Quantum Computability and Complexity and the Limits of Quantum Computation

Eric Benjamin, Kenny Huang, Amir Kamil, Jimmy Kittiyachavalit
University of California, Berkeley

December 7, 2003

This paper will present an overview of the current beliefs of quantum complexity theorists and discuss in detail the impacts these beliefs may have on the future of the field. In section one we give a brief fundamental overview of classical complexity theory, defining the time and space hierarchies and providing examples of problems that fit into several important categories thereon. In section two we introduce quantum complexity and discuss its relationship to classical complexity. Section three presents an overview of the successful existing quantum algorithms, and section four presents a discussion on the actual practical gains that might be realized by quantum computation. Finally in section five we speculate briefly on the direction we believe the field to be headed, and what might reasonably be expected in the future.

1 Introduction to Computability and Complexity

In computability and complexity theory, all problems are reduced to language recognition, where a *language* is defined as a set of strings over some alphabet, typically $\{0, 1\}$. For example, $L_1 = \{0, 00, 000, \cdots\}$ is the language that contains all strings that contain only the letter 0. Once we have a language L, there are some interesting questions we can ask about it. Given an input x, can a machine decide if $x \in L$? If so, how long does it take to make this decision, with respect to the size of the input $|x|$? In order to answer these questions, it is necessary to define a model of computation.

1.1 Computational Models

There are many useful models of computation. Some examples include Boolean circuits, push-down automata with two stacks, Java programs, and Turing machines. Of these, Turing machines will be the most useful to our discussion. (For a detailed formal discussion of Turing machines see [11]). All reasonable deterministic models are essentially equivalent, since they can simulate one another in polynomial time. We will take advantage of this result by using the particular models that are most convenient for our discussion.

1.2 Classical Computability

Many people, even those who have studied computer science, erroneously believe that any conceivable well-defined problem can be computed. In fact, a language is *computable* (also called *decidable*) if and only if there exists a Turing machine that halts on all inputs, such that the final state is the accept state if the given input is in the language, and the final state is the reject state if not. The simplest example of a problem that fails this criteria, and is therefore uncomputable, is the *halting problem*: Given a machine M and its input x, is it possible to decide if M will ever halt on x? A naïve way to try answering this question is to run on M on x. But how do we know when to stop? Perhaps if we run M a little longer, it might halt. Since we have no idea how long M might run on x, this solution fails. In fact it can be formally proven that, no solution exists [11]. This problem is uncomputable.

A useful property of computability is that it is a property of a language. That is to say, the decidability of a language does not depend on any particular computational model. This means, for example, that anything that can be computed on a nondeterministic machine can also be computed on a deterministic machine [11].

1.3 Classical Complexity Classes

Once we have determined that a language is computable, we are often interested in how much it costs to decide the language. There are two resources of interest: the amount of time required, and the amount of space necessary to compute the language. Languages are classified into *complexity classes* according to how much of each of the resources they require.

1.3.1 The Class P

The set of all languages that can be computed in polynomial time on a deterministic Turing machine is called the class **P**. Familiar examples of problems in **P** include sorting a list of elements, computing the maximum flow between two points in a network of pipes, and finding the shortest path between two points in a graph.

1.3.2 The Class NP

The nondeterministic analog to **P** is the class **NP**. This is the set of all languages that can be computed in polynomial time on a nondeterministic Turing machine. Examples of problems in this class include factoring and determining whether or not a Boolean formula is satisfiable by some input. That is, given a boolean formula $f(x_1, x_2, \cdots, x_n)$ over n boolean variables is the following statement true:

$$\exists x_1 \ \exists x_2 \ \cdots \ \exists x_n \ f(x_1, x_2, \cdots, x_n).$$

This particular problem is known as SAT.

Another equivalent definition of **NP** is that it contains all languages that are *verifiable* in polynomial time on a deterministic machine. This means that given an input and a certificate of the input's membership in a language, a machine can check if the certificate is valid in polynomial time. For example, in determining whether a number k is composite, one of the factors of k could be used as the certificate. A machine could then determine the validity of the certificate by dividing k by it and checking that the remainder is 0.

Since a deterministic Turing machine is just a specific type of nondeterministic machine, $\mathbf{P} \subseteq \mathbf{NP}$.

