

Revisiting a Limit on Efficient Quantum Computation

Tarsem S. Purewal Jr.*
Department of Computer Science
University of Georgia
Athens, GA 30602
purewal@cs.uga.edu

ABSTRACT

In this paper, we offer an exposition of a theorem originally due to Adleman, Demarrais and Huang that shows that the quantum complexity class BQP (Bounded-error Quantum Polynomial time) is contained in the classical counting class PP (Probabilistic Polynomial time). Our proof follows the one given by Fortnow and Rogers that relates quantum computing to counting complexity classes by way of GapP functions. The contribution of this paper is an exposition of an important result that assumes a minimal background in computational complexity theory and no knowledge of quantum mechanics.

Keywords

Computational Complexity, Quantum Computing

1. INTRODUCTION

In the race to build a quantum computer, the question of whether or not such a machine will be more useful than a classical computer is often overlooked. This is not surprising, since the physicists and engineers have plenty of obstacles to overcome in their attempts to build such a machine without worrying about practical applications! In addition, the construction of the first scalable quantum computer will be such a significant scientific achievement that we'll probably be able to overlook the fact that we don't have any interesting programs to run on it.

The question, however, remains a fundamental issue for computer scientists to address. One approach to answering the question affirmatively is to search for new quantum algorithms that outperform the best known classical algorithms. Peter Shor's impressive algorithm for integer factorization [17] and Lov Grover's clever search algorithm [13] seemed to foreshadow a barrage of fast quantum algorithms for prac-

tical problems. Unfortunately, such algorithms remain elusive. The lack of progress in the area is quite disheartening.

Complexity theory provides an alternative approach to answering this question by comparing the types of problems solvable by classical models of computation to the types of problems solvable by quantum analogues of these models. In this approach, it is useful to separate problems into classes based on models of computation that can successfully solve them with varying resources. If we can prove that the quantum classes are larger, then we would know for sure that the quantum models offer an advantage over classical models, and therefore more practical quantum algorithms that outperform classical algorithms are likely to exist. Although it has not been proven that quantum computers are more powerful than their classical counterparts in any practical sense, this approach has provided us with a wealth of weaker evidence that this is likely true.

We can also use the complexity-theoretic approach to answer a related, but slightly different question: *what are the limitations on models of computation that are allowed to use quantum effects?* By showing that quantum classes are contained in classical classes that are well-understood, we can gain some insight into limitations of these quantum models. This study might lead us to understand the types of problems at which quantum computers will excel.

David Deutsch [7] pioneered this line of research by showing that the universal quantum Turing Machine could be simulated by a classical Turing Machine, thereby showing that the class of problems solvable by a quantum computer is no larger than the class of problems solvable by a classical computer. The question of efficiency was addressed by Bernstein and Vazirani [6], where it was shown that the class of problems solvable on a quantum computer in polynomial-time is no bigger than the class of problems solvable in polynomial-space on a classical computer. Adleman, Demarrais and Huang [3] improved this result by showing that this class is contained in the more restrictive class PP.

In this paper, we describe a relatively straight-forward proof of this last result, namely that the quantum complexity class BQP, which stands for Bounded-error Quantum Polynomial time, is contained in the classical complexity class PP, which stands for Probabilistic Polynomial time. Since BQP is widely regarded as the class of problems that are efficiently solvable on a quantum computer this result shows limitations on efficient quantum computation.

The original proof of Adleman, *et al.* gives an explicit construction of the PP machine that accepts the language. Our proof is a complete account of the one discovered by

*Work supported in part by a grant from NSA's Mathematical Sciences Program

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM SE'06 March 10-12, 2006, Melbourne, Florida, USA
Copyright 2006 ACM 1-59593-315-8/06/0004 ...\$5.00.

Fortnow and Rogers [11] which relates the complexity of quantum computation to the complexity of functions that count the number of accepting computations of a nondeterministic Turing machine. The construction is implicit, and uses the fact that PP can be characterized in terms of GapP functions. Fortnow and Rogers actually prove a stronger result, namely that BQP is contained in AWPP. This is useful because it gives us more information about how BQP relates to PP. In order to preserve clarity, however, we do not discuss AWPP.

