On the Power of Quantum Computation *

Daniel R. Simon †

Abstract

The quantum model of computation is a probabilistic model, similar to the probabilistic Turing Machine, in which the laws of chance are those obeyed by particles on a quantum mechanical scale, rather than the rules familiar to us from the macroscopic world. We present here a problem of distinguishing between two fairly natural classes of function, which can provably be solved exponentially faster in the quantum model than in the classical probabilistic one, when the function is given as an oracle drawn equiprobably from the uniform distribution on either class. We thus offer compelling evidence that the quantum model may have significantly more complexity theoretic power than the probabilistic Turing Machine. In fact, drawing on this work, Shor has recently developed remarkable new quantum polynomial-time algorithms for the discrete logarithm and integer factoring problems.

1 Introduction

You have nothing to do but mention the quantum theory, and people will take your voice for the voice of science, and believe anything.

— Bernard Shaw, *Geneva* (1938)

The suggestion that the computational power of quantum mechanical processes might be beyond that of traditional computation models was first raised by Feynman ([Fey]). Benioff ([Beni]) had already determined that such processes were at least as powerful as Turing Machines; Feynman asked in turn whether such quantum processes could in general be efficiently simulated on a traditional computer. He also identified some reasons why the task appears difficult, and pointed out that a "quantum computer" might be imagined that could perform such simulations efficiently. His ideas were elaborated on by Deutsch ([Deu1]), who proposed that such machines, using quantum mechanical processes, might be able to perform computations that "classical" computing devices (those that do not exploit quantum mechanical effects) can only perform very inefficiently. To that end, he developed a(n at least theoretically) physically realizable model for the "quantum computer", that he conjectured might be more efficient than a classical Turing Machine for certain types of computations.

Since the construction of such a computer is beyond the realm of present technology, and would require overcoming a number of daunting practical barriers, it is worth asking first whether the proposed model even theoretically offers any substantial computational benefits over the classical Turing Machine model. The first hint of such a possibility was given by Deutsch and Jozsa ([DJ]), who presented a simple "promise problem" that can be solved efficiently without error on Deutsch's quantum computer, but that requires exhaustive search to solve deterministically without error in a classical setting. Brassard and Berthiaume ([BB1]) recast this problem in complexity theoretic terms, constructing an oracle relative to which the quantum computer is exponentially more efficient than the classical (zero-error) probabilistic Turing Machine. In [BB2], they exhibited a similar separation for non-deterministic (zero-error) Turing Machines. (See also [BB3].)

Unfortunately, the problems explored in [DJ, BB1, BB2, BB3] are all efficiently solved by a (classical) probabilistic Turing Machine with exponentially small error probability. However, Bernstein and Vazirani ([BV]) subsequently constructed an oracle which produces a superpolynomial relativized separation between the quantum and (classical) probabilistic models. They also gave the first efficient construction of a universal quantum computer which can simulate any quantum computer (as defined by Deutsch, subject

*This work was done while the author was at l'Université de Montréal, and was supported by Gilles Brassard's NSERC research grant.

†Microsoft Corp., One Microsoft Way, Redmond WA 98052-6399; dansimon@microsoft.com

0272-5428/94 $04.00 © 1994 IEEE

to a slight constraint later removed in [Yao]) with only polynomial overhead (Deutsch's universal quantum computer was subject to exponential slowdown).

In this paper, we present an expected polynomial-time algorithm for a quantum computer that distinguishes between two reasonably natural classes of polynomial-time computable function. This task appears computationally difficult in the classical setting; in particular, if the function is supplied as an oracle, then distinguishing (with non-negligible probability) between a random function from one class and a random member of the other would take exponential time for a classical probabilistic Turing Machine. (A direct consequence is an oracle which produces an exponential relativized gap between the quantum and classical probabilistic models.) Recently Shor ([Sho]), drawing on the general approach presented here and using a number of ingenious new techniques, has constructed quantum polynomial-time algorithms for the discrete logarithm and integer factoring problems.