1.3.3 The NP-Complete Languages

The hardest problems in **NP** are known as the ***NP**-complete* languages. These languages have an interesting characteristic: a solver for an **NP**-complete language can be used to solve *any* problem in **NP**, with only polynomial overhead. The classic example of an **NP**-complete language is SAT. Intuitively, this is because Boolean circuits are the building blocks of a computer, and we can thus build a circuit that simulates a Turing machine.

Other **NP**-complete problems include finding a given size fully-connected subgraph of a graph, determining whether the nodes of a graph can be colored using only three colors such that no two nodes of the same color are adjacent, and determining whether a set of numbers contains a subset that adds up to a particular number. Though these problems appear unrelated, any **NP** problem can be quickly converted into each of them.

1.3.4 Is P = NP?

Perhaps the most famous open question in computer science is whether or not the classes **P** and **NP** are equivalent. Does nondeterminism increase the power of a Turing machine, to the point where we can quickly solve problems that we otherwise couldn't? Although it may appear "obvious" that these two classes are not equivalent, given that a nondeterministic machine can perform parallel computation, no one has been able to prove this distinction.

1.3.5 The Class BPP

A language L is in the complexity class ***BPP** (Bounded-Error Probabilistic Polynomial-Time)*, if there exists a probabilistic Turing machine, M, that runs in polynomial time such that :

1. if x is in L, then M accepts x with probability $\geq \frac{2}{3}$, and

2. if x is not in L, then M accepts x with probability $\leq \frac{1}{3}$.

In other words, **BPP** is the class of decidable problems solvable by a probabilistic Turing machine in polynomial time with an error probability of at most $\frac{1}{3}$. Note, however, that the constant $\frac{1}{3}$ is arbitrary. Any constant value in the range $(0, \frac{1}{2})$ gives an equivalent definition of the class **BPP**, since repeated executions of the probabilistic machine M on x can bring the probability of correct behavior arbitrarily close to 1 [11].

It is clear that $\mathbf{P} \subseteq \mathbf{BPP}$, since a deterministic algorithm that runs in polynomial time, is like a probabilistic algorithm that runs in polynomial time with an error probability of 0, which is less than $\frac{1}{3}$. It is still an open question whether $\mathbf{P} \subset \mathbf{BPP}$ or $\mathbf{P} = \mathbf{BPP}$, though currently there exist problems which are known to lie in **BPP** but not known to lie in **P**, such as the problem of finding square roots modulo a prime number. It is also an open question whether $\mathbf{BPP} \subseteq \mathbf{NP}$ or $\mathbf{NP} \subseteq \mathbf{BPP}$.

1.3.6 The Class PSPACE

The class ***PSPACE*** is the set of decision problems that can be solved by a Turing machine using a polynomial amount of tape, given an unlimited amount of time. Analogously, ***EXPSPACE*** is the set of decision problems that can be solved by a Turing machine using an exponential amount of tape, given an unlimited amount of time. It is clear that $\mathbf{BPP} \subseteq \mathbf{PSPACE}$, since a polynomial time machine can only using a polynomial amount of tape. It's an open question whether this inclusion is strict, though it is generally believed that $\mathbf{BPP} \subset \mathbf{PSPACE}$. It has been proven that $\mathbf{NP} \subseteq \mathbf{PSPACE}$, and that $\mathbf{PSPACE} \subset \mathbf{EXPSPACE}$ [11].

2 Quantum Computability and Complexity

In beginning a discussion of quantum complexity theory it is natural to compare the computability and complexity of a quantum computer to that of a classical computer. Specifically we are interested in the questions: *can a quantum computer compute any problems that are incomputable classically?* and *can a quantum computer compute problems more efficiently than a classical computer? If so, which, why, and how?*

2.1 Quantum Computational Models

2.1.1 Quantum Circuits

The quantum analog to a Boolean circuit that operates on n bits is a quantum circuit that operates on n *qubits*. A qubit, like a classical bit, can be either 0 or 1, but unlike a classical bit, can also be in a normalized superposition of these states. The state of n qubits, then is a normalized vector in $N = 2^n$-dimensional Hilbert space (essentially $\mathbb{C}^N$ with defined geometry).

