Contents

Chapter 1

Counting complexity and quantum computation

Stephen A. Fenner[1]

Computer Science and Engineering Department, University of South Carolina, Columbia, SC 29208 USA, fenner@cse.sc.edu

Abstract We survey several applications to quantum computing of computational complexity theory, especially the complexity of problems related to counting. We show how the connection of quantum computation to counting is very close. We define counting complexity classes, relativization of both classical computation and quantum circuits, and present a number of results from disparate sources, recast into a single consistent formalism. We assume prior knowledge of the mathematical formalism of quantum mechanics, but we present the concepts of computational complexity in an introductory manner.

[1]Partially supported by a South Carolina Commission on Higher Education SCRIG Grant R-01-0256.

1.1 Introduction

This chapter is primarily aimed at people (physicists, perhaps) who know more about quantum mechanics than they do about the theory of computation. It explores a close relationship between quantum computing and the complexity (difficulty) of counting the number of solutions to classical search problems. The latter is well studied, going back to the 1970s [46]. Although the concept of a quantum computer has also been around for a long time [8, 21], it was only in the 1990s that quantum computing began to receive intense scrutiny. Our point (and we do have one) is that the older study of computational complexity, especially counting complexity, has a lot to say about quantum computers—their ultimate strengths and limitations.

There are a number of good research papers on the complexity of quantum computation, relating it to classical complexity. A primary example is the thorough and detailed treatment of the quantum Turing machine (QTM) model of Deutsch [15] given by Bernstein and Vazirani [10]. Their paper proves a number of fundamental results—they give clear evidence that QTMs are powerful enough to efficiently implement any "reasonable" quantum algorithm; but more importantly, they construct a universal QTM that can efficiently simulate any other QTM to good approximation. An efficient universal quantum computer shows that there is a small, finite handful of primitives that can combine to implement any reasonable quantum algorithm efficiently—namely, the quantum operations used by the universal machine. As a result, the quantum computational model has an especially simple and easy-to-analyze structure.

An extensive summary of results in quantum complexity can be found in Gruska [29].

The current chapter collects a number of results from other sources; some of these are more recent improvements on earlier, better known results. Although they appear in disparate sources, these results and the techniques used to prove them are related to each other, and they deserve to be brought together under a single framework. That is the main purpose of this chapter. We give detailed proofs of selected results; often, the technique used in the proof is just as interesting and useful as the result itself.

We use the quantum circuit model [16] for describing quantum algorithms. Yao [49] showed that quantum circuits were equivalent to QTMs,

each able to simulate the other efficiently to good approximation. By "efficiently" we mean with at most a polynomial slow-down of the machine or a polynomial blow-up in circuit size. Quantum circuits are conceptually easier than Turing machines for rendering quantum algorithms, and most researchers prefer them. In Section 1.2, after defining the basic circuit model for both classical Boolean circuits and quantum circuits, we review the basic concepts of "standard" complexity theory: the classes P and NP, oracles and relativization, reducibility, and completeness. We then build on these concepts to define the function class GapP, which is then used to describe counting problems. Lastly, we prove a close connection between quantum computation and GapP—and hence counting problems.

The rest of the chapter is loosely organized into two parts: the first gives positive results about the power of quantum computation (Section 1.4), and the second gives negative results (Section 1.5). In Section 1.4, we give a proof of the existence (first shown by Green and Pruim [26]) of a black-box problem that is quantum computable with zero error but cannot be computed deterministically, even with free access to an arbitrary NP oracle. The proof serves as a good illustration of oracle construction—a venerable and often-used technique in complexity theory. In Section 1.5, we use the equivalence of GapP and quantum computation to prove a "lowness" property of efficiently quantum computable sets. We show that free access to such a set is useless in solving a counting problem—that is, any problem solvable by counting with access to the set is also solvable by counting without such access. This fact provides strong evidence that quantum computers are not powerful enough to solve arbitrary counting problems.

Quantum complexity is an extremely rich field of study. Due to space limitations, we make no attempt at a complete or unbiased survey. We apologize in advance for leaving out many interesting results, even some that are closely related to the current topic, for the sake of fewer and more detailed proofs.

1.2 Preliminaries

We assume prior knowledge of the mathematical underpinnings of quantum mechanics—Hilbert spaces, operator algebras, Dirac bracket

notation, tensor products, and the like. All Hilbert spaces mentioned in this chapter are finite-dimensional. We will also suppose some basic knowledge of the theory of computation, including basic graph theoretic concepts and Turing machines (TMs), and we will briefly review some relevant concepts in complexity theory. More background can be found in several good textbooks on the theory of computation [30, 42] and on complexity theory in particular [4, 5, 24, 35].

We let $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ stand for the natural numbers $\{0, 1, 2, \ldots\}$, integers, rationals, reals, and complex numbers, respectively. For $z \in \mathbb{C}$, we write $\overline{z}$ for the complex conjugate of z, and we write $|z|$ for $\sqrt{z\overline{z}}$, the absolute value of z.

We let $\|X\|$ stand for the cardinality of finite set X. We fix an alphabet $\Sigma = \{0, 1\}$, and let Σ^* be the set of all finite strings of symbols from Σ. For any string $w \in \Sigma^*$, we let $|w|$ denote the length of w (number of symbols in w). For $n \in \mathbb{N}$, we let Σ^n denote the set of all strings of Σ^* of length n. We often identify classes of finitely describable objects—for example, Boolean values, truth tables, natural numbers, integers, rational numbers, algebraic numbers, graphs, TMs, and strings over various alphabets, as well as pairs, finite lists, and finite sets of these—with strings in Σ^* via reasonable and concise encodings. For example, we identify natural numbers with strings via the usual binary representation, and we identify the Boolean values *true* and *false* with 1 and 0, respectively. We sometimes represent a natural number n in unary notation as 1^n, i.e., a string of n 1s. We will also identify subsets of some (assumed) universal set (such as Σ^n or Σ^*) with their characteristic Boolean functions over the universal set.

All polynomials mentioned will be univariate unless otherwise specified. If a polynomial p represents the running time (number of primitive steps) of an algorithm or the lengths of strings, then we assume it has all integer coefficients, and that $p(n) \geq 0$ for all $n \in \mathbb{N}$. If $c \in \mathbb{R}$, we say $p > c$ to mean that $p(n) > c$ for all $n \in \mathbb{N}$.

A *circuit* is a directed acyclic graph with some arbitrary ordering of the vertices. If C is a circuit, then the vertices with indegree zero are the *initial* vertices, and those with outdegree zero are the *final* vertices. We designate the first few initial vertices of C as the *inputs* to C, the first few final vertices as the *outputs*, and all other vertices are *gates*. Gates that are not initial or final are called *intermediate gates*. The edges are called *wires*. It may seem strange at first to allow initial and final vertices that are not inputs or outputs, but this will be useful when we describe our quantum circuit model. A *Boolean circuit* is a circuit

where each intermediate gate is labeled with a Boolean connective $\wedge$, $\vee$, or $\neg$, and all intermediate gates have indegree two except $\neg$-gates, which have indegree one. Noninput initial gates are labeled with the constant 0 (false), and final gates are unlabeled. A labeling of the inputs of a Boolean circuit with Boolean values uniquely determines a labeling of the outputs, computed in the usual manner. So a Boolean circuit with n inputs and m outputs computes a unique function $f : \Sigma^n \to \Sigma^m$.

When we draw a circuit, the initial vertices are on the left and the final vertices on the right. Input and output vertices are represented as bare ends of wires.

1.2.1 Qubits, quantum gates, and quantum circuits

The following definitions are adapted primarily from Nielsen and Chuang [33] and Aharonov, Kitaev, and Nisan [2]. We refer to those sources for more detail and motivation.

For two Hilbert spaces $\mathcal{H}$ and $\mathcal{J}$, we denote the space of linear maps from $\mathcal{H}$ to $\mathcal{J}$ as $\mathcal{L}(\mathcal{H}, \mathcal{J})$, and we abbreviate $\mathcal{L}(\mathcal{H}, \mathcal{H})$ by $\mathcal{L}(\mathcal{H})$, the space of linear operators on $\mathcal{H}$. We will use the density matrix formulation of quantum states; so we regard a quantum state in a Hilbert space $\mathcal{H}$ as a Hermitian operator $\rho \in \mathcal{L}(\mathcal{H})$ that is positive semidefinite ($\rho \geq 0$) and has unit trace ($\mathrm{tr}(\rho) = 1$). A *quantum operation* from $\mathcal{H}$ to $\mathcal{J}$ is a linear map A from $\mathcal{L}(\mathcal{H})$ to $\mathcal{L}(\mathcal{J})$ that is trace-preserving and *completely positive* (A is also called a superoperator). Intuitively, completely positive means that if we embed $\mathcal{H}$ into some larger system, then the standard lifting of A to the larger system preserves positive (semi)definiteness, and thus states get mapped to states. Formally, this means that for any Hilbert space $\mathcal{K}$, the linear map $A \otimes I_{\mathcal{K}} : \mathcal{L}(\mathcal{H} \otimes \mathcal{K}) \to \mathcal{L}(\mathcal{J} \otimes \mathcal{K})$, where $I_{\mathcal{K}}$ is the identity map on $\mathcal{L}(\mathcal{K})$, preserves positive definiteness: for any $\rho \in \mathcal{L}(\mathcal{H} \otimes \mathcal{K})$, if $\rho > 0$, then $(A \otimes I_{\mathcal{K}})(\rho) > 0$. [For $\tau_1 \in \mathcal{L}(\mathcal{H})$ and $\tau_2 \in \mathcal{L}(\mathcal{K})$, $(A \otimes I_{\mathcal{K}})(\tau_1 \otimes \tau_2) = A(\tau_1) \otimes \tau_2$.]

A *qubit* is any quantum physical system S representable by a two-dimensional Hilbert space $\mathcal{H}_S$, e.g., photon polarization, spin of a spin-$\frac{1}{2}$ particle, etc. We will assume some preferred observable on $\mathcal{H}_S$ with eigenvalues 0 and 1 (the *value* of the qubit), and we will fix an orthonormal basis of $\mathcal{H}_S$ of corresponding eigenvectors $|0_S\rangle$ and $|1_S\rangle$, respectively. This will be called the *standard basis*. We will usually drop the subscripts, in which case the system S is understood implicitly. We let $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$ be the orthogonal projection operators onto the subspaces spanned by $|0\rangle$ and $|1\rangle$, respectively.

A combined system $\mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_n$ of n qubits has dimension 2^n and has a natural product basis of vectors

$$\begin{aligned}
|00\cdots00\rangle &= |0_1\rangle|0_2\rangle\cdots|0_{n-1}\rangle|0_n\rangle \\
|00\cdots01\rangle &= |0_1\rangle|0_2\rangle\cdots|0_{n-1}\rangle|1_n\rangle \\
|00\cdots10\rangle &= |0_1\rangle|0_2\rangle\cdots|1_{n-1}\rangle|0_n\rangle \\
|00\cdots11\rangle &= |0_1\rangle|0_2\rangle\cdots|1_{n-1}\rangle|1_n\rangle \\
&\vdots \\
|11\cdots11\rangle &= |1_1\rangle|1_2\rangle\cdots|1_{n-1}\rangle|1_n\rangle
\end{aligned}$$

We will call such a system an n-qubit system, the above basis being its standard basis. We may also write each basis vector above as $|i\rangle$ for $0 \leq i < 2^n$, corresponding to the binary representation of i using n bits. For any binary string x of length at most n, we let P_x be the orthogonal projection operator onto the subspace spanned by those basis states $|z\rangle$ where z has x as a prefix—that is, the subspace spanned by $\{|z\rangle : z \text{ has prefix } x\}$.

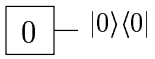
For any Hilbert space $\mathcal{H}$, a standard basis $|v_1\rangle, \ldots, |v_n\rangle$ of $\mathcal{H}$ naturally lifts to a standard basis $\{|v_i\rangle\langle v_j| : 1 \leq i, j \leq n\}$ of $\mathcal{L}(\mathcal{H})$. This means that we can identify quantum operations with matrices over the standard bases involved, and composition of quantum operations corresponds to matrix multiplication.

One can assume that all information in a quantum computation is stored in qubits, so we will restrict each quantum operation to map from one multiqubit system to another (perhaps the same one). A *quantum circuit* C is a circuit whose inputs and outputs are labeled with distinct qubits, whose gates are labeled with quantum operations drawn from some prespecified set, and where the wires leading into each gate are ordered. If the gate has n input wires and m output wires, then its quantum operation will map n-qubit states to m-qubit states. If C has k inputs, then an *input state* of C is some k-qubit state.

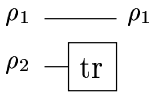
We can restrict our quantum gates to three types of quantum operations:

Unitary. Maps an n-qubit state $\rho \in \mathcal{L}(\mathcal{H})$ to the state $U\rho U^\dagger \in \mathcal{L}(\mathcal{H})$, where $U \in \mathcal{L}(\mathcal{H})$ is some unitary operator that we call the *underlying* unitary operator. (The matrix entry of a unitary gate corresponding to the pair of basis vectors $|x\rangle\langle x'|$ and $|y\rangle\langle y'|$ is $\langle x|U|y\rangle\overline{\langle x'|U|y'\rangle}$.)

Ancilla Introduction. Maps an n-qubit state ρ to the $n+1$-qubit state $\rho \otimes |0\rangle\langle 0|$.[2] For example,

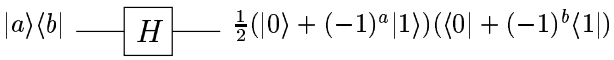


Partial Trace. Maps an $n+1$-qubit state $\rho \in \mathcal{L}(\mathcal{H}_1 \otimes \mathcal{H}_2)$ to the n-qubit state $\mathrm{tr}_{\mathcal{H}_2}(\rho) \in \mathcal{H}_1$.[3] For example,

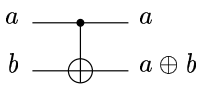


(Note that one can effectively trace out several qubits at once simply by tracing out each qubit one at a time; so our "primitive" Partial Trace operation need only act on a single qubit.)

