CSE 291 / Math 277A - Quantum Complexity Theory October 11, 2022

Lecture 6

Lecturer: Daniel Grier *Scribe: Nicholas Karris*

1 Warm Up

We begin by recalling the definition of BQP:

Definition 1. BQP *is the class of languages L for which there is a poly-time uniform (for definition of poly-time uniformity, please refer to Definition 1 of Lecture 5) family of circuits $\{Q_n\}_{n=1}^{\infty}$ such that*

- *if $x \in L$, then the probability of measuring 1 on the first qubit of $Q|x, 0\dots0\rangle$ is at least $2/3$*
- *if $x \notin L$, then the probability of measuring 1 on the first qubit of $Q|x, 0\dots0\rangle$ is at most $1/3$.*

Looking at these definitions, (at least) one natural question arises: what happens if we omit the requirement that the families of circuits be "poly-time uniform" in the BQP case? Why is the requirement that these circuits must be generated by a poly-time Turing Machine important?

The answer is that otherwise you could sneak some computation into the definition of the circuit itself. At one extreme, you could define the circuits in such a way that you could solve uncomputable functions. Consider the language

$$L := \{1^n : M_n \text{ halts}\}$$

where M_n is the nth Turing machine in some enumerated list $M_1, M_2, \dots$ (recall that the set of all TM's is countable). Note that this is essentially the halting problem under unary encoding, and so L is uncomputable. But if we define circuits Q_n by

$$Q_n := \begin{cases} X^{\otimes n} & 1^n \in L \\ I^{\otimes n} & 1^n \notin L \end{cases},$$

where X is the gate that just flips all bits (i.e., a NOT gate) and I is the identity, then

$$Q_n |0\rangle^{\otimes n} = \begin{cases} |1\rangle^{\otimes n} & 1^n \in L \\ |0\rangle^{\otimes n} & 1^n \notin L \end{cases},$$

which correctly decides L. If this family of circuits were allowed, then the unary halting problem would be in BQP, which is perhaps not what we intended by a class intended to capture efficient quantum computation. By restricting the class to be poly-time uniform, we know we are at least not bestowing any more power upon BQP than we already had with classical computation.

2 Complexity Hierarchy Results

To start, let us give a circuit definition of the class of polynomial time languages:

Definition 2. P *is the class of languages L for which there is a poly-time uniform family of classical circuits $C_n \colon \{0,1\}^n \to \{0,1\}$ built from* AND *and* NOT *gates such that $x \in L$ iff $C_n(x) = 1$.*

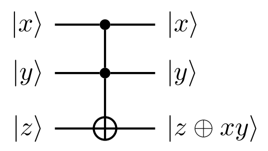
With this definition, we can prove a result which we have stated as true, but never fully explored:

Theorem 3. P $\subseteq$ BQP.

Proof. We need to use quantum gates (i.e. unitary operators) to simulate the classically complete set of gates (AND and NOT) from Definition 2. Note that NOT is already unitary operator, so we only need to find a unitary that simulates AND. For this, we introduce the Toffoli (or CCNOT) gate:

$$\text{Toffoli}(x, y, z) = (x, y, z \oplus (xy)).$$

The Toffoli gate is represented in a circuit diagram as



This is a classical gate, but it is also unitary as a quantum gate. The transition matrix is given by

$$\text{Toffoli} = \begin{bmatrix} I_6 & 0 \\ 0 & X \end{bmatrix} \in \mathbb{C}^{8\times 8}$$

where $I_6 \in \mathbb{C}^{6\times 6}$ is the identity and $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \in \mathbb{C}^{2\times 2}$ is the NOT gate, and so it is both Hermitian and self-inverse, hence unitary.

With the Toffoli gate, we can simulate AND since Toffoli$(x, y, 0) = (x, y, xy)$, where the third output is exactly $xy = \text{AND}(x, y)$. Thus, we can simulate a complete set of classical gates with unitary operators. Furthermore, this reduction is efficient, and so we can take any poly-time uniform circuit of classical gates and build a poly-time uniform circuit of quantum gates that simulates the behavior of the classical circuit. Therefore, $\mathsf{P} \subseteq \mathsf{BQP}$. $\square$

Also recall the following result also stated and proved at the end of Lecture 5:

Theorem 4. $\mathsf{BQP} \subseteq \mathsf{EXP}$.

Sketch of Proof. An n-qubit state can be represented as a 2^n-dimensional complex vector, and an n-qubit unitary is a $2^n \times 2^n$ complex matrix. Therefore, the result of a unitary transformation is simply a matrix-vector, which takes at most $O(4^n)$ time. $\square$

Putting these two together, we know that BQP is somewhere between P and EXP, which are respectively the smallest and largest complexity classes that were introduced last time. We now move towards narrowing down exactly where BQP fits in the complexity picture.

Theorem 5. $\mathsf{BQP} \subseteq \mathsf{PSPACE}$.