2 Quantum probability and computation

2.1 Classical and quantum probability

We can represent a (classical) probabilistic computation on a Turing Machine (TM) as a levelled tree, as follows: each node corresponds to a state of the machine (ie., a configuration), and each level represents a step of the computation. The root corresponds to the machine's starting configuration, and each other node corresponds to a different configuration reachable with non-zero probability, in one computation step, from the configuration represented by its parent node. Each edge, directed from parent to child, is associated with the probability that the computation follows that edge to the child node's configuration once reaching the parent node's configuration. Obviously, configurations may be duplicated across a single level of the tree, as children of different parents, as well as appearing on different levels of the tree; nevertheless we represent each such appearance by a separate node. Also, we say that any such computation tree is *well-defined*, meaning that the probabilities on the edges emanating from a parent node, and the configurations associated with its children, are strictly a function of the configuration associated with the parent node, regardless of the node's position in the tree.

Of course, this tree must necessarily conform not only to the constraints set by the definition of the TM whose computation it represents, but also to the laws of probability. For example, the probability of a particular path's being followed from the root to a node is simply the product of the probabilities along its edges. Hence we can associate a probability with each node, corresponding to the probability that that node is reached in the computation, and equal to the product of the probabilities assigned to the edges in the path leading to it from the root. Moreover, the probability that a particular configuration is reached at a certain step i in the computation is simply the sum of the probabilities of all the nodes corresponding to that configuration at level i in the tree. (For example, the probability of a particular final configuration is the sum of the probabilities of all leaf nodes corresponding to that configuration). Finally, the sum of the probabilities of all the configurations at any level of the tree must always be 1, regardless of the starting configuration. A necessary and sufficient condition for a well-defined computation tree always to satisfy this constraint is that the sum of the probabilities on edges leaving any single node always be 1.

A familiar equivalent representation of our well-defined computation, of course, is the Markov chain, in which a vector of probabilities for each possible configuration at a given step is multiplied by a fixed matrix to obtain the vector of probabilities of each configuration at the next step. For example, a space-$S(n)$-bounded computation can be represented by a Markov process with $2^{O(S(n))}$ states. Such a process can always be translated into a probabilistic TM (PTM), as long as (a) it never takes one configuration to another with nonzero probability unless the second can be obtained from the first via a single TM operation (ie., changing the control state, and/or changing the contents of the cell under the tape head, and/or moving the head position by one cell), and (b) it assigns probabilities to new configurations consistently for any set of original configurations in which the control state and the contents of the cell under the tape head are identical. We say that processes with this property are *local*; obviously, the computation of any PTM can be represented as a computation tree which is not only well-defined but also local.

A computation on a quantum Turing Machine, or QTM (as described in [Deu1]) can be represented by a similar tree, but the laws of probability in the world of quantum mechanics require that we make some adjustments to it. Instead of a probability, each edge is associated with an *amplitude*. (In general, an amplitude is a complex number with magnitude at most 1, but it is shown in [BV] that it is sufficient for complexity theoretic purposes to consider only real amplitudes in the interval $[-1, 1]$.) As before, the amplitude of a node is simply the product of the amplitudes of the edges on the path from the root to that node. The amplitude of a particular configuration at any step in the computation is simply the sum of the amplitudes of all nodes corresponding to that configuration, at the level in the tree corresponding to that step. In the vector-matrix representation corresponding to the classical Markov process, a quantum probabilistic step corresponds to multiplying the vector of amplitudes of all possible configurations at the current step by a fixed matrix, to obtain the vector representing the amplitude of each configuration in the next step.

Now, the probability of a configuration at any step is the *square* of its amplitude. For example, the probability of a particular final configuration is the square of the sum (*not* the sum of the squares) of the amplitudes of all leaf nodes corresponding to that configuration. This way of calculating probability has some remarkable consequences; for instance, a particular configuration c could correspond to two leaf nodes with amplitudes α and $-\alpha$ respectively, and the probability of c being the final configuration would therefore be zero. Yet the parent nodes of these two nodes might both have nonzero probability. In fact, the computation would produce c with probability α^2 if only the configuration of *one* of the leaf nodes were in some way different. Similarly, if both leaf nodes had amplitude α, then the probability of c being the final configuration would be, not $2\alpha^2$, but rather $4\alpha^2$—that is, more than twice the probability we would obtain if either of the nodes corresponded to a different configuration. This mutual influence between different branches of the computation is called *interference*, and it is the reason why quantum computation is conjectured to be more powerful, in a complexity theoretic sense, than classical probabilistic computation.

