Simple Quantum Algorithms

Lecture Notes for Quantum Computing

Lecturer: Dorit Aharonov
Scribe: Nahliel Wygoda and Elad Noor

March 18-23, 2001

Abstract

In this week we will define the computation class BQP, which is the class considered to contain the functions which can be computed efficiently on a quantum computer. We first show that this class contains all functions which can be computed efficiently on a classical probabilistic computer. WE then start our journey towards understanding the computational power of quantum computers. WE begin with the early quantum algorithms, Deutsch-Josza and Simon.

1 BPP vs BQP

1.1 Classical complexity: BPP

We define the complexity class BPP, which is considered to be the class of functions that can be computed efficiently classically.

BPP stands for Bounded Probability Polynomial.

Definition 1 BPP:

- *One definition of BPP is all the boolean functions* $f(0,1)^* \rightarrow (0,1)$ *that can be computed by a statistical Turing machine running in time that is polynomial to the size of the input, and for which the output is correct with probability* $\geq \frac{2}{3}$.

- *an equivalent definition for BPP is all the boolean functions such that f_n can be computed in a circuit of size polynomial in n which gets random inputs and returns a correct output with probability $\geq \frac{2}{3}$. WE require in addition that there exists a deterministic Turing machine T which, if given n, finds the ciruit C_n (which computes f_n) in time polynomial to n.*

The requirement that the circuits can be computed by a Turing machine is called the *uniformity* restriction, and the model of circuits which can be computed by a polynomial Turing machine is called the *uniform* circuit model. The model without this restriction is called the non uniform model. The reason why we work with the uniform model is because the non-uniform model is much too strong; An answer to an undecidable question for an input n can be hardwired into the circuit C_n! To make the model reasonable and realizable, we require that such peculiarities do not occur, i.e. that we can easily construct the design of the circuit given the input size. This is what makes the two definitions, the one using Turing machines and the one using uniform circuits, equivalent. For more about these classical complexity issues consider Papadimitriou's book, computational complexity.

1.2 Quantum complexity: BQP

Definition 2 BQP (Bounded Quantum Polynomial) *BQP is defined just like the BPP second definition, except that the circuits can use quantum gates (which are unitary matrices) instead of the and and or gates that we're used to.*

2 BQP contains BPP

Our first goal is to show that $P \subseteq BQP$, and thus that we haven't lost computation power in the quantum model. First, we notice that the quantum model is reversible (its gates are unitary matrices, which are reversible). The regular model, on the other hand, isn't reversible at first glance: for example, an *and* gate, which looks like $a, b \rightarrow a \cap b$ isn't reversible, since by knowing the output we can't trace back what was the input. so there seem to be more restrictions on the quantum model, and that might suggest that it is weaker.

Remember that the definition of a reversible computation is a one to one function $f : \{0,1\}^d \rightarrow \{0,1\}^d$. How to compute functions in P using only reversible gates? For example, we can make an *and* gate reversible in the following manner (called a Toffoli gate), which is a function $f(a,b,c) \rightarrow (a,b,c \oplus a \cdot b)$ (c is used as an additional parameter which enables us to do the computation while keeping a and b as they are, so that the process can then be reversed):

$$\begin{array}{c} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 01 \\ & & & & & & 10 \end{pmatrix}$$

(the numbers on the left of the array represent the members of the basis for $\{0,1\}^3$, since this is the relevant space here. in every three numbers, the first one is the value of a, the second of b, and the third of c). So *and* is reversible. It is also obvious that *not* is reversible, and therefore every gate is reversible. We showed that P can be made reversible without changing the number of gates (and still using gates on a constant number of bits) and thus $P = revP$.

To show that $revP$ is in BQP we simply notice the following fact. If a certain gate computes the one to one function $f : \{0,1\}^d \rightarrow \{0,1\}^d$, we can write the way it operates on the input bits as a $2^d \times 2^d$ matrix U, in which $|i\rangle \rightarrow |f(i)\rangle$. Then we actually get a permutation on the basis of $\{0,1\}^d$, and therefore $U^\dagger$ (the transposed complex conjugate matrix) is the inverse permutation, such that $U \cdot U^\dagger = I$, and therefore U is unitary. Hence reversible computation is a special case of quantum computation, and indeed $P = revP \subseteq BQP$.