Two important examples of unitary gates are the 1-qubit Hadamard gate



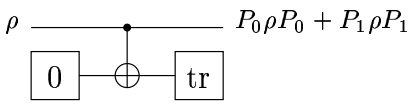
with underlying $U = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, and the 2-qubit controlled NOT gate $\wedge_1(\sigma_x)$



with underlying $U = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \end{bmatrix}$. The ancilla and partial trace gates act on all the available wires, even though they connect just to wires that are being created or deleted. Other quantum operations can be expressed as compositions of these three types. For instance, the 1-qubit measurement gate, which maps ρ to $P_0\rho P_0 + P_1\rho P_1$, can be implemented by the circuit

[2] More accurately, this gate maps the unique zero-qubit "vacuum" state in the Hilbert space $\mathbb{C}$ to the one-qubit state $|0\rangle\langle 0|$.

[3] Similarly to ancilla introduction, this gate is more accurately described as mapping any one-qubit state to the zero-qubit state in $\mathbb{C}$.



Which unitary operations are "reasonable?" This is a somewhat thorny question. The search for a practical implementation of a quantum circuit is still in its infancy, so it is not currently clear what types of gates can be easily built and which cannot. In lieu of practical experience, we must therefore rely on mathematical heuristics. Certainly, we should try to restrict ourselves to some minimal (preferably finite) set of simple operations. It is likely that in practice a quantum gate can only act on a small number of qubits at a time. The matrix entries of a quantum gate should be "simple" numbers in some sense: they should at least be polynomial-time approximable (see below); more preferably, they should be rational numbers with small numerators and denominators, or perhaps algebraic numbers of small degree. Finally, it is much more likely that a gate can be reliably fabricated if it can be simulated in a fault-tolerant manner (there is a large body of literature on fault-tolerant quantum computing; see [36] for a good survey).

Much good work has been done at isolating sets of quantum gates that do well under all these heuristics and that can efficiently simulate much bigger classes of operations to arbitrarily close approximation [12, 38]. We will have more to say on this later. There is an obvious trade-off here, however; restricting the repertoire of gates may require increasing the size and complexity of the circuits and may also lose the exactitude of the computation. An example of this is with the Quantum Fourier Transform (QFT) [39], which is a useful module for a wide variety of quantum algorithms. The QFT on n qubits can be implemented exactly with a reasonably simple circuit containing Hadamard gates and controlled conditional phase shift gates with phase shift $e^{2\pi i/2^n}$, where n grows with the size of the input. There can be no fixed finite palette of gates, however, that can be used to build circuits for computing QFT exactly for all n. This is because all the matrix elements of such a circuit would belong to some fixed finitely generated field extension of $\mathbb{Q}$, but the field generated by the matrix elements of QFT (ranging over all input sizes) is not finitely generated over $\mathbb{Q}$. Thus, a circuit family computing QFT exactly requires an infinite palette of gates, although a finite palette suffices to compute QFT to good approximation (which is enough for all current applications) [14, 39].

1.2.2 Classical complexity

There are several definitions of Turing machine, all equivalent. For concreteness, we use the model (single one-way infinite tape) of Sipser [42]. One can also forget about Turing machines and think more abstractly about algorithms if one wishes.

1.2.2.1 P and NP

A *language* or *decision problem* is any subset of Σ^*. A language can also naturally be viewed as a predicate, that is, a Boolean- or 0,1-valued function on Σ^*. A Turing machine M *decides* or *computes* a language L if M halts on all inputs (either accepting or rejecting the input), and, for each $w \in \Sigma^*$, $w \in L$ if and only if M accepts input w. Let $t : \Sigma^* \to \mathbb{R}$ be any function. A language L is *computable in time* t if there is a TM M deciding L such that M halts in at most $t(w)$ steps on any input w.

A language is computable in *polynomial time* if it is computable in time $t(w) = p(|w|)$ for some polynomial p. We let P denote the class of all polynomial-time computable languages. P is a *complexity class* that captures the notion of "easy to compute deterministically," at least in a broad theoretical sense. Decision problems in P have fast, deterministic algorithms. The class P is quite *robust* in the sense that its definition does not really depend on which model of computation is used (TMs, random access machines, uniform Boolean circuit families, Pascal programs, etc.); all reasonable computational models give rise to the same class P.

Many decision problems are not known to be in P but are in the more inclusive complexity class NP. A language L is in NP if and only if there is a predicate (language) $R \in \mathrm{P}$ and a polynomial p such that, for all $x \in \Sigma^*$,

$$x \in L \iff (\exists y \in \Sigma^r)\,[\, R(x,y)\,]\,,$$

where $r = p(|x|)$. (We tacitly assume a suitable encoding of pairs of strings as single strings.) If a string x is in L, then we call a y satisfying $R(x,y)$ a *certificate* or *witness* to x's membership in L. One thinks of a TM M computing R as a "verifier" that, when handed some alleged proof that $x \in L$, can verify the correctness of the proof in a short amount of time. Thus, P is the class of problems whose instances are easily *decided*, and NP is the class of problems whose solutions can be easily *verified*. A typical NP problem is: given an undirected graph G and integer s, does G have a clique (complete subgraph) of size s? This problem is known as the CLIQUE problem (it is customary to name

problems with all capital letters). If G did in fact have a clique of size s, then an easy-to-verify witness would be a description of an actual s-clique in G. Such a clique may be difficult to *find*, but it is easy to verify. The CLIQUE problem is not known to be in P.

If $\mathcal{C}$ is a complexity class, then co$\mathcal{C}$ is the class of complements of languages in $\mathcal{C}$, i.e.,

$$\text{co}\mathcal{C} = \{\Sigma^* - L : L \in \mathcal{C}\}.$$

Clearly P = coP $\subseteq$ NP $\cap$ coNP. It is not known whether the CLIQUE problem is in coNP.

It is a major open question in complexity theory, and in mathematics as a whole, whether or not P = NP. The common intuition is that the two classes are not equal, but a proof of this has yet to be found.

All complexity classes that we mention in this chapter are subclasses of PSPACE, the class of problems solvable by using only a polynomial (in the input size) amount of memory, but unlimited time.

1.2.2.2 Reducibility and completeness

Let A and B be any languages. We say that A is *Turing reducible (T-reducible)* to B ($A \leq_{\text{T}} B$) if there is an algorithm to compute A in polynomial time that can freely ask questions about membership in B. The algorithm is called a *reduction* of A to B.[4] We now formalize this concept. An *oracle Turing machine (OTM)* is a TM M equipped with an additional writable tape (the *query tape*), and three special states $q_?$ (the query state), q_{yes} and q_{no}. Suppose B is any language. A *computation of M with (or relative to) B* is defined similarly to an ordinary computation, except that at any time during the computation M may ask a question of the form, "Is $x \in B$?" for some string $x \in \Sigma^*$, and receive the answer as follows: first, M writes x on its query tape then enters state $q_?$; in the next step, M enters either q_{yes} if $x \in B$ or q_{no} if $x \notin B$. B is called an *oracle*, and we say that M *queries* the oracle, where x is the query. The answer to the query is "recorded" by the resulting state q_{yes} or q_{no}.

We define polynomial time in this model as before, based on the input to the computation, independent of the oracle, and we stipulate that any reduction must run in polynomial time. Turing reducibility captures the

[4] It is more common to say "polynomial-time reducible" and use the notation $A \leq_{\text{T}}^{p} B$ in this case. Our use is justified because all our reductions will run in polynomial time.

notion of relative difficulty. $A \leq_{\mathrm{T}} B$ means that A is easy *relative to* B. Here is another way to say it: A is no more difficult than B, or equivalently, B is at least as difficult as A. It is easy to see that if $A \leq_{\mathrm{T}} B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$. Contrapositively, if $A \leq_{\mathrm{T}} B$ and $A \notin \mathrm{P}$, then $B \notin \mathrm{P}$. The class P^B is the class of all languages T-reducible to B.

A restricted form of T-reducibility is called m-reducibility ("many-one" reducibility). We say that A is *m-reducible* to B ($A \leq_m B$) if $A \leq_{\mathrm{T}} B$ by a reduction M which, on any input, makes exactly one query to B, accepts if the answer is yes, and rejects otherwise. Equivalently, $A \leq_m B$ if there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that, for every $w \in \Sigma^*$, $w \in A \iff f(w) \in B$.

If $\mathcal{C}$ is a complexity class and L is a language, then we say that L is *$\mathcal{C}$-hard* if $A \leq_m L$ for all $A \in \mathcal{C}$. If in addition $L \in \mathcal{C}$, then L is *$\mathcal{C}$-complete*. The CLIQUE problem, among many other interesting problems, is known to be NP-complete. If any NP-complete problem is shown to be in P, then this would prove that P = NP. We can define hardness and completeness for Turing reducibility as well, in which case we would say that L is *T-hard* or *T-complete* for $\mathcal{C}$.

1.2.2.3 Counting classes

Many interesting complexity classes can be defined in terms of the *number* of witnesses to instances of NP problems. To reify this concept, Valiant [46] defined the class #P consisting of functions $f : \Sigma^* \to \mathbb{N}$ as follows: a function f belongs to #P if and only if there is a P predicate R and a polynomial p such that for all $x \in \Sigma^*$,

$$f(x) = \left\| \left\{ y \in \Sigma^{p(|x|)} : R(x, y) \right\} \right\|.$$

For example, the number of cliques of a given size s in a given graph G is a #P function of (G, s). A variant of #P that is more algebraically useful is the class GapP [18] consisting of functions $f : \Sigma^* \to \mathbb{Z}$. A function f is in GapP if and only if $f = g - h$ for some $g, h \in \#\mathrm{P}$. By manipulating predicates in the right way, the following closure properties of GapP are routinely verified:

PROPOSITION 1.1
[[18]]

1. *The identity function $\mathbb{Z} \to \mathbb{Z}$ is in* GapP*, and if $f : \Sigma^* \to \mathbb{Z}$ is computable in polynomial time and $g \in$ GapP, then $g \circ f \in$ GapP.*

These two facts imply that $f \in \mathrm{GapP}$ *also. Furthermore,* $\#\mathrm{P} \subseteq \mathrm{GapP}$.

2. *If* $f \in \mathrm{GapP}$, *then* $-f \in \mathrm{GapP}$.

3. *If* $g, h \in \mathrm{GapP}$, *then* $g + h \in \mathrm{GapP}$. *More generally, if* $f \in \mathrm{GapP}$, *and* p *is a polynomial, then* $s \in \mathrm{GapP}$, *where*

$$s(x) = \sum_{y:|y| \leq p(|x|)} f(x, y).$$

4. *If* $g, h \in \mathrm{GapP}$, *then* $gh \in \mathrm{GapP}$ *(*gh *is the pointwise product of* g *and* h, *not the composition). More generally, if* $f \in \mathrm{GapP}$ *and* q *is a polynomial, then* $p \in \mathrm{GapP}$, *where*

$$p(x) = \prod_{i=0}^{q(|x|)} f(x, i).$$

In other words, GapP contains all easy-to-compute functions and is closed under negation, uniform exponential sums, and uniform polynomial products. We will use the following useful fact about GapP in the proof of Theorem 1.2.

LEMMA 1.1
[[18]] If $f \in \mathrm{GapP}$, *then there is a* P *predicate* R *and a polynomial* p *such that*

$$f(x) = \frac{1}{2}\left(\|\{y \in \Sigma^r : R(x, y)\}\| - \|\{y \in \Sigma^r : \neg R(x, y)\}\|\right),$$

for all $x \in \Sigma^*$ *with* $r = p(|x|)$.

PROOF Let $S_1, S_2 \in \mathrm{P}$ be predicates and q a polynomial such that $f(x) = \|\{y \in \Sigma^s : S_1(x, y)\}\| - \|\{y \in \Sigma^s : S_2(x, y)\}\|$ for all $x \in \Sigma^*$ with $s = q(|x|)$. Let $p = q + 1$, and let R be the predicate

"On input (x, z) with $x, z \in \Sigma^*$:
1. Let $r = p(|x|)$.
2. If $|z| \neq r$ then reject.
3. If $z = 0y$ for some y and $S_1(x, y)$, then accept.
4. If $z = 1y$ for some y and $\neg S_2(x, y)$, then accept.
5. Reject."

A straightforward calculation shows that R and p satisfy the lemma. ∎

Lemma 1.2 below will show us that GapP is very useful in describing quantum computation. Meanwhile, GapP can provide simple characterizations of most counting complexity classes. We'll define only the two most relevant ones here: PP [25] and $C_=P$ [48].

- A language L is in PP if there is an $f \in$ GapP such that, for all $x \in \Sigma^*$,
$$x \in L \iff f(x) > 0.$$
- A language L is in $C_=P$ if there is an $f \in$ GapP such that, for all $x \in \Sigma^*$,
$$x \in L \iff f(x) = 0.$$

A typical problem in PP is: "Given an undirected graph G and natural numbers s and n, does G contain more than n distinct cliques of size s?" A typical problem in $C_=P$ is: "Given an undirected graph G and natural numbers s and n, does G contain exactly n distinct cliques of size s?"

To illustrate the uses of GapP we present a simple proof of the following well-known facts about counting classes:

PROPOSITION 1.2
[[25, 41, 48]] NP $\subseteq$ co$C_=P \subseteq$ PP *and* $C_=P \subseteq$ PP.