However, even a quantum computation tree must obey the property that the sum of the probabilities of configurations at any level must always equal 1. The choice of amplitudes on the edges leading from a node to its children must therefore be restricted so as to ensure that this condition is always obeyed, regardless of the starting configuration. Now, it turns out *not* to be sufficient simply to require that for each node the sum of the squares of the amplitudes on edges leading to its children be 1. In fact, even *deterministic* computation steps, in which a single outgoing edge to a single child has amplitude 1, can violate this constraint, by causing previously different configurations in different branches of the tree to become identical. Such an event might change the pattern of interference, thereby altering the sum of the probabilities of the configurations.

Computation steps which never violate this constraint are called *unitary*, because they are equivalent to multiplying the vector of amplitudes of all possible configurations by a unitary matrix. (Recall that a unitary matrix is one whose inverse is its conjugate transpose; when we restrict ourselves to real amplitudes, such a matrix becomes orthogonal—that is, equal to the inverse of its transpose.) A QTM must always execute unitary steps; for instance, its deterministic steps must be *reversible*, in the sense that the preceding configuration can always be determined given the current one. (This restriction eliminates the aforementioned problem of distinct configurations suddenly becoming identical.) Probabilistic steps, to be unitary, must also be reversible, in the sense that some unitary probabilistic step "undoes" the step. Such "unflipping" of quantum coins is made possible by the magic of interference, which can cause alternative branches to cancel each other out, leaving the remaining ones (possibly all leading to an identical outcome) certain.

Deutsch's QTM model of computation is simply a PTM which obeys the rules of quantum, rather than classical, probability. Just as the computation tree of a classical probabilistic computation is always well-defined and local, with probabilities always summing to 1, the computation tree of a quantum computation is always well-defined, local and unitary. At each step, the amplitudes of possible next configurations are determined by the amplitudes of possible current configurations, according to a fixed, local, unitary transformation representable by a matrix analogous to the stochastic matrix of a Markov process.

It is important to note that the standard equivalent characterization of a classical probabilistic computation tree, in which a deterministic machine simply reads a tape containing pre-written outcomes of

independent fair coin tosses, does not appear to have a counterpart in the quantum model. It is true that an efficient universal QTM was shown in [BV] to require only a fixed, standard set of amplitudes for all its "probabilistic" steps. However, the reversibility condition guarantees that no new interference will be introduced once those steps have been completed (say, after all the "quantum coins" have been tossed), and any remaining computation will thus be unable to exploit quantum effects. Hence the probabilistic and deterministic parts of the quantum computation tree cannot be "teased apart" the way they can in the classical case, and we must always keep an entire tree in mind when we deal with quantum computation, rather than assuming we can just follow a particular (deterministic) branch after some point. We therefore refer to a quantum computation as resulting, at any one step, in a *superposition* of all the branches of its tree simultaneously.

2.2 Notation and an example

It is useful to have a notation to denote superpositions (that is, entire levels of a computation tree). We say that at any step i, the computation is in a superposition of all the configurations $|c_1\rangle, \ldots, |c_k\rangle$ corresponding to nodes that appear in level i of the tree representing the computation, each $|c_j\rangle$ having amplitude α_j. (Borrowing quantum mechanics notation, we distinguish symbols representing configurations from those representing amplitudes by placing $|\rangle$ brackets around configuration symbols.) An abbreviated notation for this superposition is $\sum_j \alpha_j |c_j\rangle$; as we shall see, the suggestive addition/summation notation for superpositions is quite appropriate.

