-ZEROS (and other PP problems), but our construction more precisely defines the relationship between the two runtimes and defines explicitly how the algorithm operates (as opposed to simply stating that it exists). Our lower bounds on the simulation time of IQP, QAOA, and Clifford + $T$ represent an important theoretical advance toward understanding what simulation time is optimal. This is particularly relevant for Clifford + $T$ simulation, since any quantum circuit can be decomposed as a Clifford + $T$ circuit, and many experimental systems aim to operate within that framework.

## 6.3 Future work

There are several directions that this work could go in the future. As mentioned, there is room to optimize and refine the process to achieve better results. Based on

our analysis of the bottlenecks in the argument, the most likely way to do this is to find more natural PP-complete problems for QAOA or Clifford + $T$ circuits or more efficient constructions to solve the MAJ-ZEROS problem. Additionally, more thought could be put into strengthening Conjecture 4.3.1.

Moreover, it should be possible to extend this process to other quantum supremacy proposals. For example, a similar analysis of the BosonSampling problem should allow for an estimation of the number of photons needed to guarantee that classically sampling the output distribution is intractable. For quantum supremacy avenues relying on other lines of complexity theoretic reasoning, such as the random circuits proposal, perhaps the high-level idea of imposing fine-grained conjectures to conclude fine-grained results could be fruitful, if not the exact details of our argument.

Finally, our results only provide a lower bound on the simulation time for approximate weak simulations in the multiplicative sense. Additive approximations are more general, and perhaps more attainable experimentally, but existing quantum supremacy arguments require additional conjectures to guarantee that simulation is still hard [2, 10]. In a future extension of this work, it would be interesting to apply our ideas to the additive case.

# Appendix A

# Proofs

## A.1 Proof of Theorem 4.3.6: $\Sigma_3$TIME algorithm for problems in PostBPTIME

**Theorem 4.3.6.** *If a problem $\mathcal{P}$ has a postselected bounded-probability algorithm running in time $h(n)$, it has a $\Sigma_3$TIME algorithm running in $c(h(n))$ where*

$$c(x) = \log(kx)(kx)\Big(4(kx)^2 + 6(kx) + 9\Big) + \log(kx)^2 \tag{A.1}$$

*and $k = \max(16, 4\log(8x)/3)$. In other words, $\mathcal{P} \in$ PostBPTIME$(h(n))$ implies $\mathcal{P} \in \Sigma_3\text{TIME}(c(h(n)))$.*

*Proof.* The result from [23] that PostBPP $= \text{BPP}_{\text{PATH}} \subseteq \text{P}^{\Sigma_2\text{P}} \subseteq \Sigma_3\text{P}$ implies that the conversion from PostBPTIME to $\Sigma_3$TIME should be possible, and our proof takes on a similar strategy. These ideas rely heavily on the concept of Stockmeyer (approximate) counting, which was first introduced in [36].

Let the postselected probabilistic machine that solves $\mathcal{P}$ be denoted $M$, which acts on the length-$n$ input to the problem $w$ and a random string $r$ of length equal

to the run time $h(n)$ of $M$. Additionally, we define

$$\begin{aligned} A(w) &= \{r \text{ such that } M(w,r) = 1\} \\ R(w) &= \{r \text{ such that } M(w,r) = 0\} \\ C(w) &= \{r \text{ such that } M(w,r) = \text{ cancel}\} \end{aligned} \tag{A.2}$$

The bounded-error condition tells us that if $\mathcal{P}(w) = 1$ then $|A(w)| \geq 2|R(w)|$, and if $\mathcal{P}(w) = 0$ then $2|A(w)| \leq |R(w)|$. If we can approximate $|A(w)|$ and $|R(w)|$ up to a multiplicative factor of $\sqrt{2}$ we can determine which is larger and subsequently output the correct answer to $\mathcal{P}(w)$. We do so as follows.

Consider a new machine $M_k$ that acts on $w$ along with a random string $r$ of length $k \cdot h(n)$ by simply running the algorithm $M$ $k$ times. $M_k$ accepts only if all $k$ iterations of algorithm $M$ accept, rejects only if all $k$ iterations of algorithm $M$ reject, and cancels the computation otherwise. If $A_k$, $R_k$, and $C_k$ are defined as in Eq. (A.2), except using $M_k$ in place of $M$, then $|A_k(w)| = |A(w)|^k$ and $|R_k(w)| = |R(w)|^k$ since $M_k$ accepts (rejects) any $k \cdot h(n)$ length string $r$ only if each of the $k$ length-$h(n)$ substrings of $r$ is in $A(w)$ ($R(w)$). Let $k = \max(16, 4\log(8h(n))/3)$, for reasons that will make sense later.

