CMSC 858L: Quantum Complexity

Instructor: Daniel Gottesman

Spring 2023

3 Lecture 3: BQP and NP

Correction: In the definition of space-bounded complexity, the amount of space used should not count the input *or output.* E.g., we can use a 3-tape Turing machine, of which the first tape is read-only (the input) and the third tape is write-only (the output). Then only the space used on the second tape is relevant. For decision problems, this makes no real difference, since the answer is just a bit, but functional problems — like determining what is a uniform family of circuits — it potentially matters a lot, since otherwise you would require space at least equal to the size of the output.

3.1 BQP

BQP is the central quantum complexity class, the class which captures what we mean by something that can be efficiently solved on a quantum computer (at least for decision problems). Because quantum measurement is random, we have to apply the same kind of definition as for probabilistic classical algorithms.

There is an additional complication in that we need to specify what kinds of quantum circuits we allow. First, there is a question of how it interfaces with the classical input, the instance x. I will assume the quantum circuit itself depends only on $|x|$ and the exact instance is provided as input in a basis state at the start of the circuit.

Definition 1. *If $Q(|\psi\rangle)$ is a quantum circuit with input $|\psi\rangle$, let $M_Q(|\{\psi\rangle)$ be the random variable produced by running the circuit on the input $|\psi\rangle$ and measuring the first qubit at the end of the circuit.*

Then there is the question of what gates to allow. It would be natural to allow any universal set of quantum gates of bounded size. However, this is a definition that would allow us to sneak in extra computational power. For instance, imagine we have a gate R_ϕ for arbitrary ϕ that does a phase rotation by ϕ: $R_\phi|0\rangle = |0\rangle$, $R_\phi|1\rangle = e^{i\phi}|1\rangle$. This is fine as far as it goes, but what if ϕ is an uncomputable number? By applying R_ϕ many times in a phase estimation algorithm, we would be able to find ϕ to arbitrary precision (given enough time), and calculate this uncomputable number. To disallow tricks like this, we should restrict attention to gates that are themselves efficiently computable.

Definition 2. *Let $\mathcal{G}$ be a set of quantum gates acting on a bounded number of qubits (usually either 2 or 3). We say that $\mathcal{G}$ is a* universal *set of gates if for any unitary U, there exists a circuit (of any size) containing gates from $\mathcal{G}$ that realizes the transformation U. $\mathcal{G}$ is* approximately universal *if, for any unitary U and any error ϵ, there exists a circuit composed of gates from $\mathcal{G}$ that realizes a unitary U_ϵ such that $\|U - U_\epsilon\|_{sup} < \epsilon$. $\mathcal{G}$ is efficiently computable if every matrix element of every member of $\mathcal{G}$ can be computed to accuracy ϵ in a time $O(\mathrm{poly}(\log 1/\epsilon))$ (on a classical computer).*

In other words, a universal set of gates can exactly reproduce any unitary, whereas an approximately universal set can be merely get close to any unitary.

We will discuss the ins and outs of universal and non-universal sets of gates in more detail once we are done definining the major complexity classes. For now, just notice that any (exactly) universal gate set must be infinite (since finite products of them give arbitary unitaries) and cannot be efficiently computable (since there exist some unitaries, as discussed above, that are not; and if the gate set can exactly reproduce one of them, it must also contain some noncomputable gates).

With these considerations, we therefore define BQP as follows:

Definition 3. *Fix a particular efficiently computable approximately universal gate set $\mathcal{G}$. Let BQP be the set of languages L such that there exists a uniform family of polynomial-size quantum circuits Q_n with the following properties: For any instance x,*

1. *The circuit $Q_{|x|}$ is composed of gates from $\mathcal{G}$.*
2. *If $x \in L$, then $\text{Prob}(M_{Q_{|x|}}(|x\rangle) = 1) \geq 2/3$.*
3. *If $x \notin L$, then $\text{Prob}(M_{Q_{|x|}}(|x\rangle) = 0) \geq 2/3$.*

"BQP" stands for "bounded quantum polynomial." Different gate sets $\mathcal{G}$ satisfying the conditions all define the same class BQP, as we will discuss later.

