Strengths and weaknesses of quantum computing

Charles H. Bennett
IBM Research *

Ethan Bernstein
Microsoft Corporation †

Gilles Brassard ‡
Université de Montréal §

Umesh Vazirani ¶
UC Berkeley ‖

11 December 1996

Abstract

Recently a great deal of attention has focused on quantum computation following a sequence of results [4, 16, 15] suggesting that quantum computers are more powerful than classical probabilistic computers. Following Shor's result that factoring and the extraction of discrete logarithms are both solvable in quantum polynomial time, it is natural to ask whether all of **NP** can be efficiently solved in quantum polynomial time. In this paper, we address this question by proving that relative to an oracle chosen uniformly at random, with probability 1, the class **NP** cannot be solved on a quantum Turing machine in time $o(2^{n/2})$. We also show that relative to a permutation oracle chosen uniformly at random, with probability 1, the class $\mathbf{NP} \cap \mathbf{co}$–$\mathbf{NP}$ cannot be solved on a quantum Turing machine in time $o(2^{n/3})$. The former bound is tight since

* IBM T. J. Watson Research Laboratory, Yorktown Heights, New York, NY 10598, USA. email: bennetc@watson.ibm.com.

† 1 Microsoft Way, Redmond, WA 98052 – 6399, USA. email: ethanb@microsoft.com.

‡ Supported in part by Canada's NSERC and Québec's FCAR.

§ Département IRO, Université de Montréal, C.P. 6128, succursale centre-ville, Montréal (Québec), Canada H3C 3J7. email: brassard@iro.umontreal.ca.

¶ Supported by NSF Grant No. CCR-9310214.

‖ Computer Science Division, University of California, Berkeley, CA 94720, USA. email: vazirani@cs.berkeley.edu.

recent work of Grover [13] shows how to accept the class **NP** relative to any oracle on a quantum computer in time $O(2^{n/2})$.

1 Introduction

Quantum computational complexity is an exciting new area that touches upon the foundations of both theoretical computer science and quantum physics. In the early eighties, Feynman [12] pointed out that straightforward simulations of quantum mechanics on a classical computer appear to require a simulation overhead that is exponential in the size of the system and the simulated time; he asked whether this is inherent, and whether it is possible to design a universal quantum computer. Deutsch [9] defined a general model of quantum computation — the quantum Turing machine. Bernstein and Vazirani [4] proved that there is an efficient universal quantum Turing machine. Yao [17] extended this by proving that quantum circuits (introduced by Deutsch [10]) are polynomially equivalent to quantum Turing machines.

The computational power of quantum Turing machines (QTMs) has been explored by several researchers. Early work by Deutsch and Jozsa [11] showed how to exploit some inherently quantum mechanical features of QTMs. Their results, in conjunction with subsequent results by Berthiaume and Brassard [5, 6], established the existence of oracles under which there are computational problems that QTMs can solve in polynomial time with certainty, whereas if we require a classical probabilistic Turing machine to produce the correct answer with certainty, then it must take exponential time on some inputs. On the other hand, these computational problems are in **BPP** [1] relative to the same oracle, and therefore efficiently solvable in the classical sense. The quantum analogue of the class **BPP** is

[1] **BPP** is the class of decision problems (languages) that can be solved in polynomial time by probabilistic Turing machines with error probability bounded by 1/3 (for all inputs). Using standard boosting techniques, the error probability can then be made exponentially small in k by iterating the algorithm k times and returning the majority answer.

the class **BQP** [2] [5]. Bernstein and Vazirani [4] proved that $\mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}$, thus establishing that it will not be possible to conclusively prove that $\mathbf{BQP} \neq \mathbf{BPP}$ without resolving the major open problem $\mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$. They also gave the first evidence that $\mathbf{BQP} \neq \mathbf{BPP}$ (polynomial-time quantum Turing machines are more powerful than polynomial-time probabilistic Turing machines), by proving the existence of an oracle relative to which there are problems in **BQP** that cannot be solved with small error probability by probabilistic machines restricted to running in $n^{o(\log n)}$ steps. Since **BPP** is regarded as the class of all "efficiently computable" languages (computational problems), this provided evidence that quantum computers are inherently more powerful than classical computers in a model-independent way. Simon [16] strengthened this evidence by proving the existence of an oracle relative to which **BQP** cannot even be simulated by probabilistic machines allowed to run for $2^{n/2}$ steps. In addition, Simon's paper also introduced an important new technique which was one of the ingredients in a remarkable result proved subsequently by Shor [15]. Shor gave polynomial-time quantum algorithms for the factoring and discrete logarithm problems. These two problems have been well-studied, and their presumed intractability forms the basis of much of modern cryptography. In view of these results, it is natural to ask whether $\mathbf{NP} \subseteq \mathbf{BQP}$; i.e. can quantum computers solve **NP**–complete problems in polynomial time? [3]

In this paper, we address this question by proving that relative to an oracle chosen uniformly at random [3], with probability 1, the class **NP** cannot be solved on a quantum

[2] **BQP** is the class of decision problems (languages) that can be solved in polynomial time by quantum Turing machines with error probability bounded by 1/3 (for all inputs)—see [4] for a formal definition. We prove in Section 4 of this paper that, as is the case with **BPP**, the error probability of **BQP** machines can be made exponentially small.

[3] Actually it is not even clear whether $\mathbf{BQP} \subseteq \mathbf{BPP}^{\mathbf{NP}}$; i.e. it is unclear whether nondeterminism together with randomness is sufficient to simulate quantum Turing machines. In fact, Bernstein and Vazirani's [4] result is stronger than stated above. They actually proved that relative to an oracle, the recursive Fourier sampling problem can be solved in **BQP**, but cannot even be solved by Arthur-Merlin games [1] with a time bound of $n^{o(\log n)}$ (thus giving evidence that nondeterminism on top of probabilism does not help). They conjecture that the recursive Fourier sampling cannot even be solved in the unrelativized polynomial-time hierarchy.

Turing machine in time $o(2^{n/2})$. We also show that relative to a permutation oracle chosen uniformly at random, with probability 1, the class $\mathbf{NP} \cap \mathbf{co{-}NP}$ cannot be solved on a quantum Turing machine in time $o(2^{n/3})$. The former bound is tight since recent work of Grover [13] shows how to accept the class **NP** relative to any oracle on a quantum computer in time $O(2^{n/2})$. See [7] for a detailed analysis of Grover's algorithm.