We now describe an oracle called $\text{HASH}_A$ that allows $|A_k(w)|$ to be approximated using time polynomial in $h(n)$. The input to $\text{HASH}_A$ is the string $w$ and an integer $j$, and is defined by

$$(\exists\{H_1, H_2, \ldots, H_j\})\ (\forall r \in A_k(w))\ (\exists i \leq j)\ (\forall s \in A_k(w), s \neq r)\ H_i(s) \neq H_i(r) \tag{A.3}$$

where each $H_i$ is a linear hash function from $k \cdot h(n)$ bits to $j$ bits in the form of a $j$ by $k \cdot h(n)$ matrix of 0s and 1s. In words, the oracle outputs 1 if, given the integer $j$, there exists a set of $j$ linear hash functions such that for all elements $r_k$ of $A_k(w)$, we can choose one of these hash functions $H_i$ and be guaranteed that $H_i$ creates no collisions with $r_k$, that is, the hash function $H_i$ sends no other string $s_k$ to the same compressed string $H_i(r_k)$. Intuitively, the larger $j$ is, the more likely $\text{HASH}_A$ is to be true on input

$(w, j)$, since it is easier to construct a hash function yielding no collisions if the output strings from the hash function are longer. By replacing $A_k$ with $R_k$ in Eq. (A.3), we can form the oracle $\text{HASH}_R$. We can then define the oracle HASH whose input is $w$, $j$, and a single bit of input $z$, such that $\text{HASH}(w, r, z) = \text{HASH}_A(w, r)$ if $z = 1$ and $\text{HASH}(w, r, z) = \text{HASH}_R(w, r)$ if $z = 0$.

Looking at Eq. (A.3), we notice that $\text{HASH}_A$ has four quantifiers. However, we can compress this into effectively two quantifiers since the third quantifier ranges over only a polynomial (in $h(n)$) number of possibilities, so we can remove it and combine the $\forall$ quantifiers on either side. We present the details of this conversion in the following series of expressions, where each line is equal to the line above it and equal to $\text{HASH}_A$.

$$\text{NOT}\,((\forall\{H_1, \ldots H_j\})\ (\exists r \in A_k(w))\ (\forall i \leq j)\ (\exists s \in A_k(w), s \neq r)\ H_i(s) = H_i(r))$$
$$\text{NOT}\,((\forall\{H_1, \ldots H_j\})\ (\exists r \in A_k(w))\ (\exists (s_1, \ldots, s_j) \in A_k(w)^j)\ H_i(s_i) = H_i(r)\ \forall i \leq j)$$
$$\text{NOT}\,((\forall\{H_1, \ldots H_j\})\ (\exists (r, s_1, \ldots, s_j) \in A_k(w)^{j+1})\ H_i(s_i) = H_i(r)\ \forall i \leq j)$$
$$(\exists\{H_1, \ldots H_j\})\ (\forall (r, s_1, \ldots, s_j) \in A_k(w)^{j+1}, r \neq s_i \forall i)\ (\exists i \leq j\ H_i(s_i) \neq H_i(r))$$

In this form, we see how $\text{HASH}_A$ can be computed with a $\Sigma_2$TIME machine running in time polynomial in $h(n)$: we write the last line of $\text{HASH}_A$ as $\exists u \forall v R((w, j), u, v)$ where $R$ is a deterministic algorithm operating as follows.

(1) $R$ checks that $r \neq s_i\ \forall i$, and that $(r, s_1, \ldots s_j) \in A_k(w)^{j+1}$ by running $M$ on each set of random bits. If any of these fail, $R$ accepts.

(2) $R$ computes $H_i(r)$ and $H_i(s_i)$ for each $i \leq j$ and accepts only if $H_i(s_i) \neq H_i(r)$ for at least some $i$. Otherwise $R$ rejects.

