Quantum Query Complexity Revisited

by

Daniel C Preda

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Computer Science and Engineering

and

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2003
[June 2003]

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2003

Certified by .....
Madhu Sudan
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Quantum Query Complexity Revisited

by

Daniel C Preda

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2003, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Computer Science and Engineering
and
Bachelor of Science in Computer Science and Engineering

Abstract

In this thesis, we look at the polynomial method for quantum query complexity and relate it to the $\mathbf{BQP}^A = \mathbf{P}^A$ question for a random oracle A. We will also look at some open problems and improve some bounds relating classical and quantum complexity.

Thesis Supervisor: Madhu Sudan
Title: Professor

Acknowledgments

The author wishes to thank Prof. Madhu Sudan, Dr. Howard Barnum, Dr. Harry Buhrman and Dr. Lance Fortnow for extensive guidance on this project, and Mihai Badoiu, Bogdan Fedeles and Hein Roehrig for support and useful discussions.

Contents

1 Introduction **6**

1.1 Quantum computers 6

1.2 Qubits 7

1.3 Operators 8

1.4 Quantum algorithms 8

1.5 Quantum Complexity 10

1.5.1 The Oracle Model 11

1.5.2 Query complexity 12

2 The Polynomial Method **13**

2.1 Classical Query Model 13

2.2 Quantum Query Model 14

2.2.1 Quantum states 14

2.2.2 Query Complexity 14

2.3 Complexity Measures 15

2.3.1 Sensitivity & Block Sensitivity 15

2.3.2 Certificate complexity 16

2.3.3 Degree of Representing/Approximating Polynomial 17

3 Query Complexity **21**

3.1 Classical 21

3.2 Quantum 23

3.3 $\mathbf{P}^{A,\mathbf{PSPACE}}$ versus $\mathbf{BQP}^{A,\mathbf{PSPACE}}$ 24

3.4 $pr\mathbf{P}^{A,\mathbf{PSPACE}}$ versus $pr\mathbf{BQP}^{A,\mathbf{PSPACE}}$ 25

4 Conclusions and Future Work **28**

Chapter 1

Introduction

1.1 Quantum computers

Very recently people have begun to consider other options to classical computation models. By increasing the speed of the electronic computers, the dimensions of their components approach the atomic limit, and the computation model used in present will no longer be valid. Moreover, there are problems for which no polynomial-time algorithm is known. These problems have key practical importance in fields such as cryptography (factoring a large number), or even everyday life (traveling salesman problem, which belongs to a class of so-called NP-complete problems; a polynomial-time algorithm that solves one of them can be modified to solve any of them). Some of the most important new computation models are quantum, molecular, and DNA computing. Of these, the most promising results, both theoretical and experimental, have been achieved within the quantum computation model.

1.2 Qubits

The basic "unit" of information is the bit. Classically, it is represented by a Boolean variable, and it can have values 0 or 1. From a physical perspective, a certain state of the system is assigned to each value of the bit. For example, by using nuclear spin, spin-up is mapped to the logical value 0, and spin-down to the logical value 1. If using polarized photons, 0 is assigned to left-right polarization, and 1 to up-down polarization. In general, the only condition is to distinguish between the two logical values. Hence ,the corresponding physical representations need to be orthogonal. Generally, the logical 0 will be denoted by $|0\rangle$, and the logical 1 by $|1\rangle$.

There are some important differences between the classical and the quantum representations of $|0\rangle$ and $|1\rangle$. Although a classical Boolean variable has a definite logical value, a quantum bit (called qubit) can be in a superposition of $|0\rangle$ and $|1\rangle$: thus, in the case of spins, $|0\rangle + |1\rangle$ points along the x-axis, and, in the case of photons, $|0\rangle+|1\rangle$ corresponds to a $\pi/4$ polarization. A spin in the $|0\rangle+|1\rangle$ (or $|0\rangle-|1\rangle$) state has probability of 50% of being up (along the z-axis), and probability 50% of being down (along the $-z$-axis) when measured. However, this is not just a classical probability distribution, because the superposition of $|+\rangle(= |0\rangle + |1\rangle)$ and $|-\rangle(= |0\rangle - |1\rangle)$ gives back $|0\rangle$. If it were classical, the state $|+\rangle + |-\rangle$ would have 50%-50% probability of getting $|0\rangle$ or $|1\rangle$ because each of its components ($|+\rangle$ and $|-\rangle$) has a 50%-50% probability. Moreover, due to the Heisenberg uncertainty principle, non-commuting observables cannot be measured with infinite precision, and hence measurement disturbs the system. By measuring an observable O and obtaining the value o, the system will be left in an eigenstate of O with eigenvalue o. This fact is connected to another pure quantum property, the so-called no-cloning theorem: no quantum state

can be perfectly copied without disturbing the original state.