In the same way a Boolean circuit is built from NAND gates, a quantum circuit is built from *universal gates*, represented by unitary $N \times N$ matrices, that can be composed to produce any unitary $N \times N$ matrix. The result of applying a gate is the product of its matrix and the vector representing the state being operated on. One additional operation on a quantum state is *measurement*, which probabilistically determines a physical property of the state, and simultaneously removes elements of the initial superposition (if any) that are inconsistent with the result of the measurement.

Unlike classical circuits, a quantum circuit must also be reversible, which follows from the fact that only unitary operations are allowed (which in turn follows from the unitary evolution of the Hamiltonian. For a more complete discussion on this topic see [8]).

2.1.2 Quantum Turing Machines

An alternative, but equivalent computational model to quantum circuits is the *quantum Turing machine* [3]. A quantum Turing machine is defined as a triple (Q, Σ, δ), where Q is a finite set of states that includes a start and final state, Σ is the tape alphabet and includes the blank letter '#', and δ is a transition function $\delta : Q \times \Sigma \to \tilde{\mathbb{C}}^{Q \times \Sigma \times D}$ where $D = \{L, R\}$ is the direction in which the tape head moves. The set $\tilde{\mathbb{C}}$ is all complex numbers whose kth bit can be computed in time polynomial in k. The configuration of the

quantum machine is a superposition of *configurations*, where a configuration is an element of $\Sigma^{\mathbb{Z}} \times Q \times \mathbb{Z}$, the first member of which corresponds to the tape contents, the second the state, and the last the tape head position. The function δ must be unitary. A quantum machine halts when its state is a superposition of only those configurations that are in the final state. The output of the machine is the corresponding superposition of the tape contents. For a decider, the output contains a 0 in the start cell if it rejects, 1 if it accepts. The probability that the decider accepts is the total amplitude of accepting configurations in its output superposition.

2.2 Quantum Computability

Bennet showed that any classical circuit can be converted into an equivalent reversible circuit, and further that this conversion can be done efficiently [1]. Thus, we see immediately that a quantum computer is at least as powerful as a classical computer. To demonstrate that a quantum computer is no more powerful than a classical computer we present a classical Turing machine that simulates an arbitrary quantum circuit.

The quantum gates of the circuit are given, suitably encoded, as input to the Turing machine as $N \times N$ matrices, along with the measurements made on the circuit, in the order of application. In addition, the states of the Turing machine contain all the information necessary about the physical properties of each of the quantum basis states. The machine also receives an encoding of the N-dimensional initial state in its input. The machine then applies each quantum gate to the current state by doing a matrix multiplication.

One detail that may not be immediately obvious is the manner in which our simulation should handle measurement, since measurement appears to rely entirely on the physical properties of the system and true "randomness." It turns out, however, that a non-deterministic Turing machine will still compute correctly providing some computation path succeeds. Therefore we appeal to nondeterminism and let our Turing machine decide the outcome of the measurement nondeterministically, and then use the information it knows about each of the quantum basis states to delete inconsistent elements. Finally, it renormalizes the resulting vector. Thus the Turing machine can simulate each of the operations of a quantum circuit, and therefore compute anything that can be computed on a quantum circuit.

Notice, that this simulation is very inefficient. It requires space, and therefore time, exponential in the number of qubits in the quantum circuit. Bernstein and Vazirani have given a simulation that takes polynomial space, but exponential time [3], which we outline in §2.3.2. In fact, no one has yet been able to present an efficient classical simulation of a quantum computer, which has contributed to the idea that a quantum computer may be inherently exponentially faster. Note too, however, that no one has proved that such an efficient simulation does not exist.

2.3 Quantum Complexity

Having proven that a quantum computer can compute exactly the same set of languages as a classical computer, we now examine quantum complexity.

2.3.1 The Class BQP

The quantum analog to **BPP** is the class ***BQP***. This consists of all languages for which a quantum machine gives the right answer at least $\frac{2}{3}$ of the time. Shor's algorithm (see section 3) proves that factoring is in **BQP**, while it is not known to be in **BPP**. So where then, does **BQP** fit in the complexity hierarchy? It is known that **BPP** $\subseteq$ **BQP**, since anything that can be computed on a classical machine can be computed on a quantum machine with little overhead. However, it is not known whether or not this containment is strict. While there are problems such as factoring and computing a discrete log that are in **BQP** but not known to be in **BPP**, no one has actually *proven* that these problems are not in **BPP**. Similarly, the relation between **NP** and **BQP** is unknown. The argument of Bennet, Bernstein, Brassard and Vazirani that a brute force approach to quantum computation results in only quadratic gains suggests that **BQP** does not contain **NP** [2]. However, the relationship between **NP** and **BPP** is currently unknown, so whether or not **NP** contains **BQP** is also unknown [9].