We use a model where comparing two bits takes one time step. The first step takes $k \cdot h(n) \cdot j$ steps to compare all the bits and then $k \cdot h(n) \cdot (j+1)$ steps to run $M_k$ $j + 1$ times. The second step is simply matrix multiplication and takes $k \cdot h(n) \cdot 2j^2$ time steps to compute $H_i(r)$ and $H_i(s_i)$ for each $i$, as well as $k \cdot h(n) \cdot j$ time steps to

compare the results, giving a total number of time steps

$$k \cdot h(n) \cdot (2j^2 + 3j + 1). \tag{A.4}$$

Note that if $h(n)$ were a polynomial (generally not true), then the runtime of $R$ would be a polynomial in the length of its inputs, putting $\text{HASH}_A$ in the class $\Sigma_2\text{P}$. By allowing $R$ to take an extra bit of input $z$, it can instead solve the HASH problem by using that bit to decide in step (1) whether to check if the strings are $A_k(w)$ or $R_k(w)$. This costs no additional time steps.

The crucial fact for approximate counting is the following.

**Lemma A.1.1.** *If $j_A$ is the least integer such that $HASH_A(w, j_A)$ is true, then*

$$2^{j_A - 3} \leq |A_k(w)| \leq j_A 2^{j_A}. \tag{A.5}$$

*The lemma extends similarly to $HASH_R$.*

*Proof.* The proof appears in [23], which draws from ideas presented by Stockmeyer [36], and crucially relies on Sipser's Coding Lemma for universal hashing [34]. □

Now we describe how to complete the approximate counting given access to an oracle for HASH. We attempt to find $j_A$ and $j_R$, the smallest integers such that $\text{HASH}_A(w, j_A)$ and $\text{HASH}_R(w, j_R)$ are true, respectively. This can be done by binary searching the set of integers between 1 and $k \cdot h(n)$ using the HASH oracle, requiring $2\log(k \cdot h(n))$ oracle queries. We know that $|A(w)|^k = |A_k(w)|$ and that $2^{j_A - 3} \leq |A_k(w)| \leq j_A 2^{j_A}$. We also know $j_A \leq k \cdot h(n)$. Hence we can estimate $|A(w)|$ by the quantity $\hat{A} = (j_A 2^{2j_A - 3})^{1/(2k)}$ which is approximately correct up to a factor of $(8j_A)^{1/(2k)} \leq (8k \cdot h(n))^{1/(2k)}$. Since $k \geq 16$, $k^{1/(2k)} \leq 2^{1/8}$, and since $k \geq 4\log(8h(n))/3$, $(8h(n))^{1/(2k)} \leq (8h(n))^{3/(8\log(8h(n)))} = 2^{3/8}$, meaning our approximation is good to a factor of $\sqrt{2}$. We do the same to get an estimate $\hat{R}$ for $|R(w)|$, and then accept if $\hat{A} > \hat{R}$ and reject if $\hat{R} > \hat{A}$. This gives a correct deterministic algorithm for the problem $\mathcal{P}$, but requires access to the HASH oracle. It will henceforth be referred to as $N^{\text{HASH}}$

The number of time steps needed is at most 2 to compute the input to each subsequent oracle query (since binary searching requires at most 2 bits be flipped to find the next input), giving $4\log(k \cdot h(n))$, plus several arithmetic steps at the end of the algorithm to compute $\hat{A}$ and $\hat{R}$. This part will depend on the exact computational model, but we'll say it costs 3 steps to compute each bit of $\hat{A}$ and $\hat{R}$, giving $3\log(k \cdot h(n))$ steps for a grand total of $7\log(k \cdot h(n))$ steps. If this calculation does not seem rigorous enough, take solace in the fact that these time steps will ultimately be negligible when $n$ is large.

Since the oracle HASH has a $\Sigma_2$TIME algorithm, we can convert $N^{\text{HASH}}$ for the problem $\mathcal{P}$ into a $\Sigma_3$TIME algorithm using the intuition behind the fact that $\text{P}^{\Sigma_2\text{P}} \subseteq \Sigma_3\text{P}$.

