CS 810: Complexity Theory 3/19/2007

Lecture 21: Quantum Effects

Instructor: Dieter van Melkebeek Scribe: Seeun William Umboh

Today we will take a quick look at the quantum computing model. We will define the quantum model of computing as a variant of the probabilistic model of computing, present a common technique in the design of quantum algorithms, and give currently known upper bounds on the power of quantum computation.

1 Motivation

Quantum computation presents a challenge to the *Strong Church-Turing Thesis*, which says that every physically realizable computing device can be efficiently simulated by the Turing machine model presented in the first lecture. It is not clear at this point that problems solvable in polynomial time on quantum computers can be solved in polynomial time on deterministic Turing machines or randomized machines. One reason is that we know how to factor efficiently on quantum machines but not on classical machines. However, the consensus in the community is that quantum computers cannot solve NP-complete problems and some even think that there is an efficient classical factoring algorithm. Another caveat is that it is still uncertain whether quantum computers are actually physically realizable.

2 Idea

We would like to exploit quantum effects to solve computational problems more efficiently, in terms of time. The key idea for search problems is that we would like to use quantum interference in such a way that the "good" solutions interfere constructively and the "bad" solutions interfere destructively. So, in the end, only the "good" solutions remain.

3 Turing Machine Models

3.1 Probabilistic

It is useful to relate the quantum model to the probabilistic model by viewing the probabilistic model from the perspective of Markov chains. We give an alternate definition of the probabilistic model, and then define the quantum model.

Definition 1 (State). *We represent the state of the probabilistic machine as a probability distribution over configurations using the* "ket" *notation:*

$$\sum_c p_c|c\rangle$$

where $\sum_c p_c = 1, p_c \in [0,1]$. p_c *is the probability of being in configuration* c *and* $|c\rangle$ *is the column vector with zeros everywhere except a 1 at the position representing the configuration* c. c *runs over all configurations.*

Definition 2 (Computation). *The computation on a probabilistic machine consists of 2 phases:*

- *A sequence of local(acting only on a few bits, such as the tapehead, state, etc) linear* [1] *transformations that transform a probability distribution into a probability distribution,* i.e. *stochastic matrices, induced by the transition function* δ [2]*:*

$$\delta : Q \times \Gamma^k \times Q \times \Gamma^k \times \{L, R\} \to [0, 1]$$

- *Final observation of part of configuration* C*:*

$$\Pr[ANSWER\ is\ y] = \sum_c p_c$$

  where c *runs over all configurations giving the answer* y*.*

From the Markov chain view, at each point in time, the *state* of the machine is a superposition of all possible configurations, represented by the column vector $\sum_c p_c|c\rangle$. Note that the set of single-configuration vectors $\{|c\rangle\}_c$ then forms a basis for the linear space, over $\mathbb{R}$, of all state vectors. Then, at the end of the computation, we observe the output bit(s) and the probability of observing, say 1, is the probability that the machine is in some configuration giving the output 1. This is so far merely a rephrasing of the probabilistic model we presented in an earlier lecture. Now we move on to the quantum model.

3.2 Quantum

Definition 3 (State). *The state of a quantum machine is defined as a linear superposition of all possible configurations* c

$$\sum_c \alpha_c|c\rangle$$

where $\alpha_c \in \mathbb{C}, \sum_c |\alpha_c|^2 = 1$*.*

Definition 4 (Computation). *The computation on a quantum machine consists of 2 phases:*

- *A sequence of local(acting only on a few bits, such as the tapehead, state, etc) linear transformations that transform a vector* v *with* $\|v\|_2 = 1$ *into a vector* v' *with* $\|v'\|_2 = 1$*,* i.e. *unitary matrices, induced by the transition function* δ*:*

$$\delta : Q \times \Gamma^k \times Q \times \Gamma^k \times \{L, R\} \to \mathbb{C}$$

- *Final observation of part of configuration* C*:*

$$\Pr[ANSWER\ is\ y] = \sum_c |p_c|^2$$

  where c *runs over all configurations giving the answer* y*.*

[1] We would like the transformation to depend on the configuration not on the overall distribution.

[2] We usually require the transition probabilities to be efficiently approximable, for example the rationals, to avoid dealing with machines with say, the halting sequence as a transition probability.

Here, we view the state of the quantum machine as a wave function. The coefficients, called *amplitudes*, are vectors in the imaginary plane with length at most 1. The fact that the amplitudes can be negative, allowing destructive interference, is what underlies a lot of the power in quantum computing. Also, note that the probability of being in a particular configuration is now the square of the absolute value of the amplitude associated with it.