What is the relevance of these oracle results? We should emphasize that they do not rule out the possibility that $\mathbf{NP} \subseteq \mathbf{BQP}$. What these results do establish is that there is no black-box approach to solving **NP**–complete problems by using some uniquely quantum-mechanical features of QTMs. That this was a real possibility is clear from Grover's [13] result, which gives a black-box approach to solving **NP**–complete problems in square-root as much time as is required classically.

One way to think of an oracle is as a special subroutine call whose invocation only costs unit time. In the context of QTMs, subroutine calls pose a special problem that has no classical counterpart. The problem is that the subroutine must not leave around any bits beyond its computed answer, because otherwise computational paths with different residual information do not interfere. This is easily achieved for deterministic subroutines since any classical deterministic computation can be carried out reversibly so that only the input and the answer remain. However, this leaves open the more general question of whether a **BQP** machine can be used as a subroutine. Our final result in this paper is to show how any **BQP** machine can be modified into a *tidy* **BQP** machine whose final superposition consists almost entirely of a tape configuration containing just the input and the single bit answer. Since these tidy **BQP** machines can be safely used as subroutines, this allows us to show that $\mathbf{BQP}^{\mathbf{BQP}} = \mathbf{BQP}$. The result also justifies the definition of oracle quantum machines that we now give.

2 Oracle Quantum Turing Machines

In this section and the next, we shall assume without loss of generality that the Turing machine alphabet (for each track or tape) is $\{0, 1, \#\}$, where "#" denotes the blank symbol. Initially all tapes are blank except that the input tape contains the actual input surrounded by blanks. We shall use Σ to denote $\{0, 1\}$.

In the classical setting, an oracle may be described informally as a device for evaluating some Boolean function $A : \Sigma^* \to \Sigma$, on arbitrary arguments, at unit cost per evaluation. This allows to formulate questions such as "if A were efficiently computable by a Turing machine, which other functions (or languages) could be efficiently computed by Turing machines?". In the quantum setting, an equivalent question can be asked, provided we define oracle quantum Turing machines appropriately—which we do in this section—and provided bounded-error quantum Turing machines can be composed—which we show in Section 4 of this paper.

An oracle QTM has a special *query tape* (or track), all of whose cells are blank except for a single block of non-blank cells. In a well-formed oracle QTM, the Turing machine rules may allow this region to grow and shrink, but prevent it from fragmenting into non-contiguous blocks.[4] Oracle QTMs have two distinguished internal states: a pre-query state q_q and a post-query state q_a. A query is executed whenever the machine enters the pre-query state. If the query string is empty, a no-op occurs, and the machine passes directly to the post-query state with no change. If the query string is nonempty, it can be written in the form $x \circ b$ where $x \in \Sigma^*$, $b \in \Sigma$, and "$\circ$" denotes concatenation. In that case, the result of a call on oracle A is that internal control passes to the post-query state while the contents of

[4] This restriction can be made without loss of generality and it can be verified syntactically by allowing only machines that make sure they do not break the rule before writing on the query tape.

the query tape changes from $|x \circ b\rangle$ to $|x \circ (b \oplus A(x))\rangle$, where "$\oplus$" denotes the exclusive-or (addition modulo 2). Except for the query tape and internal control, other parts of the oracle QTM do not change during the query. If the target bit $|b\rangle$ is supplied in initial state $|0\rangle$, then its final state will be $|A(x)\rangle$, just as in a classical oracle machine. Conversely, if the target bit is already in state $|A(x)\rangle$, calling the oracle will reset it to $|0\rangle$. This ability to "uncompute" will often prove essential to allow proper interference among computation paths to take place. Using this fact, it is also easy to see that the above definition of oracle Turing machines yields unitary evolutions if we restrict ourselves to machines that are well-formed in other respects, in particular evolving unitarily as they enter the pre-query state and leave the post-query state.

The power of quantum computers comes from their ability to follow a coherent superposition of computation paths. Similarly oracle quantum machines derive great power from the ability to perform superpositions of queries. For example, oracle A might be called when the query tape is in state $|\psi \circ 0\rangle = \sum_x \alpha_x |x \circ 0\rangle$, where α_x are complex coefficients, corresponding to an arbitrary superposition of queries with a constant $|0\rangle$ in the target bit. In this case, after the query, the query string will be left in the entangled state $\sum_x \alpha_x |x \circ A(x)\rangle$. It is also useful to be able to put the target bit b into a superposition. For example, the conditional phase inversion used in Grover's algorithm can be achieved by performing queries with the target bit b in the nonclassical superposition $\beta = (|0\rangle - |1\rangle)/\sqrt{2}$. It can readily be verified that an oracle call with the query tape in state $x \circ \beta$ leaves the entire machine state, including the query tape, unchanged if $A(x) = 0$, and leaves the entire state unchanged while introducing a phase factor -1 if $A(x) = 1$.

It is often convenient to think of a Boolean oracle as defining a length-preserving function on Σ^*. This is easily accomplished by interpreting the oracle answer on the pair (x, i) as the i^{th} bit of the function value. The pair (x, i) is encoded as a binary string using any standard pairing function. A *permutation oracle* is an oracle which, when interpreted as a length-preserving function, acts for each $n \geq 0$ as a permutation on Σ^n. Henceforth, when no confusion may arise, we shall use $A(x)$ to denote the length-preserving function associated with oracle A rather than the Boolean function that gives rise to it.

Let us define $\mathbf{BQTime}(T(n))^A$ as the sets of languages accepted with probability at least 2/3 by some oracle QTM M^A whose running time is bounded by $T(n)$. This bound on the running time applies to each individual input, not just on the average. Notice that whether or not M^A is a **BQP**-machine might depend upon the oracle A—thus M^A might be a **BQP**-machine while M^B might not be one.

Note: The above definition of a quantum oracle for an arbitrary Boolean function will suffice for the purposes of the present paper, but the ability of quantum computers to perform general unitary transformations suggests a broader definition, which may be useful in other contexts. For example, oracles that perform more general, non-Boolean unitary operations have been considered in computational learning theory [8] and for hiding information against classical queries [14].