The HASH oracle can be written as $\exists u \forall v R(x, u, v)$ on input $x$. The algorithm $N^{\text{HASH}}$ makes several calls to the HASH oracle while it runs. Let $a_i$ be the outcome of the $i$th oracle call on input $x_i$. Suppose $m_0$ of the oracle calls output 0 and $m_1$ output 1. If the $i$th overall oracle call (regardless of outcome) is the $j$th oracle call that outputs 0 or if it's the $j$th oracle call that outputs 1 then we let $f(i) = j$. Then $N^{\text{HASH}}$ accepting on input $w$ is equivalent to

$$(\exists\{x_i\}\{a_i\}u_1^{(1)} \dots u_{m_1}^{(1)})(\forall v_1^{(1)} \dots v_{m_1}^{(1)} u_1^{(0)} \dots u_{m_0}^{(0)})(\exists v_1^{(0)} \dots v_{m_0}^{(0)}) T(\text{all inputs}) = 1$$

where $T$ is a machine that simulates $N^{\text{HASH}}$, then, whenever $N^{\text{HASH}}$ is about to make an oracle query, verifies that the input to this oracle query is $x_i$ and that $R(x_i, u_{f(i)}^{(a_i)}, v_{f(i)}^{(a_i)}) = a_i$ for each oracle call $i$. The algorithm $T$ does not have access to the HASH oracle, so it cannot compute the outcome of the oracle itself, so in order to proceed with simulating $N^{\text{HASH}}$, it just uses $a_i$ as the oracle outcome. At the end, $T$ verifies that $N^{\text{HASH}}$ accepts. If any of these verifications come out false, $T$ rejects, but otherwise accepts. The fact that this is equivalent to $N^{\text{HASH}}$ accepting is because if $\text{HASH}(x_i) = 1$ then by definition $\exists u_{f(i)}^{(1)} \forall v_{f(i)}^{(1)} R(x_i, u_{f(i)}^{(1)}, v_{f(i)}^{(1)}) = 1$, whereas if $\text{HASH}(x_i) = 0$ then $\forall u_{f(i)}^{(0)} \exists v_{f(i)}^{(0)} R(x_i, u_{f(i)}^{(0)}, v_{f(i)}^{(0)}) = 0$.

Now we count the number of time steps that $T$ uses. Simulating $N^{\text{HASH}}$ takes $7\log(k \cdot h(n))$. Comparing each of the strings $x_i$ takes $2(\log(k \cdot h(n)))^2$ since each $x_i$ has $\log(k \cdot h(n))$ bits. Running $R$ for all of the oracle call takes

$$2\log(k \cdot h(n)) \cdot k \cdot h(n) \cdot \Big(2(k \cdot h(n))^2 + 3(k \cdot h(n)) + 1\Big)$$

steps, where we have used Eq. (A.4) and imposed $j \leq k \cdot h(n)$. This gives a total run time of

$$c(h(n)) = \log(k \cdot h(n)) \cdot (k \cdot h(n)) \cdot \Big(4(k \cdot h(n))^2 + 6(k \cdot h(n)) + 9\Big) + \log(k \cdot h(n))^2.$$

Thus, $T$ is an algorithm using $c(h(n))$ satisfying the form of a $\Sigma_3$TIME algorithm which solves the problem $\mathcal{P}$. This proves the theorem. □

## A.2 Proof of Theorem 4.3.7: Oracle separation between PP and $\Sigma_3$TIME

**Theorem 4.3.7.** *There is an oracle $A$ such that* $\text{PP}^A \not\subset \Sigma_3\text{TIME}^A(\frac{2^{n/5}}{92} - \frac{n}{8})$.

*Proof.* The proof has several elements. A lower bound on the size of any classical low-depth circuit computing the parity function is fundamentally what provides the force for this result. This lower bound implies a lower bound on low-depth circuits that compute the majority function (which is how it will eventually be connected to the class PP). Low-depth circuits can be connected to PH machines with oracles, where the circuit depth corresponds to the level of the PH, which allows us to complete the proof.

Background for classical Boolean circuits is described in [15]. Boolean circuits resemble quantum circuits in the sense that a set of inputs are acted upon by gates to yield outputs, in this case AND, OR, and NOT gates. The depth of these circuits is the minimum number of layers of gates and the size of the circuit is the total number of gates. The output of these circuits is either a 0 or 1, so they compute a Boolean

function $g(z)$ on their $n$-bit input $z$. The parity function is defined as the sum of the bits of $z$ mod 2. The majority function is defined to be 1 if more than half the bits are 1 and 0 if at least half are 0.