It is currently known that **BQP** sits between **BPP** and **PSPACE** in the complexity hierarchy (see below for the proof that **BQP** $\subseteq$ **PSPACE**). Thus **BQP** contains all of **P** and **BPP**, and potentially some

problems in **NP** but probably none that are **NP**-complete, and perhaps some problems in **PSPACE** that are not in **NP** [13]. The latter two postulations, however, have not been proven.

2.3.2 BQP $\subseteq$ PSPACE

In the simulation of quantum circuits given above, the amount of space used by the simulation is exponential in the size of the input. Here, we outline a polynomial-space simulation of a polynomial-time quantum Turing machine [3].

First, we assume that we are simulating a quantum Turing machine $M \in \mathbf{BQP}$ for which the transition amplitudes can be computed exactly in polynomial time. We assume M runs in time $p(n)$. Then the depth of M's computational tree is at most $p(n)$. We use a depth-first search on this computational tree, using at most $p(n)$ space, and add the amplitude of this path to a running total if the final configuration is accepting. Since this amplitude can be computed exactly in polynomial time, the total only requires polynomial space to store.

Now given an arbitrary quantum Turing machine $M' \in \mathbf{BQP}$, Bernstein and Vazirani showed that it suffices to use a machine that is similar to M' but for which each transition amplitude can be calculated exactly, and to then compare the total amplitude we computed to $\frac{7}{12}$ [3]. If the amplitude is at least this amount, we accept. The total space required for this simulation is polynomial, so **BQP** $\subseteq$ **PSPACE**. Note that the total time required for this simulation is exponential, since the number of possible final configurations is in $\mathrm{O}(p(n) \cdot 2^{p(n)})$.

2.3.3 Relativized Results for Quantum Complexity

A result is said to be *relativized* if it assumes that a particular problem can be solved efficiently, whether or not we can actually do so. Although nothing universal has been proven about the relationship between the relative speeds of quantum and classical computation, several such relativized results have made progress that suggests the direction that future work will eventually go.

The first results discovered appeared to suggest that quantum computation was significantly faster than classical computation. Among these was the discovery by Deutsch, Jozsa, Berthiaume, and Brassard, of oracles under which problems could be found for which a quantum machine computed the answer with certainty in polynomial time, whereas requiring a probabilistic classical machine to solve the same problem with certainty required exponential time for some inputs [4]. Note, however, that relative to the same oracle, these problems were in **BPP**, indicating that only the requirement of certainty pushed the problem across the exponential time barrier.

Bernstein and Vazirani also showed that there exist oracles under which there exist problems that are in **BQP** but not **BPP**, which is now taken as some of the earliest evidence that QTMs are more powerful than probabilistic Turing Machines [2]. Building upon this, Simon proved the stronger result that there exists an oracle relative to which **BQP** cannot even be simulated by a probabilistic Turing Machine allowed to run for an exponential number of steps [10].

Despite this evidence as to the power of quantum computation, the above results do not appear to break any major ground in terms of computational complexity. That is, although they suggest that quantum computation is at least somewhat more powerful than classical computation, they do not lend any real insight as to whether it will allow us to compute **NP**-complete problems. But results by Bennet, Bernstein, Brassard, and Vazirani do. They demonstrated that relative to an oracle chosen uniformly at random, the class **NP** can never be solved in less than $\sqrt{2^n}$ time. (This result is tight, as we show in §3.4.) They then go on to explain that while their results do not prove that **NP** cannot be contained in **BQP**, they do establish the fact that there is no intrinsic property of quantum computation that will function like a black box to solve **NP**-complete problems [2].

3 Quantum Algorithms

We now turn to some concrete examples of problems that can be computed faster on quantum computers than on classical computers.