PROOF Let L be a language in NP with corresponding P predicate R and polynomial p. R and p also naturally define a function $f \in \#P$ such that $x \in L \iff f(x) > 0 \iff f(x) = 0$ for all $x \in \Sigma^*$. By Closure Property (1) of Proposition 1.1, f is also in GapP, so $L \in$ co$C_=P$ via f.

Now let L be any language in co$C_=P$. There is a $g \in$ GapP such that $x \in L \iff g(x) \neq 0$ for all $x \in \Sigma^*$. By Closure Property (4) of Proposition 1.1, the function $g^2 \in$ GapP, and $x \in L \iff (g(x))^2 > 0$. Thus $L \in$ PP via the function g^2.

The proof that $C_=P \subseteq$ PP is similar, except that we use the GapP function $1 - g^2$ instead of g^2. ∎

For our purposes, the most important property of GapP is that it is closed under uniform multiplication of a polynomial number of matrices,

each of which can have exponential size. Roughly speaking, if a small number of large matrices have entries computed by a GapP function, then their product's entries are also computed by a GapP function. Since quantum operations are essentially matrix multiplication, this means we can simulate them with GapP.

We now make this notion precise. Suppose a is a GapP function that takes two parameters $i, j \in \mathbb{N}$ represented in binary, and possibly other parameters $\vec{x}$. Then for numbers $m, n \in \mathbb{N}$ (which may depend on $\vec{x}$) we let $[\ a(\vec{x})\ ]^{m \times n}$ denote the $m \times n$ matrix whose (i,j)th entry is $a(\vec{x}; i, j)$, for $0 \leq i < m$ and $0 \leq j < n$. A lemma similar to the following was proved by Fortnow and Rogers [22].

LEMMA 1.2
Let $a(\vec{x}, y; i, j)$ be a GapP *function and let $s(\vec{x}, y)$ be a polynomial-time computable function. Then there is a* GapP *function $b(\vec{x}, y; i, j)$ such that for all $\vec{x}$ and for all $r \in \mathbb{N}$,*

$$[\ b(\vec{x}, 1^r)\ ]^{s_r \times s_0} = [\ a(\vec{x}, 1^r)\ ]^{s_r \times s_{r-1}} [\ a(\vec{x}, 1^{r-1})\ ]^{s_{r-1} \times s_{r-2}} \cdots [\ a(\vec{x}, 1)\ ]^{s_1 \times s_0},$$

where $s_\ell = s(\vec{x}, 1^\ell)$ for $0 \leq \ell \leq r$.

PROOF For $0 \leq i_r < s_r$ and $0 \leq i_0 < s_0$, the (i_r, i_0)th entry on the right hand side of the above equation is

$$\sum_{i_1=0}^{s_1-1} \sum_{i_2=0}^{s_2-1} \cdots \sum_{i_{r-1}=0}^{s_{r-1}-1} \prod_{u=1}^{r} a(\vec{x}, u; i_u, i_{u-1}).$$

This is a uniform exponential size sum of uniform polynomial size products of a GapP function. By the closure properties of GapP given in Proposition 1.1, it is a GapP function of $\vec{x}$, 1^r, i_r, and i_0. ∎

1.2.2.4 Relativization

Many results in complexity theory are "oracle" results. Let A be any language and $\mathcal{C}$ be one of the classes we have defined thus far in terms of P predicates. We can define the class $\mathcal{C}^A$ analogously with our definition of $\mathcal{C}$, except that we now allow the P predicate R free access to A as an oracle; that is, R is now a P^A predicate. This is a natural way of *relativizing* a class $\mathcal{C}$ to an oracle A, just as we relativized P

earlier. If a language L is in $\mathcal{C}^A$, then we say that "$L \in \mathcal{C}$ relative to A." Most standard techniques and results in complexity theory carry over when the classes involved are relativized to any oracle. For example, GapP^A has all the closure properties of Proposition 1.1 relativized to A, including $\#\mathrm{P}^A \subseteq \mathrm{GapP}^A$. Thus the proof of Proposition 1.2 also easily relativizes to any oracle A to show that $\mathrm{NP}^A \subseteq \mathrm{coC_{=}P}^A \subseteq \mathrm{PP}^A$ and $\mathrm{C_{=}P}^A \subseteq \mathrm{PP}^A$.

An *oracle result* is a statement about complexity classes that holds relative to *some* oracle. Oracle results have great heuristic value in complexity theory: if some property P about complexity classes holds relative to some oracle, then this provides some evidence that P holds without an oracle (unrelativized); at least it shows that the standard, relativizable techniques of complexity theory will not suffice in refuting P, because otherwise P would be false relative to all oracles. In 1975, Baker, Gill, and Solovay [3] constructed an oracle A such that $\mathrm{P}^A \neq \mathrm{NP}^A$. This result suggests that it will not be easy to show that P = NP. However, in the same paper, they constructed another oracle B such that $\mathrm{P}^B = \mathrm{NP}^B$, which suggests that it will not be easy showing that P $\neq$ NP, either. These two oracles underscore the difficulty of the P versus NP question. Any resolution of this question will need new, possibly radically new, techniques.[5] Most of the current open problems in complexity theory relativize in both directions, and hence are probably difficult to solve. There are many oracle results relating to quantum complexity classes, some of which we will present after we define relativization in the quantum computation model.

Oracles are also useful for defining new complexity classes. If $\mathcal{C}$ and $\mathcal{D}$ are complexity classes, then we define $\mathcal{C}^{\mathcal{D}} = \bigcup \{\mathcal{C}^A : A \in \mathcal{D}\}$. For example, P^{NP} is the class of all languages that are easily decidable given access to an NP oracle. A typical problem in P^{NP} is: "Given a graph G and $s \in \mathbb{N}$, is s the size of the largest clique in G?" This problem is not known to be either in NP or in coNP, but it can be decided using binary search on s with access to the CLIQUE problem as an oracle.

[5] There are some nonrelativizing techniques. A prime example is the technique of low-degree polynomial interpolation, which has been used to show that all PSPACE languages have interactive proofs [32, 37], even though there is an oracle relative to which some languages in coNP do not have interactive proofs [23]. Despite their early promise, however, and despite their great utility in other areas, these techniques have yet to resolve any open questions about more "traditional" time-bounded complexity classes.

1.2.2.5 Quantum algorithms and FQP

Classical algorithms can be expressed efficiently with Boolean circuits, where an input string of length n is regarded as the Boolean settings of the input nodes of an n-input circuit. Since an algorithm should run on inputs of any length, we model the algorithm by an infinite family $C_0, C_1, C_2, \ldots$ of Boolean circuits, where C_i has i input nodes and handles all input strings of length i. If the algorithm decides a language, then each C_i should have exactly one output; an output value of 1 signifies acceptance, and 0 signifies rejection. If an algorithm is efficient (polynomial time, say) and we wish to express it efficiently by a family of circuits, then we must have some efficient way of generating each circuit in the family. A family $\{C_i\}_{i\in\mathbb{N}}$ of circuits is *p-uniform* if there is a polynomial time algorithm which, on input 1^i, outputs a full description of C_i. In this case, note that each C_i is not too large, and so can be simulated efficiently. Indeed, we have the following well-known proposition (see, e.g., [47]):

PROPOSITION 1.3
For any language L, $L \in \mathrm{P}$ if and only if L is decided by a p-uniform family of Boolean circuits.

Quantum algorithms can be expressed using quantum circuits. Here, an input string x of length n is "fed" into an n-input quantum circuit by setting the quantum state of the input wires (qubits) to $|x\rangle\langle x|$. Our practical considerations in Section 1.2.1 suggest that a quantum operation cannot operate reliably on more than a few qubits at a time; therefore, we set a fixed limit on the indegree and outdegree of any quantum gate in any quantum circuit. A constant of three is convenient, although two actually suffices; and in fact, the only two-qubit gate needed is the controlled NOT gate [6]. For efficient quantum computation, we again insist on a p-uniform family of quantum circuits.

The p-uniformity requirement raises some subtle issues. First, one may ask if p-uniformity is too strict; perhaps some *quantum* process for circuit fabrication is more powerful than a classical one. Second, our requirement implies that each quantum gate must be completely represented by a finite (polynomial size) amount of classical information, whereas the matrix representation of a general quantum operation may contain arbitrary complex numbers, which are not finitely representable.

To address the first issue, we must have a precise concept of a circuit-fabricating quantum process. Such a process should be algorithmic and allow for all possible inputs (of all possible lengths). Here we can turn to the QTM model [15], which was historically the first reasonable model of quantum computation. A single, finitely describable QTM can handle inputs of any and all lengths, so a QTM can be used to fabricate quantum circuits. It was later shown by Yao [49] that QTMs can be simulated efficiently by p-uniform families of quantum circuits (and vice versa). Thus, a family of quantum circuits fabricated by a quantum process as above is no more or less computationally powerful than a p-uniform family of quantum circuits.

The second issue is explained based on physics and some intuition. Since quantum computers do not currently exist, the first physical process for building a quantum computer must be described and implemented classically. So it is reasonable to suppose that any physical procedure for efficiently fabricating a quantum computer, including calibrating the various components, must ultimately rely on an efficient classical algorithmic description. In keeping with these considerations, Bernstein and Vazirani settled upon the requirement that all the transition amplitudes in a QTM should be polynomial-time approximable [10]. [A complex number z is *polynomial-time approximable* if there is a polynomial time algorithm that on input 1^r outputs three integers s, x, y such that $|z' - z| \leq 2^{-r}$, where $z' = 2^{-s}(x + iy)$.] An additional *de facto* requirement is that any QTM M only uses a finite set of transition amplitudes, since these amplitudes are part of the finite description of M's transition function. These two requirements are not overly strict. All currently known and forseeable quantum algorithms easily fit this requirement. Furthermore, any algorithm that violated this requirement would, at the very least, require significant justification, and even then it may not be universally accepted by the research community.

These requirements are also not overly lax; the univeral QTM constructed by Bernstein and Vazirani [10], which can efficiently approximate any QTM satisfying these requirements, uses only classical transitions (amplitudes in $\{0, 1\}$) and the one-qubit Hadamard transition (amplitudes in $\left\{-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}$).

Yao's simulation of a QTM by a quantum circuit family and vice versa [49] does not disturb the set of amplitudes involved too much, except for arithmetic operations. In particular, all the gates used in a p-uniform quantum circuit family simulating a QTM can have matrix entries drawn from some finite set depending only on the QTM being

simulated. These entries will also lie in the field over $\mathbb{Q}$ generated by the transition amplitudes of the QTM. The converse also holds: such a p-uniform quantum circuit family F can be efficiently simulated by a QTM whose transition amplitudes all lie in the field over $\mathbb{Q}$ generated by the matrix entries in the gates of F. We will therefore adopt the reasonable requirement that for every circuit family there is a fixed finite set of amplitudes from which all matrix entries of all gates in the family are drawn.

Bernstein and Vazirani's universal QTM together with Yao's results imply that there is a single finite set of amplitudes that works for all efficient quantum circuit families. In fact, circuits with Toffoli gates (described in Section 1.2.3 below) and Hadamard gates suffice to simulate the universal QTM, and hence they suffice for all efficient quantum computation. A Toffoli gate itself can be simulated (exactly) using

- the one-qubit Hadamard gate H,
- the controlled NOT gate $\wedge_1(\sigma_x)$, and
- the one-qubit conditional phase shift, or $\pi/8$ gate with underlying unitary matrix $\sigma_z^{1/4} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$ [6, 33].

Thus we call these three gates a *universal set* in analogy with classical Boolean circuits. The gates in this set can all be implemented fault-tolerantly [12]. Several other universal sets have been found [6]. (One should add to any such list some gates for nonunitary operations, such as measurement and partial trace.) There are other universal sets where all amplitudes in the underlying unitary matrices are drawn from the set $\{-1, -\frac{4}{5}, -\frac{3}{5}, 0, \frac{3}{5}, \frac{4}{5}, 1\}$ [43].

For any subset $B \subseteq \mathbb{C}$, we say that a quantum circuit family F is *over* B if all matrix elements of all gates of all circuits in F are elements of B. We will get stronger negative results if we do not insist on a fixed finite universal set of gates for all circuit families, but rather allow different kinds of gates in different circuit families. We will, however, generally restrict B to be the field of algebraic numbers. Algebraic numbers are all polynomial-time approximable (by Newton's method, for example), and such a set of operations is clearly universal. Moreover, all currently known quantum algorithms are expressed directly and easily using algebraic amplitudes.

We now define the input-output behavior of a quantum circuit C with n inputs and m outputs. Each gate g of C represents a quantum

operation from a multiqubit Hilbert space $\mathcal{H}$ to a multiqubit space $\mathcal{J}$, taking any state ρ of $\mathcal{H}$ to the state $g(\rho)$ of $\mathcal{J}$. Let $g_1, \ldots, g_s$ be a (topologically sorted) list of the gates of C, so that no output wires of any g_i are inputs to g_j for $j \leq i$. Suppose the input qubits are in state ρ_0 of space $\mathcal{H}_0$. We first apply g_1 by expressing $\mathcal{H}_0$ as the tensor product $\mathcal{H}_0' \otimes \mathcal{H}_0''$, where $\mathcal{H}_0'$ is the space of qubits entering gate g_1 and $\mathcal{H}_0''$ is the space of qubits bypassing g_1. The gate g_1 maps states of $\mathcal{H}_0'$ to states of some space $\mathcal{H}_1'$, in which case the state of all the qubits after applying g_1 is $\rho_1 = (g_1 \otimes \mathcal{I})(\rho_0)$ of the Hilbert space $\mathcal{H}_1 = \mathcal{H}_1' \otimes \mathcal{H}_0''$.