**Lemma A.2.1** (Corollary 6.38 in [15])**.** *For sufficiently large $n$, the parity function on $n$ bits cannot be computed by a depth-$d$ circuit of size at most $2^{0.1n^{1/d}}$.*

*Proof.* The proof appears in [15]. □

Circuits that are able to compute the majority function can be used to compute the parity function.

**Lemma A.2.2.** *If the majority function can be computed on $n$ bits with a circuit of depth $d$ and size $S$, then the parity function can be computed on $n/2$ bits with a circuit of depth $d+1$ and size $nS/2+1$.*

*Proof.* This proof follows an idea presented in [27]. Let the Boolean function $\mathrm{Thr}_k^{n/2}$ for $1 \leq k \leq n/2$ on $n/2$ bits be 1 if at least $k$ of the input bits are 1 and 0 if not. Now, we form a circuit with $n/2$ input variables, and another $n/2$ inputs fixed at 1, for a total of $n$ inputs. To compute $\mathrm{Thr}_k^{n/2}$, we use our size-$S$ circuit for computing the majority function acting on the $n/2$ variable bits and $n/2-2k+2$ of the bits that are fixed to 1, for a total of $n-2k+2$ inputs. Thus, if the majority function outputs 1, then strictly more than $n/2-k+1$ bits are 1. But since $n/2-2k+2$ of these bits are fixed to 1, we conclude that the majority function only outputs 1 if at least $k$ of the $n/2$ variable inputs are 1, which is precisely the function $\mathrm{Thr}_k^{n/2}$. Thus, with a circuit of size at most $S$ and depth $d$, we can compute $\mathrm{Thr}_k^{n/2}$.

The Boolean function that outputs 1 if exactly $k$ of the inputs are 1 can be expressed as

$$(\mathrm{Thr}_k^{n/2} \wedge \neg \mathrm{Thr}_{k+1}^{n/2})$$

where $\neg$ means the negation of the output. So the parity function can be expressed as the OR of this expression over all odd numbers $k$ less than $n/2$. Since each Thr function can be computed with a subcircuit of size $S$ and there are $n/2$ of them, the

total size is $nS/2 + 1$ after the single OR gate has been included. This circuit will have depth at most $d + 1$. □

**Corollary A.2.2.1.** *For sufficiently large $n$, the majority function on $n$ bits cannot be computed by a depth-$d$ circuit of size at most $2^{0.1(n/2)^{1/(d+1)}}/n$.*

*Proof.* If such a circuit existed, it could be used to compute the parity function, as stated in Lemma A.2.2, with size less than $2^{0.1n^{1/(d+1)}}$ and depth $d + 1$, breaking the bound in Lemma A.2.1. □

Now we describe the connection between PH machines equipped with an oracle and classical low-depth circuits. Suppose we are given an oracle-machine in the $k$th level of the TIME hierarchy of the form $\sigma(A, x) = \exists y_1 \forall y_2 \ldots Q_k y_k R(x, y_1, \ldots y_k)$ where $A$ is a set which $R$ is allowed to compute in a single time step (i.e., $R$ has an $A$ oracle), $n = |x|$, the length of $y_i$ are bounded by the function $q(n)$, and the run time of the deterministic machine $R$ with access to oracle $A$ is bounded by function $p(n)$.

Using $\sigma(A, x)$, we construct a classical Boolean circuit. Let the circuit contain $2^{p(n)+2} - 2$ bits labeled by the strings of length at most $p(n)$ and their negations, which we denote by the symbol $\neg$. The circuit will have the property that, if we set the input value of bit $x_i$ corresponding to string $i$ to $A(i)$ and $\neg x_i$ to $\neg A(i)$, the output of the circuit will be 1 if and only if $\sigma(A, x)$ is true. Here we are following Ref. [15].

We fix all of the $y_i$ and simulate the running of $R$. When $R$ calls the oracle, we imagine a branch into two possible paths corresponding to the two possible answers it could receive. At the end, we have a tree of possible computation paths, some of which accept. Each accepting path corresponds to a specific set of oracle answers, so for each accepting path, at the bottom level of the circuit we make an AND gate between all of the $x_i$ or $\neg x_i$ that must be set to 1 for $R$ to take that path. Then, at the second level, we OR the outputs of these AND gates. This subcircuit outputs a 1 if and only if $R$ accepts for input $x$ and our fixed values of $y_i$. Suppose $k = 1$, so $\sigma(A, x) = \exists y_1 R(x, y_1)$. In this case, all we need to do is OR together the output of each of these subcircuits for the various possible $y_i$, and the output of the circuit will

agree with $\sigma(A, x)$. This OR gate can be combined with the OR gates at the level below it, making a depth-2 circuit.