1.3 Operators

The laws of physics require that the operators applied to any quantum state should be reversible. Also, the l-2 norm of the states (seen as a vector) should be preserved. These conditions require that operators are unitary: orthogonal states are mapped into orthogonal states, or, equivalently, the operators preserve the angle between vectors.

1.4 Quantum algorithms

One of the first goals of quantum computing was to overcome the speed limits of known classical computation models. This goal seems plausible because the quantum computation using n qubits is performed in a huge Hilbert space with dimension 2^n. Furthermore, there are non-local quantum correlations that have no classical equivalent, including entanglement and interference.

For instance, consider the following "quantum gambling" game [7] that takes advantage of superposition. Suppose there are two persons, Alice and Bob. Alice prepares a secret quantum state, and hands it to Bob. Without looking at the state, Bob either applies or does not apply an operator U (that they agreed on) to the state (e.g. he either flips the qubit or does not flip it); then, he hands it back to Alice. She applies another operator to the state, and then, without measuring, she announces the state she has. Classically, Alice has 50% chances of winning, but, by taking advantage of the quantum mechanics, she can guess the state with 100% probability.

If Bob can apply U or I(=identity, equivalent to not applying U) to the state, then Alice prepares the initial state to be a eigenstate of U. Then, whatever Bob does, he actually does not change the state, and Alice receives the same state she prepared in the first place. This example shows how quantum computers can outperform the classical ones.

The most important quantum algorithms up to date are the Grover's database search algorithm [6] (that runs faster than the classical one) and Shor's factoring algorithm [11] that achieves an exponential speed-up relative to the best classical factorization algorithm. Recently, algorithms that might solve NP-complete problems faster have been proposed, but no bound on the running time has been found so far. Using Grover's basic idea of searching, several other algorithms have emerged, including an algorithm for the collision problem: given a function that is 2-to-1, find two elements that are mapped to the same element. Similarly, there are some quantum algorithms that use the period-finding technique to perform better than their classical analogs.

Other applications of the quantum computation model in cryptography gave rise to powerful quantum cryptography codes (e.g. quantum key distribution: agreement on a secret cryptographic key is secure against attack by a quantum computer or by an unlimited computational power.)

In practice it was found that the non-local correlations are extremely fragile and tend to decay very rapidly, mainly due to the interaction with the environment. However, even if one could eliminate most of the interaction with the surroundings, the logic gates used in the computation cannot be ideally implemented. Thus, if quantum error-correction codes (QECC) were not discovered, a quantum computer

would have a very small chance of working properly.

1.5 Quantum Complexity

In order to be able to compare in a formal way the power of a quantum computer, some definitions are required.

- **P** is the class of languages decidable in classical polynomial deterministic time.
- **BPP** is the class of languages decidable in classical polynomial probabilistic time with bounded error.
- **NP** is the class of languages decidable in classical polynomial non-deterministic time.
- **PP** is the class of languages decidable in classical polynomial probabilistic time with unbounded error.
- **PSPACE** is the class of languages decidable in classical polynomial deterministic space.
- **BQP** is the class of languages decidable in quantum polynomial probabilistic time with bounded error..

The only strict inclusion known is $\mathbf{P} \subset \mathbf{EXP}$. The known inclusions are: $\mathbf{P} \subseteq \mathbf{BPP}$, $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$. All these are believed to be strict, except $\mathbf{P} \subseteq \mathbf{BPP}$.

Where does **BQP** stand? It has been proven in 1997 that $\mathbf{BQP} \subseteq \mathbf{PSPACE}$ and shortly after that $\mathbf{BQP} \subseteq \mathbf{PP}$ [4]. Since then, there has been no progress in capturing the power of **BQP**. There are some reasons why this result could be improved: first, all the languages with known efficient quantum algorithms lie in **NP**; second, the unitarity condition of the operators is hardly used in the proof.

There have been little other results in this field. Most notably, the result of Yao, who showed that quantum Turing machines and quantum circuits are equivalent. We will use only the circuit framework, because of its simplicity. A given sequence of quantum operators U_1 through U_T are applied to the input string (state), where the only limitation is that any operator only acts on a constant number of qubits (it can be shown that acting on two qubits is enough for universal quantum computing).