One issue we need to consider is that the condition that the matrices induced by δ be stochastic or unitary imposes restrictions on the set of allowable transition functions. In the probabilistic setting though, we only need that the transition function be "stochastic". That is, for a fixed configuration, the sum of the probabilities of the configurations it can move to in one step is 1. In particular, we can show that setting the transition probability to be either 1, 0 or 1/2 is enough. In the quantum setting, the natural thing to do is to have δ be "unitary", in the appropriate sense. However, it turns out that δ has to satisfy some *orthogonality conditions* as well. These conditions are unnatural and so instead of considering quantum machines as Turing machines during algorithm design, we prefer to think in terms of circuits.

4 Circuit Models

We now define circuit models for both probabilistic and quantum computations. In order to satisfy the above conditions, we would like the gates to act on a finite number of bits, in particular it is sufficient that they act on at most 3 bits, and to induce stochastic(unitary) matrices. To this end, we define the following notions:

- The *register* is the analogous notion of Turing machine configuration in the circuit model. The contents of the register reflect the results of the computation so far. We retain the notation $|x_o \dots x_m\rangle$ to denote the contents of the register.

- The *state* of a register is represented as a probability distribution(linear superposition) over all possible contents of the register. Again, we note that the set of vectors $\{|c\rangle\}c$ form a basis for the linear space, over the reals(complex numbers), of all state vectors.

- An *operation* G on an m-bit register is specified by a linear stochastic(unitary) transformation $F : \mathbb{R}^{2^3} \to \mathbb{R}^{2^3}(F : \mathbb{C}^{2^3} \to \mathbb{C}^{2^3})$ acting on 3 distinct bits with indices $j, k, l \in \{1, \dots, m\}$, leaving others unmodified, such that for every $x_1, \dots, x_m \in \{0, 1\}$, applying G on $|x_1, \dots, x_m\rangle$ gives $|y_1, \dots, y_m\rangle$ where $|y_j y_k y_l\rangle = F(|x_j x_k x_l\rangle)$.

- A computation consists of a sequence of operations followed by a final observation of the register.

Note that the operations can be represented by stochastic(unitary) matrices, hence the models defined satisfy that condition.

For uniformity, we can simply require that a single Turing machine can compute the operation at step i, for all i. In addition, for the $T(n)$-time bounded versions of these models, we require that the Turing machine take time at most $T(i)$ to compute the operation at step i.

4.1 Probabilistic

For the probabilistic model, it is sufficient to have classical AND, OR and NOT gates to simulate deterministic computation, and a *coin flip* gate to allow access to a fair coin.

4.1.1 Deterministic Gates

The AND, OR and NOT operations are defined by the transformations F, G, H with $F|xyz\rangle = |xy(x \wedge y)\rangle, G|xyz\rangle = |xy(x \vee y)\rangle, H|x\rangle = |\bar{x}\rangle$, respectively. The matrix for AND is:

$$F = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Remember that the input bits are unaffected by the gate, and since we multiply F from the right, $F_{i,j}$ denotes the probability of the register having contents $|i_2 i_1 i_0\rangle$ if it started in configuration $|j_2 j_1 j_0\rangle$ where l_k is the kth bit of l, from the right.

4.1.2 Coin Flip Gate

The coin flip gate acts only on one bit and its output is 1 or 0 equiprobably:

$$C = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

4.2 Quantum

4.2.1 Deterministic Gates

In the quantum setting, we cannot simply use the matrices for the AND, OR and NOT gates as above, since they do not preserve the 2-norm. We note first that the 2-norm preserving condition for a matrix A is equivalent to it satisfying $A^*A = I$ since $||Ax||_2^2 = x^*A^*Ax = x^*x$ iff $A^*A = I$.[3] Hence, the gate matrices have to be at least invertible. However, the AND and OR gates are not invertible, as more than one input can map to 0 under AND, for example.

For the deterministic gates to be reversible, they need to induce permutation matrices as determinism requires the matrix to have in every column a 1 in exactly 1 entry, and zero everywhere else, and reversibility requires the matrix to have in every row a 1 in exactly 1 entry, and zero everywhere else. We can get around this by introducing additional ancilla bits. Given a boolean function $\varphi : \{0,1\}^k \to \{0,1\}$, we transform it to the reversible version $g : \{0,1\}^{k+1} \to \{0,1\}^{k+1}$, defined by $(x, b) \to (x, b \oplus \varphi(x))$.