What we said until now was true for deterministic circuits. What about probabilistic circuits? We can regard those as circuits which get a certain number of probabilistic inputs, and from then on work in a totally deterministic way. But since we already saw how to turn deterministic circuits to reversible ones, thus showing that $P \subseteq BQP$ we only need to see how to realize probabilistic inputs in the quantum model. Those inputs can be realized using **Hadamard Gates**. Given $|0\rangle$ as an input, a single Hadamard gate outputs a vector $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ which, when measured, results in $|0\rangle$ or

$|1\rangle$ with equal probabilities. So if we need s random inputs, we simply use s Hadamard gates and then measure their outputs, and so we can also represent probabilistic circuits in the quantum model. It follows that $BPP \subseteq BQP$.

3 "Clean" computation of function f

Let f be a function $f : |x\rangle|0...0\rangle \to |f(x)\rangle|junk\rangle$, where x is a vector containing the relevant inputs, and $|0...0\rangle$ are bits used for the computation (working space.) We would now like to compute f in a clean manner, that is, to perform the following computation:

Definition 3 *$f : \{0,1\}^n \longmapsto \{0,1\}^m$ is said to be computed in a "clean" way by the circuit Q if Q takes $|x\rangle|c\rangle|0...0\rangle \longmapsto |x\rangle|c \oplus f(x)\rangle|0...0\rangle$, where $x \in \{0,1\}^n, c \in \{0,1\}^m$, the xor of c and $f(x)$ is performed bit by bit, and we can use any number of zeroes as working space.*

Convention: If the state of the bits, both as input and as output, is $|0\rangle$, we can omit it from the expression. We therefore say that Q takes $|x\rangle|c\rangle \longmapsto |x\rangle|c \oplus f(x)\rangle$, using working space.

Theorem 1 *Given a reversible circuit M that computes f with k gates, then there's a circuit that computes f in a "clean" way with 2k+1 gates.*

Proof: The way to do that is:

$$|x\rangle|c\rangle|0...0\rangle \to^{M} |f(x)\rangle|c\rangle|junk\rangle \to^{Cnot}$$

$$|\oplus f(x)\rangle|c \oplus f(x)\rangle|junk\rangle \to^{M^{-1}} |x\rangle|c \oplus f(x)\rangle|0...0\rangle$$

Where M^{-1} is applied by applying the inverse of the gates in the reverse order which they appear in M. The CNOT is applied from the bit which contains $f(x)$ to the bit which contains c. The computations of M, M^{-1} are applied on all qubits except that qubit which originally contained c; this qubit is only touched during the CNOT.

4 Quantum Exponential Parallelism

Given a vector $|0...0\rangle$ of n bits, we can apply a Hadamard gate on each of the n bits. Since the Hadamard gate takes $|0\rangle$ to an equal superposition of

$|0\rangle$ and $|1\rangle$, when we apply the Hadamard on each qubit we get an equal superposition of all 2^n classical configurations, since this is the number of possible combinations for n bits. By applying n Hadamard gates in parallel, we get an exponential number of possible states. the formal notation for this process of using n Hadamard gates is H_{2^n}, and it means:

$|0...0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes ... = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$ where each of the 2^n vectors represents one of the possible states of 2^n bits.

Now we invoke linearity of quantum mechanics. Suppose $|i\rangle|c\rangle \longmapsto |i\rangle|c \oplus f(i)\rangle$ can be computed by a polynomial size quantum circuit. Applying this circuit on this state we get by linearity $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle|f(i)\rangle$. This is quite an impressive result, because we have computed an exponential number of outputs (2^n) in a polynomial time. This phenomena is referred to as *exponential parallelism.*

However, this computation does not really give us the computation of f of each of the inputs in polynomial time in a way which we can read. In order to read the results of all the possible states, we need to compute it 2^n times, because after every measurement we do, all the states collapse to what we have measured, i.e. to just one of the strings $i, f(i)$.

We will however see how to use exponential parallelism together with another property of quantum mechanics, namely *interference*, to get extra computational power.

5 The first Quantum Algorithm: [Deutsch-Josza 92]

Focus on the Oracle, or black box Model. Instead of the classical oracle $x \rightarrow f(x)$, we use the quantum oracle $|x, c\rangle \rightarrow |x, c \oplus f(x)\rangle$. This is a single call for the oracle. Notice that one call to the oracle is a generalization of CNOT. If f is a function to one bit, it applies a NOT on the second bit conditioned that the first input bits satisfy some restriction, namely that $f(x) = 1$. In particular, CNOT can be achieved if we make $\forall x : f(x) = x$, and x of one bit.