1.5.1 The Oracle Model

An oracle for a language L is a device that can decide in unit time the language L. A machine A can use the oracle in the following way: at any step in the computation, the machine can ask the oracle a question and go on after learning the answer - the resulting device is represented by A^L. In the quantum framework, a little more technical details are needed in order to ensure unitary operation.

There are quite a few (expected) results: $\mathbf{BQP}^{\mathbf{BQP}} = \mathbf{BQP}$ [3], $\mathbf{PP}^{\mathbf{BQP}} = \mathbf{PP}$ [5], for most oracles A $\mathbf{BQP}^A$ is not a subset of $\mathbf{NP}^A$ [3].

Daniel Simon asked if the existence of one-way functions is enough to guarantee the hardness of **BQP**. Fortnow and Rogers [5] proved that, relative to an oracle, this is not true. However, they did not use cryptographic one-way functions (functions that are hard to invert on most of the inputs).

They also proved that, if $\mathbf{P} = \mathbf{BQP}$ relative to a random oracle, then $\mathbf{BPP} = \mathbf{BQP}$. Moreover, they raised the interesting (and possibly easier) question of whether for a random oracle A, $\mathbf{P}^{A,\mathbf{PSPACE}} = \mathbf{BQP}^{A,\mathbf{PSPACE}}$.

1.5.2 Query complexity

This model is very similar to the oracle one, but we measure the number of queries instead of the running time. It has been shown [1] that for total functions, the classical deterministic query complexity is upper bounded by a polynomial in the quantum query complexity. On the other hand, for very sparse functions, there can be an exponential gap between classical and quantum query complexities.

Though, there is an open question whether the gap is still polynomial on almost all almost-total functions. This issue is linked to the previous question (if for a random oracle A, $pr\mathbf{P}^{A,\mathbf{PSPACE}} = pr\mathbf{BQP}^{A,\mathbf{PSPACE}}$), in the sense that a polynomial gap implies a positive answer.

Chapter 2

The Polynomial Method

In this chapter we will give an overview of the mathematical tools needed by the polynomial method for quantum query complexity introduced by Beals *et all.* in [1]. We will also prove some new bounds for almost all functions.

2.1 Classical Query Model

Let f be a Boolean function of n Boolean variables: $f(x) : \{0,1\}^n \to \{0,1\}$ (from now on we'll assume that any f has this form unless otherwise indicated). The easiest way to understand the query complexity is to look at the decision tree for f. This is a binary tree whose nodes are labeled by variables x_i and leafs by a 0 or 1. The tree is evaluated recursively, at each node moving left if the corresponding variable is 0, and to the right if it's 1. The complexity of such a tree is defined as it's depth, equivalent to the maximum number of queries over all the inputs. The query (decision) complexity of f, $D(f)$ is defined as the minimal complexity taken over all the trees that compute f.

2.2 Quantum Query Model

2.2.1 Quantum states

Each n-qubit state is a *superposition* of all n-bit Boolean strings of length n:

$$|\psi\rangle = \sum_{i \in \{0,1\}^n} \alpha_i |i\rangle$$

where all α_i's are complex numbers that obey $\sum_i |\alpha_i|^2 = 1$.

We can either apply a unitary operator to this state, or chose to measure it. In the latter case, we have probability $|\alpha_i|^2$ to measure string i. If this happens, we collapse the state to just $\alpha_i|i\rangle$

2.2.2 Query Complexity

Because all quantum operations are unitary, we have to be careful how we define a query in this case. The transformation O (seen as a call to the oracle) maps $\sum_i |i, b, z\rangle$ into $\sum_i |i, b \oplus x_i, z\rangle$ (i indicates which variable we query, $\oplus$ is the *xor* operation, and z represents the rest of the computer memory).

A T-query quantum decision tree can be built in the following way: start from the state $|0^n\rangle$, apply a unitary transformation U_0, than a query O, than another unitary operation U_1, another query O and so on, until the last unitary operation U_T. The final state $\psi = U_T O U_{T-1} \ldots U_1 O U_0 |0^n\rangle$. We can define the output of the tree as value of the first qubit after we measure the final state.

A quantum decision tree computes f exactly if the output is always $f(x)$. It computes f with bounded error if the probability that the output is $f(x)$ is at least $2/3$ for all x. We define $Q_E(f)$ to be the minimum number of queries to compute f exactly, and $Q_2(f)$ to be the minimum number of queries to compute f with bounded error. Clearly, $Q_2(f) \leq Q_E(f) \leq D(f) \leq n$.