A simple example of a unitary quantum probabilistic step is the quantum "fair coin flip" performed upon a single bit. It is represented by the following matrix M:

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

M acts on 2-element column vectors whose top and bottom entries represent the amplitudes of the states $|0\rangle$ and $|1\rangle$ respectively. A bit in state $|0\rangle$ is transformed by M into a superposition of $|0\rangle$ and $|1\rangle$, both with amplitude $1/\sqrt{2}$. Similarly, a bit in state $|1\rangle$ is transformed into a superposition of $|0\rangle$ and $|1\rangle$ with amplitude of magnitude $1/\sqrt{2}$ in each case, but with the sign, or *phase* of the amplitude of $|1\rangle$ being negative. In other words, the state $|0\rangle$ is transformed into $(1/\sqrt{2})|0\rangle + (1/\sqrt{2})|1\rangle$, and $|1\rangle$ becomes $(1/\sqrt{2})|0\rangle + (-1/\sqrt{2})|1\rangle$.

It turns out that this transformation is its own inverse. For example, performing it a second time on a bit that was originally in state $|0\rangle$ produces $(1/\sqrt{2})((1/\sqrt{2})|0\rangle + (1/\sqrt{2})|1\rangle) + (1/\sqrt{2})((1/\sqrt{2})|0\rangle + (-1/\sqrt{2})|1\rangle)$. Collecting like terms in this expression (here we see the aptness of the addition/summation notation) allows us to obtain the amplitude of each distinct configuration, which in this case is 1 for $|0\rangle$ and 0 for $|1\rangle$. Similarly, performing this same transformation twice on the initial configuration $|1\rangle$ gives us $|1\rangle$ (with certainty) again.

In a system of n bits, with 2^n possible configurations, we can perform such a transformation on each bit independently in sequence. The matrices representing these transformations will be of dimension $2^n \times 2^n$, of course; their rows, each corresponding to a different configuration, will each have two non-zero entries, taken from either the top or bottom row of M. Their columns will similarly have two non-zero entries each, taken from either the left or right column of M. Also, they will all be unitary, since they each represent a local, unitary transformation.

The result of performing these n different transformations in sequence will be a superposition of all possible n-bit strings. The amplitude of each string at the end of the n transformations will have magnitude $2^{-n/2}$. As the transformations are applied in turn, the phase of a resulting configuration is changed when a bit that was previously a 1 remains a 1 after the transformation is performed. Hence, the phase of the amplitude of string x is determined by the parity of the dot product of the original configuration string and x. More precisely, if the string w is the original configuration, then performing the product transformation composed of these n transformations in sequence will result in the superposition

$$2^{-n/2} \sum_x (-1)^{w \cdot x} |x\rangle.$$

This product transformation was introduced in [DJ], and is referred to in [BV] as the Fourier transformation F.

3 Using quantum probability

3.1 Problem: is a function invariant under some xor-mask?

Suppose we are given a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$, with $m \geq n$, and we are promised that either f is 1–to–1, or there exists a non-trivial s such that $\forall x \neq x'(f(x) = f(x') \Leftrightarrow x' = x \oplus s)$, where $\oplus$ denotes bitwise exclusive-or. We wish to determine which of these conditions holds for f, and, in the second case, to find s.

We now present an algorithm for a QTM which solves the above problem, with zero error probability, in expected time $O(nT_f(n) + G(n))$, where $T_f(n)$ is the time required to compute f on inputs of size n, and $G(n)$ is the time required to solve an $n \times n$ linear system of equations over $\mathbb{Z}_2$. The algorithm is very simple, consisting essentially of (an expected) $O(n)$ repetitions of the following routine:

Routine **Fourier-twice**

1. perform the transformation F described above on a string of n zeroes, producing $2^{-n/2}\sum_x |x\rangle$.
2. compute $f(x)$, concatenating the answer to x, thus producing $2^{-n/2}\sum_x |(x, f(x))\rangle$.
3. perform F on x, producing $2^{-n}\sum_y \sum_x (-1)^{x \cdot y} |(y, f(x))\rangle$.