Deutsch-Jozsa Problem: We shall limit the oracle to only Balanced or Constant function (A Balanced function is one that returns 1 for half the inputs, and 0 for the other half). Determining whether the oracle is Balanced or Constant is the problem, and the goal is to use as few queries to the oracle

as possible.
Promise: The fact that the oracle computes a function which is either balanced or constant is called a Promise. We are promised that the oracle's function belongs to some restricted regime; We do not care what our algorithm will do in other cases.

Assume f is the oracle, then $f : \{0,1\}^n \to \{0,1\}$. A classical algorithm will need $2^{n-1}+1$ calls to the oracle, to solve the problem with no error. This is because in the worst case, we will read 2^{n-1} 0s (or 1s) and only another call will tell us if all the others are also 0s (or 1s).

However, there is a quantum algorithm that can resolve the Promise Problem with only 2 calls to the oracle:

Deutsch-Josza Algorithm:

- $Init : |0..0\rangle \in \{0,1\}^n$
- $H_{2^n} : \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle|0\rangle$
- $U_f : \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle|f(i)\rangle$
- $Z(phaseflip) : \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle(-1)^{f(i)}|f(i)\rangle$
- $U_f : \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} (-1)^{f(i)}|i\rangle|0\rangle$ (*)
- $H_{2^n} : |\alpha\rangle$
- Measure the result. If the output is a string of 0's, output constant. If the output is anything but a string of 0's, output balanced.

(*) Notice that applying U_f on $|i\rangle|f(i)\rangle$ will return $|i\rangle|f(i) \oplus f(i)\rangle = |i\rangle|0\rangle$.

If f is constant, then before applying the last H_{2^n} we will have $\frac{1}{\sqrt{2^n}} \pm \sum |i\rangle$. And since H is the inverse matrix of itself, applying it will yield $\pm|0..0\rangle$ (exactly the opposite of the first step in the algorithm). This means that in this case the output will be constant with probability 1.

If f is balanced then $(\sum_i (-1)^{f(i)}|i\rangle, \sum_j |j\rangle) = \sum_i (-1)^{f(i)} = 0$, because for half of the instances, $(-1)^{f(i)} = 1$ and for the other half $(-1)^{f(i)} = (-1)$. H is unitary therefore conserves the inner product, so if we apply H_{2^n} on both vectors, we will get $0 = (|\alpha\rangle, |0..0\rangle)$. This means that the result is perpendicular to $|0..0\rangle$ so if we measure $|\alpha\rangle$ according to the standard basis, we will get $|0..0\rangle$ with zero probability, so for sure we will output balanced.

There is another version of DJ, with only one call to the oracle. Instead of applying f when the last qubit is initialized as $|0\rangle$, we initialize it to $|\beta\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Notice that $NOT|\beta\rangle = \frac{1}{\sqrt{2}}(|0 \oplus 1\rangle - |1 \oplus 1\rangle) = \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -|\beta\rangle$.

Then if $f(i) = 1$ then $|i\rangle|\beta\rangle \rightarrow^{U_f} |i\rangle NOT|\beta\rangle = -|i\rangle|\beta\rangle$ and if $f(i) = 0$ then $|i\rangle|\beta\rangle \rightarrow^{U_f} |i\rangle|\beta\rangle = |i\rangle|\beta\rangle$

So altogether: $\sum_i |i\rangle|\beta\rangle \rightarrow^{U_f} \sum_i (-1)^{f(i)}|i\rangle|\beta\rangle$. The state of the first register is exactly the state we need two steps before the end of the algorithm. Hence, if we modify the algorithm to first apply a NOT gate, and then Hadamard on the last qubit, to get $|\beta\rangle$ on the last qubit, we can replace the three steps $U_f, CNOT, U_f$ in the algorithm by only one call U_f, and proceed to the last few steps of the algorithm. This is an algorithm which uses only one call, and gives the result with no error.

Note that this seems like a very large gap between classical and quantum: exponential number of queries vs. 1. Note, however, that if we allow errors, classical computation can also do with a constant number of queries. In the next algorithm, we will see a gap which is preserved even when we allow errors. But let us first understand why the quantum algorithm worked so well.

6 Fourier Transform (FT)

H_{2^n} is a special case of a transformation called Fourier Transform. Let us study it a little more. Each vector $c = (c_0, c_1, .., c_{d-1}) \in C^d$ can be seen as a function $\{0, 1, .., d-1\} \rightarrow C$, such that $\forall i \in \{0, 1, .., d-1\} : f_c(i) = c_i$

According to this observation, the standard basis $|0..00\rangle, |0..01\rangle, .., |1..11\rangle$ are 2^n delta functions $\{0,1\}^n \rightarrow C$. This is a base for all the functions $\{0,1\}^n \rightarrow C$.