3.1 Quantum Fourier Transforms (QFT)

As digital media has become commonplace, one of the most important, and most often performed classical computations is the *Fourier transform*. For an N-dimensional space, the best known classical algorithm takes $O(N \log N)$ time. By contrast the best known quantum algorithm operates on a superposition of all states in the space, and takes only $O(\log^2 N)$ time, an exponential speedup.

When performing the quantum Fourier transform on a superposition states, however, we can only obtain a random sampling of the transformation. This places limitations on the applicability of the algorithm for working with classical data. One useful application is to find the period of a function, which is used by Shor's algorithm below.

3.2 Shor's Algorithm

Many widely-used cryptographic protocols are based on the difficulty of factoring on a classical computer. No efficient classical algorithm for factoring is known. In 1994, Shor published an algorithm that proved factoring can be done efficiently on a quantum computer.

Classical number theory tells us that computing the period of the function $f(x) = a^x \mod N$, where $\gcd(a, N) = 1$, allows us to factor N. On a classical computer the only known ways to compute this period take exponential time. On a quantum computer, however, we can use the quantum Fourier transform to efficiently find the period. The quantum algorithm, then, can factor exponentially faster than the best known classical algorithm [8].

3.3 Grover's Algorithm

Another important problem for which a quantum algorithm exists that is faster than the best classical algorithm is searching. Given a set S of n unsorted items and a marking function $f : S \to \{0, 1\}$, find an element $x \in S$ such that $f(x) = 1$. The best classical algorithm is to search through the entire list, which takes $O(m \cdot n)$ time, where the computation of $f(x)$ takes $O(m)$ time.

The best known quantum algorithm for searching, *Grover's algorithm*, takes $O(m \cdot \sqrt{n})$ time by searching a superposition of all the elements in S. Given such a superposition, it inverts the coefficients of the states x for which $f(x) = 1$ by applying the inversion circuit in figure 1, where CZ is the controlled phase flip gate

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

It then applies a unitary transformation that reflects each coefficient about the average of all of them. The coefficients of the unmarked states decrease slightly, but the coefficients of the marked states increase to greater than their initial values. Repeatedly applying the inversion circuit and the reflection operation can increase the total amplitude of the marked items, and in fact, $O(\sqrt{n})$ repetitions are required to get close to 1 [7]. The running time of the search, then, is $O(m \cdot \sqrt{n})$, since it does $\sqrt{n}$ iterations, each of which apply $f(x)$ and thus take $O(m)$ time. This is a quadratic improvement over the classical algorithm.

3.4 Grover's Upper Bound on NP Problems

Using Grover's algorithm, we can construct a solver for an arbitrary **NP** problem that runs in $O(\sqrt{2^n})$ time, where n is the size of the required witness to verify the problem. We illustrate the technique by constructing such a solver for the **NP**-complete problem SAT [5].

Since $SAT \in \mathbf{NP}$, there exists a polynomial time verifier for SAT, that takes in an input x and a witness y and outputs 0 if the witness is invalid, and 1 if it is valid. In the case of SAT, x is a boolean formula in some n variables, and y is an assignment to those variables. Note that $|y| = n$, so the number of possible witnesses is 2^n. Call the quantum circuit implementing this verifier $f(x, y)$.

Now there are two possible cases. Either there does not exist a valid y, or there exists at least one, and possibly many. In the first case, $x \notin SAT$, and in the second case, $x \in SAT$. Thus we need to distinguish between these two cases.

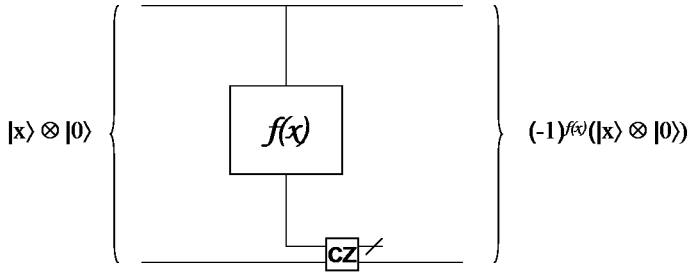

Figure 1: A circuit that inverts states x for which $f(x) = 1$. The state takes in a superposition of all states plus an ancilla bit, and outputs a superposition in which the marked elements are inverted, plus the unchanged ancilla bit.