Most broadly, a quantum oracle may be defined as a device that, when called, applies a fixed unitary transformation U to the current contents $|z\rangle$ of the query tape, replacing it by $U|z\rangle$. Such an oracle U must be defined on a countably infinite-dimensional Hilbert space, such as that spanned by the binary basis vectors $|\epsilon\rangle, |0\rangle, |1\rangle, |00\rangle, |01\rangle, |10\rangle, |11\rangle, |000\rangle, \ldots,$ where ϵ denotes the empty string. Clearly, the use of such general unitary oracles still yields unitary evolution for well-formed oracle Turing machines. Naturally, these oracles can map inputs onto superpositions of outputs, and vice versa, and they need not even be length-preserving. However, in order to obey the dictum that a single machine cycle ought not to make infinite changes in the tape, one might require that $U|z\rangle$ have amplitude zero on all but finitely many basis vectors. (One could even insist on a uniform and effective version of the above restriction.) Another natural restriction one may wish to impose upon

U is that it be an involution, $U^2 = I$, so that the effect of an oracle call can be undone by a further call on the same oracle. Again this may be crucial to allow proper interference to take place. Note that the special case of unitary transformation considered in this paper, which corresponds to evaluating a classical Boolean function, is an involution.

3 On the Difficulty of Simulating Nondeterminism on QTMs

The computational power of QTMs lies in their ability to maintain and compute with exponentially large superpositions. It is tempting to try to use this "exponential parallelism" to simulate non-determinism. However, there are inherent constraints on the scope of this parallelism, which are imposed by the formalism of quantum mechanics.[5] In this section, we explore some of these constraints.

To see why quantum interference can speed up **NP** problems quadratically but not exponentially, consider the problem of distinguishing the empty oracle ($\forall_x A(x) = 0$) from an oracle containing a single random unknown string y of known length n (i.e. $A(y)=1$, but $\forall_{x \neq y} A(x)=0$). We require that the computer never answer yes on an empty oracle, and seek to maximize its "success probability" of answering yes on a nonempty oracle. A classical computer can do no better than to query distinct n–bit strings at random, giving a success probability $1/2^n$ after one query and $k/2^n$ after k queries. How can a quantum computer do

[5] There is a superficial similarity between this exponential parallelism in quantum computation and the fact that probabilistic computations yield probability distributions over exponentially large domains. The difference is that in the probabilistic case, the computational path is chosen by making a sequence of random choices—one for each step. In the quantum-mechanical case, it is possible for several computational paths to interfere destructively, and therefore it is necessary to keep track of the entire superposition at each step to accurately simulate the system.

better, while respecting the rule that its overall evolution be unitary, and, in a computation with a nonempty oracle, all computation paths querying empty locations evolve exactly as they would for an empty oracle? A direct quantum analog of the classical algorithm would start in an equally-weighted superposition of 2^n computation paths, query a different string on each path, and finally collapse the superposition by asking whether the query had found the nonempty location. This yields a success probability $1/2^n$, the same as the classical computer. However, this is not the best way to exploit quantum parallelism. Our goal should be to maximize the separation between the state vector $|\psi_k\rangle$ after k interactions with an empty oracle, and the state vector $|\psi_k(y)\rangle$ after k interactions with an oracle nonempty at an unknown location y. Starting with a uniform superposition

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle,$$

it is easily seen that the separation after one query is maximized by a unitary evolution to

$$|\psi_1(y)\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{\delta_{x,y}} |x\rangle = |\psi_0\rangle - \frac{2}{\sqrt{2^n}} |y\rangle.$$

This is a phase inversion of the term corresponding to the nonempty location. By testing whether the post-query state agrees with $|\psi_0\rangle$ we obtain a success probability

$$1 - |\langle\psi_0|\psi_1(y)\rangle|^2 \approx 4/2^n$$

approximately four times better than the classical value. Thus, if we are allowed only one query, quantum parallelism gives a modest improvement, but is still overwhelmingly likely to fail because the state vector after interaction with a nonempty oracle is almost the same as after interaction with an empty oracle. The only way of producing a large difference after one query would be to concentrate much of the initial superposition in the y term before the query, which cannot be done because that location is unknown.

Having achieved the maximum separation after one query, how best can that separation be increased by subsequent queries? Various strategies can be imagined, but a good one (called "inversion about the average" by Grover [13]) is to perform an oracle-independent unitary transformation so as to change the phase difference into an amplitude difference, leaving the y term with the same sign as all the other terms but a magnitude approximately threefold larger. Subsequent phase-inverting interactions with the oracle, alternating with oracle-independent phase-to-amplitude conversions, cause the distance between $|\psi_0\rangle$ and $|\psi_k(y)\rangle$ to grow linearly with k, approximately as $2k/\sqrt{2^n}$ when $k \leq \sqrt{N}/2$. This results in a quadratic growth of the success probability, approximately as $4k^2/2^n$ for small k. The proof of Theorem 3.5 shows that this approach is essentially optimal: no quantum algorithm can gain more than this quadratic factor in success probability compared to classical algorithms, when attempting to answer **NP**-type questions formulated relative to a random oracle.

3.1 Lower Bounds on Quantum Search

We will sometimes find it convenient to measure the accuracy of a simulation by calculating the Euclidean distance[6] between the target and simulation superpositions. The following theorem from [4] shows that the simulation accuracy is at most 4 times worse than this Euclidean distance.

Theorem 3.1 *If two unit-length superpositions are within Euclidean distance ε then observing the two superpositions gives samples from distributions which are within total variation distance*[7] *at most* 4ε.

[6] The Euclidean distance between $|\phi\rangle = \sum_x \alpha_x |x\rangle$ and $|\psi\rangle = \sum_x \beta |x\rangle$ is defined as $(\sum_x |\alpha_x - \beta_x|^2)^{1/2}$.
[7] The total variation distance between two distributions $\mathcal{D}$ and $\mathcal{D}'$ is $\sum_x |\mathcal{D}(x) - \mathcal{D}'(x)|$.

Definition 3.2 *Let $|\phi_i\rangle$ be the superposition of M^A on input x at time i. We denote by $q_y(|\phi_i\rangle)$ the sum of squared magnitudes in $|\phi_i\rangle$ of configurations of M which are querying the oracle on string y. We refer to $q_y(|\phi_i\rangle)$ as the* query magnitude *of y in $|\phi_i\rangle$.*

Theorem 3.3 *Let $|\phi_i\rangle$ be the superposition of M^A on input x at time i. Let $\varepsilon > 0$. Let $F \subseteq [0, T-1] \times \Sigma^*$ be a set of time-strings pairs such that $\sum_{(i,y)\in F} q_y(|\phi_i\rangle) \leq \frac{\varepsilon^2}{T}$. Now suppose the answer to each query $(i, y) \in F$ is modified to some arbitrary fixed $a_{i,y}$ (these answers need not be consistent with an oracle). Let $|\phi'_i\rangle$ be the time i superposition of M on input x with oracle A modified as stated above. Then $||\phi_T\rangle - |\phi'_T\rangle| \leq \varepsilon$.*