2.3 Complexity Measures

2.3.1 Sensitivity & Block Sensitivity

Let f be a Boolean function. The *sensitivity* indicates how sensitive the function is to changes in the input.

Definition 2.3.1 The *sensitivity* $s_x(f)$ of f on x is the number of x_i's such that f changes the value when x_i is flipped: $f(x) \neq f(x^i)$.

Definition 2.3.2 The *sensitivity* $s(f)$ of f is $\max_x s_x(f)$.

Definition 2.3.3 The *block sensitivity* bs(f) of a function f is the maximum number of disjoint blocks of variables $B_1, B_2, \ldots, B_b$ such that when we flip all the variables in a block, the function flips: $f(x) \neq f(x^{B_i})$

Clearly, $s(f) \leq bs(f)$.

Lemma 2.3.4 *Let B be a minimal sensitive block for f. Then $|B| \leq s(f)$*

Proof: Because B is minimal, if we flip any variable in x^B we flip the value of the function. Hence, $|B| \leq s_{x^B}(f) \leq s(f)$ ∎

2.3.2 Certificate complexity

The certificate complexity of a function f indicates how many of the variables have to be fixed in order to find the value of the function.

Definition 2.3.5 We say that an assignment A of values to some subset of the variables is *consistent* with $x \in \{0,1\}^n$ if for all the variables in the subset, $A(i) = x_i$. The size of A is the cardinality of the subset.

Definition 2.3.6 A *b-certificate* for f is an assignment A such that $f(x) = b$ for all x consistent with A.

Definition 2.3.7 The *certificate complexity* $C_x(f)$ of f on x is the size of the smallest $f(x)$-certificate that is consistent with x. The *certificate complexity* of f is $C(f) = \max_x C_x(f)$. The 0-*certificate complexity* of f is $C^0(f) = max_{x|f(x)=0} C_x(f)$. Similarly we define $C^1(f)$

Lemma 2.3.8 $bs(f) \leq C(f)$

Proof: A certificate for x has to contain at least one variable from each sensitive block (otherwise we could flip all the variable sin that block and the function changes its value), hence $bs_x(f) \leq C_x(f)$ ∎

Theorem 2.3.9 $C(f) \leq s(f)bs(f)$ *[8]*

Proof: For an input x, let $B_1, B_2, \ldots B_b$ be disjoint minimal blocks of variables which achieve the block sensitivity $b = bs_x(f)$. Consider an assignment A of variables that are set accordingly to $\bigcup_i B_i$.

Suppose A is not an $f(x)$-certificate. Then there is an input y consistent with A such that $f(x) \neq f(y)$. Define B_{b+1} by $y = x^{B_{b+1}}$ (the set of variables on which y differs from x). f is sensitive to B_{b+1} on x, and B_{b+1} is disjoint from all other B_i's (otherwise it would not be consistent with A). This implies that $bs_x(f) = b + 1$, contradiction.

Because the size of each B_i is at most $s(f)$, the size of the certificate A is at most $s(f)bs_x(f)$ ∎

2.3.3 Degree of Representing/Approximating Polynomial

Definition 2.3.10 A polynomial $p : \mathbf{R}^n \to \mathbf{R}$ *represents* f if $p(x) = f(x)$ for all x. A polynomial $p : \mathbf{R}^n \to \mathbf{R}$ *approximates* f if $|p(x) = f(x)| \leq 1/3$ for all x.

In our case, because we work with Boolean variables ($x^2 = x$), we can assume p is multilinear.

Definition 2.3.11 The degree $deg(f)$ is the degree of the polynomial representing f. The *approximate degree* $\overline{deg}(x)$ is the minimum degree among all polynomials that approximate f.

We would like to relate the degree of representing/approximating polynomial to the block sensitivity/certificate complexity of a function.