What if we use a quantum Turing machine to generate the BQP circuits? This gives the same class, as you will see on the problem set (although with a slightly different definition since I don't want to define quantum Turing machines).

3.2 NP

"NP" stands for "non-deterministic polynomial." The original definition is in terms of a non-deterministic Turing machine, which is a Turing machine that has a choice of possible transitions and can choose the "best" one. However, there is also a straightforward and more understandable definition in terms of checking the answer to the computation.

The idea is that for each yes instance x there is a "witness" w_x which proves that $x \in L$ and an efficient algorithm to check that the witness is correct. In particular, we have the following definition:

Definition 4. *Let NP be the set of languages L such that there exists a uniform family of polynomial-size circuits $C_{n,m}$ that takes two inputs (x, w) with the following properties: For any instance x,*

1. *If $x \in L$, then exists w_x with $|w_x| = O(\text{poly}(|x|))$ and $C_{|x|,|w_x|}(x, w_x) = 1$.*
2. *If $x \notin L$, then for any w_x with $|w_x| = O(\text{poly}(|x|))$, $C_{|x|,|w_x|}(x, w_x) = 0$.*

In the first case, w_x is the witness *for $x \in L$.*

That is, C is the checking algorithm, and if $x \in L$, then there exists some witness which will be accepted by the checking algorithm, whereas if $x \notin L$, than any possible witness will be rejected.

Example 1. *3-SAT is a language defined by Boolean expressions that can be satisfied. The Boolean expressions we consider are the AND ($\wedge$) of* clauses*, each of which involves the OR ($\vee$) of 3 Boolean variables (True/False valued) or their negations (written here with a line over the variable). (It is 3 variables because we are considering 3-SAT; k-SAT would involve k variables per clause.) The same variable can be repeated in a clause. For instance, the following are valid clauses:*

$$x_0 \vee x_5 \vee x_7 \tag{1}$$
$$x_1 \vee \overline{x_3} \vee \overline{x_4} \tag{2}$$
$$x_2 \vee x_2 \vee \overline{x_6} \tag{3}$$
$$x_1 \vee \overline{x_1} \vee x_8 \tag{4}$$

So, for instance, the first clause evaluates to True if one or more of x_0, x_5, x_7 is True. The last clause above always evaluates to True, since either x_1 is True or $\overline{x_1}$ is True.

Then the Boolean expression is formed from these clauses by taking the AND: e.g.,

$$(x_0 \vee x_5 \vee x_7) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2 \vee x_2 \vee \overline{x_6}) \wedge (x_1 \vee \overline{x_1} \vee x_8). \tag{5}$$

This evaluates to True iff all the individual clauses making it up are True. So, for instance, if all the variables x_0 through x_8 are True, then this particular expression is True, but if x_1 is False and all other variables are True, the overall expression is False since the second clause is False.

A Boolean expression $B \in$ 3-SAT if there exists a truth assignment x_i to the variables of B such that $B(x_0, \ldots, x_n) = \text{True}$.

Then 3-SAT is in NP: If $B \in$ 3-SAT, let the witness w_B be a satisfying truth assignment $(x_0, \ldots, x_n)$. The checking circuit evaluates B on the truth assignment. This checking circuit clearly runs in polynomial time and when we have a valid truth assignment, the checking circuit accepts. On the other hand, if $B \notin$ 3-SAT, then for any witness (any truth assignment), evaluating B on that truth assignment gives False, and the witness is rejected.

Note that there is an inherent asymmetry to the definition of NP: When $x \in L$, we can prove that and efficiently verify that the proof is correct, even though it might be hard to find the proof (i.e., the witness) efficiently. However, if $x \notin L$, we can't easily prove that this is true. Any witness we try will fail, of course, but we can't be sure that we didn't just make a bad choice of witness and that there isn't another witness that would work.

Note also that P $\subseteq$ NP since we can just let the witness be empty and the checking circuit equal to the algorithm to find the answer given the instance x.

There is another class called co-NP, which is basically the same as NP except that the witness exists when $x \notin L$ and there is no witness when $x \in L$. It is believed that NP $\neq$ co-NP. Factoring and graph isomorphism are examples of problems in NP $\cap$ co-NP.