Proof. Let U be the unitary time evolution operator of M^A. Let A_i denote an oracle such that if $(i, y) \in F$ then $A_i(y) = a_{i,y}$ and if $(i, y) \notin F$ then $A_i(y) = A(y)$. Let U_i be the unitary time evolution operator of M^{A_i}. Let $|\phi_i\rangle$ be the superposition of M^A on input x at time i. We define $|E_i\rangle$ to be the error in the i^{th} step caused by replacing the oracle A with A_i. Then

$$|E_i\rangle = U_i|\phi_i\rangle - U|\phi_i\rangle.$$

So we have

$$|\phi_T\rangle = U|\phi_{T-1}\rangle = U_T|\phi_{T-1}\rangle - |E_{T-1}\rangle = \cdots = U_T \cdots U_1|\phi_0\rangle - \sum_{i=0}^{T-1} U_{T-1} \cdots U_i|E_i\rangle.$$

Since all of the U_i are unitary, $|U_{T-1} \cdots U_i|E_i\rangle| = ||E_i\rangle|$.

The sum of squared magnitudes of all of the E_i is equal to $\sum_{(i,y)\in F} q_y(|\phi_i\rangle)$ and therefore at most $\frac{\varepsilon^2}{T^2}$. In the worst case, the $U_{T-1} \cdots U_i|E_i\rangle$s could interfere constructively; however, the squared magnitude of their sum is at most T times the sum of their squared magnitudes, i.e. ε^2. Therefore $||\phi_T\rangle - |\phi'_T\rangle| \leq \varepsilon$. □

Corollary 3.4 *Let A be an oracle over alphabet Σ. For $y \in \Sigma^*$, let A_y be any oracle such that $\forall x \neq y$ $A_y(x) = A(x)$. Let $|\phi_i\rangle$ be the time i superposition of M^A on input x and $|\phi_i\rangle^{(y)}$ be the time i superposition of M^{A_y} on input x. Then for every $\varepsilon > 0$, there is a set S of cardinality at most $\frac{2T^2}{\varepsilon^2}$ such that $\forall y \notin S$ $\left| |\phi_T\rangle - |\phi_T\rangle^{(y)} \right| \leq \varepsilon$.*

Proof. Since each $|\phi_t\rangle$ has unit length, $\sum_{i=0}^{T-1} \sum_y q_y(|\phi_i\rangle) \leq T$. Let S be the set of strings y such that $\sum_{i=0}^{T-1} q_y(|\phi_i\rangle) \geq \frac{\varepsilon^2}{2T}$. Clearly $\mathrm{card}(S) \leq \frac{2T^2}{\varepsilon^2}$.

If $y \notin S$ then $\sum_{i=0}^{T-1} q_y(|\phi_i\rangle) < \frac{\varepsilon^2}{2T}$. Therefore by Theorem 3.3 $\forall y \notin S$ $\left| |\phi_i\rangle - |\phi_i\rangle^{(y)} \right| \leq \varepsilon$. □

Theorem 3.5 *For any $T(n)$ which is $o(2^{n/2})$, relative to a random oracle, with probability 1,* **BQTime**$(T(n))$ *does not contain* **NP**.

Proof. Recall from Section 2 that an oracle can be thought of as a length-preserving function: this is what we mean below by $A(x)$. Let $\mathcal{L}_A = \{y : \exists x \; A(x) = y\}$. Clearly, this language is contained in $\mathbf{NP}^A$. Let $T(n) = o(2^{n/2})$. We show that for any bounded-error oracle QTM M^A running in time at most $T(n)$, with probability 1, M^A does not accept the language $\mathcal{L}_A$. The probability is taken over the choice of a random length-preserving oracle A. Then, since there are a countable number of QTMs and the intersection of a countable number of probability 1 events still has probability 1, we conclude that with probability 1, no bounded error oracle QTM accepts $\mathcal{L}_A$ in time bounded by $T(n)$.

Since $T(n) = o(2^{n/2})$, we can pick n large enough so that $T(n) \leq \frac{2^{n/2}}{20}$. We will show that the probability that M gives the wrong answer on input 1^n is at least $1/8$ for every way of fixing the oracle answers on inputs of length not equal to n. The probability is taken over the random choices of the oracle for inputs of length n.

Let us fix an arbitrary length-preserving function from strings of lengths other than n over alphabet Σ. Let $\mathcal{C}$ denote the set of oracles consistent with this arbitrary function. Let $\mathcal{A}$ be the set of oracles in $\mathcal{C}$ such that 1^n has no inverse (does not belong to $\mathcal{L}_A$). If the oracle answers to length n strings are chosen uniformly at random, then the probability that the oracle is in $\mathcal{A}$ is at least $1/4$. This is because the probability that 1^n has no inverse is $(\frac{2^n-1}{2^n})^{2^n}$ which is at least $1/4$ (for n sufficiently large). Let $\mathcal{B}$ be the set of oracles in $\mathcal{C}$ such that 1^n has a unique inverse. As above, the probability that a randomly chosen oracle is in $\mathcal{B}$ is $(\frac{2^n-1}{2^n})^{2^n-1}$ which is at least $1/e$.

Given an oracle A in $\mathcal{A}$, we can modify its answer on any single input, say y, to 1^n and therefore get an oracle A_y in $\mathcal{B}$. We will show that for most choices of y, the acceptance probability of M^A on input 1^n is almost equal to the acceptance probability of M^{A_y} on input 1^n. On the other hand, M^A must reject 1^n and M^{A_y} must accept 1^n. Therefore M cannot accept both $\mathcal{L}_A$ and $\mathcal{L}_{A_y}$. By working through the details more carefully, it is easy to show that M fails on input 1^n with probability at least $1/8$ when the oracle is a uniformly random function on strings of length n, and is an arbitrary function on all other strings.

Let A_y be the oracle such that $A_y(y) = 1^n$ and $\forall z \neq y \; A_y(z) = A(z)$. By Corollary 3.4 there is a set S of at most $338T^2(n)$ strings such that the difference between the i^{th} superposition of M^{A_y} on input 1^n and M^A on input 1^n has norm at most $1/13$. Using Theorem 3.1 we can conclude that the difference between the acceptance probabilities of M^{A_y} on input 1^n and M^A on input 1^n is at most $1/13 \times 4 < 1/3$. Since M^{A_y} should accept 1^n with probability at least $2/3$ and M^A should reject 1^n with probability at least $2/3$, we can conclude that M fails to accept either $\mathcal{L}_A$ or $\mathcal{L}_{A_y}$.