Theorem 2.3.12 *Let $p : \mathbf{R} \to \mathbf{R}$ a polynomial such that for all integers $0 \leq i \leq n$ we have that $a \leq p(i) \leq b$, and for some real x, $0 \leq x \leq n$ we have that $|p'(x)| \geq c$. Then, $deg(p) \geq \sqrt{cn/(c + b - a)}$ [10]*

Theorem 2.3.13 *$bs(f) \leq 2\ deg(f)^2$ [9]*

Proof: Let p a polynomial of degree d that represents f, $b = bs(f)$, and a and $B_1, B_2, \ldots B_b$ the input and blocks that achieve the block sensitivity. Assume without loss of generality that $f(a) = 0$. We will map p into a polynomial q as follows:

- $x_j = y_i$ if $a_j = 0$ and $j \in B_i$
- $x_j = 1 - y_i$ if $a_j = 1$ and $j \in B_i$
- $x_j = a_j$ if j is not in any B_i

q has the following properties:

- multilinear of degree at most d
- $q(y)$ is in $\{0,1\}$ for all $y \in \{0,1\}^n$
- $q(0^b) = 0$
- for all unit vectors $e_i \in \{0,1\}^b, q(e_i) = p(x^{B_i}) = f(x^{B_i}) = 1$

Now we symmetrize q over $\{0,1\}^b$, and let r the resulting polynomial, of degree $\leq d$. For all integers $0 \leq i \leq b$ we have that $0 \leq r(i) \leq 1$. Moreover, because $r(0) = 0$ and $r(1) = 1$, we have that $r'(x) \geq 1$ for some $x \in [0,1]$. Applying the previous theorem, $d \geq \sqrt{b/2}$. ∎

Using the same idea, for the degree of approximating polynomial we get:

Theorem 2.3.14 $bs(f) \leq 6\ \widetilde{deg}(f)$ *[9]*

Now we'll prove a strong bound for almost all functions:

Theorem 2.3.15 *For almost all functions, the representing polynomial has full degree.*

Proof: We will use a counting argument. First let $A = \{x||x| \text{ is even and } f(x) = 1\}$, and $B = \{x||x| \text{ is odd and } f(x) = 1\}$. Let $p = \sum_S c_S M_S$, where c_S is the coefficient of the monomial $M_S = \prod_{i \in S} x_i$, T be a subset of S and $f(T)$ the value of the function on the input where only variables in T are 1. Then [2] we have that: $c_S = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T)$

Now let's find the value of the coefficient of the degree n monomial ($S = \{1, 2, \ldots, n\}$.

$$c_S = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T) = (-1)^n \sum_{x \in A \bigcup B} (-1)^{|x|} = (-1)^n (|A| - |B|)$$

We will count the number F of functions from $\{0,1\}^n \rightarrow \{0,1\}$ for which $|A| = |B|$ (and hence have degree $\leq n$). We can assign i 1-values to the 2^{n-1} elements of A in $\binom{2^{n-1}}{i}$. But $|A| = |B|$, so there are another $\binom{2^{n-1}}{i}$ ways of choosing the elements of B. $F = \sum_{i=0}^{2^{n-1}} \binom{2^{n-1}}{i}^2$, which turns out to be $\binom{2^n}{2^{n-1}} = \Theta(2^{2^n}/\sqrt{2^n})$.

■

Theorem 2.3.16 *For almost all functions, the degree of the approximating polynomial is at least* $n/2 - O(\sqrt{n} \log n)$

Proof:

Again, we'll use a counting argument. First, let's show that in a polynomial approximates a Boolean function, its coefficients of the degree d monomials are bounded in absolute value by 2^{nd+1}. We'll do this by induction:

- for $d = 0$, the coefficient is bounded by $4/3 \leq 2$
- consider the coefficient c of the monomial $x_{i_1}, \ldots, x_{i_d}$ the value of the polynomial when only the variable sin this monomial are 1 is the sum of c and of coefficients of monomials whose variables are a subset of the variables of our monomial. There are at most $2^d - 1$ such coefficient, and each is bounded by $2^{n(d-1)+1}$. So, $|c| \leq (2^d - 1)2^{n(d-1)+1} + 4/3 \leq 2^{nd+1}$

Now we'll bound the number of functions that can be approximated by a polynomial of degree d. Consider two polynomials p and q whose coefficients differ by at most 2^{-n-2}. Then, their values on any assignment differ by at most $2^n 2^{-n-2} = 1/4 \leq 1/3$, so they approximate the same Boolean function. By the previous observation, these coefficients are in the interval $[-2^{nd+1}, 2^{nd+1}]$. If we split this interval in intervals of size 2^{-n-2} we get $2^{O(n^2 d)}$ smaller intervals. Let's choose an interval for each coefficient. Then, there is at most one Boolean function with such coefficients. There are at most $N(n,d) = \sum_{i=0}^{d} \binom{n}{i}$ monomials of degree at most d, so there are at most $(2^{O(n^2 d)})^{N(n,d)}$ combinations.