We can accomplish this by running Grover's algorithm on the set of all possible witness, using $f'(y) = f(x, y)$ as the marking function. Since there are 2^n possible witness, this will take $O(\sqrt{2^n})$ time. If at least one valid y exists, then the search will find one with high probability $p > \frac{1}{2}$, and we can run it through the verifier to make sure it is valid. If none exists, the algorithm will return a random witness with probability 1. Thus if $x \in SAT$, we get the wrong answer with probability $1 - p$, and if $x \notin SAT$, we never get the wrong answer. We can repeat the procedure k times to get a probability of $\leq (1 - p)^k$ of getting the wrong answer in either case, which can be made arbitrarily close to 0.

Thus SAT can be solved in $O(\sqrt{2^n})$. Using the same construction as above, we can produce a solver for any **NP** problem that runs in $O(\sqrt{2^n})$ time.

4 The Practical Impacts of Modern Quantum Complexity Beliefs

When Shor first published his now-famous quantum factoring algorithm in 1994, the excitement surrounding the discovery caused hundreds of the best minds in complexity theory to ask: *how long until we can use this to solve **NP**-complete problems?* and their experimental counterparts to ask: *how long until we can get one of these machines running in a lab?* Nearly a decade later the questions asked by complexity theorists have changed dramatically. In a recent publication by well known complexity theorist Lance Fortnow, from NEC Research, and the keynote speaker at the 2003 MSRI Conference, he asks [6]: *how long until we can prove in the general case that a tight upper bound for quantum performance speedups is quadratic?*

This drastic change in the gains quantum complexity theorists expect to realize, raises several important questions: *how much longer does this mean that the technology will need to evolve before quantum computing is a viable and practical way to compute?* and: *if a quadratic speedup is in fact a tight upper bound, what does this tell us about those cases (like Shor's factoring algorithm), for which the speedup appears to be exponential?*

While the first question may fall slightly outside the scope of this paper, it provides a nice example from which to draw perspective on our theoretical discussions of basic complexity theory in sections I and II above. Imagine for a moment that we have at our disposal a classical 2 GHz desktop, and a quantum computer, which runs at 500 Hz, or 4,000,000 times slower than the classical desktop. Now imagine that, like researchers believed in 1994, quantum computing would open the door to solving **NP**-complete problems, which we can presently solve only in exponential time, in say, N^2 time. For an arbitrary such problem of size 30, the classical desktop finishes roughly three times faster. For an arbitrary such problem of size 40, the quantum computer finishes roughly 170 times faster. And for an arbitrary such problem of size 50, the quantum computer finishes roughly 110,000 times faster than the classical desktop. Since many complexity theorists believe that it will eventually be proved that $\mathbf{P} \neq \mathbf{NP}$, the implication of the above example is startling clear: even for modest sized **NP**-complete problems, a fantastically slow quantum computer would be enormously beneficial. This was the picture that Shor's algorithm helped to create in 1994, and helps to explain why for the last decade experimentalists have worked feverishly at producing just such a machine.

Let us now consider a different situation that better reflects the current beliefs of the quantum complexity community as discussed above. Imagine that we again have the same 500 Hz quantum computer, and 2 GHz

classical computer, only now the quantum computer can provide at most a quadratic speedup in processing, as supported by the relativized oracle results discussed in section II. First it should be clear that in this situation we have no hope of solving even modest sized **NP** problems, as the polynomial gain is insufficient to break the exponential barrier for N sufficiently large.

More interestingly then, let us consider the case of searching, which Grover proved could be done in $\sqrt{N}$ time on a quantum computer, as compared with N time on a classical computer. In this case, even a problem of size 4,000,000 will run 4 times slower on the quantum computer than on its classical counterpart, and a problem of size 10,000 will run roughly 80 times slower. Since most classical applications consist of many (relatively) small operations performed in sequence rather than a few instances of large problems, it is clear that for most processing applications the gains from quantum computing would be small until the technology matured a great deal.

It is also important to consider the fact that to achieve a 500 Hz quantum computer (approximately 50 times faster than the fastest currently achieved in a laboratory) nano-production techniques would need to improve significantly, and, for comparison purposes, by the time Intel is producing at the 20-micron range, they believe they will be producing chips at 25-60 GHz (personal communication).