So, each oracle $A \in \mathcal{A}$ for which M correctly decides whether $1^n \in \mathcal{L}_A$ can, by changing a single answer of A to 1^n, be mapped to at least $(2^n - \text{card}(S)) \geq 2^{n-1}$ different oracles $A_f \in \mathcal{B}$ for which M fails to correctly decide whether $1^n \in \mathcal{L}_{A_f}$. Moreover, any particular $A_f \in \mathcal{B}$ is the image under this mapping of at most $2^n - 1$ oracles $A \in \mathcal{A}$, since where it now answers 1^n, it must have given one of the $2^n - 1$ other possible answers. Therefore, the number of oracles in $\mathcal{B}$ for which M fails must be at least $1/2$ the number of oracles in $\mathcal{A}$ for which M succeeds. So, calling a the number of oracles in $\mathcal{A}$ for which M fails, M must fail for at least $a + 1/2(\text{card}(\mathcal{A}) - a)$ oracles. Therefore M fails to correctly decide whether $1^n \in \mathcal{L}_A$ with probability at least $(1/2)P[\mathcal{A}] \geq 1/8$.

It is easy to conclude that M decides membership in $\mathcal{L}_A$ with probability 0 for a uniformly chosen oracle A. $\square$

Note: Theorem 3.3 and its Corollary 3.4 isolate the constraints on "quantum parallelism" imposed by unitary evolution. The rest of the proof of the above theorem is similar in spirit to standard techniques used to separate **BPP** from **NP** relative to a random oracle [3]. For example, these techniques can be used to show that, relative to a random oracle A, no classical probabilistic machine can recognize $\mathcal{L}_A$ in time $o(2^n)$. However, quantum machines can recognize this language quadratically faster, in time $O(\sqrt{2^n})$, using Grover's algorithm [13]. This explains why a substantial modification of the standard technique was required to prove the above theorem.

The next result about **NP**$\cap$**co–NP** relative to a random permutation oracle requires a more subtle argument; ideally we would like to apply Theorem 3.3 after asserting that the total query magnitude with which $A^{-1}(1^n)$ is probed is small. However, this is precisely what we are trying to prove in the first place.

Theorem 3.6 *For any $T(n)$ which is $o(2^{n/3})$, relative to a random permutation oracle, with probability* 1, $\mathbf{BQTime}(T(n))$ *does not contain* $\mathbf{NP} \cap \mathbf{co{-}NP}$.

Proof. For any permutation oracle A, let $\mathcal{L}_A = \{y :$ first bit of $A^{-1}(y)$ is $1\}$. Clearly, this language is contained in $(\mathbf{NP} \cap \mathbf{co{-}NP})^A$. Let $T(n) = o(2^{n/3})$. We show that for any bounded-error oracle QTM M^A running in time at most $T(n)$, with probability 1, M^A does not accept the language $\mathcal{L}_A$. The probability is taken over the choice of a random permutation oracle A. Then, since there are a countable number of QTMs and the intersection of a countable number of probability 1 events still has probability 1, we conclude that with probability 1, no bounded error oracle QTM accepts $\mathcal{L}_A$ in time bounded by $T(n)$.

Since $T(n) = o(2^{n/3})$, we can pick n large enough so that $T(n) \leq \frac{2^{n/3}}{100}$. We will show that the probability that M gives the wrong answer on input 1^n is at least $1/8$ for every way of fixing the oracle answers on inputs of length not equal to n. The probability is taken over the random choices of the permutation oracle for inputs of length n.

Consider the following method of defining random permutations on $\{0,1\}^n$: let $x_0, x_1, \ldots x_{T+1}$ be a sequence of strings chosen uniformly at random in $\{0,1\}^n$. Pick π_0 uniformly at random among permutations such that $\pi(x_0) = 1^n$. Let $\pi_i = \pi_{i-1} \cdot \tau$, where τ is the transposition (x_{i-1}, x_i), i.e. $\pi_i(x_i) = \pi_{i-1}(x_{i-1})$ and $\pi_i(x_{i-1}) = \pi_{i-1}(x_i)$. Clearly each π_i is a random permutation on $\{0,1\}^n$.

Consider a sequence of permutation oracles A_i, such that $A_i(y) = A_j(y)$ if $y \notin \{0,1\}^n$ and $A_i(y) = \pi_i(y)$ if $y \in \{0,1\}^n$. Denote by $|\phi_i\rangle$ the time i superposition of $M^{A_{T(n)}}$ on input 1^n, and by $|\phi'_i\rangle$ the time i superposition of $M^{A_{T(n)-1}}$ on input 1^n. By construction, with probability exactly $1/2$, the string 1^n is a member of exactly one of the two languages $L_{A_{T(n)}}$ and $L_{A_{T(n)-1}}$. We will show that $E\left[\left||\phi_{T(n)}\rangle - |\phi'_{T(n)}\rangle\right|\right] \leq 1/50$. Here the expectation is taken over the random choice of the oracles. By Markov's bound, $P[\left| |\phi_{T(n)}\rangle - |\phi'_{T(n)}\rangle \right| \leq 2/25] \geq 3/4$. Applying Theorem 3.1 we conclude that if $\left| |\phi_{T(n)}\rangle - |\phi'_{T(n)}\rangle \right| \leq 2/25$, then the acceptance probability of $M^{A_{T(n)}}$ and $M^{A_{T(n)-1}}$ differ by at most $8/25 < 1/3$, and hence either both machines accept input 1^n or both reject that input. Therefore $M^{A_{T(n)}}$ and $M^{A_{T(n)-1}}$ give the same answers on input 1^n with probability at least $3/4$. By construction, the probability that the string 1^n belongs to exactly one of the two languages $L_{A_{T(n)}}$ and $L_{A_{T(n)-1}}$ is equal to $P[\text{first bit of } x_{T(n)-1} \neq \text{first bit of } x_{T(n)}] = 1/2$. Therefore, we can conclude that with probability at least $1/4$, either $M^{A_{T(n)}}$ or $M^{A_{T(n)-1}}$ gives the wrong answer on input 1^n. Since each of $A_{T(n)}$ and $A_{T(n)-1}$ are chosen from the same distribution, we can conclude that $M^{A_{T(n)}}$ gives the wrong answer on input 1^n with probability at least $1/8$.