Our exposition has the advantage that it is designed to be understood by those who have a modest background in theoretical computer science and no knowledge of quantum mechanics. We attempt to highlight some of the intriguing, basic aspects of quantum computers throughout. In effect, we hope that this paper gives an elementary, but non-trivial, introduction to an exciting area of research.

2. PRELIMINARIES

For an introduction to the theory of computation, see the book by Sipser [19]. We assume a rudimentary understanding of the material through Chapter 8.

In this paper, our standard model of computation will be the polynomial-time bounded single tape Turing Machine and its nondeterministic analogue, the polynomial-time bounded single tape nondeterministic Turing Machine (which we will abbreviate NTM). The computation of a NTM on a particular string gives rise to a *computational tree* of polynomial depth, where the vertices are configurations of the machine and the edges are transitions from one configuration to another. The leaves of the tree correspond to halting configurations of the machine. A *computational path* is a root-leaf path through the tree. An *accepting path* is a computational path in which the leaf represents an accepting configuration and a *rejecting path* is a path in which the leaf represents a rejecting configuration.

Without loss of generality, we can assume that a NTM is standardized in the following way. At each step of the computation, there are exactly two paths that the machine can follow. Furthermore, we can assume that every computational path on an input of a fixed length has the same depth [16, page 254]. In other words, every computational tree of an NTM can be thought of as a perfect binary tree of polynomial depth. When we refer to a machine as *precise*, we will assume it to be of this form. Not every NTM in this paper will be precise - we will specify when we make this assumption.

2.1 The Quantum Model

This description of the quantum model of computation is roughly based on one due to Simon [18]. For a more complete discussion of quantum computing see the book by Kitaev, Shen and Vyalıy [14] or Nielsen and Chuang [15].

To build a polynomial-time probabilistic model of computation from the nondeterministic model, we associate a NTM's transitions with probabilities. A *polynomial-time probabilistic Turing Machine* (PTM) is such a machine with the restriction that the sum of the transition probabilities leaving every state is 1.

The computation tree of a PTM has the transition probabilities as an additional feature associated with the edges. The probability of the machine reaching a particular configuration in the tree is the product of the transition probabili-

ties along the path from the root to that configuration. The probability of finding the machine in a particular configuration at time t is the sum of the probabilities of all appearances of that configuration at depth t in the tree. Note that by making the sum of the transitions leaving a particular configuration be 1, we force our machine to be *well-formed*, meaning that the sum of the configuration probabilities at time t is always 1. In other words, at all times of the computation the machine is in a valid probability distribution of configurations.

The quantum model is also a probabilistic model, but the rules used to compute the probability are different. The transitions are associated with *amplitudes*, rather than probabilities. In the most general setting, we allow the amplitudes to be efficiently computable complex numbers¹. As in the probabilistic model, the amplitude associated with a particular configuration in the tree is the product of the transition amplitudes along the path from the root to that configuration. The amplitude of a particular configuration at time t is the sum of the amplitudes of all appearances of that configuration at depth t in the tree. The key difference is that the probability of finding the machine in a particular configuration at time t is the *square* of the amplitude of that configuration at time t (in the case that the amplitude is complex it's the amplitude times its complex conjugate).

In the quantum case, we still need to make sure that the machine is well-formed and it turns out *not* to be sufficient to simply require that the sum of the squares of the amplitudes leaving a particular configuration be 1. We require that the machine evolve in a *unitary* fashion, meaning that it preserves a proper probability distribution. It turns out that if each local configuration transition can be performed reversibly, meaning that no information is lost at each step of the computation, then the machine evolves in a unitary manner (the converse is also true). We call a machine that has these properties and, in addition, has all branches of the computation halting after the same number of steps a *polynomial-time quantum Turing machine* (QTM) [6].

From this description, it is not obvious that the quantum Turing machine is as powerful as the classical deterministic Turing machine, due to the fact that the quantum machine is required to evolve in a unitary (and therefore reversible) manner. A deterministic Turing machine has no such restriction. Early work by Charles Bennett, however, shows that a reversible computer can efficiently simulate a classical deterministic computer [4], and therefore a quantum computer can efficiently simulate a classical computer. In addition, Bernstein and Vazirani show that a quantum computer can efficiently simulate a classical probabilistic computer [6].