3.3 Reductions and NP-completeness

It is a famous open problem to show that P = NP. One approach to studying this problem is to look at the hardest problems in NP. What does that mean?

For some pairs of problems, it is difficult to say and may not be meaningful to say that one is harder than the other, but sometimes it is clearly sensible to say that. In particular, if solving problem A automatically gives us a soltuion to problem B as well, then it makes sense to say that A is at least as hard as B. This the underlying notion of *reduction.*

Definition 5. *Language L* reduces *to language M if exists a polynomial-time computable function $f(x)$ such that for any instance x of L:*

1. *$f(x)$ is an instance of M,*
2. *If $x \in L$, then $f(x) \in M$,*
3. *If $x \notin L$, then $f(x) \notin f(M)$.*

There are a variety of different notions of reduction, but they all share a basic property with this definition: If we have an algorithm to decide M, we can combine it with the reduction to get an algorithm to decide L as well. In particular, given any instance x for L, we convert it to an instance of M using f and then determine whether $f(x) \in M$. Whatever the answer, we know it is the same as the answer to the original question : Is $x \in L$?

This lets us define a sensible partial order on problems with M "harder than" L if L reduces to M and L and M of equal hardness if they reduce to each other.

Remarkably, there are some problems which we can prove to be harder than any other problem in NP, in the sense that any NP problem can be reduced to one of them. Problems with this property are called "NP-complete."

Definition 6. *Let C be a complexity class. A language L is* C-hard *if for any $M \in C$, M reduces to L. A language L is* C-complete *if L is in C and L is C-hard.*

Example 2. *An example of an NP-complete problem is 3-SAT. To see how this works, let us consider an arbitrary NP problem M and see how to reduce it to 3-SAT. Since $M \in$ NP, there exists a family C_n of polynomial-size checking circuits for M.*

Now, a classical circuit is just a sequence of gates, say AND, OR, and NOT, connected by wires. AND and OR takes two inputs and one output; NOT takes one input and one output. Suppose we assign a new variable y_j to each wire. Then the basic gates express a relationship between the variables corresponding to their inputs and outputs. For instance, a NOT gate with input y and output y' expresses

$$(y \wedge \overline{y'}) \vee (\overline{y} \wedge y') = (y \vee y') \wedge (\overline{y} \vee \overline{y'}). \tag{6}$$

This is a Boolean formula of the form needed for 3-SAT (adding an extra redundant variable to the two clauses). Similarly for AND and OR.

By combining all the Boolean formulas for all the gates in the circuit C_n, we get a Boolean formula for which any satisfying assignment of variables is equivalent to a "history" of the circuit, giving truth values for the variables on the wires as they evolve under the gates of the circuit. However, this property doesn't constrain the input or output to the circuit. We can constrain some of the inputs to the circuit (corresponding to any ancilla bits or to the inputs for the instance we care about) by adding clauses which are true only when those bits have the desired values.

Now suppose we add one more clause. If x_f is the variable corresponding to the output wire of the circuit, add the clause $x_f \wedge x_f \wedge x_f$. With the extra clause, the Boolean formula $f(x)$ (when x is the original instance) is satisfied iff we have a set of possible input variables and a valid history of the circuit such that the output of the circuit is True (or 1). That is, the Boolean formula is satisfied exactly in those cases where the corresponding circuit input is accepted by the circuit C_n.

The input in this case is the witness for the instance x, so the Boolean formula is satisfied iff there is a witness that is accepted by the checking circuit. That is, $f(x) \in$ 3-SAT iff $x \in M$. We have reduced M to 3-SAT.

The argument in the example shows the following theorem:

Theorem 1 (Cook-Levin)**.** *3-SAT is NP-complete.*

There are many more NP-complete problems.

One significance of NP-completeness is that it offers a potential route to prove that P = NP: Simply find a polynomial-time algorithm for any NP-complete problem L. Since any other NP problem can be reduced to L, that automatically gives us an efficient algorithm for every language in NP. NP-completeness also helps us understand the relationships of complexity classes by distinguishing the hardest problems in NPfrom easier problems that are not NP-complete and therefore sometimes sit in lower complexity classes (such as NP $\cap$ co-NP).