To bound $E[\left| |\phi_{T(n)}\rangle - |\phi'_{T(n)}\rangle \right|]$, we show that $|\phi_{T(n)}\rangle$ and $|\phi'_{T(n)}\rangle$ are each close to a certain superposition $|\psi_{T(n)}\rangle$. To define this superposition, run M on input 1^n with a different oracle on each step: on step i, use A_i to answer the oracle queries. Denote by $|\psi_i\rangle$, the time i superposition that results. Consider the set of time-string pairs $S = \{(i, x_j) : \ j \geq i, \ 0 \leq i \leq T\}$. It is easily checked that the oracle queries in the computation described above and those of $M^{A_{T(n)}}$ and $M^{A_{T(n)+1}}$ differ only on the set S. We claim that the expected query magnitude of any pair in the set is at most $1/2^n$, since for $j \geq i$, we may think of x_j as having been randomly chosen during step j, *after* the superposition of oracle queries to be performed has already been written on the oracle tape. Let α be the sum of the query magnitudes for time-string pairs in S. Then

$$E[\alpha] \leq \text{card}(S)/2^n = \binom{T(n)+1}{2}/2^n \leq \frac{T(n)^2}{2^n}$$

for $T(n) \geq 4$. Let ε be a random variable such that $\alpha = \varepsilon^2/2T(n)$. Then by Theorem 3.3, $\left| |\phi\rangle - |\phi_{T(n)}\rangle \right| \leq \varepsilon$ and $\left| |\phi\rangle - |\phi'_{T(n)}\rangle \right| \leq \varepsilon$. We showed above that

$$E[\varepsilon^2/T(n)] = E[\alpha] \leq \frac{T(n)^2}{2^n}.$$

But $E[\varepsilon/\sqrt{2T(n)}]^2 \leq E[\varepsilon^2/2T(n)]$. Therefore

$$E[\varepsilon] = \sqrt{2T(n)}E[\varepsilon/\sqrt{2T(n)}] \leq \sqrt{2T(n)E[\varepsilon^2/2T(n)]} \leq \sqrt{2T(n)\frac{T(n)^2}{2^n}} \leq \sqrt{\frac{2}{100^3}} < 1/100.$$

Therefore $E[\left| |\phi\rangle - |\phi_{T(n)}\rangle \right|] \leq E[\varepsilon] < 1/100$ and $E[\left| |\phi\rangle - |\phi'_{T(n)}\rangle \right|] \leq E[\varepsilon] < 1/100$. It follows that $E[\left| |\phi_{T(n)}\rangle - |\phi'_{T(n)}\rangle \right|] < 1/50$.

Finally, it is easy to conclude that M decides membership in $\mathcal{L}_A$ with probability 0 for a uniformly random permutation oracle A. □

Note: In view of Grover's algorithm [13], we know that the constant "1/2" in the statement of Theorem 3.5 cannot be improved. On the other hand, there is no evidence that the constant "1/3" in the statement of Theorem 3.6 is fundamental. It may well be that Theorem 3.6 would still hold (albeit not its current proof) with 1/2 substituted for 1/3.

Corollary 3.7 *Relative to a random permutation oracle, with probability* 1, *there exists a* quantum one-way permutation. *Given the oracle, this permutation can be computed efficiently even with a classical deterministic machine, yet it requires exponential time to invert even on a quantum machine.*

Proof. Given an arbitrary permutation oracle A for which A^{-1} can be computed in time $o(2^{n/3})$ on a quantum Turing machine, it is just as easy to decide $\mathcal{L}_A$ as defined in the proof of Theorem 3.6. It follows from that proof that this happens with probability 0 when A is a uniformly random permutation oracle. $\square$

4 Using a Bounded-Error QTM as a Subroutine

The notion of a subroutine call or an oracle invocation provides a simple and useful abstraction in the context of classical computation. Before making this abstraction in the context of quantum computation, there are some subtle considerations that must be thought through. For example, if the subroutine computes the function f, we would like to think of an invocation of the subroutine on the string x as magically writing $f(x)$ in some designated spot (actually xoring it to ensure unitarity). In the context of quantum algorithms, this abstraction is only valid if the subroutine cleans up all traces of its intermediate calculations, and leaves just the final answer on the tape. This is because if the subroutine is invoked on a superposition of x's, then different values of x would result in different scratch-work on the tape, and would prevent these different computational paths from interfering. Since erasing is not a unitary operation, the scratch-work cannot, in general, be erased post-facto. In the special case where f can be efficiently computed deterministically, it is easy to design the subroutine so that it reversibly erases the scratch-work—simply compute $f(x)$, copy $f(x)$ into safe storage, and then uncompute $f(x)$ to get rid of the scratch work [2]. However, in the case that f is computed by a **BQP** machine, the situation is more complicated. This is because only some of the computational paths of the machine lead to the correct answer $f(x)$, and therefore if we copy $f(x)$ into safe storage and then uncompute $f(x)$, computational paths with different values of $f(x)$ will no longer interfere with each other, and we will not reverse the first phase of the computation. We show, nonetheless, that if we boost the success probability of the **BQP** machine before copying $f(x)$ into safe storage and uncomputing $f(x)$, then most of the weight of the final superposition has a clean tape with only the input x and the answer $f(x)$. Since such tidy **BQP** machines can be safely used as subroutines, this allows us to show that $\mathbf{BQP}^{\mathbf{BQP}} = \mathbf{BQP}$. The result also justifies our definition of oracle quantum machines.

The correctness of the boosting procedure is proved in Theorems 4.13 and 4.14. The proof follows the same outline as in the classical case, except that we have to be much more careful in simple programming constructs such as looping, etc. We therefore borrow the machinery developed in [4] for this purpose, and present the statements of the relevant lemmas and theorems in the first part of this section. The main new contribution in this section is in the proofs of Theorems 4.13 and 4.14. The reader may therefore wish to skip directly ahead to these proofs.

4.1 Some Programming Primitives for QTMs

In this subsection, we present several definitions, lemmas and theorems from [4].

Recall that a QTM M is defined by a triplet (Σ, Q, δ) where: Σ is a finite alphabet with an identified blank symbol #, Q is a finite set of states with an identified initial state q_0 and final state $q_f \neq q_0$, and δ, the *quantum transition function*, is a function

$$\delta \ : \ Q \times \Sigma \rightarrow \tilde{\mathbf{C}}^{\Sigma \times Q \times \{L,R\}}$$

where $\tilde{\mathbf{C}}$ is the set of complex numbers whose real and imaginary parts can be approximated to within 2^{-n} in time polynomial in n.

Definition 4.1 *A* final configuration *of a QTM is any configuration in state q_f. If when QTM M is run with input x, at time T the superposition contains only final configurations and at any time less than T the superposition contains no final configuration, then M* halts *with* running time T *on input x. The superposition of M at time T is called the* final superposition *of M run on input x. A polynomial-time QTM is a well-formed QTM which on every input x halts in time polynomial in the length of x.*

Definition 4.2 *A QTM M is called* well-behaved *if it halts on all input strings in a final superposition where each configuration has the tape head in the same cell. If this cell is always the start cell, we call the QTM* stationary.