We then apply g_2 to the state ρ_1 in a similar manner, again keeping track of the qubits bypassing g_2, to obtain a state ρ_2 of a Hilbert space $\mathcal{H}_2$. We then apply gate g_3, and so on. After all the gates have been applied, we obtain a state ρ_s of a space $\mathcal{H}_s$ of at least m qubits, where the first m qubits, say, are the designated outputs. (The state ρ_s is independent of the ordering of the gates as long as they are topologically sorted; for if two gates could be swapped in the ordering, then they act on disjoint sets of qubits and hence the corresponding operations commute.) Finally, we observe the values of the m output qubits (a simultaneous projective measurement), from which we obtain some random variable ranging over the 2^m possible outcomes (classical bit strings of length m).

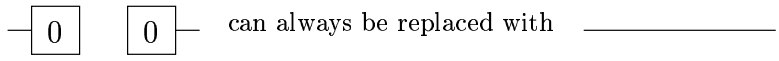
A function f on Σ^* is a *probabilistic function* if, for all $x \in \Sigma^*$, $f(x)$ is a random variable ranging over Σ^m for some m depending on $|x|$. For any set $B \subseteq \mathbb{C}$, we let FQP_B [2] denote the class of all probabilistic functions computed by p-uniform quantum circuit families over B as described above. If we drop the subscript on FQP, then we mean FQP_A, where A is the field of algebraic numbers.

As we mentioned earlier, a partial trace gate acts on all the current qubits at once but we only attach it to one wire. This means we have more freedom to place the gate in a topological sort of all the gates of the circuit. Our freedom does not lead to any ambiguity, though, because the tr-gate commutes with other tr-gates and with unitary gates. For example, both of the circuits

map an input operator $A \otimes B$ to the output operator $(\mathrm{tr}B)UAU^\dagger$. Similarly, ancilla gates commute with each other and with unitary gates. These facts allow us to place all the ancilla gates to the left (first in the topological sort) and all the tr-gates to the right (last), with all the

intermediate gates being unitary. We will say that such a circuit is in *standard form* with *width* equal to the number of qubits present at any point between the ancillæ and tr-gates.

Occasionally, when we "trace out" a qubit (that is, apply a tr-gate to it) in a circuit C, we can guarantee that the state of the qubit is $|0\rangle\langle 0|$. If this is the case, we may label the tr-gate with a 0 instead of the usual tr. If we embed C into a larger circuit, asserting that a qubit is in the zero-state allows it to be reused later as an ancilla, making the larger circuit more efficient. Put another way,



Another good reason to reset ancilla qubits to 0 before tracing them out is that this makes the input–output behavior of C to be unitary. This arises especially when we simulate a classical gate (see Section 1.2.3 below): if the ancilla is in an arbitrary "garbage" state (which may be entangled with the other qubits) when we trace it out, then we will in general get a mixed state result, even though the input was a pure state. This happens when the input state is a nontrivial superposition of basis states. Circuits that avoid this problem by resetting their ancillæ before tracing them out are called *clean circuits*. If a classical circuit is not clean, there is a straightforward way to clean it up that essentially doubles the number of gates. This is often not the most efficient way, though.

1.2.2.6 The classes BPP and BQP

Decision problems with efficient probabilistic algorithms make up the class BPP (Bounded-error Probabilistic Polynomial time). Decision problems with efficient bounded error quantum algorithms make up the class BQP [10]. AWPP is a counting class defined in [19]. After formally defining these classes, we will give proofs that $\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{BQP} \subseteq \mathrm{AWPP} \subseteq \mathrm{PP}$. The first inclusion is obvious and well known. The second inclusion was proved by Bernstein and Vazirani [10], the third by Fortnow and Rogers [22], and the last appeared in [19].

DEFINITION 1.1 *A language L is in* BPP *if there is a* P *predicate R and a polynomial p such that, for all $x \in \Sigma^*$ and for $m = p(|x|)$,*

$$x \in L \Rightarrow \|\{y \in \Sigma^* : |y| = m \wedge R(x,y)\}\| \geq (2/3)2^m,$$
$$x \notin L \Rightarrow \|\{y \in \Sigma^* : |y| = m \wedge R(x,y)\}\| \leq (1/3)2^m.$$

Given x and m as above, if y is chosen uniformly at random among the strings of length m, then $R(x,y)$ computes $L(x)$ correctly with a probability of at least $2/3$. By repeating the computation $R(x,y)$ several times with independently chosen y's and then taking the majority answer, we can make the probability of error exponentially close to zero.

PROPOSITION 1.4
If $L \in$ BPP, then for every polynomial q there is a polynomial p and a P *predicate R such that, for all $x \in \Sigma^*$ (setting $r = q(|x|)$ and $m = p(|x|)$),*

$$x \in L \Rightarrow \|\{y \in \Sigma^* : |y| = m \wedge R(x,y)\}\| \geq 2^m - 2^{m-r},$$
$$x \notin L \Rightarrow \|\{y \in \Sigma^* : |y| = m \wedge R(x,y)\}\| \leq 2^{m-r}.$$

A well-known problem in BPP that is not known to be in P is the PRIMALITY problem: "Given a natural number x, is x prime?" Primality testing is especially useful for public key cryptography.

The class BQP can be defined in terms of FQP. A *binary probabilistic function* is a probabilistic function whose output random variables range over the set $\{0,1\}$. A language L is in BQP if and only if there is a binary probabilistic function $f \in$ FQP such that, for all $x \in \Sigma^*$,

$$x \in L \Rightarrow \text{Prob}[\, f(x) = 1 \,] \geq 2/3,$$
$$x \notin L \Rightarrow \text{Prob}[\, f(x) = 1 \,] \leq 1/3.$$

By combining several simultaneous computations of f, we can diminish the error probability as we did with BPP.

LEMMA 1.3
If $L \in$ BQP, then for every polynomial p there is a binary probabilistic $f \in$ FQP such that, for all $x \in \Sigma^*$ (setting $r = p(|x|)$),

$$x \in L \Rightarrow \text{Prob}[\, f(x) = 1 \,] \geq 1 - 2^{-r},$$
$$x \notin L \Rightarrow \text{Prob}[\, f(x) = 1 \,] \leq 2^{-r}.$$

The set of possible *exact* distributions output by FQP functions may be quite sensitive to the choice of allowed gates or amplitudes in the quantum circuit model, but the set of possible *approximable* distributions—and hence the class BQP—is not very sensitive at all. As we mentioned before, Bernstein and Vazirani's universal QTM uses only classical transitions (i.e., those that preserve basis states) and the one-qubit Hadamard transform. The corresponding circuit family thus only

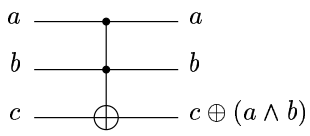
needs classical gates (see below) and the one-qubit Hadamard gate. These gates have all amplitudes in $\mathbb{Q}$—actually in the set $\{0, \pm\frac{1}{2}, 1\}$. We can therefore simulate any QTM *à la* [10] to good approximation and thus get the same class BQP, even if we restrict our gates to using these amplitudes. Actually there is a bit of a cheat here. Although the Hadamard gate (the superoperator) uses rational matrix elements, its underlying unitary operator has elements $\pm\frac{1}{\sqrt{2}} \notin \mathbb{Q}$. Adleman, DeMarrais and Huang [1] and independently Solovay and Yao [43] showed that rational transition amplitudes in the set $\{0, \pm\frac{3}{5}, \pm\frac{4}{5}, \pm 1\}$ suffice even for the underlying unitary operators for a universal QTM. All these results show that BQP is a very robust class.

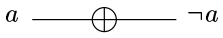
An interesting subclass of BQP is EQP (Exact Quantum Polynomial time [1]), defined just as with BQP except no error is allowed—that is, the allowed acceptance probabilities are 1 and 0. Strangely, EQP is even less sensitive to the allowed amplitudes than BQP. Adleman et al. showed that EQP remains the same even if arbitrary complex amplitudes are allowed, and that this is not the case with BQP.

1.2.3 Classical computations on a quantum circuit

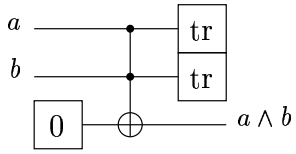
A unitary gate is *classical* if its matrix elements are in the set $\{0, 1\}$. Classical gates map basis states onto basis states and so do not introduce quantum superpositions. If a circuit's intermediate gates are all classical, then a classical input (basis state) yields a classical output (basis state). An important result in quantum computation is that any Boolean circuit (and hence any classical computation) can be simulated efficiently by a quantum circuit made up of classical gates. The difficulty is that classical gates in a quantum circuit must obey restrictions that Boolean gates in a classical circuit need not. Boolean gates may be irreversible, losing information from input to output, whereas classical gates in a quantum circuit must be reversible since they correspond to unitary operations. In addition, the output of a Boolean gate may be freely duplicated on several output wires, but this is not possible with a classical gate in a quantum circuit.

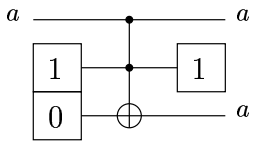
Both of these difficulties can be overcome by allowing more ancilla qubits for the simulating quantum circuit. Two types of classical gates—the *Toffoli gate* and the NOT gate—



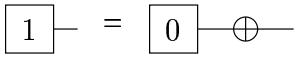


suffice to build circuits simulating any classical Boolean circuit when provided with a small number of extra ancillæ. The Boolean AND and COPY gates are simulated as follows:

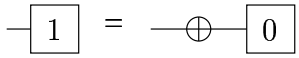




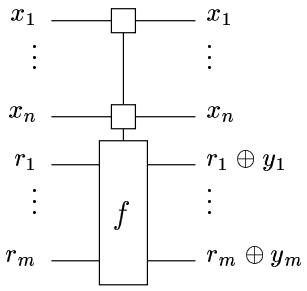
where



and



We will allow classical gates in quantum circuits that encapsulate arbitrary efficient classical algorithms, as is justified by the previous considerations. More precisely, for any polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ whose output length depends only on the input length, and for any $n \in \mathbb{N}$, we have the $(n+m)$-qubit gate U_f, which we depict as

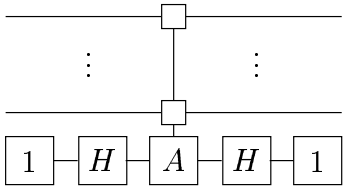


where $f(x_1 \cdots x_n) = y_1 \cdots y_m$. We call such a gate an *f-gate*. Our decidedly nonstandard depiction of this gate is meant to distinguish the output wires (bottom) from the input wires (top) by connecting the inputs via the little boxes. One can view this gate as shorthand for a quantum subcircuit computing f on strings of length n. Note that f must be computed by a clean circuit to guarantee that U_f is a unitary gate.

If A is an n-ary Boolean function outputting a single bit, then we will call the *A-inversion gate* the unitary gate whose underlying unitary operator I_A is defined by

$$I_A|x\rangle = (-1)^{A(x)}|x\rangle$$

for all $x \in \Sigma^n$. Inversion gates are used quite often in quantum algorithmics. The I_A gate is easy to implement with a single A-gate:



1.2.4 Relativizing quantum computation

To prove oracle results relating to quantum computation, we need to define exactly what it means for a quantum circuit to have access to an oracle. Let $f : \Sigma^* \to \Sigma^*$ be any function (not necessarily computable) whose output length depends only on the input length. There is an f-gate for every $n \in \mathbb{N}$. Note that f-gates cannot be viewed as shorthand for subcircuits, for f may not be computable by any circuit at all. A quantum circuit gains access to f as an oracle by using f-gates. This is completely analogous to the standard way of relativizing Boolean circuits. The relativized class FQP^f can now be defined just as with FQP, except now we allow the quantum circuits to have f-gates and we also allow the polynomial-time circuit-fabrication algorithm access to f as an oracle (the latter allowance is actually unnecessary). A language A can be accessed by a quantum circuit as an oracle since it corresponds to a function with output length 1. In this case, we refer to the oracle gates as A-gates.

An oracle in a quantum algorithm is often referred to as a "black box," and the algorithm is a "black-box algorithm." One can think of a black-box algorithm as computing with some unknown classical gate behind an abstraction barrier. We regard the classical gate—the black box—as another kind of input to the algorithm, and the algorithm extracts information about the black box. Many interesting quantum algorithms are black-box algorithms. We will mention two early black-box algorithms, a variant of one due to Deutsch and Josza [17] and the other due to Simon [40] in Section 1.4.1 below.

1.3 Equivalence of FQP and GapP

In this section we relate FQP closely to GapP. In doing so, we get a deep connection between the complexity of quantum computation and classical counting complexity. The following theorem shows that GapP is at least as powerful a class as FQP. It is adapted from [20], which itself is a straightforward generalization of a result by Fortnow and Rogers [22], which in turn improves an earlier result announced by Valiant (see [10]) essentially showing that $\mathrm{BQP} \subseteq \mathrm{P}^{\mathrm{GapP}}$.

If we consider some field extension K of $\mathbb{Q}$ with finite basis $\alpha_1, \ldots, \alpha_e$ over $\mathbb{Q}$, then a quantum circuit over K can be finitely described by representing its matrix elements as vectors of rational coefficients of the α_i.

THEOREM 1.1

Fix a field $K \subseteq \mathbb{C}$ with finite dimension over $\mathbb{Q}$, and fix a basis $\alpha_1, \ldots, \alpha_e$ for K over $\mathbb{Q}$ such that $\alpha_1, \ldots, \alpha_d$ (for some $d \leq e$) span $K \cap \mathbb{R}$ (as a vector space over $\mathbb{Q}$). There are GapP *functions $h_1, \ldots, h_d$ that behave as follows: let C be any quantum circuit over K with set of matrix coefficients $B \subseteq K$, and suppose C has n inputs and m outputs. Then there is an integer $D_B > 0$ such that for all $x \in \Sigma^n$,*

$$\mathrm{Prob}[\ C \text{ outputs } y \text{ on input } x\ ] = D_B^{-r} \sum_{i=1}^{d} h_i(C, x, y)\alpha_i,$$

where r is the number of unitary gates in C. Moreover, D_B depends only on B and is polynomial-time computable from B (or from C).