So now we can apply this transformation to all the gates of any deterministic classical circuit to obtain a circuit usable by our quantum machine. Given a circuit C computing $f : \{0,1\}^n \to \{0,1\}$, the resulting function after transforming C's gates is $f' : \{0,1\}^{n+m+1} \to \{0,1\}^{n+m+1}$, defined by $(x, 0^m, 0) \to (x, \mathit{garbage}, f(x))$, where the number of ancilla bits m is on the order of the number of gates in C.

Now, we need to have the m ancilla bits set to zero, as otherwise the interference patterns would be different. In particular, the garbage in the ancilla bits may cause certain computation paths to

[3]By A^*, we mean the complex conjugate of A.

not interfere when we would like them to. Since simply resetting the bits to zero is an irreversible operation, we do it by applying f' in reverse, and using an extra bit to preserve the result computed by f':

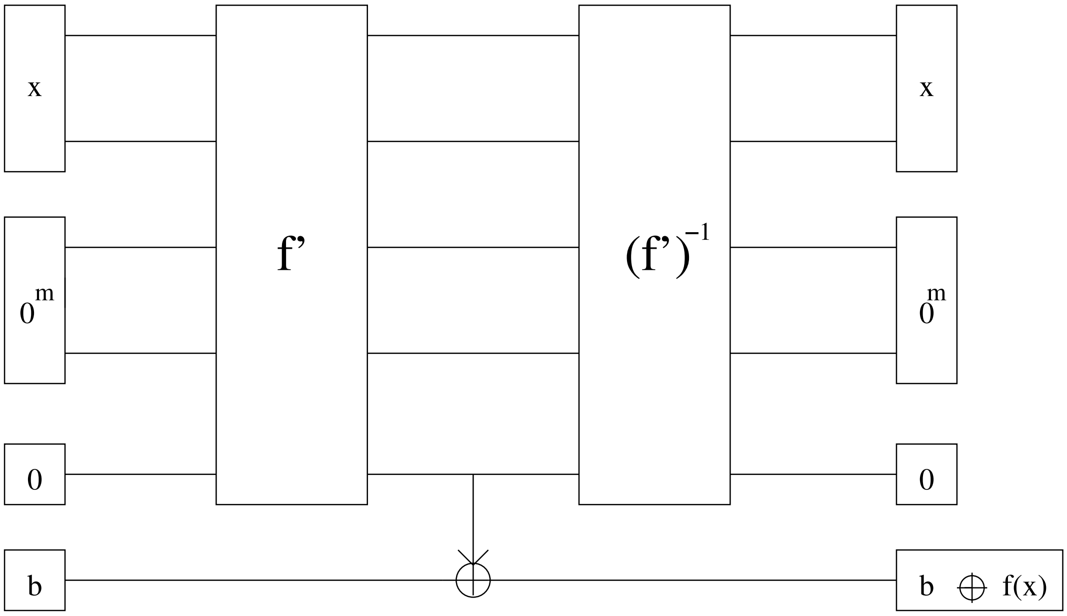

Figure 1: The schematic for the reversible simulation of f.

The complexity of this reversible simulation, denoted as $\tilde{f}$, is a constant factor greater than the complexity for f since we only need to apply the transformation to every gate, and then run the transformed circuit twice.

Note that this also gives us P $\subseteq$ BQP, where BQP is the class of decision problems solvable in polynomial on quantum Turing machines. Since the output of a quantum Turing machine is a random variable, we say that it decides a language L if the probability of deciding correctly the membership of x in L is at least $2/3$.

4.2.2 Hadamard Gate

The quantum analog of the classical coin flip gate is called the *Hadamard gate*. The matrix for this gate is

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

Exercise 1. *Verify that the Hadamard matrix is unitary and orthogonal.*

The effect of applying this gate to a single bit is:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

So, if we observe the bit after applying the Hadamard gate, we get 0 or 1 equiprobably, just as in a fair coin flip. However, we also have that $H^2 = I$. So, applying the Hadamard gate twice consecutively in sequence, will give us the original bit, which is not the same result as if we did the same with the coin flip gate. One way to see this is in terms of interference amongst the computation paths.

Hence, we can simulate randomized computation by applying the Hadamard gate to some 0s and then observing those bits. Afterward we can proceed deterministically and use those bits as random bits. Thus, BPP $\subseteq$ BQP.