With this in mind, we can see why the quantum model might be more powerful than the probabilistic model - simulating a quantum computer deterministically seems more difficult. If we use a classical deterministic computer to simulate a probabilistic computation, following a single path through the computational tree will yield *some* information about the computation. If we find an accepting configuration, we can say with certainty that the machine accepts with a non-zero probability. If we find a rejecting configuration, we know that the machine rejects with a non-zero probability. In contrast, following a single path of a quantum

¹Here, a complex number $a + bi$ is efficiently computable iff the j th bit of a and b can be computed in time polynomial in j .

computation yields *no* information about the computation.

This is because the quantum model allows for alternative paths to affect each other - a phenomenon known as *quantum interference*. For example, the computation may halt with exactly two accepting configurations that appear with amplitudes $\frac{1}{2}$ and $-\frac{1}{2}$. The actual probability that the machine accepts is

$$\left(\frac{1}{2} - \frac{1}{2}\right)^2 = 0.$$

In this case, even though the machine accepts with a non-zero amplitude in more than one computational path, it never actually accepts. It is not clear how one can conclude anything about a quantum computation without keeping track of the entire computational tree. Any simulation that is required to keep track of the entire computational tree of a non-deterministic Turing machine might require an exponential amount of time.

The class BQP is the class of languages decidable on a QTM with an error-rate bounded below $\frac{1}{2}$ by some constant amount. In our definition, we use $\frac{1}{3}$ as the constant value.

DEFINITION 2.1. *A language, L , is in the class BQP iff there exists a polynomial-time quantum Turing Machine, T , with the following property.*

- $x \in L$ implies $\text{prob}[T \text{ accepts } x] \geq \frac{2}{3}$
- $x \notin L$ implies $\text{prob}[T \text{ accepts } x] \leq \frac{1}{3}$

BQP ends up not being very sensitive to the allowed transition amplitudes. Adleman, *et al.* show that for BQP we can assume that our transition amplitudes come from the set $\{0, \pm 1, \pm \frac{4}{5}, \pm \frac{3}{5}\}$ [3], and we will use this fact in our proof.

2.2 Counting Complexity Background

In addition to BQP, we'll have the occasion to examine some of the more exotic specimens from the complexity zoo [1]. These classes all relate to the number of accepting computational paths yielded by a NTM on a particular string. We'll start by defining the class PP, which stands for *Probabilistic Polynomial Time*. This class was originally studied by Gill [12].

DEFINITION 2.2. *A language, L , is in the class PP iff there exists a NTM, T , with the property that the computation of T on a string $x \in L$ halts with more than half of the computational paths accepting [16]. If $x \notin L$ then at least half of the computational paths are rejecting. We say that T accepts L by majority.*

A typical language in PP is *MAJORITYSAT*, which consists of the encodings of boolean formulas for which more than half of the possible assignments to the variables makes the formula evaluate to true. In fact, *MAJORITYSAT* turns out to be complete for PP, which roughly means that it is at least as hard as any other language in the class [16, page 257].

We'll also have the occasion to talk about the class of functions that count the number of accepting computational paths precisely. This class of functions is #P, a class originally studied by Valiant [20].

DEFINITION 2.3. *A function, $f : \Sigma^* \rightarrow \mathbb{N}$ is in the class #P iff there exists a NTM, T , with the property that the computation of T on x halts with exactly $f(x)$ accepting computational paths.*

A typical function in #P is #SAT, which is a function that takes in encodings of boolean formulas and outputs the number of different assignments that make the formula evaluate to true.

Using simple properties of NTM's, we can see that the #P functions are closed under multiplication and addition. In the case of addition, we build a new machine that non-deterministically simulates the two machines simultaneously. In the case of multiplication, we build a new machine that simulates one of the machines, and then simulates the other machine from the accepting configurations of the first.

LEMMA 2.4. *#P is closed under addition and multiplication.*

Unfortunately, the #P functions are not closed under subtraction - they are defined as non-negative valued-functions. It's still useful, however, to talk about the difference of two #P functions. Therefore, we define the GapP functions, which are the subtractive closure of the #P functions².