End **Fourier-twice**

Note that the (deterministic) computation of $(x, f(x))$ from x in time $T_f(n)$ in step 2 can always be made reversible (and hence unitary) at the cost of only a constant factor in the number of computation steps. This is due to a result obtained independently by Lecerf ([Lec]) and Bennett ([Benn]).

Suppose f is 1–to–1. Then after each performance of **Fourier-twice**, all the possible configurations $|(y, f(x))\rangle$ in the superposition will be distinct, and their amplitudes will therefore all be 2^{-n}, up to phase. Their probabilities will therefore each be 2^{-2n}, and k independent repetitions of **Fourier-twice** will thus yield k configurations each distributed uniformly and independently over configurations of the form $|(y, f(x))\rangle$.

Now suppose that there is some s such that $\forall x \neq x'(f(x) = f(x') \Leftrightarrow x' = x \oplus s)$. Then for each y and x, the configurations $|(y, f(x))\rangle$ and $|(y, f(x \oplus s))\rangle$ are identical, and the amplitude $\alpha(x, y)$ of this configuration will be $2^{-n}((-1)^{x \cdot y} + (-1)^{(x \oplus s) \cdot y})$. Note that if $y \cdot s \equiv 0 \pmod 2$, then $x \cdot y \equiv (x \oplus s) \cdot y \pmod 2$, and $\alpha(x, y) = 2^{-n+1}$; otherwise $\alpha(x, y) = 0$. Thus k independent repetitions of **Fourier-twice** will yield k configurations distributed uniformly and independently over configurations of the form $|(y, f(x))\rangle$ such that $y \cdot s \equiv 0 \pmod 2$.

In both cases, after an expected $O(n)$ repetitions of **Fourier-twice**, sufficiently many linearly independent values of y will have been collected that the nontrivial string s^* whose dot product with each is even will be uniquely determined. s^* can then easily be obtained by solving the linear system of equations defined by these values of y. In the second case, this string s^* must be the s we are looking for, since we know that $y \cdot s \equiv 0 \pmod 2$ for each y generated in the second case. On the other hand, in the first case, where f is 1–to–1, s^* will simply be a random string. Hence, evaluation of, say, $f(0^n)$ and $f(s^*)$ will reveal whether we have found the true s (in the second case) or simply selected a random string (in the first case).

If we allow a bounded error probability, we can use essentially the same algorithm to solve slightly less constrained promise problems. For example, in the case where f is 1–to–1, the outputs of n/ϵ repetitions of **Fourier-twice** (for constants $\epsilon < 1$) will with probability $1 - 2^{O(n)}$ contain a basis for $(\mathbb{Z}_2)^n$. On the other hand, if there exists an s such that for a fraction at least $1 - \epsilon/n$ of possible choices of x, $f(x) = f(x \oplus s)$, then the outputs of n/ϵ repetitions of **Fourier-twice** will still all satisfy $y \cdot s \equiv 0 \pmod 2$, with constant probability, regardless of any other properties of f. Hence we can efficiently distinguish between these two classes of function (for appropriate ϵ) on a quantum computer with negligible error probability.

3.2 Relativized hardness of our problem

Now, in a relativized setting, suppose that an oracle is equiprobably either an oracle uniformly distributed among permutations on n-bit values, or an oracle uniformly distributed among those 2–to–1 functions f for which there exists a unique nontrivial s such that $f(x)$

always equals $f(x \oplus s)$. Then a classical probabilistic oracle TM would require exponentially many oracle queries to successfully distinguish the two cases with probability non-negligibly greater than 1/2.