5 Illustration of Quantum Power

Even though it is widely believed in the community that interference is not enough to solve NP-complete problems, we have managed to use interference to efficiently solve certain problems for which we do not have efficient classical algorithms. An example is factoring, and *Simon's Problem.* While this problem is unnatural, the technique used in solving it underlies many quantum algorithms such as factoring.

Definition 5 (Simon's Problem). *Given a poly-time length-preserving(it maps strings of length n to strings of length n) function $f : \{0,1\}^* \to \{0,1\}^*$ with the promise that*

$$(\forall n)(\exists 0 \neq s_n \in \{0,1\}^n)(\forall x, y \in \{0,1\}^n)[f(x) = f(y) \iff x + y = s_n],$$

where addition is defined as bitwise XOR, find s_n on input 0^n.

In other words, for strings of length n, the function f is exactly 2-to-1 and the 2 strings that map to a certain string differ by a *shift* s_n, and we would like to find s_n. Also note that this is a promise problem.

This problem is in the second-level of the Polynomial Hierarchy since we can guess a shift and then verify for all pairs (x, y) of n-length strings that $f(x) = f(y) \iff x+y = s_n$. f is a poly-time function and we can easily bitwise-XOR x with y and see if we get s_n, so the verification can be done efficiently.

For classical machines, we do not know how to do better than exponential time. Even using randomness, the best algorithm we have is to guess x and y and see if $f(x) = f(y)$. If we get a collision, then we can easily determine s_n. However, the expected number of trials to get a collision is exponential.

On quantum machines though, this problem is solvable in polynomial time. We first start off in the state $|0^n\rangle|0^n\rangle$, where we would like to use the first n bits to encode all possible inputs for f. We achieve this by applying the Hadamard gate to each of the n bits. We denote this operation as $H^{\otimes^n}$. [4] So, now we are in the state $\frac{1}{\sqrt{N}} \sum_x |x\rangle|0^n\rangle$, where $N = 2^n$. Then, we apply $\tilde{f}$ with the first n bits as input bits and the last n bits as the output bits, and get the state $\frac{1}{\sqrt{N}} \sum_x |x\rangle|f(x)\rangle$. Finally, we apply $H^{\otimes^n}$ to the first n bits again and we leave it as an exercise to verify that we end up in $\frac{1}{N} \sum_x \sum_y (-1)^{x \cdot y} |y\rangle|f(x)\rangle$, where $(\cdot)$ denotes inner product. We can rewrite this as $\sum_{y,z} \alpha_{y,z}|y\rangle|z\rangle$, a linear superposition over all possible contents of both registers.

So, the probability of observing y in the first register is $\sum_z |\alpha_{y,z}|^2$. Now, if z is not in the range of f, $\alpha_{y,z}$ is 0. Otherwise, z is in the range and f maps exactly 2 strings to it, so $\alpha_{y,z} =$

[4] In linear algebra, this is the n-fold tensor product of the Hadamard gate matrix.

$\frac{1}{N}[(-1)^{x\cdot y} + (-1)^{(x+s_n)\cdot y}]$ where $z = f(x)$ for some x. Because we can take out the common factor $(-1)^{x\cdot y}$, and since if we sum over all n-length strings, we will count each $f(x)$ twice(as $f(x)$ and $f(x+s_n)$), the probability that a particular y is observed is $\frac{1}{2}\sum_x \frac{1}{N^2}|1 + (-1)^{s_n\cdot y}|^2$. As there are $2^n = N$ possible x's, this is then $\frac{1}{2N}$ if $s_n \cdot y = 0 \mod 2$, and 0 if $s_n \cdot y = 1 \mod 2$.

Therefore, the output of this computation y is chosen uniformly at random from those y such that $s_n \cdot y = 0 \mod 2$. We observe that $s_n \cdot y = 0 \mod 2$ is a linear combination of the components of y where the coefficients are the corresponding components of s_n. This is a linear equation of n variables over $\mathbf{GF}(2)$. So, we can run the above routine an order of n times and collect the output y_i from each run until the equations $s_n \cdot y_i$ form a homogeneous system of rank $n-1$. From elementary linear algebra, it follows that it has a unique non-trivial solution which is s_n.