For a suitable α in $d = n/2 - \alpha\sqrt{n}\log n$ we can make $N(n,d) \leq 2^n/n^4$, so the total number of combinations turns out to be at most $2^{2^n/n}$ ∎

Chapter 3

Query Complexity

In this chapter we will give an overview of the polynomial method for quantum query complexity introduced by Beals *et all.* in [1]. Then we will apply this method to prove that for any oracle A, $\mathbf{BQP}^{\mathbf{PSPACE},A} = \mathbf{P}^{\mathbf{PSPACE},A}$. We will try the same approach for the *promise* version of the theorem, and show that it fails. Finally, we will prove some new bounds and improve old ones regarding the relationship between classical and quantum query complexity.

3.1 Classical

In this section we give Beals *et all.*'s result connecting the deterministic query complexity to the complexity measures presented in the previous section. We also present some improvements to this bound.

Theorem 3.1.1 $D(f) \leq bs(f) \min(C^0(f), C^1(f))$ *[1]*

Proof:

Without loss of generality, suppose that the minimum certificate is $C^1(f)$. The algorithm that computes $f(x)$ has the following structure:

1. Repeat at most $bs(f)$ times: pick a consistent 1-certificate C (one that agrees with the variables queried so far) and query the variables whose values are still unknown. If there is no such C, return 0. If all the queried variables agree with C, return 1.

2. Pick a consistent x and return $f(x)$.

There are at most $bs(f)$ steps and at each step we query at most $C^1(f)$ variables. We have to prove the algorithm is correct.

If it returns an answer in part 1 it's either because there are no more 1-certificates (and hence f is 0), or we found a consistent 1-certificate (and hence f is 1).

Suppose it returns an answer in part 2. We have to show that all x's give the same value for f. Suppose not: there are consistent x, y such that $f(y) = 0$ and $f(x) = 1$. We have already queried $b = bs(f)$ 1-certificates $C_1, C_2, \ldots, C_b$, and x also contains another 1-certificate C_{b+1}. For each $1 \leq i \leq b+1$ let B_i be the variables on which C_i and y disagree. y^{B_i} agrees with C_i, so $f(y_i^B) = 1$. This means that f is sensitive to each B_i on y. Because all C_i's are chosen consistent with the variables queried so far, all B_i's are disjoint. We arrived at a contradiction, that f has block sensitivity $b+1$.

■

In terms of the degree of the representing / approximating polynomial, we have: $D(f) \leq 8\ deg(f)^6$ and $D(f) \leq 216\ \overline{deg}(f)^6$. The result for the degree of approximating polynomial can be improved to:

Theorem 3.1.2 $D(f) \leq 2\ deg(f)^4$

Proof:

First we'll prove that each monomial of maximum degree has a sensitive block for 0. From f, obtain g by setting all variables outside a monomial of maximum degree to 0. Clearly this g cannot be constant, so there is a subset of the variable sin the monomial such that $g(0^B) \neq g(0)$ and hence $f(0^B) \neq f(0)$

Because each monomial contains a sensitive block for 0 (and there are at most $bs(f)$ such blocks), these is a set of $deg(f)bs(f)$ variables that intersect all monomials of maximum degree (just take all the variables in the monomials as long as there is still a disjoint monomial). If we query all these variable, we obtain a polynomial g with $deg(g) \leq deg(f)$ and $bs(g) \leq bs(f)$. This procedure can go on at most $deg(f)$ times before we are left with a constant polynomial, hence we only need $deg(f)^2 bs(f) \leq 2\ deg(f)^4$

■

3.2 Quantum

Theorem 3.2.1 *If a quantum decision tree makes T queries, then there exists complex valued multilinear polynomials α_i of degree at most T such that the final state is $\sum_i \alpha_i(x)|i\rangle$, where x represents the queried variables. [1]*

Proof: Each query maps state $|i, b, z\rangle$ into $|i, b \oplus x_i, z\rangle$, so $\alpha|i, 0, z\rangle + \beta|i, 1, z\rangle$ is mapped into $((1 - x_i)\alpha + x_i\beta)|i, 0, z\rangle + (x_i\alpha + (1 - x_i)\beta)|i, 1, z\rangle$ So the coefficients of the states are polynomials in x whose degrees increase by at most 1 for each query. ■