We will say that a QTM M is in *normal form* if all transitions from the distinguished state q_f lead to the distinguished state q_0, the symbol in the scanned cell is left unchanged, and the head moves right, say. Formally:

Definition 4.3 *A QTM $M = (\Sigma, Q, \delta)$ is in* normal form *if*

$$\forall \sigma \in \Sigma \quad \delta(q_f, \sigma) = |\sigma\rangle|q_0\rangle|R\rangle$$

Theorem 4.4 *If f is a function mapping strings to strings which can be computed in deterministic polynomial time and such that the length of $f(x)$ depends only on the length of x, then there is a polynomial-time, stationary, normal form QTM which given input x, produces output $x; f(x)$, and whose running time depends only on the length of x.*

If f is a one-to-one function from strings to strings that such that both f and f^{-1} can be computed in deterministic polynomial time, and such that the length of $f(x)$ depends only on *the length of x, then there is a polynomial-time, stationary, normal form QTM which given input x, produces output $f(x)$, and whose running time depends only on the length of x.*

Definition 4.5 *A* multi-track *Turing machine with k tracks is a Turing machine whose alphabet Σ is of the form $\Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_k$ with a special blank symbol $\#$ in each Σ_i so that the blank in Σ is $(\#, \ldots, \#)$. We specify the input by specifying the string on each "track" (separated by ';'), and optionally by specifying the alignment of the contents of the tracks.*

Lemma 4.6 *Given any QTM $M = (\Sigma, Q, \delta)$ and any set Σ', there is a QTM $M' = (\Sigma \times \Sigma', Q, \delta')$ such that M' behaves exactly as M while leaving its second track unchanged.*

Lemma 4.7 *Given any QTM $M = (\Sigma_1 \times \cdots \times \Sigma_k, Q, \delta)$ and permutation $\pi : [1, k] \to [1, k]$, there is a QTM $M' = (\Sigma_{\pi(1)} \times \cdots \times \Sigma_{\pi(k)}, Q, \delta')$ such that the M' behaves exactly as M except that its tracks are permuted according to π.*

Lemma 4.8 *If M_1 and M_2 are well-behaved, normal form QTMs with the same alphabet, then there is a normal form QTM M which carries out the computation of M_1 followed by the computation of M_2.*

Lemma 4.9 *Suppose that M is a well-behaved, normal form QTM. Then there is a normal form QTM M' such that on input $x; k$ with $k > 0$, the machine M' runs M for k iterations on its first track.*

Definition 4.10 *If QTMs M_1 and M_2 have the same alphabet, then we say that M_2* reverses *the computation of M_1 if the following holds: for any input x on which M_1 halts, let c_x and ϕ_x be the initial configuration and final superposition of M_1 on input x. Then M_2 on input the superposition ϕ_x, halts with final superposition consisting entirely of configuration c_x. Note that for M_2 to reverse M_1, the final state of M_2 must be equal to the initial state of M_1 and vice versa.*

Lemma 4.11 *If M is a normal form QTM which halts on all inputs, then there is a normal form QTM M' that reverses the computation of M with slowdown by a factor of 5.*

Finally, recall the definition of the class **BQP**.

Definition 4.12 *Let M be a stationary, normal form, multi-track QTM M whose last track has alphabet $\{\#, 0, 1\}$. We say that M accepts x if it halts with a 1 in the last track of the start cell. Otherwise we say that M rejects x.*

A QTM accepts *the language $\mathcal{L} \subseteq (\Sigma - \#)^*$* with probability p *if M accepts with probability at least p every string $x \in \mathcal{L}$ and rejects with probability at least p every string $x \in (\Sigma - \#)^* - \mathcal{L}$. We define the class* **BQP** (bounded-error quantum polynomial time) *as the set of languages which are accepted with probability $2/3$ by some polynomial-time QTM. More generally, we define the class* $\mathbf{BQTime}(T(n))$ *as the set of languages which are accepted with probability $2/3$ by some QTM whose running time on any input of length n is bounded by $T(n)$.*

4.2 Boosting and Subroutine Calls

Theorem 4.13 *If QTM M accepts language $\mathcal{L}$ with probability $2/3$ in time $T(n) > n$, with $T(n)$ time-constructible, then for any $\varepsilon > 0$, there is a QTM M' which accepts $\mathcal{L}$ with probability $1 - \varepsilon$ in time $cT(n)$ where c is polynomial in $\log 1/\varepsilon$ but independent of n.*

Proof. Let M be a stationary QTM which accepts the language $\mathcal{L}$ in time $T(n)$.

We will build a machine that runs k independent copies of M and then takes the majority vote of the k answers. On any input x, M will have some final superposition of strings $\sum_i \alpha_i |x_i\rangle$. If we call A the set of i for which x_i has the correct answer $M(x)$ then $\sum_{i \in A} |\alpha_i|^2 \geq 2/3$. Now running M on separate copies of its input k times will produce $\sum_{i_1,\ldots,i_k} \alpha_{i_1} \cdots \alpha_{i_k} |x_{i_1}\rangle \cdots |x_{i_k}\rangle$. Then the probability of seeing $|x_{i_1}\rangle \cdots |x_{i_k}\rangle$ such that the majority have the correct answer $M(x)$ is the sum of $|\alpha_{i_1}|^2 \cdots |\alpha_{i_k}|^2$ such that the majority of $i_1, \ldots, i_k$ lie in A. But this is just like taking the majority of k independent coin flips each with probability at least $2/3$ of heads. Therefore there is some constant b such that when $k = b \log 1/\varepsilon$, the probability of seeing the correct answer will be at least $1 - \varepsilon$.

So, we will build a machine to carry out the following steps.

1. Compute $n = T(|x|)$.
2. Write out k copies of the input x spaced out with $2n$ blank cells in between, and write down k and n on other tracks.
3. Loop k times on a machine that runs M and then steps n times to the right.
4. Calculate the majority of the k answers and write it back in the start cell.

We construct the desired QTM by building a QTM for each of these four steps and then dovetailing them together.

Since Steps 1, 2, and 4 require easily computable functions whose output length depend only on k and the length of x, we can carry them out using well-behaved, normal form QTMs, constructed using Theorem 4.4, whose running times also depend only on k and the length of x.