A very important different basis for these functions is the Fourier Vectors Basis: for each $s \in \{0,1\}^n$ we define the function $\chi_s(x) = \frac{(-1)^{s \cdot x}}{\sqrt{2^n}}$, where $s \cdot x = \sum_{l=1}^{n} s_l x_l$ mod 2. This gives us 2^n functions; again, functions can be viewed as vectors.

For example: if n=1 the Fourier Basis is

$$\chi_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \chi_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

for n=2:

$$\chi_{00} = \begin{pmatrix} +\frac{1}{2} \\ +\frac{1}{2} \\ +\frac{1}{2} \\ +\frac{1}{2} \end{pmatrix}, \chi_{01} = \begin{pmatrix} +\frac{1}{2} \\ -\frac{1}{2} \\ +\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}, \chi_{10} = \begin{pmatrix} +\frac{1}{2} \\ +\frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix}, \chi_{11} = \begin{pmatrix} +\frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \\ +\frac{1}{2} \end{pmatrix}$$

Exercise: show that all χ_s are orthonormal.

Let $f : \{0,1\}^n \to C$. Then $|f\rangle = \sum_i f(i)|i\rangle$ and $|f\rangle = \sum_s \hat{f}_s|\chi_s\rangle$. Notice that $(\chi_q, f) = \sum_s \hat{f}_s(\chi_q, \chi_s) = \hat{f}_q$, so we can write: $|f\rangle = \sum_s (\chi_s, f)|\chi_s\rangle$.

We will call $|\hat{f}\rangle = \sum \hat{f}_s|s\rangle$ the Fourier Transform of $|f\rangle$. We shall prove that applying H_{2^n} upon $|f\rangle$ gives the Fourier Transform.

if $n = 1$ then:

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$|0\rangle \to_H \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$|1\rangle \to_H \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

or in other words:

$$H|x\rangle = \frac{1}{\sqrt{2}} \sum (-1)^{x \cdot y}|y\rangle$$

for $n > 1$:

$$H \otimes .. \otimes H|x_1..x_n\rangle = \frac{1}{\sqrt{2^n}} \sum (-1)^{x_1 y_1}|y_1\rangle..(-1)^{x_n y_n}|y_n\rangle =$$

$$\frac{1}{\sqrt{2^n}} \sum (-1)^{x_1 y_1 + x_2 y_2 + .. + x_n y_n}|y_1..y_n\rangle = \frac{1}{\sqrt{2^n}} \sum (-1)^{x \cdot y}|y_1..y_n\rangle$$

in vector syntax:

$$H_{2^n}|\bar{x}\rangle = \frac{1}{\sqrt{2^n}} \sum (-1)^{\bar{x} \cdot \bar{y}}|\bar{y}\rangle$$

so if $|f\rangle = \sum_s f_s|s\rangle$:

$$H_{2^n}|f\rangle = \frac{1}{\sqrt{2^n}} \sum_{s,r} (-1)^{r \cdot s} f_s|r\rangle = \sum_r \left(\sum_s \frac{(-1)^{r \cdot s} f_s}{\sqrt{2^n}} \right) |r\rangle =$$

$$\sum_r \left(\sum_s \chi_r(s) f_s \right) |r\rangle = \sum_r (\chi_r, f)|r\rangle = |\hat{f}\rangle$$

Therefore, H_{2^n} is a Fourier Transform. Observe that each argument in the Fourier Transform of $|f\rangle$ is the sum of contributions of f for each χ_r. Actually, this is a calculation of an exponential number of values in a constant time. This is where the advantage of quantum parallel computation comes to light.

7 Bernstein Vazirani Algorithm [93]

Let f_a be a linear oracle, that is:

$$f_a(x) = a \cdot x = \sum_{l=1}^{n} a_l x_l$$

In the classical sense, it is necessary to call the oracle n times in order to find a. But, quantum computation can reduce this to only one call. We apply the same algorithm exactly as in DJ problem.

$$\sum |x\rangle \rightarrow^{U_f} \sum (-1)^{f(x)}|x\rangle = \sum (-1)^{a\cdot x}|x\rangle = |\hat{a}\rangle \rightarrow^{H_{2^n}} |a\rangle$$

and so applying the H_{2^n} we will have a written on the qubits and we will simply measure a. This algorithm reduces linear time complexity to constant, using quantum computation. Note that this gap is still correct when we allow errors to the classical computation, since we will need $O(n)$ queries even if we allow errors.