Theorem 3.2.2 *$Q_E(f) \geq deg(f)/2$ (and similarly, $Q_2(f) \geq \overline{deg}(f)/2$ [1]*

Proof: The acceptance probability of the decision tree will be the sum over all accepting states : $\sum_{i \in S} |\alpha_i|^2$, which is a polynomial that represents f and has degree at most $2Q_E(f)$ ∎

Corollary 3.2.3 *$D(f) = O(Q_E(f)^4)$ and $D(f) = O(Q_2(f)^6)$*

3.3 $\mathbf{P}^{A,\mathbf{PSPACE}}$ versus $\mathbf{BQP}^{A,\mathbf{PSPACE}}$

Fortnow and Rogers [5] proved that if $\mathbf{P}^A = \mathbf{BQP}^A$ relative to a random oracle A, then $\mathbf{BPP} = \mathbf{P}$. However, this might be a rather strong assumption. Instead, we'll prove the following fact:

Theorem 3.3.1 *For all oracles A, $\mathbf{P}^{A,\mathbf{PSPACE}} = \mathbf{BQP}^{A,\mathbf{PSPACE}}$*

What we actually prove with this fact is that, given enough computational power to $\mathbf{P}$, it can access an oracle as efficiently as $\mathbf{BQP}$. The intuition behind the proof is that, because $\mathbf{P}^{\mathbf{PSPACE}} = \mathbf{BQP}^{\mathbf{PSPACE}}$, we can use the fact that the number of queries (classical and quantum) are polynomially related.

Proof: Let L be a language decided by a $\mathbf{BQP}$ Turing machine M with access to $\mathbf{PSPACE}$ and A. We can see the output of such a machine as a function of the input $x = x_1 x_2 \ldots x_n$, of the queries to the oracle A $a_1 a_2 \ldots a_m$, and of the queries to the $\mathbf{PSPACE}$ machine $b_1 b_2 \ldots b_p$. Because the machine runs in poly-time, m and p are poly(n).

Using **Theorem 3.2.1** we conclude that the output is a **PSPACE**-computable multilinear polynomial p (with the corresponding Boolean function f) of the answers A gave to the m queries. From **Corollary 3.2.3** we know that there is a deterministic algorithm that makes only poly(m) queries and computes the same polynomial. We only have to prove that this algorithm can be run on a **P** machine with access to oracle A and a **PSPACE** machine.

The algorithm is actually contained in the proof of **Theorem 3.1.1**:

1. Repeat at most $bs(f)$ times: pick a consistent 1-certificate C (one that agrees with the variables queried so far) and query the variables whose values are still unknown. If there is no such C, return 0. If all the queried variables agree with C, return 1.

2. Pick a consistent x and return $f(x)$.

The algorithm tales at most $bs(f) + 1$ =poly(m)=poly(n) steps. Each step involves either picking a consistent 1-certificate (which can be done in **PSPACE**: try all combinations of variables until we find a certificate), or querying A (at most $\min(C^0(f))$ =poly(m)=poly(n) variables). This can be done on a **P** machine with access to A and a **PSPACE** machine.

■

3.4 $pr\mathbf{P}^{A,\mathbf{PSPACE}}$ versus $pr\mathbf{BQP}^{A,\mathbf{PSPACE}}$

One might want to try the previous approach to check whether $pr\mathbf{P}^{A,\mathbf{PSPACE}} = pr\mathbf{BQP}^{A,\mathbf{PSPACE}}$. Then, two things have to be proven:

- $pr\mathbf{P}^{\mathbf{PSPACE}} = pr\mathbf{BQP}^{\mathbf{PSPACE}}$

- For any non-total function f (defined only on some of the inputs), $D(f)$ =poly$(Q_2(f))$

Theorem 3.4.1 $pr\mathbf{P}^{\mathbf{PSPACE}} = pr\mathbf{BQP}^{\mathbf{PSPACE}}$

Proof: Any **BQP** quantum network can be simulated in **PSPACE**, so using the same idea we have $pr\mathbf{BQP} \subseteq \mathbf{PSPACE}$

So, $pr\mathbf{BQP}^{\mathbf{PSPACE}} \subseteq \mathbf{PSPACE} \subseteq pr\mathbf{P}^{\mathbf{PSPACE}} \subseteq pr\mathbf{BQP}^{\mathbf{PSPACE}}$ ∎

Now we will look at the second condition. Actually there are functions f defined on a very small subset of the inputs for which the gap between classical and quantum query complexity is exponential [12]. However, the fact may still be true for functions defined on most of the inputs.