In the special case where $K = \mathbb{Q}$ and $\alpha_1 = 1$, there is a GapP *function h such that*

$$\mathrm{Prob}[\ C \text{ outputs } y \text{ on input } x\ ] = D_B^{-r} h(C, x, y).$$

COROLLARY 1.1

[[20, 22]] Suppose $f \in$ FQP is computed by a p-uniform quantum circuit family over a finite set B of algebraic numbers. Let K be the field extension of $\mathbb{Q}$ generated by B, and let $\alpha_1, \ldots, \alpha_d \in \mathbb{R}$ span $K \cap \mathbb{R}$. Then there exist an integer $D > 0$, an integer-coefficient polynomial p, and GapP

functions $g_1, \ldots, g_d$ *such that, for all* $x, y \in \Sigma^*$ *(setting* $r = p(|x|)$*),*

$$\text{Prob}[\, f(x) = y \,] = D^{-r} \sum_{i=1}^{d} g_i(x,y)\alpha_i.$$

In particular, if $B \subseteq \mathbb{Q}$*, we have* $\text{Prob}[\, f(x) = y \,] = D^{-r} g(x,y)$ *for some* GapP *function* g*.*

PROOF Extend $\alpha_1, \ldots, \alpha_d$ to a basis $\alpha_1, \ldots, \alpha_e$ of K. Apply Theorem 1.1 to get $h_1, \ldots, h_d$ and D_B. Set $D = D_B$. Let $C_0, C_1, C_2, \ldots$ be a p-uniform quantum circuit family computing f, where C_n has n inputs. Let $p(n)$ be a polynomial upper bound on the number r_n of unitary gates in C_n. Then for all $x \in \Sigma^*$ of length n we have

$$\text{Prob}[\, f(x) = y \,] = D^{-p(n)} \sum_{i=1}^{d} g_i(x,y)\alpha_i,$$

where $g_i(x,y) = D^{p(n)-r_n} h(C_n, x, y)$ is clearly a GapP function. ∎

PROOF of Theorem 1.1 We prove the special case where $K = \mathbb{Q}$, and sketch the proof of the general case. A more detailed proof of the general case can be found in [20].

We are given a quantum circuit C with n inputs, m outputs, and set of matrix elements $B = \left\{ \frac{n_1}{d_1}, \ldots, \frac{n_s}{d_s} \right\}$ for integers n_i and d_i. Set $D_B = \text{lcm}(d_1, \ldots, d_s)$. By rearranging gates if necessary, we can assume that C is in standard form with width q and (unitary) gates $g_1, \ldots, g_r$ in some topological order. Each gate g_ℓ corresponds to a quantum operation G_ℓ mapping $\mathcal{L}(\mathcal{H})$ to $\mathcal{L}(\mathcal{H})$ for some fixed q-qubit (2^q-dimensional) Hilbert space $\mathcal{H}$. Thus each G_ℓ is represented by a $Q \times Q$ matrix with entries in B, where $Q = 2^{2q}$. By our choice of D_B, each $D_B G_\ell$ is an integer matrix. It is clear that there is a polynomial-time computable function (and hence a GapP function) $a(C, \ell; i, j)$ which computes the (i,j)th entry of $D_B G_\ell$. By Lemma 1.2, the composition of all the G_ℓ, and thus the computation of the entire circuit, is computed by a function $D_B^{-r} b(C; i, j)$, where $b \in$ GapP because

$$[\, b(C) \,]^{Q \times Q} = [\, a(C,r) \,]^{Q \times Q} \cdots [\, a(C,1) \,]^{Q \times Q}.$$

(Note that we do not need to supply r as an input to b, because it can be computed from C.)

Let $M_C = D_B^{-r}[\, b(C)\,]^{Q\times Q}$. An input string $x \in \Sigma^n$ is fed to C via the input state $\rho_0 = |x\rangle\langle x| \otimes |0^{q-n}\rangle\langle 0^{q-n}|$ which includes the ancilla qubits. If we write ρ_0 as a column vector V of length Q, then the final state of the circuit (just before the tr-gates are applied and the output qubits are measured) is $\rho = M_C V$ (in column vector form). Using Lemma 1.2 again and reshaping the column vector as a matrix, we get a GapP function $c(C, x; i, j)$ such that ρ is represented by the matrix $D_B^{-r}[\, c(C,x)\,]^{2^q\times 2^q}$.

We can write ρ as the sum

$$\rho = \sum_{y_1, y_2 \in \Sigma^m} |y_1\rangle\langle y_2| \otimes A_{y_1 y_2},$$

where the $|y_1\rangle\langle y_2|$ correspond to the m output qubits and the $A_{y_1 y_2}$ are operators on the space of the remaining $q-m$ qubits. For any $y \in \Sigma^m$, the probability of y being the value of the final output measurement is

$$\mathrm{tr}(P_y \rho) = \mathrm{tr} A_{yy} = D_B^{-r} \sum_{z:|z|\,=\,q \text{ and } z \text{ has prefix } y} c(C, x; z, z).$$

The sum on the right-hand side is clearly a GapP function of C, x, and y by Proposition 1.1. Letting $h(C, x, y)$ be this sum proves the special case of the theorem.

We now sketch the proof of the general case. Let K, d, and $\alpha_1 \ldots, \alpha_e$ be given as in the theorem. For all $1 \le i, j, k \le e$, let $c_{jk}^i \in \mathbb{Q}$ be such that $\alpha_j \alpha_k = \sum_{i=1}^e c_{jk}^i \alpha_i$. By rescaling the α_i if necessary, we can assume that all the c_{jk}^i are integers. Now the idea is that we can represent scalars in K such as individual matrix entries uniquely as vectors over $\mathbb{Q}$ of length e. Addition of scalars corresponds to vector addition, and scalar multiplication is computed using the c_{jk}^i. GapP is closed under all of these operations, and so we can generalize Lemma 1.2 to the case where matrix entries are elements of K. We then apply the generalized lemma much as before. Given circuit C with set $B = \{b_1, \ldots, b_s\}$ of matrix elements, we must choose D_B so that we always compute with integer entries in the scalar representations: Let $n_{ij}, d_{ij} \in \mathbb{Z}$ be such that each $b_i = \sum_{j=1}^e \frac{n_{ij}}{d_{ij}} \alpha_j$. We can let D_B be the least common multiple of all the d_{ij}.

Analoguous with the previous argument, we now get GapP functions $c_1(C, x; i, j), \ldots, c_e(C, x; i, j)$ such that the final state

$$\rho = D_B^{-r} \sum_{\ell=1}^{e} \alpha_\ell [\, c_\ell(C, x)\,]^{2^q \times 2^q}.$$

And for $y \in \Sigma^m$, we get

$$\text{Prob}[\, f(x) = y \,] = \sum_{z:|z| = q \text{ and } z \text{ has prefix } y} \rho_{zz},$$

where $\rho_{zz} = D_B^{-r} \sum_{\ell=1}^{e} \alpha_\ell c_\ell(C, x; z, z)$, the (z,z)th entry of the matrix ρ. Since ρ is Hermitian, each ρ_{zz} is real, and so by the linear independence of the α_ℓ we must have $c_\ell(C, x; z, z) = 0$ for $d < \ell \leq e$. Therefore, setting

$$h_\ell(C, x, y) = \sum_{z:|z| = q \text{ and } z \text{ has prefix } y} c_\ell(C, x; z, z),$$

for $1 \leq \ell \leq d$, it is clear that each $h_\ell \in \text{GapP}$, and the equality

$$\text{Prob}[\, f(x) = y \,] = D_B^{-r} \sum_{\ell=1}^{d} h_\ell(C, x, y) \alpha_\ell$$

holds as desired. ∎

COROLLARY 1.2
If $f \in \text{FQP}_B$ where $B = \left\{0, \pm\frac{1}{2}, \frac{1 \pm i}{\sqrt{2}}, \pm 1\right\}$ as in the case for the universal set of gates described in Section 1.2.2.5 above, then there are $h_1, h_2 \in \text{GapP}$ such that

$$\text{Prob}[\, f(x) = y \,] = 2^{-r} \left(h_1(x, y) + \sqrt{2} h_2(x, y) \right),$$

where r is a polynomial in $|x|$.

There is a partial converse to Corollary 1.1. The following theorem was shown in [20].

THEOREM 1.2
Let $B = \left\{0, \pm\frac{1}{2}, 1\right\}$. For any $h \in \text{GapP}$ there is an $f \in \text{FQP}_B$ and an integer-coefficient polynomial p such that for all $x \in \Sigma^*$,

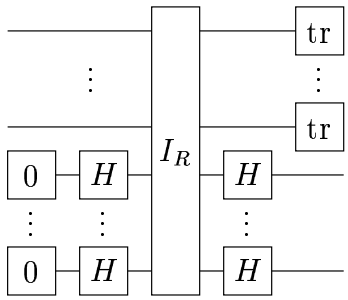
$$\text{Prob}[\, f(x) = 0^r \,] = 2^{-2(r+1)} (h(x))^2,$$

where $r = p(|x|)$.

PROOF By Lemma 1.1 there is a predicate $R \in \text{P}$ and polynomial p such that for any $x \in \Sigma^*$,

$$h(x) = \frac{1}{2} \left(\|\{y \in \Sigma^r : R(x, y)\}\| - \|\{y \in \Sigma^r : \neg R(x, y)\}\| \right),$$

where $r = p(|x|)$. For $n \in \mathbb{N}$, let C_n be the quantum circuit with n inputs and r ancillæ, where $r = p(n)$, depicted below.



On input $|x\rangle\langle x|$ where $|x| = n$, the circuit C_n transforms the quantum state as follows:

$$\begin{aligned}
|x\rangle\langle x| &\overset{0}{\mapsto} |x, 0^r\rangle\langle x, 0^r| \\
&\overset{H}{\mapsto} 2^{-r} \sum_{y,y' \in \Sigma^r} |x, y\rangle\langle x, y'| \\
&\overset{I_R}{\mapsto} 2^{-r} \sum_{y,y'} (-1)^{R(x,y)+R(x,y')} |x, y\rangle\langle x, y'| \\
&\overset{H}{\mapsto} 2^{-2r} \sum_{y,y',z,z'} (-1)^{R(x,y)+R(x,y')+y\cdot z+y'\cdot z'} |x, z\rangle\langle x, z'| \\
&\overset{\mathrm{tr}}{\mapsto} 2^{-2r} \sum_{y,y',z,z'} (-1)^{R(x,y)+R(x,y')+y\cdot z+y'\cdot z'} |z\rangle\langle z'|.
\end{aligned}$$

The probability that $f(x)$ equals 0^r is given by the coefficient of $|0^r\rangle\langle 0^r|$ in the final state, which is

$$\begin{aligned}
& 2^{-2r} \sum_{y,y'} (-1)^{R(x,y)+R(x,y')} \\
&= 2^{-2r} \left(\sum_{y} (-1)^{R(x,y)} \right)^2 \\
&= 2^{-2r} (-2h(x))^2 \\
&= 2^{-2(r+1)} (h(x))^2.
\end{aligned}$$

∎

An easy application of Corollary 1.1 and Theorem 1.2 is a characterization of the class NQP, a quantum analogue of NP, defined by Adleman, DeMarrais, and Huang [1].

DEFINITION 1.2 *[[1]] Let $B \subseteq \mathbb{C}$. A language L is in the class NQP_B if there is an $f \in \mathrm{FQP}_B$ such that for all $x \in \Sigma^*$,*

$$x \in L \iff \mathrm{Prob}[\, f(x) = 0 \,] > 0.$$

THEOREM 1.3 [20]
For any finite set B of algebraic numbers containing $\left\{0, \pm\frac{1}{2}, 1\right\}$, we have $\mathrm{NQP}_B = \mathrm{coC_{=}P}$.

PROOF Let $K \subseteq \mathbb{C}$ be the field generated by B and let $\alpha_1, \ldots, \alpha_d$ be a basis of $K \cap \mathbb{R}$ over $\mathbb{Q}$. Suppose $L \in \mathrm{NQP}_B$ via some $f \in \mathrm{FQP}_B$, and let $D > 0$ and $g_1, \ldots, g_d \in \mathrm{GapP}$ be as in Corollary 1.1. Fix an input x. Since the α_i are linearly independent over $\mathbb{Q}$, we have $\mathrm{Prob}[\, f(x) = 0 \,] = 0$ just in the case that $g_1(x, 0) = \cdots = g_d(x, 0) = 0$. It follows that $L \in \mathrm{coC_{=}P}$ via the GapP function

$$h(x) = \sum_{i=1}^{d} (g_i(x, 0))^2,$$

that is, $x \in L \iff h(x) \neq 0$.

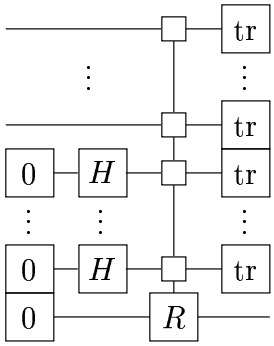
Conversely, suppose $L \in \mathrm{coC_{=}P}$ via some $h \in \mathrm{GapP}$. Let $f \in \mathrm{FQP}_B$ be given by Theorem 1.2. Clearly, $L \in \mathrm{NQP}_B$ witnessed by f. ∎

1.4 Strengths of the quantum model

In this section we give results that show that the class BQP is large. That is, we place lower bounds on the size of BQP. We first show that $\mathrm{BPP} \subseteq \mathrm{BQP}$. This is an old and relatively easy result, first proved by Bernstein and Vazirani [10], but it remains the strongest result of its kind—showing that a previously studied complexity class is contained in BQP.