Proof. To get some intuition, consider the following example: Compute $H^2|0\rangle$, where H is the Hadamard gate. To do this, we have

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

We know the result is simply $|0\rangle$ after simplification, but we don't have to simplify the expression. Instead, we can keep it as a linear combination of $|0\rangle, |1\rangle, |0\rangle, |1\rangle$ with respective coefficients $\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, -\frac{1}{2}$. Why might we want to do this? Notice that keeping track of every coefficient of a pure state requires an exponential amount of space (in terms of the number of Hadamard gates we apply).

Writing the computation in the above form allows us to view the quantum computation as many "classical computation paths" in the form of a tree. Each leaf-node corresponds to a weighted classical state and summing the weights for one classical state over the leaf-nodes then gives rise to the amplitude of that classical state in the final quantum superposition result.

$$\begin{tikzcd}
& {|0\rangle} \arrow[dl] \arrow[dr] & & & {} \arrow[d, "H"] \\
{\frac{1}{\sqrt{2}}|0\rangle} \arrow[d] \arrow[dr] & & {\frac{1}{\sqrt{2}}|1\rangle} \arrow[d] \arrow[dr] & & {} \arrow[d, "H"] \\
{\frac{1}{2}|0\rangle} & {\frac{1}{2}|1\rangle} & {\frac{1}{2}|0\rangle} & {\frac{-1}{2}|1\rangle} & {}
\end{tikzcd}$$

Notice that though there may potentially be exponentially many paths, tracing through one path only requires polynomial amount of memory and time. This suggests the possibility of simulating the entire computation with only polynomial amount of space with some clever space reusing tricks. On the other hand, if we are going to work out the simplified vector representation of all the intermediate states, the amount of time required is at least exponential.

This idea generalizes to arbitrary unitaries on arbitrary states. Specifically, given a problem in BQP, we have a family of circuits Q_n, and critically, the number of unitary gates required in each circuit is polynomial in n. Thus, the branching tree described above (where we start with $|0\rangle$, apply the first unitary, then apply the second unitary to each of the outputs in turn, etc.) is poly-depth. This means we can, in poly-space, compute one particular branch of this tree.

Keeping this in mind, remember the goal: we want to produce an algorithm that uses poly-space which returns yes if and only if the quantum circuit accepts more than 2/3 of the time, i.e. if

$$\sum_{y\in\{0,1\}^{n-1}} |\alpha_{1y}|^2 \geq 2/3.$$

So we need to compute the above sum using only polynomial space. Intuitively, we want to compute the entire tree and then add up all the coefficients in the bottom row that correspond to states with a 1 in the first qubit, but the entire tree is itself has exponentially many nodes. However, we can compute any particular branch using only poly-space, so we propose the following fix: go through the branches one at a time, record the coefficients, then afterward add up all the coefficients that correspond to the same state, square the sums, and then add the sums together. But this again runs into the issue of needing exponential space because the number of states (and hence the number of buckets we need to keep track of the running total coefficient of each state) is exponential.

We are saved by the following (extremely inefficient computation-wise) approach: for each string $|y\rangle \in \{0,1\}^{\otimes n-1}$, iterate through each branch of the tree. If the state at the leaf of that branch is exactly the state $|1y\rangle$, then add the coefficient to the running total for $|1y\rangle$, and if it's a different state at the leaf, do nothing. After we have checked every branch, square the overall coefficient for $|1y\rangle$ and add it to the overall total probability of seeing a 1 in the first qubit. Then move to the next string y. Repeating this process for each string y gives a way to compute the total probability of seeing a 1 while only needing poly-space, which is what we want. $\square$

We'll give one final hierarchy theorem:

Theorem 6. $\mathsf{BQP} \subseteq \mathsf{PP}$.

Proof. Similar to the above, we want to produce a poly-time classical algorithm that accepts $> 50\%$ of the time whenever the quantum algorithm accepts, and we'll construct this classical algorithm by using the same tree idea. We'll make a simplifying assumption that the only quantum gates are Toffoli and Hadamard (recall that these gates are universal for quantum computation). This means that the magnitude of the coefficients at a given depth of the tree are all the same, which means the magnitude corresponding to each state is directly proportional to the (signed) number of leaves that end in that state. More specifically, let the final state before measurement of the quantum algorithm be

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle .$$

Let a_x be the fraction of all leaf nodes that are in state $|x\rangle$ and have positive weight, and let b_x be the fraction of all leaf nodes that are in state $|x\rangle$ and have positive weight. In other words, we can express the amplitude of the state as

$$\alpha_x = a_x - b_x$$

for real numbers $a_x, b_x \geq 0$.

Now consider the following classical algorithm. Randomly and uniformly follow one branch of the tree and record the state $|x\rangle$ at the leaf, then return to the beginning of the tree and randomly follow another branch of the tree and record the state $|y\rangle$ at the leaf. If $x \neq y$, then flip a coin to determine if we accept or reject the input. However, if $x = y$, then we want to determine if the signs of these states are likely to add constructively or destructively. Explicitly, if the first qubit of $x = y$ is a 1, then accept if the coefficients on x and y have the same sign and reject if they have opposite signs, and if the first qubit of $x = y$ is a 0, then accept if the coefficients have opposite signs and reject if they have the same sign.