Using the same method as in theorem **2.3.16** we can prove:

Theorem 3.4.2 *There is a constant c such that for almost all f defined on at least a c-fraction of the inputs, $\overline{deg}(f) = O(n)$.*

There are two ways we though of in order to prove a polynomial gap:

1. Use the same approach as for total functions directly.

2. Show that for any almost-total function f, there is a total function g that approximates f on almost all the inputs and $\overline{deg}(g)$ =poly$(\overline{deg}(f))$

Now we'll give some intuition on why the first method fails. We can still define the *block sensitivity* and *certificate complexity* in the same way, by restricting all inputs to be within the domain of the function.

Theorems **2.3.13** and **3.1.1** would still hold. However, lemma **2.3.4** used in theorem **2.3.9** fails: when we flip any variable in x^B we have no guarantee we'll remain within the domain of the function. So far all attempts to overcome this problem have been unsuccessful.

However, we still believe that:

Conjecture: *There is a constant c such that for all functions defined on at least a c-fraction of the inputs,* $D(f) = poly(Q_2(f))$

A weaker (and possibly easier to prove) conjecture would be:

Conjecture: *There is a constant c such that for all functions defined on at least a c-fraction of the inputs,* $D^x(f) = poly(Q_2^x(f))$ *for almost all x's, where the index x represents the query complexity on input x.*

Chapter 4

Conclusions and Future Work

The polynomial method was the first framework in which quantum and classical query complexities can be studied on a unified basis. Theorem **3.2.2** provides a lower bound for the number of queries. However, we do not take at all into account the fact that the unitary transformations applied between consecutive calls to the oracle have to be efficiently computable. Hence, we expect the gap between $D(f)$ and $Q_2(f)$ to be much smaller. The largest gap known so far is quadratic (for the OR function), and it wouldn't be surprising if this were the largest separation. Another open question is whether $D(f) = O(deg(f))$. This happens to be true for all the functions studied so far

The relation $D(f) = O(deg(f)^4)$ can be improved on certain assumptions:

Theorem 4.0.3 *If all the monomials of maximum degree intersect, then* $D(f) = O(deg(f)^2)$

Proof: If we query all the variables from a monomial of maximum degree we reduce the degree of the polynomial by at least one (because we queried at least one variable from each monomial of maximum degree). By repeating this at most $deg(f)$ times, we can find the value of the function. ∎

A new approach to query complexity has been proposed by Howard Barnum [personal communication]. It involves expressing a quantum decision tree in terms of positive semidefinite matrices. Using the dual of the system, one could bound the required number of queries. Although this method is not equivalent to the polynomial approach, so far no new bounds have been proven.

Bibliography

[1] Beals, R., Buhrman, H., Cleve, R., Mosca, M., *Quantum Lower Bounds by Polynomials*, quant-ph/9802049.

[2] Beigel, R., *The polynomial method in circuit compexity*, Proceedings of the 8^{th} IEEE Structure in Complexity Theory Conference, pp. 82-95, 1993.

[3] Bennett, C., Bernstein, E., Brassard, G., Vazirani, U., *Strengths and weaknesses of quantum computing*, SIAM Journal on Computing, 26(5):1524-1540, 1997.

[4] Bernstein, E., Vazirani, U., *Quantum complexity theory*, SIAM Journal on Computing, 26(5):1411-1473, 1997.

[5] Fortnow, L., Rogers, J., *Complexity Limitations on Quantum Computing*, quant-ph/9811023.

[6] Grover, L., *A fast quantum mechanical algorithm for database search*, Proceedings of the 28^{th} ACM Symposium on the Theory of Computing, pp. 212–219, ACM, New York, 1996.

[7] Lloyd, S., Lecture notes, Quantum Computation, Fall 1999.

[8] Nisan, N., *CREW PRAMs and decision trees*, SIAM Journal of Computing, 20(6):999-1007, 1991.

[9] Nisan, N., Szegedy, M., *On the degree of Boolean functions as real polynomials*, Computational Complexity, 4(4):301-313, 1994.

[10] Rivlin, T.J., Cheney, E.W., *A comparison of uniform approximations on an interval and a finite subset thereof*, SIAM Journal on Numerical Analysis, 3(2):311-320, 1966.

[11] Shor, P., *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing, 26(5):1484-1509, 1997.

[12] Simon, D., *On the power of quantum computation*, SIAM Journal of Computing, 26(5):1474-1483, 1997.