So, we complete the proof by constructing a QTM to run the given machine k times. First, using Theorem 4.4 we can construct a stationary, normal form QTM which drags the integers k and n one square to the right on its work track. If we add a single step right to the end of this QTM and apply Lemma 4.9, we can build a well-behaved, normal form QTM moves which n squares to the right, dragging k and n along with it. Dovetailing this machine after M, and then applying Lemma 4.9 gives a normal form QTM that runs M on each of the k copies of the input. Finally, we can dovetail with a machine to return with k and n to the start cell by using Lemma 4.9 two more times around a QTM which carries k and n one step to the left. □

The extra information on the output tape of a QTM can be erased by copying the desired output to another track, and then running the reverse of the QTM. If the output is the same in every configuration in the final superposition, then this reversal will exactly recover the input. Unfortunately, if the output differs in different configurations, then saving the output will prevent these configurations from interfering when the machine is reversed, and the input will not be recovered. We show is the same in most of the final superposition, then the reversal must lead us close to the input.

Theorem 4.14 *If the language $\mathcal{L}$ is contained in the class* $\mathbf{BQTime}(T(n))$*, with $T(n) > n$ and $T(n)$ time-constructible, then for any $\varepsilon > 0$, there is a QTM M' which accepts $\mathcal{L}$ with probability $1-\varepsilon$ and has the following property. When run on input x of length n, M' runs for time bounded by $cT(n)$, where c is a polynomial in $\log 1/\varepsilon$, and produces a final superposition in which $|x\rangle|\mathcal{L}(x)\rangle$, with $\mathcal{L}(x) = 1$ if $x \in \mathcal{L}$ and 0 otherwise, has squared magnitude at least $1-\varepsilon$.*

Proof. Let $M = (\Sigma, Q, \delta)$ be a stationary, normal form QTM which accepts language $\mathcal{L}$ in time bounded by $T(n)$.

According to Theorem 4.13, at the expense of a slowdown by factor which is polynomial in $\log 1/\varepsilon$ but independent of n, we can assume that M accepts $\mathcal{L}$ with probability $1-\varepsilon/2$ on every input.

Then we can construct the desired M' by running M, copying the answer to another track, and then running the reverse of M. The copy is easily accomplished with a simple two-step machine that steps left and back right while writing the answer on a clean track. Using Lemma 4.11, we can construct a normal form QTM M^R which reverses M. Finally, with appropriate use of Lemmas 4.6 and 4.7, we can construct the desired stationary QTM M' by dovetailing machines M and M^R around the copying machine.

To see that this M' has the desired properties, consider running M' on input x of length n. M' will first run M on x producing some final superposition of configurations $\sum_y \alpha_y|y\rangle$ of M on input x. Then it will write a 0 or 1 in the extra track of the start cell of each configuration, and run M^R on this superposition $|\phi\rangle = \sum_y \alpha_y|y\rangle|b_y\rangle$. If we were to instead run M^R on the superposition $|\phi'\rangle = \sum_y \alpha_y|y\rangle|M(x)\rangle$ we would after $T(n)$ steps have the superposition consisting entirely of the final configuration with output $x; M(x)$.

Clearly, $\langle\phi|\phi'\rangle$ is real, and since M has success probability at least $1-\varepsilon/2$, $\langle\phi|\phi'\rangle \geq \sqrt{1-\varepsilon}$. Therefore, since the time evolution of M^R is unitary and hence preserves the inner product, the final superposition of M' must have an inner product with $|x\rangle|M(x)\rangle$ which is real and at least $1-\varepsilon/2$. Therefore, the squared magnitude in the final superposition of M' of the final configuration with output $x; M(x)$ must be at least $(1-\varepsilon/2)^2 \geq 1-\varepsilon$. $\square$

Corollary 4.15 $\mathbf{BQP}^{\mathbf{BQP}} = \mathbf{BQP}$.

Acknowledgement

We wish to thank Bob Solovay for several useful discussions.

References

[1] Babai, L. and Moran, S., "Arthur–Merlin games: A randomized proof system, and a hierarchy of complexity classes", *Journal of Computer and System Sciences*, vol. 36, 1988, pp. 254–276.

[2] Bennett, C. H., "Logical reversibility of computation", *IBM Journal of Research and Development*, vol. 17, 1973, pp. 525–532.

[3] Bennett, C. H. and Gill, J., "Relative to a random oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq \mathbf{co}\text{–}\mathbf{NP}^A$ with probability 1", *SIAM Journal on Computing*, vol. 10, 1981, pp. 96–113.

[4] Bernstein, E. and Vazirani, U., "Quantum complexity theory", *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 1993, pp. 11–20.

[5] Berthiaume, A. and Brassard, G., "The quantum challenge to structural complexity theory", *Proceedings of 7th IEEE Conference on Structure in Complexity Theory*, 1992, pp. 132–137.

[6] Berthiaume, A. and Brassard, G., "Oracle quantum computing", *Journal of Modern Optics*, vol. 41, no. 12, December 1994, pp. 2521–2535.

[7] Boyer, M., Brassard, G., Høyer, P. and Tapp, A., "Tight bounds on quantum searching", *Proceedings of the Fourth Workshop on Physics and Computation*, Boston, November 1996, New England Complex Systems Institute, pp. 36–43. Available online in the *InterJournal* at `http://interjournal.org`.

[8] Bshouty, N. and Jackson, J., "Learning DNF over uniform distribution using a quantum example oracle", *Proceedings of 8th Annual ACM Conference on Computational Learning Theory*, 1995, pp. 118–127.

[9] Deutsch, D., "Quantum theory, the Church-Turing principle and the universal quantum computer", *Proceedings of the Royal Society, London*, vol. A400, 1985, pp. 97–117.

[10] Deutsch, D., "Quantum computational networks", *Proceedings of the Royal Society, London*, vol. A425, 1989, pp. 73–90.

[11] Deutsch, D. and Jozsa, R., "Rapid solution of problems by quantum computation", *Proceedings of the Royal Society, London*, vol. A439, 1992, pp. 553–558.

[12] Feynman, R., "Simulating physics with computers", *International Journal of Theoretical Physics*, vol. 21, nos. 6/7, 1982, pp. 467–488.

[13] Grover, L., "A fast quantum mechanical algorithm for database search", *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 1996, pp. 212–219.

[14] Machta, J., "Phase information in quantum oracle computing", Physics Department, University of Massachusetts at Amherst, manuscript, May 1996.

[15] Shor, P.W., "Algorithms for quantum computation: Discrete logarithms and factoring", *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[16] Simon, D., "On the power of quantum computation", *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994, pp. 116 – 123.

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