We briefly discuss why this combination of accept/reject rule would give you an algorithm which accepts more than 50% of the time only when the quantum algorithm accepts. Notice that when $x \neq y$, our decision rule is simply a coin flip, which succeeds with probability 50% regardless of the truth. It suffices to show our decision rule succeeds with probability strictly more than 1/2 when $x = y$. Notice that the probability of seeing the state $|x\rangle$ twice with constructive weights is given by

$$\Pr[\text{seeing } |x\rangle \text{ twice with constructive weights}] = a_x^2 + b_x^2.$$

The probability of seeing the state $|x\rangle$ twice with destructive weights is given by

$$\Pr[\text{seeing } |x\rangle \text{ twice with destructive weights}] = 2a_x b_x$$

Suppose the quantum algorithm accepts (i.e., measures "1" on the first qubit) with probability p_{acc}. We get

$$p_{\text{acc}} = \sum_{y \in \{0,1\}^{n-1}} \alpha_{1y}^2 = \sum_{y \in \{0,1\}^{n-1}} (a_{1y} - b_{1y})^2 . \tag{1}$$

On the other hand, this implies that the algorithm rejects with probability

$$1 - p_{\text{acc}} = \sum_{y \in \{0,1\}^{n-1}} \alpha_{0y}^2 = \sum_{y \in \{0,1\}^{n-1}} (a_{0y} - b_{0y})^2 \tag{2}$$

Subtracting Equation (2) from (1), we get

$$\begin{aligned} 2p_{\text{acc}} - 1 &= \sum_y (a_{1y} - b_{1y})^2 - \sum_y (a_{0y} - b_{0y})^2 \\ &= \sum_y (a_{1y}^2 + b_{1y}^2 + 2a_{0y}b_{0y}) - \sum_y (a_{0y}^2 + b_{0y}^2 + 2a_{1y}b_{1y}) \end{aligned}$$

Notice that the first sum on the right hand side corresponds exactly to the probability of seeing either (i) two states starting with 1 and with constructive weights, or (ii) two states starting with 0 and with destructive weights. Ignoring the 50-50 coin tosses from the classical simulation procedure that don't end in the same state, this is exactly the probability that the procedure outputs YES. Similarly, the second sum is the probability that our classic simulation procedure outputs NO. This then gives

$$2p_{\text{acc}} - 1 = \Pr[\text{simulation outputs YES}] - \Pr[\text{simulation outputs NO}],$$

Hence, when $p_{\text{acc}} \geq 2/3$ left hand side is positive. Since the classic simulation is more likely to output YES, and so the PP machine accepts. When $p_{\text{acc}} \leq 1/3$, the left hand side is negative, so the classical simulation is more likely to output NO, and therefore the PP machine rejects. This concludes the proof. □

3 Quantum Algorithm Prelude

Overall, we've now shown that $\mathsf{BPP} \subseteq \mathsf{BQP} \subseteq \mathsf{PP}$. We don't know where BQP sits in relation to NP, and we'll spend some time later in the course investigating this relationship. But before we do that, now that we have some notion of what a quantum algorithm is, we should probably give some examples of quantum algorithms.

We begin with an easy example. Consider the problem where we have $f \colon \{0,1\} \to \{0,1\}$ and we want to determine if this function is constant or not, i.e. if $f(0) = f(1)$ or not. Classically, there is no way to do this without making both function calls $f(0)$ and $f(1)$ and checking if they return the same value. But the claim is that there's a way to do it with only one query using a quantum algorithm.

The quantum algorithm for computing this, known as Deutsch's algorithm, is

$$\begin{aligned}|0\rangle &\xrightarrow{H} \frac{|0\rangle + |1\rangle}{\sqrt{2}} \xrightarrow{O_f} \frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}} \xrightarrow{H} \frac{(-1)^{f(0)}\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) + (-1)^{f(1)}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)}{\sqrt{2}} \\ &= \frac{1}{2}\left[\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle + \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle\right]\end{aligned}$$

where O_f represents the phase oracle: $O_f|x\rangle = (-1)^{f(x)}|x\rangle$. Thus if $f(0) = f(1)$, then we get $\pm|0\rangle$ and hence we will see the state 0 with 100% probability when we measure, and if $f(0) \neq f(1)$, then we get $\pm|1\rangle$ and hence will see the state 1 with 100% probability when we measure. In other words, if we take the 0 state, apply $H \circ O_f \circ H$ and then measure, we will know with confidence whether $f(0) = f(1)$.

And notice that we only used one query here! The quantum circuit only uses one instance of the phase oracle, whereas we needed two queries in the classical case. This may not seem like a big improvement—they're both constant after all—but we'll see more examples in the next lecture where similar ideas lead to much more substantial separations.