DEFINITION 2.5. *A function, $f : \Sigma^* \rightarrow \mathbb{Z}$ is in the class GapP iff there exist functions $f_1, f_2 \in \#P$ such that $f(x) = f_1(x) - f_2(x)$.*

The following properties of GapP functions are easily verified.

LEMMA 2.6. *GapP is closed under addition, subtraction and multiplication.*

3. RELATING BQP TO PP

We begin by proving a simple theorem that relates PP to GapP functions. This theorem is originally due to Fenner, *et al.* [8]

THEOREM 3.1. *A language, L , is in PP iff there exists a GapP function, f , with the following property:*

$$x \in L \text{ iff } f(x) > 0.$$

PROOF. If L is in PP, then we construct a GapP function with the specified property in the following manner. Let T_1 be the NTM that accepts L by majority and let T_2 be the same as T_1 , but with the accept and reject states interchanged. If we let f_1 be the #P function defined by T_1 and let f_2 be the #P function defined by T_2 , then

$$f(x) = f_1(x) - f_2(x)$$

is a GapP function with the specified property, i.e. $x \in L$ iff T_1 yields more accepting configurations than T_2 .

Now if we have a GapP function, f , where $x \in L$ iff $f(x) > 0$, then by definition we have $f_1, f_2 \in \#P$ where $f(x) = f_1(x) - f_2(x)$. Let T_1 be the machine associated with f_1 and T_2 be the machine associated with f_2 . Without loss of generality, we can assume that T_1 and T_2 are precise, meaning that their associated computational trees are perfect binary trees. Furthermore, we can assume that these machines are standardized in such a way that they both always halt after the same number of steps on an input of a

²Technically, this requires us to show that the #P functions are a subset of the GapP functions, but this easily follows from the fact that the function that is identically zero on all strings is in #P.

particular length³. In other words, we can assume that on input x , both T_1 and T_2 halt after exactly n steps. Using these two machines, we'll construct a machine that accepts L by majority.

First we construct T'_2 , a machine that is the same as T_2 but with the accept and reject states interchanged. Now we construct a machine T that acts as follows: on input x , it nondeterministically simulates T_1 and T'_2 on x and accepts or rejects accordingly.

Now fix a string x and suppose T_1 and T_2 halt after n steps. Then T yields exactly 2^{n+1} halting configurations. We can easily count the number of accepting configurations, $a(x)$, as

$$a(x) = f_1(x) + (2^n - f_2(x)) = 2^n + (f_1(x) - f_2(x))$$

Now if $x \in L$ we have $f(x) = f_1(x) - f_2(x) > 0$ and if $x \notin L$ we have $f(x) = f_1(x) - f_2(x) \leq 0$. Here, we are adding the value of $f(x)$ to 2^n which is exactly half of T 's halting configurations. Therefore T accepts L by majority and so $L \in \text{PP}$. \square

We are now ready to prove the main theorem of the paper.

THEOREM 3.2. $\text{BQP} \subseteq \text{PP}$

PROOF. Let $L \in \text{BQP}$. Then we have a QTM, M , that decides L with an error rate below $\frac{1}{3}$. Now fix an input $x \in \{0, 1\}^*$ and suppose the computation of M on x halts after n steps. Since M decides a language in BQP , we can assume that all transition amplitudes come from the set $\{0, \pm 1, \pm \frac{4}{5}, \pm \frac{3}{5}\}$. Therefore, on input x , the machine yields halting configurations whose amplitudes are all rational, and the denominator of each of these rational amplitudes is 5^k for some $0 \leq k \leq n$. Each one of these halting amplitudes is either positive or negative.

Let A_+ represent the multiset of positive valued amplitudes of the accepting configurations and A_- represent multiset of the absolute value of the negative valued accepting configurations. Define R_+ and R_- similarly for the rejecting configurations.