We have already argued the efficiency of some of the components of the routine: $H^{\otimes^n}, \tilde{f}$. We also know how to solve homogeneous linear systems of equations efficiently. Lastly, we leave the fact that we only need on average $O(n)$ runs of the routine as an exercise:

Exercise 2. *Verify that with high probability, $O(n)$ runs suffice.*

6 Hidden Subgroup Generalization

Definition 6 (Hidden Subgroup Problem). *Given a group G and a poly-time $f : G \to \{0,1\}^*$, and the promise that there exists some subgroup H of G such that $f(x) = f(y)$ iff x and y belong to the same coset of H in G, find generators for H.*

There are several familiar instances of this problem:

Example:[Simon's Problem] The group for Simon's Problem is $G = (\{0,1\}^n, +)$, where + denotes bit-wise XOR. We have f as defined in Definition 5, and so the subgroup we are interested in is $H = \langle s_n \rangle = \{0^n, s_n\}$, since x and y belong to the same coset of H iff $x = 0^n + y = y$ or $x = s_n + y$. ⊠

This next example is critical for the factoring algorithm, although we do not discuss the factoring algorithm here..

Example:[Finding Order r of a mod b] Given a, b relatively prime, we would like to find the order of $a \mod b$. So, we are interested in the group $G = (Z_b, \cdot)$, with $(\cdot)$ denoting multiplication mod b, and the poly-time function $f(x) = a^x \mod b$. Then, H is $\langle r \rangle$ since $f(x) = f(y)$ iff $r|(x-y)$. This can be solved in a way similar to Simon's Problem but we use a general *Fast Fourier Transform* instead of the Hadamard gate, which is just a Fourier Transform over $\mathbf{GF}(2)$. ⊠

Example:[Discrete Log] We leave this as an exercise for the reader. ⊠

Example:[Graph Isomorphism] Consider 2 connected graphs G_1, G_2 on n vertices. In order to cast this as a hidden subgroup problem, we consider the symmetric group $G = S_{2n}$ on $2n$ elements, and the function $f(\pi) = \pi(G_1 \cup G_2)$ which gives the result of applying the permutation π on the vertices of the disjoint union of the 2 graphs. So, H is $\mathrm{Aut}(G_1 \cup G_2)$ since $f(\pi) = f(\sigma)$ if and only if there is some automorphism such that applying the automorphism after applying π gives the same result as applying σ. Note that the graphs are isomorphic if and only if some generator of H swaps G_1, G_2, since we can decompose automorphisms into automorphisms that do not swap G_1, G_2 and those that do. Thus, if we can find the generators for H, we can easily determine if the graphs are isomorphic. ⊠

The main technique in the algorithms for some of these examples is *Fourier sampling*. We set up a linear superposition of all possible inputs to f in the first register, store the output of f in the second register, apply an appropriate Fourier transform and then measure the system. However, this technique only works when the group is abelian, which is the case for factoring, Simon's problem, and discrete logarithm. So, in the case of Graph Isomorphism, since the symmetric group is not abelian, we cannot use the same technique. More specifically, the probability distributions given by Fourier sampling on positive instances of Graph Isomorphism and negative instances are indistinguishable.

7 Upper Bounds

Now that we have seen some of the power of quantum computers, we would like to see if we can upper bound quantum computers. Note that the outcome of a quantum computation depends on the, possibly negative, amplitudes of paths. Thus, the probability distribution is a GapP function. Thus, we get that $\mathrm{BQP} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$, and also $\mathrm{BQP} \subseteq \mathrm{PP}$(this is a class we will define in a future lecture).

The biggest open problem on quantum computation in complexity theory is whether or not BQP is contained within the Polynomial Hierarchy. In the probabilistic setting, approximate counting was good enough to show containment within the Hierarchy. In the quantum setting however, the possibility of interference makes it harder to just rely on approximate counting. In fact, it is conjectured that exact counting is required.

On the physics side, we do not yet know if we can build reliable and scalable quantum computers. So far, 2 models have been proposed. The first uses the spin of a single electron trapped in a silicon lattice. This scales up well once we can implement it on a few bits, since silicon is abundant. However, we are still trying to implement it on those few bits. Another model is the optical lattice. In this model, the electron is trapped in an optical lattice using lasers. While we have managed to implement some limited quantum computers using this model, scaling is a problem since the bottleneck is now laser power.

8 Next Lecture

With today's lecture, we have completed the first part of the course where we look at different models of computation and compare them with regards to time and space resources. In the next lecture we will start the next section of the course, where our goal is to use metrics other than time and space.