For higher values of $k$, we use induction. We fix $y_1$ and assume we have a depth-$k$ circuit $\mathcal{C}_{y_1}$ for the function $\exists y_2 \forall y_3 \ldots Q_{k-1} y_k \ \neg R(x, y_2, \ldots, y_k)$. If we switch all the AND gates with OR gates in this circuit, and switch every bit with its negation, we compute the negation of this function on $k-1$ bits. Then, by performing an OR gate over all such circuits $\mathcal{C}_{y_1}$, we correctly compute $\sigma(A, x)$, with a depth $k+1$ circuit. By induction, this is possible for all integers $k$.

Moreover, the fanin (maximum number of inputs for a single gate) at the bottom level of this circuit is at most $p(n)$ since that is the maximum number of oracle queries for any computational path. The max fanin at higher levels is $2^{q(n)+p(n)}$ (we have to add the $q(n)$ since at the 2nd level we are combining the OR from all the computation paths with the OR associated with the $\exists$).

Finally, we use this circuit-hierarchy machine correspondence to choose an oracle $A$ that satisfies the statement of the theorem. For any oracle $A$, let the language $L_A = \{0^n : \text{the majority of the strings of length } n \text{ are in } A\}$. For any $A$, $L_A$ is clearly in the class $\mathrm{PP}^A$, since a probabilistic machine can guess one of the strings of length $n$, check if it is in $A$, and then accept or reject accordingly, and end up with the correct answer more than half the time.

We want to find an $A$ such that $L_A$ is not in $\Sigma_k\mathrm{TIME}(t(n))$ for some function $t(n)$. If $L_A$ were in $\Sigma_k\mathrm{TIME}(t(n))$, it would be decided by some machine at level $k$. We enumerate all of the machines at level $k$ of the hierarchy, and construct the oracle $A$ in stages by declaring certain strings in the language $A$ at each stage, declaring certain strings not in $A$ at each stage, and deferring the rest of the strings to be decided at a later stage.

On stage $j$, let $\sigma_j$ be the $j$th machine in level $k$, which has bounds $q_j(n)$ on the length of its inputs and $p_j(n)$ on the length of its runtime. At this point it has been decided that some strings will be in $A$ and that some strings will not be in $A$, but only for a finite number of such strings. Choose $n$ so that $n$ is larger than the length

of all these strings and also so that

$$s_j(n) \equiv 2^{(k+1)(q_j(n)+p_j(n))} < 2^{0.1 \cdot 2^{(n-1)/(k+2)}}/(2^n), \tag{A.6}$$

which we assume for now is possible. We construct the circuit of depth $k+1$ that corresponds to $\sigma_j$. We assign all input bits $i$ with length less than $n$ the appropriate value $A(i)$ to keep consistency with previous stages, and we temporarily assign all strings greater than $n$ the value 0. This circuit has size at most $s_j(n)$ and, by Corollary A.2.2.1, it cannot compute the majority function on $2^n$ bits. Since all the inputs of length not equal to $n$ have been fixed, we can view the circuit as solely a function of the $2^n$ inputs corresponding to bit strings of length $n$. Since we know the circuit does not compute the majority function, there must be some setting of the $2^n$ inputs where the output of the circuit is inconsistent with the majority. We assign the oracle $A$ these values on strings of length $n$ (which had previously been left undetermined), so that the $j$th predicate (which agrees with the output of the circuit) is guaranteed to output the wrong answer on the input $0^n$ and hence cannot decide the language $L_A$.