Theorem 3.1 *Let O be an oracle constructed as follows: for each n, a random n-bit string $s(n)$ and a random bit $b(n)$ are chosen. If $b(n) = 0$, then the function $f_n : \{0,1\}^n \leftarrow \{0,1\}^n$ chosen for O to compute on n-bit queries is a random 1-to-1 function; otherwise, it is a random 2-to-1 function such that $f_n(x) = f_n(x \oplus s(n))$ for all x, where $\oplus$ denotes bitwise exclusive-or. Then any PTM that queries O no more than $2^{n/4}$ times (except for finitely many n) cannot correctly guess $b(n)$ with probability greater than $(1/2)+2^{-n/2}$ (except for finitely many n), over choices made in the construction of O, and its own probabilistic choices.*

Proof: (sketch) Consider any such PTM M. We say that M's choice of the first k queries is *good* for n if M queries O at two n-bit input values whose exclusive or is $s(n)$. If M makes a good choice of $2^{n/4}$ queries for n, then the distribution on answers given by O differs depending on $b(n)$; otherwise, the distributions are identical (ie., completely random distinct values for each distinct query). Since the probability that M guesses $b(n)$ is only greater than 1/2 when its choices are good for n, we need only calculate that probability to obtain a bound on M's probability of guessing $b(n)$.

Now, since O's answers are randomly chosen wherever they are not required to be identical, they reveal no information other than sameness or distinctness. Hence M's queries can be assumed chosen independently of previously given query answers. But for any k queries, the number of distinct pairs of input values queried (and hence the number of distinct values of s for which the queries might be good for n) is less than k^2. The probability that M's (assumed independently chosen) $2^{n/4}$ queries are good for n is therefore no better than $(2^{n/4})^2/2^n$, or $2^{-n/2}$, over choices of $s(n)$. It follows that M cannot estimate $b(n)$ with probability better than $(1/2) + 2^{-n/2}$. ∎

We can also use the above theorem to prove the existence of a specific oracle relative to which there is an exponential gap (in terms of classical computing time) between BPP and its quantum analogue, BQP (defined in the natural way; see [BV]). Let E be the (countable) set of classical oracle PTM's making at most $2^{n/4}$ queries on input 1^n. We say that $M \in E$ *solves* an oracle O generated as in Theorem 3.1 if for all n, M computes $b(n)$, with error bounded away from 1/2, on input 1^n. By Boole's inequality, the probability that there exists such an M, for an O so generated, is at most the sum over the choices of M of the probability that M solves O. Since this latter probability is zero for all $M \in E$, an oracle O chosen as described in Theorem 3.1 will with probability 1 be solved by no $M \in E$. Hence with probability 1, the language $\{1^n | b(n) = 1\}$, for $b(n)$ chosen as in Theorem 3.1, cannot be accepted with error bounded away from 1/2 by any $M \in E$.

Theorem 3.2 *There exists an oracle O relative to which $BQP \not\subseteq PTIME(2^{cn})$ (with two-sided error).*

4 Conclusion

Since any quantum computer running in polynomial time can be fairly easily simulated in $PSPACE$, as was pointed out in [BV], we are unlikely to be able to prove anytime soon that BQP is larger than P. However, Shor ([Sho]) has recently made a huge advance towards establishing the complexity-theoretic advantage of the quantum model compared to the classical one, by giving quantum polynomial-time algorithms for two well-known presumed-hard problems: computing discrete logarithms modulo an arbitrary prime, and factoring integers. His algorithms follow the very rough outline of the ones presented here, but with many additional sophistications that allow them to work over the field $\mathbb{Z}_p^*$ (for primes p such that $p-1$ is smooth) rather than $(\mathbb{Z}_2)^n$, and to extract much more than a single bit of information per iteration.

A logical next step might be to try to separate BPP and BQP based on a more general complexity-theoretic assumption such as $P \neq NP$ or the existence of one-way functions. Alternatively, it may be possible to prove limits to the advantages of quantum computation through simulation results of some kind. (In [BBBV], oracle methods are used to give evidence that $NP \not\subseteq BQP$.) Further possible simplifications of the model should also be explored; for example, does the "fair quantum coin flip" suffice as a universal non-classical step, the way its classical counterpart, the fair coin flip, suffices as a universal (classical) probabilistic step?