THEOREM 1.4 [10]
$\mathrm{BPP} \subseteq \mathrm{BQP}_{\mathbb{Q}}$.

PROOF Let L be in BPP with corresponding polynomial p and predicate $R \in \mathrm{P}$. For each $n \in \mathbb{N}$ and $r = p(n)$, consider the circuit with n inputs, $r+1$ ancillæ, and one output, depicted below.



On input $|x\rangle\langle x|$ where $x \in \Sigma^*$, the state after the R-gate is

$$2^{-r} \sum_{y,y' \in \Sigma^r} |x, y, R(x,y)\rangle\langle x, y', R(x,y')|.$$

After the tr-gates are applied, the final state is

$$2^{-r} \sum_{y} |R(x,y)\rangle\langle R(x,y)|.$$

The probability that the output qubit has value 1 is the coefficient of $|1\rangle\langle 1|$ in the final state. This probability is $2^{-r}\|\{y \in \Sigma^r : R(x,y)\}\|$, which is equal to the probability of $R(x,y)$ being true for random y. Thus the above family of circuits computes L as a BQP language. ∎

The other known positive results about BQP show that specific problems are in BQP. These problems come in two types: (1) black-box problems and (2) non-black-box problems (*concrete* problems) that are not known to be in BPP. Primary examples of such concrete problems are INTEGER FACTORIZATION and the DISCRETE LOGARITHM problem [39]. These two problems are easily computable relative to some NP oracle, but have no known efficient randomized solutions.

We will concentrate on black-box problems. These will give us oracle results of the form $\mathrm{BQP}^A \not\subseteq \mathcal{C}^A$ for some oracle A, where $\mathcal{C}$ is a complexity class. We will also obtain oracles for EQP as well.

1.4.1 Oracle results

A natural question to ask is: does NP contain BQP, or even EQP? Equivalently, does every problem decidable by an efficient quantum algorithm (with or without error allowed) have easily verifiable solutions? Another natural question is: does BPP = BQP? In other words, can quantum algorithms (with bounded error) be efficiently simulated by classical probabilistic algorithms (with bounded error)? If so, then quantum computation would be a lot less interesting.

We will address the second question first. INTEGER FACTORIZATION is in BQP and it is not *known* to be in BPP, so this gives credibility to the conjecture that BPP $\neq$ BQP, although it is no proof. Credibility of a different sort comes by way of an oracle B such that $\mathrm{BPP}^B \neq \mathrm{BQP}^B$, in fact, $\mathrm{EQP}^B \not\subseteq \mathrm{BPP}^B$. This oracle can be constructed as an instance of Simon's black-box problem [40], which we now describe. Suppose we are given a black-box function $f : \Sigma^n \to \Sigma^m$ with $n \leq m$, and there is an $s \in \Sigma^n$ such that, for all distinct $x, y \in \Sigma^n$,

$$f(x) = f(y) \iff x \oplus y = s.$$

Simon showed that deciding if $s = 0$ is a BQP problem relative to f. Moreover, the BQP^f algorithm can be used as a subroutine to find s with high probability. Later, Brassard and Høyer [13] showed how to find s with zero error probability, yielding an EQP^f algorithm. The set oracle B just codes the output bits of f. Existence of B signifies that proving BPP = BQP or even EQP $\subseteq$ BPP will at least be very difficult.

We now address the first question. It is clear that if $L \in \mathrm{BQP}$, then $\Sigma^* - L \in \mathrm{BQP}$ also. Therefore if BQP $\subseteq$ NP, then BQP $\subseteq$ NP $\cap$ coNP. The same goes for EQP replacing BQP. All concrete problems currently known to be in BQP are also in NP $\cap$ coNP.

Such a containment would be quite difficult to prove, however. There is an oracle A such that $\mathrm{EQP}^A \not\subseteq \mathrm{NP}^A$, so any proof that EQP $\subseteq$ NP would need nonrelativizable techniques. A can be obtained via the black-box Balance problem of Deutsch and Jozsa [17]. We will present a stronger oracle result here due to Green and Pruim [26] based on ideas of Boyer, Brassard, Høyer, and Tapp [11]: there is an oracle A such that $\mathrm{EQP}^A \not\subseteq (\mathrm{P}^{\mathrm{NP}})^A$. This result, as do most oracle results, comes in two pieces: first we describe a particular problem computed by an algorithm (here a quantum algorithm) that uses a single type of black-box gate. Then we concoct a particular black box that plugs into the quantum algorithm so that (1) the quantum algorithm computes the problem with zero probability of error and (2) the problem cannot be

solved by any P^{NP} algorithm, even one with access to the same black box as an oracle.

How do you relativize a class like P^{NP} to an oracle A, a class that is already defined in terms of oracles? You should at least let the polynomial time TM access an oracle $L \in \mathrm{NP}^A$ (instead of just NP). The "standard" heuristic for relativizing any class is to allow all computations free access to the oracle. In keeping with this, you should also allow the TM direct access to A itself, on a separate oracle tape, say. It turns out that this latter requirement is unnecessary, because information about A can be encoded directly into an appropriately chosen NP^A oracle L, so that the TM can access A indirectly through L. The preceding trick is expressed symbolically by the equation $(\mathrm{P}^{\mathrm{NP}})^A = \mathrm{P}^{(\mathrm{NP}^A)}$, with the right-hand side usually being written as just $\mathrm{P}^{\mathrm{NP}^A}$. This is the standard definition of P^{NP} relativized to A.

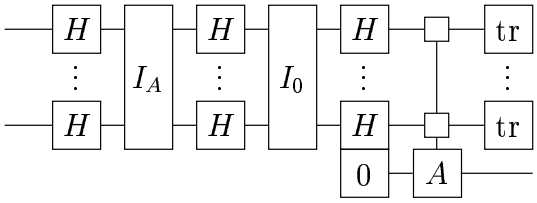
THEOREM 1.5 [26]

There is an oracle A such that $\mathrm{EQP}^A \not\subseteq \mathrm{P}^{\mathrm{NP}^A}$.

PROOF Green and Pruim use a quantum circuit C_n implicitly described by Boyer et al. [11] to solve a variant of the Deutsch–Josza Balance problem [17] with zero error. Fix a language $A \subseteq \Sigma^*$ as a black box with the promise that, for any length n, A either contains exactly one quarter or exactly three quarters of the strings of length n. The Modified Balance Problem relative to A (MBP^A) is the problem of determining which is the case for each length n, given input 0^n: accepting for one quarter, rejecting for three quarters. The circuit C_n that solves MBP has n inputs and employs two inversion gates I_0 and I_A, where for any $x \in \Sigma^n$,

$$I_0|x\rangle = \begin{cases} -|x\rangle & \text{if } x = 0, \\ |x\rangle & \text{if } x \neq 0, \end{cases}$$

That is, I_0 is the NOR-inversion gate. Note that I_0 has underlying unitary operator $I - 2|0^n\rangle\langle 0^n|$, where I is the identity operator on n qubits. The ciruit C_n is



The unitary operator underlying most of C_n is $H^{\otimes n} I_0 H^{\otimes n} I_A H^{\otimes n}$, where $H^{\otimes n}$ is simultaneous application of H on n qubits. (The operator $H^{\otimes n} I_0 H^{\otimes n} I_A$ is used in quantum search algorithms to find an element of A and is sometimes referred to as the Grover iterate [27, 28, 29].)

On input $|0^n\rangle\langle 0^n|$, we compute the state $|\psi\rangle\langle\psi|$ of C_n just before the last A-gate. For any $X \subseteq \Sigma^n$ let $|X\rangle = \sum_{x \in X} |x\rangle$ (unnormalized). For two such sets X and Y, note that $\langle X|Y\rangle = \|X \cap Y\|$. Let $A_n = A \cap \Sigma^n$ with cardinality s. We have $H^{\otimes n}|0^n\rangle = 2^{-n/2}|\Sigma^n\rangle$, and so up to a harmless global phase factor,

$$\begin{aligned}
|\psi\rangle &= H^{\otimes n} I_0 H^{\otimes n} I_A H^{\otimes n} |0^n\rangle \\
&= 2^{-n/2} H^{\otimes n} (I - 2|0^n\rangle\langle 0^n|) H^{\otimes n} I_A |\Sigma^n\rangle \\
&= 2^{-n/2} (I - 2^{1-n} |\Sigma^n\rangle\langle\Sigma^n|) I_A |\Sigma^n\rangle \\
&= 2^{-n/2} (I - 2^{1-n} |\Sigma^n\rangle\langle\Sigma^n|) (|A_n\rangle - |\Sigma^n - A_n\rangle) \\
&= 2^{-n/2} (|A_n\rangle - |\Sigma^n - A_n\rangle - 2^{1-n}(2s - 2^n)|\Sigma^n\rangle) \\
&= 2^{-n/2} (3|A_n\rangle + |\Sigma^n - A_n\rangle - 2^{2-n} s |\Sigma^n\rangle).
\end{aligned}$$

If $s = \frac{1}{4}2^n$, then $|\psi\rangle = 2^{1-n/2}|A_n\rangle$; and if $s = \frac{3}{4}2^n$, then $|\psi\rangle = -2^{1-n/2}|\Sigma^n - A_n\rangle$. If we observed the value of the first n qubits at this point, then we are guaranteed to see an element of A_n in the former case and $\Sigma^n - A_n$ in the latter. The last A-gate distinguishes between these two cases with certainty. This completes the description of C_n: if A satisfies the promise on every length, then $\mathrm{MBP}^A \in \mathrm{EQP}^A$ via the family of circuits C_n.

Our remaining job is to construct a particular A satisfying the promise such that $\mathrm{MBP}^A \notin \mathrm{P}^{\mathrm{NP}^A}$ (i.e., MBP^A is not "NP^A-easy"). To do this, we use a time-honored technique from logic and computer science known as diagonalization. A $\mathrm{P}^{\mathrm{NP}^A}$ "machine" involves a P^A predicate $R(x, y)$ and polynomial p, together with a polynomial time oracle TM M computing with oracle L^A, where L^A is the NP^A language determined by R and p. Each such machine has a finite description $\langle R, p, M\rangle$, so we can enumerate these machines as $N_1, N_2, \ldots$ in some way. If MBP^A is NP^A-easy, then there is some i such that N_i correctly computes MBP^A

on each input 0^n. Equivalently, MBP^A is *not* NP-easy if for each i there is a length n_i such that N_i computes the wrong value for MBP^A on input 0^{n_i}. We define A in stages $1, 2, \ldots$. At stage i we "diagonalize against N_i" by explicitly choosing a length n_i and specifying the set $A_{n_i} = A \cap \Sigma^{n_i}$ so as to *force* N_i to give the wrong answer on input 0^{n_i}. All the while, we must ensure that A satisfies the promise at each length; this keeps MBP^A a member of EQP^A.

How do we fool machine N_i on input 0^{n_i}? First we commit just enough of the oracle A to force N_i to give *some* answer. This does not require us to commit too many strings. Then we add just enough additional strings of length n_i to A to satisfy the promise in the sense opposite to N_i's answer. For lengths n other than the n_i, we satisfy the promise trivially by setting A_n to be the lexicographically least 2^{n-2} strings of length n—that is, all strings starting with 00.

For each $i \geq 1$ we let N_i correspond to R_i, p_i, M_i as described above, we let $q_i(n)$ be a strictly monotone polynomial bounding the running time of M_i on all inputs of length $\leq n$, and we let $r_i(n) > 0$ be a strictly monotone polynomial bounding the running time of R_i on all inputs (x, y) where $|x| \leq q(n)$ and $|y| = p(|x|)$. Also as described above, for any oracle X let L_i^X be the NP^X language corresponding to p_i and R_i relative to X. We let $n_0 = 0$, and for $i \geq 1$ we let $n_i \in \mathbb{N}$ be least such that $\frac{1}{4}2^{n_i} \geq q_i(n_i)r_i(n_i)$ and $n_i > r_j(n_{i-1})$ for all $j < i$. Given the way we defined the r_j, we have $n_i > n_{i-1}$ and all queries to our oracle made on behalf of N_j on input 0^{n_j} for $j < i$ have length $< n_i$. (This means that we can freely put strings of length n_i into or out of the oracle without disturbing any of the computations we have already forced.) We set $T = \{n_0, n_1, n_2, \ldots\}$.

At each stage of the construction, we will commit strings to be either in or out of the oracle we are building: for $i \in \mathbb{N}$ we define two sets $A_i \subseteq B_i \subseteq \Sigma^*$ such that $A_0 \subseteq A_1 \subseteq A_2 \subseteq \ldots$ and $B_0 \supseteq B_1 \supseteq B_2 \supseteq \ldots$. A_i is the set of strings we have committed to be in the oracle by the end of Stage i, and B_i is the set of strings we have *not* committed to be *out of* the oracle by the end of Stage i (thus the A_i sets get bigger as i increases, and the B_i sets get smaller). Strings in $B_i - A_i$ are *uncommitted*. All strings of length n_i are committed by the end of Stage i, and all strings with lengths not in T are committed at the beginning of the construction. The final oracle will be $A = \bigcup_i A_i = \bigcap_i B_i$.