Bernetein and Vazirani showed an algorithm that reduces super-polynomial time complexity ($O(n^{log(n)})$) to polynomial quantum time, using a recursion on the above idea. This was the first result that showed a superpolynomial gap (in the oracle model) between quantum and classical, even if we allow the classical computer to err; Simon's algorithm is another example like this, which cam somewhat later. We will present it due to its simplicity.

8 Simon's algorithm [94]

Let f be an oracle with a period a, that is:

$$\forall x : f(x) = f(x \oplus a)$$

$$\forall x, y : x \neq y, x \neq a \oplus y \Rightarrow f(x) \neq f(y)$$

How can we find a?

Classically, after K questions (call for f on $x_1,..,x_K$) if $f(x_i) \neq f(x_j)$ for each $i \neq j$ we can discard only $\left(\begin{array}{c} K \\ 2 \end{array}\right)$ possible a's. Therefore, we still have to check $(2^{n-1}-1) - \left(\begin{array}{c} K \\ 2 \end{array}\right)$. So $K = \Omega(2^{n/2})$ calls have to be made to reduce the number of possible a's to 1. (This is not a rigorous argument; we haven't shown that all a's that were not discarded are all equally likely. we will leave this as an exercise to give a rigorous proof.)

A quantum algorithm can perform this task in a polynomial number of queries.

- Init : $|0..0\rangle|0\rangle$
- H_{2^n} : $\sum_i |i\rangle|0\rangle$
- U_f : $\sum_x |x\rangle|f(x)\rangle = (|x\rangle + |x \oplus a\rangle)f(x) + (|x'\rangle + |x' \oplus a\rangle)f(x') + ... = \sum_x (|x\rangle + |x \oplus a\rangle) \otimes |f(x)\rangle$
- measure $f(x)$ (which are the last n-1 qubits): $\frac{|x\rangle+|x\oplus a\rangle}{\sqrt{2}} \otimes |f(x)\rangle$
- H_{2^n} : $\sum_y \left((-1)^{x\cdot y} + (-1)^{(x\oplus a)\cdot y}\right)|y\rangle \otimes |f(x)\rangle = \sum_y (-1)^{x\cdot y}\left(1 + (-1)^{a\cdot y}\right)|y\rangle \otimes |f(x)\rangle$
- measure the first n qubits ($|y\rangle$)

Notice that $(1 + (-1)^{a\cdot y})$ is 0 if $a \cdot y = 1$, and 1 if $a \cdot y = 0 mod 2$. Therefore, the last measurement can only give a state $|y\rangle$ that satisfies $y \cdot a = y_1 a_1 + y_2 a_2 + .. + y_n a_n = 0$. Moreover, all such y's have the same amplitude, so we get a random such y.

We can view this as if we are working over the vector space of dimension n, over the field F_2, i.e. each coordinate can be either 0 or 1, and we have n such coordinates. We have now a process to pick a uniformly random vector in the subspace orthogonal to a in $\{0,1\}^n$. Let us call this subspace $a^\perp$.

If we run the algorithm m times, then we have m vectors $y^1,..,y^m$ (or a set of m equations). If $y^1,..,y^m$ span $a^\perp$ then we can easily discover a using Gauss elimination. Hence, if the y''s span the $n-1$ dimensional subspace which is orthogonal to a, we are done. What is the probability of increasing the dimensionality of the subspace spanned by the y's if it is not yet $n-1$? We claim that it is at least 1/2, as long as we haven't spanned the subspace orthogonal to a yet; This is because if $span(y_1, ...y_k)$ has dimension $l < n-1$, then the number of vectors in it is 2^l which is at most half the number of vectors in $a^\perp$. Since we toss a random vector in $a^\perp$, we get a vector not spanned by $span(y_1, ...y_k)$ with probability at least half. This increases the dimension of the subspace spanned by the y's by 1. Suppose we perform cn such trials, for some $c > 2$. Suppose these vectors did not span the $n-1$ subspace. This means that we have succeeded in less than $n-1$ of these cn trials. But the expected number of successes is half the number of trials, i.e. it is larger than n; Using Chernoff bound, the probability that the actual number of successes will be so much smaller than the expectation drops exponentially with n, and hance the probability that the vectors have not spanned $a^\perp$ is exponentially small. So we can achieve any desired probability to span the subspace orthogonal to in a linear number of repetitions. For Chernoff bound, see any probability book, eg. Feller, or an accessible presentation in Raghavan's randomized algorithms.