By the rules of quantum probability and the definition of BQP , we can compute the acceptance probability as follows.

$$\text{prob}[M \text{ accepts } x] = \left(\sum_{\alpha \in A_+} \alpha - \sum_{\alpha \in A_-} \alpha \right)^2$$

where $\alpha = a/5^k$ for some integers a and $0 \leq k \leq n$. We can compute the rejection probability in a similar fashion.

$$\text{prob}[M \text{ rejects } x] = \left(\sum_{\rho \in R_+} \rho - \sum_{\rho \in R_-} \rho \right)^2$$

We define a new function $a(x)$ as the acceptance probability multiplied by 5^{2n} and $r(x)$ as the rejection probability

³A technicality - when we construct these "standardized" machines, we have to make sure that we don't change the number of accepting configurations yielded by any input, so that the $\#\text{P}$ functions do not change. When we add the additional transitions from the accepting states, we do so in such a way that a previous accepting configuration can only yield a single accepting configuration - we force the rest to reject. This makes it so the $\#\text{P}$ functions of the "standardized" machines are the same as those of the original machines.

multiplied by 5^{2n} . These functions are integer valued because the 5^n will cancel out the denominator of 5^k for each summand. Now notice that if $x \in L$ we have

$$a(x) = \left(\sum_{\alpha \in A_+} 5^n \alpha - \sum_{\alpha \in A_-} 5^n \alpha \right)^2 \geq \frac{2}{3} 5^{2n}$$

$$r(x) = \left(\sum_{\rho \in R_+} 5^n \rho - \sum_{\rho \in R_-} 5^n \rho \right)^2 \leq \frac{1}{3} 5^{2n}$$

and if $x \notin L$ we have

$$a(x) = \left(\sum_{\alpha \in A_+} 5^n \alpha - \sum_{\alpha \in A_-} 5^n \alpha \right)^2 \leq \frac{1}{3} 5^{2n}$$

$$r(x) = \left(\sum_{\rho \in R_+} 5^n \rho - \sum_{\rho \in R_-} 5^n \rho \right)^2 \geq \frac{2}{3} 5^{2n}$$

So clearly the function $f(x) = a(x) - r(x)$ has the property that $x \in L$ iff $f(x) > 0$. We now show that each of the four summations in $a(x)$ and $r(x)$ are in $\#\text{P}$. It will follow that $f(x)$ is in GapP since GapP is closed under multiplication and subtraction (as per Lemma 2.5).

To this end, we construct four NTMs from the description of M . We describe the construction of the machine that computes $\sum_{\alpha \in A_+} 5^n \alpha$. The other three constructions are completely analogous.

First of all, we can construct a NTM by simply stripping M of transition amplitudes. In addition, we select an unused portion of the tape to keep track of the computational path's amplitude (in particular, the numerator and the exponent of the denominator). When we reach a halting state, we reject if the original state in M was not an accepting configuration with a positive amplitude. Otherwise, we compute $i = n - k$, where k specifies the power of 5 in the denominator of the original positive accepting amplitude. Then $5^i a$ represents the number of accepting configurations that we need to produce, where $a = 5^k \alpha$. Now while $i > 1$, we decrement i and spawn 5 configurations that repeat this process. When $i = 1$ we spawn a accepting configurations. This gives us exactly $5^i a$ accepting configurations without significantly increasing the running time of the computation. Therefore this summation is in $\#\text{P}$.

Now since $f(x) \in \text{GapP}$, $L \in \text{PP}$. \square

4. CONCLUDING REMARKS

The proof technique used here has been generalized (see, for example, Fenner, Green, Homer, and Pruijm [9]), as it nicely characterizes the interference aspect of quantum computation. Unfortunately, it doesn't characterize the unitary evolution of a quantum computer, which is likely the biggest restriction on the quantum model (see §2.1). It seems that one might be able to put BQP in a smaller complexity class by using the fact that a quantum computer is required to evolve in a unitary manner.

The result described here implies that it is unlikely that a quantum computer will be able to solve problems like MAJORITYSAT unless $\text{BQP} = \text{PP}$. It seems very unlikely that this equality holds as PP is a large class of problems, many of which are probably very difficult. There is more compelling evidence that $\text{BQP} \neq \text{PP}$, however. In their original

paper, Fortnow and Rogers go a step further and show that, in fact, BQP is *low* for PP [11]. This implies that showing $BQP = PP$ would have some consequences that are thought to be very unlikely, namely the collapse of the *counting hierarchy* (see reference [1] for a definition).