Before Stage 1 we let

$$A_0 = \{00y : y \in \Sigma^* \wedge |00y| \notin T\} \quad \text{and} \quad B_0 = A_0 \cup \bigcup_{n \in T} \Sigma^n.$$

Construction of A
Stage $i \geq 1$:
Set $A_i = A_{i-1}$ and $B_i = B_{i-1}$.
Simulate M_i on input 0^{n_i}:
Whenever M_i makes a query x, do:
If $x \in L_i^C$ for some C with $A_i \subseteq C \subseteq B_i$, then
Let $y \in \Sigma^{p_i(|x|)}$ be least such that $R_i^C(x, y)$.
Let Q be the set of queries made while computing $R_i^C(x, y)$.
/* Commit the queries in Q: */
Set $A_i = A_i \cup (Q \cap C)$ and $B_i = B_i - (Q - C)$.
Answer "yes" to M_i's query.
Else,
Answer "no" to M_i's query.
Let $m = \|\Sigma^{n_i} \cap A_i\|$.
If M_i accepts in our simulation, then:
Add to A_i the first $\frac{3}{4}2^{n_i} - m$ uncommitted strings in Σ^{n_i}, and remove the rest of the length n_i uncommitted strings from B_i.
Else,
Add to A_i the first $\frac{1}{4}2^{n_i} - m$ uncommitted strings in Σ^{n_i}, and remove the rest of the length n_i uncommitted strings from B_i.
End of Stage i.
Set $A = \bigcup_i A_i$.
End of Construction

We complete the proof by making three observations. First, there are always enough uncommitted strings to add to A_i at the end of Stage i. All strings of length n_i are uncommitted at the start of Stage i because n_i is too big for any of these strings to be committed at earlier stages. Each query set Q in the simulation has cardinality at most $r_i(n_i)$ due to the running time of R_i. M_i can make at most $q_i(n_i)$ queries; so the total number of strings we commit during the simulation is at most $q_i(n_i)r_i(n_i) \leq \frac{1}{4}2^{n_i}$, by the choice of n_i.

Second, N_i cannot "change its mind" about 0^{n_i} after the simulation is finished in Stage i, no matter how we commit strings afterwards. If one of M_i's queries x is in L_i^C for some C such that $A_i \subseteq C \subseteq B_i$, then we commit the queries made by R_i^C as it accepts some appropriate (x, y), so R_i's behavior on (x, y) cannot change, and x will also be in L_i^A. Otherwise, if there is no such C, then $x \notin L_i^A$ since $A_i \subseteq A \subseteq B_i$. Thus we always answer M_i's query according to L_i^A, which makes M_i's accept/reject behavior in the simulation the same as it is relative to A.

Third and finally, A satisfies the promise at all lengths, and M_i (and thus N_i) with oracle A computes MBP^A incorrectly at length n_i. These facts follow from the definitions of A_0 and B_0 and from the number of strings we put into A at the end of Stage i. ∎

1.5 Limitations of the quantum model

Bernstein and Vazirani showed that $\mathrm{BQP} \subseteq \mathrm{P}^{\mathrm{PP}}$ $(= \mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{GapP}})$, establishing the first connection between quantum complexity and counting complexity. Fortnow and Rogers [22] tightened this connection by showing that $\mathrm{BQP} \subseteq \mathrm{AWPP}$, a counting class we will define shortly. Their results, together with previous results about AWPP [19], give the strongest evidence yet that $\mathrm{NP} \not\subseteq \mathrm{BQP}$, and thus NP-complete problems cannot be decided efficiently with quantum circuits. In particular, there is an oracle G such that $\mathrm{P}^G = \mathrm{BQP}^G \neq \mathrm{NP}^G$. This complements an earlier result of Bennett, Bernstein, Brassard, and Vazirani [9], which states that $\mathrm{NP}^R \not\subseteq \mathrm{BQP}^R$ with probability 1, where the oracle R is chosen at random. (They also show that $\mathrm{NP}^{R'} \cap \mathrm{coNP}^{R'} \not\subseteq \mathrm{BQP}$ with probability 1, where R' is a permutation oracle chosen at random.)

The class AWPP is somewhat analogous to BPP.

DEFINITION 1.3 *[[19, 31]] A language L is in* AWPP *if and only if, for every polynomial q, there is a polynomial-time computable function $p > 0$ and a* GapP *function f such that*

$$x \in L \Rightarrow (1 - 2^{-r})m \leq f(x) \leq m,$$
$$x \notin L \Rightarrow 0 \leq f(x) \leq 2^{-r}m$$

for all $x \in \Sigma^*$, where $r = q(|x|)$ and $m = p(1^{|x|})$.

It is easy to see that $\mathrm{AWPP} \subseteq \mathrm{PP}$: for $L \in \mathrm{AWPP}$, setting $q(|x|) = 2$ gives an f and $p > 0$ as in the definition, whence the GapP function $g(x) = f(x) - \lfloor p(1^{|x|})/2 \rfloor$ witnesses that $L \in \mathrm{PP}$.

THEOREM 1.6 [22]
$\mathrm{BQP} \subseteq \mathrm{AWPP}$.

PROOF Let L be a language in BQP, and let q be any polynomial. We will find a polynomial-time computable function $p > 0$ and GapP function f that satisfy Definition 1.3 for L. By Lemma 1.3 there is a binary $g \in \mathrm{FQP}_{\mathbb{Q}}$ such that

$$x \in L \Rightarrow \mathrm{Prob}[\, g(x) = 1 \,] \geq 1 - 2^{-r},$$
$$x \notin L \Rightarrow \mathrm{Prob}[\, g(x) = 1 \,] \leq 2^{-r},$$

for any $x \in \Sigma^*$ with $r = q(|x|)$. By Theorem 1.1, there is an integer $D > 0$, polynomial s and GapP function f such that $\mathrm{Prob}[\, g(x) = 1 \,] = D^{-s(|x|)} f(x)$ for all $x \in \Sigma^*$, and so

$$x \in L \Rightarrow 1 - 2^{-r} \leq D^{-s} f(x) \leq 1,$$
$$x \notin L \Rightarrow 0 \leq D^{-s} f(x) \leq 2^{-r},$$

where $s = s(|x|)$. The theorem follows immediately by letting $p(1^n) = D^{s(n)}$ for all $n \in \mathbb{N}$. ∎

COROLLARY 1.3
[[1]] $\mathrm{BQP} \subseteq \mathrm{PP}$.

There is a stronger limitation on AWPP: languages in AWPP are *low* for PP—that is, if $A \in \mathrm{AWPP}$, then $\mathrm{PP}^A = \mathrm{PP}$. If some PP-complete language is low for PP, then $\mathrm{PP}^{\mathrm{PP}} = \mathrm{PP}$, which is considered highly unlikely.

PROPOSITION 1.5
[[31]] If $A \in \mathrm{AWPP}$, *then* $\mathrm{PP}^A = \mathrm{PP}$.

PROOF Let L be any language in PP^A via some function in GapP^A, which itself corresponds by Lemma 1.1 to a polynomial q and P^A predicate R. That is,

$$x \in L \iff \left\| \left\{ y \in \Sigma^{q(|x|)} : R(x,y) \right\} \right\| - \left\| \left\{ y \in \Sigma^{q(|x|)} : \neg R(x,y) \right\} \right\| > 0.$$

We may assume that $R(x,y)$ depends on (i.e., its computation makes queries to) A only on strings of length $s(|x|)$, where $s > 0$ is some polynomial, and makes exactly $s(|x|)$ such queries. (If this is not the case, then we can replace A with a "padded" version of A:

$$A' = \{y10^n : y \in A \wedge n \in \mathbb{N}\}.$$

It is obvious that $A' \in \mathrm{AWPP} \iff A \in \mathrm{AWPP}$ and that $\mathrm{PP}^A = \mathrm{PP}^{A'}$. Moreover, there is a $\mathrm{P}^{A'}$ computation of R that satisfies our assumption above.)

The algorithmic nature of R can be decomposed into two pieces—a polynomial-time computable function $a(x, y, b, i)$ and a P predicate $Q(x, y, b)$. Here, $b = b_1 \cdots b_{s(|x|)}$ is a string of bits representing guesses of the answers to R's queries. Given x, y, b, we simulate the computation of $R(x, y)$. When the computation makes its ith query to the oracle, we answer with b_i. In this simulation it is easy to compute what the ith query will be, and we let $a(x, y, b, i)$ be this query. When the simulation ends, we let $Q(x, y, b)$ be the result. Now if b corresponds to all correct query answers (all according to A), then $Q(x, y, b)$ is the same as $R(x, y)$. In other words,

$$Q(x, y, b_1 b_2 \cdots b_{s(|x|)}) = R(x, y)$$

whenever each bit $b_i = A(a(x, y, b, i))$. Clearly, a is computable in polynomial time and $|a(x, y, b, i)| = s(|x|)$.

Let r be a large enough polynomial that we will choose later. The membership of A in AWPP gives us, by Definition 1.3, a polynomial-time computable $p > 0$ and $f \in \mathrm{GapP}$ such that

$$z \in A \Rightarrow (1 - 2^{-r})m \leq f(z) \leq m, \tag{1.1}$$
$$z \notin A \Rightarrow 0 \leq f(z) \leq 2^{-r}m \tag{1.2}$$

for all $z \in \Sigma^*$, where $r = r(|z|)$ and $m = p(1^{|z|}) > 0$. By doubling both p and f, we can assume that m is always even.

Fix an arbitary $x \in \Sigma^*$ and let $n = |x|$, $q = q(n)$, $s = s(n)$, $r = r(s) = r(s(n))$, and $m = p(1^s) = p(1^{s(n)}) > 0$ and even. Let $in_x = \|\{y \in \Sigma^q : R(x, y)\}\|$ and let $out_x = \|\{y \in \Sigma^q : \neg R(x, y)\}\|$. We want to define an $h \in \mathrm{GapP}$ (with no oracle) such that $h(x) > 0 \iff in_x - out_x > 0$ ($\iff x \in L$). Define

$$g(x, y, b) = \prod_{i=1}^{s} [b_i f(a(x, y, b, i)) + (1 - b_i)(m - f(a(x, y, b, i)))]$$

for all $b = b_1 \cdots b_s \in \Sigma^s$ and $y \in \Sigma^q$. It is evident by Proposition 1.1 that $g \in \mathrm{GapP}$. For all $1 \leq i \leq s$ we have, by (1.1) and (1.2) above,

$$b_i = A(a(x, y, b, i)) \Rightarrow (1 - 2^{-r})m \leq \lambda_i \leq m,$$
$$b_i \neq A(a(x, y, b, i)) \Rightarrow 0 \leq \lambda_i \leq 2^{-r}m,$$

where $\lambda_i = b_i f(a(x,y,b,i)) + (1-b_i)(m - f(a(x,y,b,i)))$. This means that

$$(\forall i)\,[\, b_i = A(a(x,y,i))\,] \Rightarrow (1-2^{-r})^s m^s \leq g(x,y,b) \leq m^s, \quad (1.3)$$

$$(\exists i)\,[\, b_i \neq A(a(x,y,i))\,] \Rightarrow 0 \leq g(x,y,b) \leq 2^{-r} m^s. \quad (1.4)$$

The point here is that if all the guesses b_i to query answers are correct, then $g(x,y,b)$ is large (close to m^s), and if not, then $g(x,y,b)$ is close to zero. Now if we define

$$h(x) = \sum_{y \in \Sigma^q} \sum_{b \in \Sigma^s} g(x,y,b)(2Q(x,y,b) - 1) - \frac{m^s}{2},$$

then $h \in$ GapP, which is clear by Proposition 1.1. Roughly speaking, each term in the sum contributes to $h(x)$ a negligible amount if some of the guesses in b are incorrect. If the guesses in b are all correct, then the term contributes approximately m^s, if $Q(x,y,b)$, or approximately $-m^s$, if $\neg Q(x,y,b)$. But if all the guesses in b are correct, then $Q(x,y,b) = R(x,y)$, and so this makes $h(x)$ close enough to $m^s(in_x - out_x)$ that it witnesses that $L \in$ PP. The $m^s/2$ correction is to ensure that $h(x) \leq 0$ when $x \notin L$.

More precisely, for every x, y let $b_{x,y} \in \Sigma^s$ be the unique string of correct oracle answers during the computation of $R(x,y)$. We have

$$\begin{aligned} h(x) + \frac{m^s}{2} &= \sum_y g(x,y,b_{x,y})(2Q(x,y,b_{x,y}) - 1) \\ &\quad + \sum_y \sum_{b \neq b_{x,y}} g(x,y,b)(2Q(x,y,b) - 1) \\ &= \sum_y g(x,y,b_{x,y})(2R(x,y) - 1) \\ &\quad + \sum_y \sum_{b \neq b_{x,y}} g(x,y,b)(2Q(x,y,b) - 1). \end{aligned}$$

We can bound the second sum on the right-hand side using (1.4) above:

$$\left| \sum_y \sum_{b \neq b_{x,y}} g(x,y,b)(2Q(x,y,b) - 1) \right| < 2^{q+s-r} m^s. \quad (1.5)$$

The first sum splits into two:

$$S = \sum_{y:R(x,y)} g(x,y,b_{x,y}) - \sum_{y:\neg R(x,y)} g(x,y,b_{x,y}).$$

Using (1.3) above, we get

$$(1-2^{-r})^s in_x - out_x \leq S/m^s \leq in_x - (1-2^{-r})^s out_x. \tag{1.6}$$

Combining (1.5) and (1.6) gives us

$$\begin{aligned}(1-2^{-r})^s in_x - out_x - 2^{q+s-r} &< h(x)/m^s + 1/2 \\ &< in_x - (1-2^{-r})^s out_x + 2^{q+s-r}.\end{aligned}$$

If $x \in L$, then $out_x + 1 \leq 2^{q-1} \leq in_x - 1$, so

$$h(x)/m^s > (1-2^{-r})^s(2^{q-1}+1) - 2^{q-1} + 1/2 - 2^{q+s-r}. \tag{1.7}$$

If $x \notin L$, then $in_x \leq 2^{q-1} \leq out_x$, so

$$h(x)/m^s < 2^{q-1}(1-(1-2^{-r})^s) + 2^{q+s-r} - 1/2. \tag{1.8}$$

Now if we choose the polynomial r to be at least $q+s+2$, then much tedious calculation reveals that the right-hand side of (1.7) is positive and the right-hand side of (1.8) is negative. Thus $h(x)$ is as desired. ∎

COROLLARY 1.4
All BQP *languages are low for* PP.