Let us now turn to the issue of reconciling those quantum algorithms that appear to provide an exponential speedup in processing time with a suspected upper bound of quadratic speedups. Following the publication of Shor's algorithm in 1994, many people believed that the effect of superposition in quantum computation was equivalent to being able to perform an exponential amount of processing in parallel [12]. A more recent view of the subject suggests that rather than making an exponential amount of raw computing power available, quantum computation does the same amount of raw processing as a classical computer, it simply takes advantage of unique quantum mechanical structures that are well suited for certain problems [12]. This idea suggests that one possible explanation for the exponential speedup seen in Shor's algorithm and by the QFT could be the following: there are certain problems for which the superposition structure of quantum computation is particularly well suited such that despite the fact that in general quantum computation is at most quadratically faster than classical computing, certain exploitable structures allow for even greater gains in the same way that a binary tree allows for much faster searching than does an unsorted list.

While there is no evidence at present to dispute such an explanation, it does seem unlikely that such structures could widely exist, but that ten years of concerted effort by the complexity theory community would fail to find more than a few examples. By contrast it seems equally hard to imagine that such structures are in fact so rare and so esoteric that it would be realistic for complexity theorists to have only discovered a handful of them in ten years. In light of these observations and the fact that neither factoring nor QFT have been shown to be **NP**-complete, an alternative and more compelling explanation might be: quantum computation makes it easier to exploit complicated structures that exist in previously considered problems, but that are so hard to conceptualize we have not yet discovered how to leverage them. It therefore suggests that there may exist many exploitable structures within these problems that clever quantum algorithms could use to achieve polynomial running time, but that these structures are so complex or well disguised that they have not yet been discovered. Notice that this explanation says nothing about the gains quantum computation yields with respect to **NP**-complete problems, as these problems are not believed to have any structure at all. It simply suggests that quantum computation supplies a new set of tools with which to approach problems, and that though these tools may not be inherently "faster," they may be easier to apply to certain problems.

5 Conclusions

Perhaps unsurprisingly, quantum complexity theory now faces many of the same big questions that have faced the classical complexity community for many years. And unfortunately there doesn't appear to be a solution on the horizon. Despite a great deal of effort by people hoping to prove both sides of the issues, virtually nothing concrete is known about the true power of quantum computation. Since we are left then to sift through relativized results that, at best offer mere glimpses at what might be the larger picture, we choose to take a cautious position.

We believe that the present evidence suggests that the realistic gains to be reaped from quantum computing are going to be polynomial, and not exponential. Nevertheless we acknowledge that a quadratic speedup

in processing time is significant, and as nano-production continues to develop, we expect that practical implementations of quantum computation will become a reality. Initially the applications may be limited to database systems and other specific applications that require enormous problem sizes (with which to exploit Grover's algorithm), but we believe that in time other examples of modest, but substantial, polynomial gains will be discovered.

6 References

[1] Bennet, C., "Logical reversibility of computation," *IBM J. Res. Develop.*, Vol. 17, 1973, pp. 525-532.

[2] Bennet, C., Bernstein, E., Brassard, G., and Vazirani, U., "Strengths and weaknesses of quantum computation," *Special issue on Quantum Computation of the Siam Journal of Computing*, Oct. 1997.

[3] Bernstein, E. and Vazirani, U., "Quantum complexity theory," *Special issue on Quantum Computation of the Siam Journal of Computing*, Oct. 1997.

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

[5] Boyer, M., Brassard, G., Høyer, P., and Tapp, A., "Tight bounds on quantum searching," *arXiv e-print quant-ph/9605034*, 1996.

[6] Dam, W. V. "Quantum complexity theory," *SQuInT Retreat 2003*, lecture, June 2003.

[7] Fortnow, L., "One complexity theorist's view of quantum computing," *Theoretical Computer Science*, 292(3):597-610, 2003.

[8] Grover, L., "A framework for fast quantum mechanical algorithms," *lanl e-print quant-ph/9711043*, Nov. 1997.

[9] Nielsen, M. and Chuang, I, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[10] Robinson, S., "Emerging insights on limitations of quantum computing shape quest for fast algorithms," *SIAM News*, vol. 36, no. 1, Jan./Feb. 2003.

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

[12] Sipser, M., *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[13] Steane, M., "A quantum computer only needs one universe," *lanl e-print quant-ph/0003084*, Mar. 2003.