Another issue is that of alternative models of quantum computation. Yao ([Yao]) has presented a quantum circuit model (following [Deu2]) and proven it equivalent to the QTM. In contrast, it is not yet known whether a quantum cellular automaton is equivalent or more powerful, or even how reasonably to define such a machine. Still other distinct quantum-based computational models may exist, as well. For example, any unitary "evolution" matrix describing a quantum computation (in any model) is related (by Schrodinger's equation) to a corresponding Hermitian "Hamiltonian" matrix which describes the same process. There is also a natural notion of locality for Hamiltonians–but evolution matrices and their associated Hamiltonians are not necessarily both local or both nonlocal. It is therefore unclear whether even the definition of BQP (for QTMs or for any other model) is the same for operator-based and Hamiltonian-based encodings.

Beyond the question of models is the matter of their implementation. For example, any physical realization of a quantum computer would necessarily be subject to some error; exact superpositions would end up being represented by approximations just as deterministic discrete computations and random coin flips are approximated in modern computers using analog quantities such as voltages. Considerable work has been done on the feasibility of resiliently simulating true randomness with "approximate randomness" (see, for example, [VV], [CG]); similar work is necessary to determine if computation using approximations of quantum superpositions can be made comparably resilient. Resolution of these and other theoretical issues would be a crucial step towards understanding both the utility and the ultimate feasibility of implementing a quantum computer.

Acknowledgements

Many thanks to Charles Bennett, Ethan Bernstein, Gilles Brassard, Jeroen van de Graaf, Richard Jozsa, and Dominic Mayers for valuable insights and helpful discussion.

References

[Beni] P. Benioff, *Quantum Mechanical Hamiltonian Models of Turing Machines*, J. Stat. Phys. **29**, pp. 515–546 (1982).

[Benn] C. H. Bennett, *Logical Reversibility of Computation*, IBM J. Res. Develop. **17**, pp. 525–532 (1973).

[BBBV] C.H. Bennett, E. Bernstein, G. Brassard and U. Vazirani, *Strengths and Weaknesses of Quantum Computing*, manuscript (1994).

[BB1] A. Berthiaume and G. Brassard, *The Quantum Challenge to Structural Complexity Theory*, Proc. 7th IEEE Conference on Structure in Complexity Theory (1992).

[BB2] A. Berthiaume and G. Brassard, *Oracle Quantum Computing*, Proc. Physics of Computation (1992).

[BB3] A. Berthiaume and G. Brassard, *Oracle Quantum Computing*, J. Modern Optics, to appear.

[BV] E. Bernstein and U. Vazirani, *Quantum Complexity Theory*, Proc. 25th ACM Symp. on Theory of Computation, pp. 11–20 (1993).

[CG] B. Chor and O. Goldreich, *Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity*, SIAM J. Comput. **17**, pp. 230–261 (1988).

[Deu1] D. Deutsch, *Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer*, Proc. R. Soc. Lond. **A400**, pp. 73–90 (1985).

[Deu2] D. Deutsch, *Quantum Computational Networks*, Proc. R. Soc. Lond. **A425**, pp. 73–90 (1989).

[DJ] D. Deutsch and R. Jozsa, *Rapid Solution of Problems by Quantum Computation*, Proc. R. Soc. Lond. **A439**, pp. 553–558 (1992).

[Fey] R. Feynman, *Simulating Physics with Computers*, International Journal of Theoretical Physics **21**, pp. 467–488 (1982).

[Lec] Yves Lecerf, *Machines de Turing reversibles. Récursive insolubilité en $n\epsilon N$ de l'équation $u = \theta^n$ ou θ est un "isomorphism de codes"*. Comptes Rendus de L'Academie Francaise des Sciences **257**, 2597-2600 (1963).

[Sho] P. Shor, *Algorithms for Quantum Computation: Discrete Log and Factoring*, Proc. 35th IEEE Symp. on Foundations of Computer Science, 1994.

[VV] U.V. Vazirani and V.V. Vazirani *Random Polynomial Time is Equal to Slightly-Random Polynomial Time*, Proc. 26th IEEE Symp. on Foundations of Computer Science, pp. 417–428 (1985).

[Yao] A. Yao, *Quantum Circuit Complexity*, Proc. 34th IEEE Symp. on Foundations of Computer Science, 1993.