The *counting hierarchy* is the chain of classes

$$\mathrm{PP} \subseteq \mathrm{PP}^{\mathrm{PP}} \subseteq \mathrm{PP}^{\mathrm{PP}^{\mathrm{PP}}} \subseteq \cdots.$$

It is widely believed that this hierarchy *does not collapse*, i.e., all classes in the chain are distinct.

COROLLARY 1.5
If $\mathrm{BQP} = \mathrm{PP}$, *then* $\mathrm{PP}^{\mathrm{PP}} = \mathrm{PP}$ *and so the counting hierarchy collapses to* PP.

1.5.1 Black-box problems

As we mentioned earlier, a significant and growing number of quantum algorithms solve black-box problems. It would be useful for us to understand the inherent limitations of these algorithms. We have some strong,

unconditional results that reveal sharp restrictions on the use of quantum circuits to solve black-box problems, unlike the case with concrete problems. Using certain properties of low-degree multivariate polynomials, Beals, Buhrman, Cleve, Mosca, and de Wolf [7] have proven lower bounds on the depths of quantum circuits that decide various important properties of black-box functions.

The key to the results in [7] lies in making two observations: (1) Theorem 1.1 and Lemma 1.1 both relativize *uniformly* to any oracle, as do most results in complexity theory (see Lemma 1.4 below), and (2) a relativized GapP function varies with the oracle in a simple way, according to a low-degree multivariate polynomial. Oracles and black-box problems are conceptually the same, and understanding the implications of these observations will give us a key result in [7] that ties quantum circuits to multivariate polynomials.

Theorem 1.1 and Lemma 1.1 relativize to give us the following lemma, which is the first observation above, and which is implicit in [19]. For simplicity, we will restrict our attention to quantum circuits with rational matrix entries.

LEMMA 1.4

For every oracle X, there is a function $h^X \in \mathrm{GapP}^X$ that behaves as follows: let C be any quantum circuit over $\mathbb{Q}$ with X-gates and set of matrix coefficients $B \subseteq \mathbb{Q}$, and suppose C has n inputs and m outputs. Then for all $z \in \Sigma^n$ and $w \in \Sigma^m$,

$$\mathrm{Prob}[\ C \text{ outputs } w \text{ given input } z\ ] = D^{-r} h^X(C, z, w),$$

where r is the number of unitary gates in C, and D is the least positive integer such that $DB \subseteq \mathbb{Z}$.

Moreover, h^X is computed uniformly relative to X, i.e., there is a single polynomial p and single polynomial-time oracle TM M that

1. *makes at most $2r$ oracle queries along any path, and*
2. *relative to any oracle X, computes a predicate $R^X \in \mathrm{P}^X$ such that*

$$h^X(x) = \frac{1}{2}\left(\left\|\Sigma^{p(|x|)} \cap R^X(x)\right\| - \left\|\Sigma^{p(|x|)} - R^X(x)\right\|\right),$$

for all $x \in \Sigma^*$, where $R^X(x) = \{y : R^X(x, y)\}$.

(Here we interpret x as encoding the triple (C, z, w).)

PROOF SKETCH The proof is just as with that of Theorem 1.1. The only added feature of C is that it can use X-gates. Each matrix element of an X-gate is either 0 or 1 and is easily computed with two queries to X (recall that X is a superoperator which needs to act on basis vectors of the form $|w\rangle\langle w'|$). It remains to see that, along any single path, M only needs to compute one matrix entry of each X-gate. This follows from a straightforward analysis of how the matrix product is computed in the proof of Theorem 1.1 relativized to X, and we will not go into further detail here. ∎

Given M and p computing h^X as in Lemma 1.4, how does h^X vary with X? To address this question, we associate a real-valued variable v_z for every $z \in \Sigma^*$. We then let any oracle X correspond to a unique setting of each v_z to either 1 (if $z \in X$) or 0 (if $z \notin X$). Now fix an input (x, y) and let the v_z vary over $\{0, 1\}$. The machine M makes oracle queries as it runs on input (x, y), so M's output is some function of all the v_z such that M can possibly query z on input (x, y). This function can be expressed as a multivariate polynomial $q_{(x,y)}$ of low degree over the v_z.

PROPOSITION 1.6

Let M and p compute the function h^X as in Lemma 1.4 above, fix an (x, y) with $|y| = p(|x|)$, and let r be the maximum number of oracle queries made by M on input (x, y) along any computation path. The function $q_{(x,y)}$ described above is a multivariate polynomial over the variables v_z of degree at most r with rational coefficients; hence, the value of $h^X(x)$ is described by a multivariate polynomial s_x over the v_z of degree at most r.

PROOF We use some ideas from the proof of Proposition 1.5. We can assume that M makes exactly r queries on any computation path on input (x, y). Let R^X and h^X be as in Lemma 1.4. Fix some $b = b_1 \cdots b_r \in \Sigma^r$. We simulate M on input (x, y); when M makes its ith query $z_{b,i}$ to the oracle, we answer according to b_i. We say that M *accepts on* b if M accepts in this simulation. Now define the degree r polynomial $m_b = \prod_{i=1}^{r} u_i$, where

$$u_i = \begin{cases} v_{z_{b,i}} & \text{if } b_i = 1, \\ 1 - v_{z_{b,i}} & \text{if } b_i = 0. \end{cases}$$

Let X be any oracle and let the variables v_z be set according to X. If $X(z_{b,i}) = b_i$ for each i, then M takes the same path and makes the same queries relative to oracle X as it did in our simulation above, and $m_b = 1$; otherwise, one of M's queries $z_{b,i}$ is not answered according to X in the simulation, so $m_b = 0$. If we now set

$$q_{(x,y)} = \sum_{b \in \Sigma^r} m_b a_b,$$

(where $a_b = 1$ if M accepts on b, and $a_b = 0$ otherwise), then $q_{(x,y)}$ is a degree r multivariate polynomial over the v_z that takes the value $R^X(x,y)$ when its variables are set according to X. Thus the multivariate polynomial

$$s_x = \frac{1}{2} \sum_{y \in \Sigma^{p(|x|)}} \left(2q_{(x,y)} - 1\right)$$

has degree r and takes the value $h^X(x)$ when its variables are set according to X. ∎

As our final result, we see that Lemma 1.4 and Proposition 1.6 immediately combine to give us a uniform version of the central lemma in Beals et al. [7]. It shows how the behavior of a quantum circuit C with X-gates varies as the oracle X varies.

PROPOSITION 1.7

Let C be a quantum circuit with n inputs, m outputs, and r many X-gates. For any input $z \in \Sigma^n$, and output $w \in \Sigma^m$, there is a degree $2r$ multivariate polynomial s over the variables v_z such that, for all oracles X,

$$\text{Prob}[\ C \text{ outputs } w \text{ given input } z\ ] = s^X,$$

where s^X is the value of s when the variables v_z are set according to X. Moreover, s is the multivariate polynomial s_x defined in the proof of Proposition 1.6 where $x = (C, z, w)$.

1.6 Conclusions

Given the close connection we have seen between quantum complexity and counting complexity, we should continue to expect results in one area to apply to the other, especially the latter to the former, since counting complexity is an older subject. This has clearly been the case so far. For example, the oracle G such that $\mathrm{P}^G = \mathrm{BQP}^G \neq \mathrm{NP}^G$ was discovered in the context of the counting class AWPP in [19] before any of the authors knew about BQP (it was shown there that $\mathrm{P}^G = \mathrm{AWPP}^G \neq \mathrm{NP}^G$). It was only later, when Fortnow and Rogers showed that $\mathrm{BQP} \subseteq \mathrm{AWPP}$ [22], that the significance of the oracle G to quantum complexity was recognized.

Another example regards the class NQP (Definition 1.2). Showing that $\mathrm{NQP} = \mathrm{coC_{=}P}$ [20] immediately imported a wealth of knowledge about $\mathrm{C_{=}P}$ into the quantum realm. For instance, $\mathrm{coC_{=}P}$ contains problems that are at least as hard as any in the *polynomial hierarchy*: $\mathrm{NP} \subseteq \mathrm{NP}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}} \subseteq \cdots$ [44, 45], so the same is true of NQP.

Finally, the connection between black-box quantum circuits and low-degree multivariate polynomials [7] allows important results about the decision tree complexity of Boolean functions [34] (already applied to counting classes in [19]) to be applied directly to quantum complexity.

There are many interesting open questions and potentially fruitful lines of further research connecting quantum computation with complexity theory. Can we put BQP inside a class smaller than AWPP—perhaps a counting class? On the other hand, can we put BQP out of a class bigger than P^{NP} with respect to some oracle? How about showing that $\mathrm{BQP}^A \not\subseteq \mathrm{NP}^{\mathrm{NP}^A}$ for some oracle A? What are the analogous relationships between EQP and other complexity classes? Answers to any of these questions will push the field forward considerably.

References

[1] L.M. Adleman, J. DeMarrais, and M.-D.A. Huang. Quantum computability. *SIAM J. Comp.*, 26(5):1524–1540, 1997.

[2] D. Aharonov, A.Yu. Kitaev, and N. Nisan. Quantum circuits with mixed states. *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 20–30, 1998, quant-ph/9806029.

[3] T. Baker, J. Gill, and R. Solovay. Relativizations of the P = NP question. *SIAM J. Comp.*, 4:431–442, 1975.

[4] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.

[5] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 22 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1990.

[6] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52(5):3457–3467, 1995, quant-ph/9503016.

[7] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 352–361. IEEE, 1998, quant-ph/9802049.

[8] P.A. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *J. Statist. Phys.*, 22:563–591, 1980.

[9] C.H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computation. *SIAM J. Comp.*, 26(5):1510–1523, 1997, quant-ph/9701001.

[10] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comp.*, 26(5):1411–1473, 1997.

[11] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Forsch. Phys.*, 46:493–506, 1998, quant-ph/9605034.

[12] P.O. Boykin, T. Mor, M. Pulver, and V. Roychowdhury. On universal and fault-tolerant quantum computing, 1999, quant-ph/9906054.

[13] G. Brassard and P. Høyer. An exact quantum polynomial-time algorithm for Simon's problem. *Proceedings of the Israeli Sympo-*

sium on the Theory of Computing and Systems, pages 12–23, 1997, quant-ph/9704027.

[14] D. Coppersmith. An approximate fourier transform useful in quantum factoring. Research Report RC 19642. IBM, 1994.

[15] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society London A*, 400:97–117, 1985.

[16] D. Deutsch. Quantum computational networks. *Proceedings of the Royal Society London A*, 425:73–90, 1989.

[17] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society London A*, 439:553–558, 1992.

[18] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *J. Comp. Syst. Sci.*, 48(1):116–148, 1994.

[19] S. Fenner, L. Fortnow, S. Kurtz, and L. Li. An oracle builder's toolkit. *Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 120–131, 1993.

[20] S. Fenner, F. Green, S. Homer, and R. Pruim. Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy. *Proceedings of the Royal Society London A*, 455:3953–3966, 1999, quant-ph/9812056.

[21] R.P. Feynman. Simulating physics with computers. *Intl. J. Theoret. Phys.*, 21(6/7):467–488, 1982.

[22] L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *J. Comp. Syst. Sci.*, 59(2):240–252, 1999, cs.CC/9811023.

[23] L. Fortnow and M. Sipser. Are there interactive protocols for co-NP languages? *Inf. Proc. Lett.*, 28:249–251, 1988.

[24] M. Garey and D. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, 1979.

[25] J. Gill. Computational complexity of probabilistic complexity classes. *SIAM J. Comp.*, 6:675–695, 1977.

[26] F. Green and R. Pruim. Relativized separation of EQP from P(NP). Manuscript.

[27] L.K. Grover. A fast quantum mechanical algorithm for database search. *Phys. Rev. Lett.*, 78:325, 1997, quant-ph/9605043.

[28] L.K. Grover. A framework for fast quantum mechanical algorithms. *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 53–62, 1998, quant-ph/9711043.

[29] J. Gruska. *Quantum Computing.* McGraw-Hill, New York, 1999.

[30] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[31] L. Li. On the counting functions. Technical Report TR-93-12, The University of Chicago, 1993. Ph.D. thesis available at http://www.cs.uchicago.edu/research/publications/techreports/TR-93-12.

[32] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[33] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, New York, 2000.

[34] N. Nisan and M. Szegedy. On the degree of boolean functions as real polynomials. *Proceedings of the 24th ACM Symposium on the Theory of Computing*, pages 462–467, New York, 1992. ACM.

[35] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[36] J. Preskill. Fault-tolerant quantum computation, in *Introduction to Quantum Computation and Information*, Lo, Spiller, and Popescu, Eds. World Scientific, River Edge, NJ, 1997.

[37] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[38] P.W. Shor. Fault-tolerant quantum computation. *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 56–65, 1996.

[39] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26:1484–1509, 1997, quant-ph/9508027.

[40] D.R. Simon. On the power of quantum computation. *SIAM J. Comp.*, 26(5):1474–1483, 1997.

[41] J. Simon. *On Some Central Problems in Computational Complexity.* Ph.D. thesis, Cornell University, Ithaca, NY, January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.

[42] M. Sipser. *Introduction to the Theory of Computation.* PWS, Boston, 1997.

[43] R. Solovay and A. Yao. Manuscript, 1996.

[44] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comp.*, 20(5):865–877, 1991.

[45] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comp.*, 21(2):316–328, 1992.

[46] L. Valiant. The complexity of computing the permanent. *Theoret. Comp. Sci.*, pages 189–201, 1979.

[47] H. Vollmer. *Introduction to Circuit Complexity.* Springer-Verlag, Berlin, 1999.

[48] K. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[49] A. Yao. Quantum circuit complexity. *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1993.