This will only be true for the $j$th predicate if it is possible to choose an $n$ that satisfies Eq. (A.6). This constraint can be simplified to

$$p_j(n) + q_j(n) < 2^{(n-1)/(k+2)}/(10(k+1)) - n/(k+1)$$

for sufficiently large $n$. The size of the inputs $q_j(n)$ must be less than the run time $p_j(n)$ in order for all the bits to be read, so we further simplify the expression to

$$p_j(n) < 2^{(n-1)/(k+2)}/(20(k+1)) - n/(2(k+1))$$

. What we've shown is that no level-$k$ machine with runtime less than $2^{(n-1)/(k+2)}/(20(k+1)) - n/(2(k+1))$ can compute the language $L_A$, so $L_A \notin$

$\Sigma_k \mathrm{TIME}^A(2^{(n-1)/(k+2)}/(20(k+1)) - n/(2(k+1)))$. Since $L_A \in \mathrm{PP}^A$ this shows

$$\mathrm{PP}^A \not\subset \Sigma_k \mathrm{TIME}^A(2^{(n-1)/(k+2)}/(20(k+1)) - n/(2(k+1))) \tag{A.7}$$

Choosing $k = 3$ proves the theorem, where we impose the simplification $92 > 2^{1/5} \cdot 20 \cdot 4$. □

# Bibliography


[1] Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 461, pages 3473–3482. The Royal Society, 2005.

[2] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2011.

[3] Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. *arXiv preprint arXiv:1612.05903*, 2016.

[4] Leonard M Adleman, Jonathan DeMarrais, and Ming-Deh A Huang. Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540, 1997.

[5] Richard Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational Complexity*, 4(4):339–349, 1994.

[6] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *arXiv preprint arXiv:1608.00263*, 2016.

[7] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by Clifford gates. *Physical review letters*, 116(25):250501, 2016.

[8] Sergey Bravyi, David Gosset, and Robert Koenig. Quantum advantage with shallow circuits. *arXiv preprint arXiv:1704.00690*, 2017.

[9] Michael J Bremner, Richard Jozsa, and Dan J Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 467, pages 459–472. The Royal Society, 2011.

[10] Michael J Bremner, Ashley Montanaro, and Dan J Shepherd. Average-case complexity versus approximate simulation of commuting quantum computations. *Physical review letters*, 117(8):080501, 2016.

[11] Isaac L Chuang, Lieven MK Vandersypen, Xinlan Zhou, Debbie W Leung, and Seth Lloyd. Experimental realization of a quantum algorithm. *Nature*, 393(6681):143–146, 1998.

[12] Christopher M Dawson and Michael A Nielsen. The solovay-kitaev algorithm. *arXiv preprint quant-ph/0505030*, 2005.

[13] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117. The Royal Society, 1985.

[14] David Deutsch. Quantum computational networks. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 425, pages 73–90. The Royal Society, 1989.

[15] Ding-Zhu Du and Ker-I Ko. *Theory of computational complexity*, volume 58. John Wiley & Sons, 2011.

[16] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[17] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[18] Edward Farhi and Aram W Harrow. Quantum supremacy through the quantum approximate optimization algorithm. *arXiv preprint arXiv:1602.07674v1*, 2016.

[19] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[20] Daniel Gottesman. The Heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*, 1998.

[21] Daniel Gottesman and Isaac L Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402(6760):390–393, 1999.

[22] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[23] Yenjo Han, Lane A Hemaspaandra, and Thomas Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.

[24] Mark Howard and Earl Campbell. Application of a resource theory for magic states to fault-tolerant quantum computing. *Physical Review Letters*, 118(9):090501, 2017.

[25] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2):022328, 2013.

[26] Jonathan A Jones, Michele Mosca, and Rasmus H Hansen. Implementation of a quantum search algorithm on a quantum computer. *Nature*, 393(6683):344–346, 1998.

[27] Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

[28] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit RSA modulus. In *Annual Cryptology Conference*, pages 333–350. Springer, 2010.

[29] Ashley Montanaro. Quantum circuits and low-degree polynomials over $\mathbb{F}_2$. *Journal of Physics A: Mathematical and Theoretical*, 2017.

[30] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[31] John Preskill. Quantum computing and the entanglement frontier. *arXiv preprint arXiv:1203.5813*, 2012.

[32] Dan Shepherd and Michael J Bremner. Temporally unstructured quantum computation. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 465, pages 1413–1439. The Royal Society, 2009.

[33] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[34] Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 330–335. ACM, 1983.

[35] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[36] Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 118–126. ACM, 1983.

[37] Barbara M Terhal and David P DiVincenzo. Adaptive quantum computation, constant depth quantum circuits and arthur-merlin games. *arXiv preprint quant-ph/0205133*, 2002.