A more pressing question is whether or not a quantum computer will be able to solve NP-Complete problems in polynomial time. Unfortunately, it seems as though this proof technique will not be able to resolve this question. There has, however, been evidence presented to suggest that this important class of problems will not be efficiently solvable on a quantum computer [5].

5. ACKNOWLEDGEMENTS

Many thanks to Rod Canfield, Gregory Baramidze, Chris Bennett, and Fred Maier for listening to me drone on about this stuff for hours. Thanks to Yaoyun Shi for an extraordinarily enlightening discussion on quantum computing. Also, thanks to Shelby Funk and Bob Robinson for reading earlier drafts of this paper and offering suggestions.

6. REFERENCES

- [1] Aaronson, Scott and Greg Kuperberg. The Complexity Zoo. http://qwiki.caltech.edu/wiki/Complexity_Zoo
- [2] Aaronson, Scott. *Quantum Computing, Postselection, and Probabilistic Polynomial-Time*. Proceedings of the Royal Society A, Vol. 461, pp. 3473-3482, 2005.
- [3] Adleman, Leonard M., Jonathan Demarrais, and Ming-deh A. Huang. *Quantum Computability*. SIAM Journal on Computing. Vol. 26, No. 5, pp. 1524-1540, 1997.
- [4] Bennett, Charles. *Logical reversibility of computation*. IBM Journal of Research and Development. Vol. 17, pp. 525-532, 1973.
- [5] Bennett, Charles, Ethan Bernstein, Gilles Brassard and Umesh Vazirani. *Strengths and Weaknesses of Quantum Computation*. SIAM Journal on Computing. Vol. 26, No. 5, pp. 1510-1523, 1997.
- [6] Bernstein, Ethan and Umesh Vazirani. *Quantum Complexity Theory*. SIAM Journal on Computing. Vol. 26, No. 5, pp. 1411-1473, 1997.
- [7] Deutsch, David. *Quantum theory, the Church-Turing principle and the universal quantum computer*. Proceedings of the Royal Society, London. Vol. A400, pp. 97-117, 1985.
- [8] Fenner, Stephen, Lance Fortnow and Stuart Kurtz. *Gap-Definable Counting Classes*. Journal of Computer and System Sciences. Vol 48, #1, pp. 116-148, 1994.
- [9] Fenner, Stephen, Frederick Green, Steven Homer, and Randall Pruim. *Determining Acceptance Possibility for a Quantum Computation is Hard for the Polynomial Hierarchy*. Proceedings of the Royal Society, London A (1999) Vol 455, pp. 3953-3966.
- [10] Fortnow, Lance. *One complexity theorist's view of quantum computing*. Theoretical Computer Science. Vol 292, #3, pp.597-610, 2003.
- [11] Fortnow, Lance and J.D. Rogers. *Complexity Limitations on quantum computation*. Journal of Computer and System Sciences. Vol 59, #2, pp. 240-252, 1999.
- [12] Gill, J. *Computational Complexity of Probabilistic Turing Machines*. SIAM Journal on Computing. Vol. 6, No. 4, pp. 675-695, 1977.
- [13] Grover, Lov. *Quantum Mechanics helps in searching for a needle in a haystack*. Physical Review Letters. Vol. 79, No. 2, pp. 325-328, 1997.
- [14] Kitaev, A. Yu., A.H. Shen and M.N. Vyalyi. *Classical and Quantum Computation*. Graduate Studies in Mathematics Volume 47. American Mathematical Society, Providence, RI, 1999.
- [15] Nielsen, Michael and Isaac Chuang. *Quantum Computation and Information*. Cambridge University Press, 2000.
- [16] Papadimitriou, Christos. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.
- [17] Shor, Peter. *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*. SIAM Journal on Computing. Vol. 26, No. 5, pp.1484-1509, 1997.
- [18] Simon, Daniel. *On the Power of Quantum Computation*. SIAM Journal on Computing. Vol. 26, No. 5, pp. 1474-1483, 1997.
- [19] Sipser, Michael. *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, MA, 1997.
- [20] Valiant, L.G. *The complexity of computing the permanent*. Theoretical Computer Science. Vol. 8, pp. 189-201, 1979.