Lecture 22: Quantum computational complexity

April 11, 2006

This will be the last lecture of the course—I hope you have enjoyed the lectures. There is still much that we have not had time to discuss, and of course there is a great deal that remains to be discovered, but we have to stop somewhere…

In this lecture, I'm going to give a very brief introduction to quantum computational complexity theory. Because many of you have no prior background in classical computational complexity, we will be severely limited in terms of what we can cover, but I hope to at least give you the feel for some of the topics considered in quantum complexity.

Promise Problems and Complexity Classes

Let us start with the notion of a *promise problem*. We have talked about promise problems in the black box setting, but for this lecture we will assume the input is given explicitly and there are no black boxes. A promise problem is a computational problem where two disjoint sets of inputs (*yes* inputs and *no* inputs) must be distinguished. For example, here is the Graph Isomorphism problem:

Graph Isomorphism (GI)

Input:	Two simple, undirected graphs G_0 and G_1.
Yes:	G_0 and G_1 are isomorphic ($G_0 \cong G_1$).
No:	G_0 and G_1 are not isomorphic ($G_0 \not\cong G_1$).

Formally, a promise problem is a pair $A = (A_{\text{yes}}, A_{\text{no}})$ with $A_{\text{yes}}, A_{\text{no}} \subseteq \{0,1\}^*$ (i.e., A_{yes} and A_{no} are sets of binary strings) that satisfies $A_{\text{yes}} \cap A_{\text{no}} = \varnothing$. Strings in $A_{\text{yes}} \cup A_{\text{no}}$ are said to *satisfy the promise*, and may be thought of as "valid" inputs. It is not required that $A_{\text{yes}} \cup A_{\text{no}} = \{0,1\}^*$. In the Graph Isomorphism problem, for example, a string $x \in \{0,1\}^*$ that encodes a pair of graphs (G_0, G_1) is an element of GI_{yes} if $G_0 \cong G_1$, and is an element of GI_{no} if $G_0 \not\cong G_1$. A string that does not encode a pair of graphs at all is in neither set, and is said to *violate the promise*. You may think of such strings as "don't care" inputs.

Traditionally, complexity theory has dealt mostly with non-promise decision problems, where you would essentially require $A_{\text{yes}} \cup A_{\text{no}} = \{0,1\}^*$. In this case, the decision problem is identified with the *language* A_{yes}, and it is implicit that $A_{\text{no}} = \{0,1\}^* \backslash A_{\text{yes}}$. You don't generally bother writing a subscript "yes" in this case—you just talk about a language A, which contains all of the strings for which the correct answer is yes. There are some aspects of the complexity theory of languages that are lost when treating things in more generality by allowing promises, but for this lecture it won't make any difference. I have chosen to discuss promise problems rather than non-promise problems mostly because the promise problem formulation turns out to be useful and this is a good opportunity to tell you about it.

A *complexity class* is just a set of promise problems, usually coinciding with some particular resource constraint and/or computational model. For example, the following complexity classes are defined by placing time or space constraints on the deterministic and probabilistic Turing machine models:

P	The class of promise problems solvable in polynomial time on a deterministic Turing machine.
BPP	The class of promise problems solvable in polynomial time on a bounded error probabilistic Turing machine (correct on every input with probability at least 2/3).
PP	The class of promise problems solvable in polynomial time on an unbounded error probabilistic Turing machine (correct on every input with probability strictly greater than 1/2).
PSPACE	The class of promise problems solvable in polynomial space on a deterministic Turing machine.
EXP	The class of promise problems solvable in exponential time on a deterministic Turing machine.

Other complexity classes can be defined based on the notion of efficient verification. One of the most important ones is NP; a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in the class NP if and only if there exists (i) a polynomial p, and (ii) a polynomial-time computable predicate V, such that these two properties are satisfied:

Completeness: If $x \in A_{\text{yes}}$ then there exists a string y of length $p(|x|)$ such that $V(x, y) = 1$. The string y is a *proof* (or *certificate* or *witness*) that $x \in A_{\text{yes}}$.

Soundness: If $x \in A_{\text{no}}$ then $V(x, y) = 0$ for every string y of length $p(|x|)$.

You can also define NP in terms of *nondeterministic* Turing machines, but the definition above is more intuitive.

One can define a similar class where the verification is probabilistic: a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in MA if and only if there exists: (i) a polynomial p, and (ii) a polynomial-time probabilistic algorithm V, such that these two properties are satisfied:

Completeness: If $x \in A_{\text{yes}}$ then there exists a string y of length $p(|x|)$ such that V accepts (x, y) with probability at least $99/100$.

Soundness: If $x \in A_{\text{no}}$ then V rejects (x, y) for every string y of length $p(|x|)$ with probability at least $99/100$.

Several other classes that abstract the notion of efficient verification can be defined based on the notion of an *interactive proof system*. These and other related classes have played an important role in complexity theory over the last 20 years.

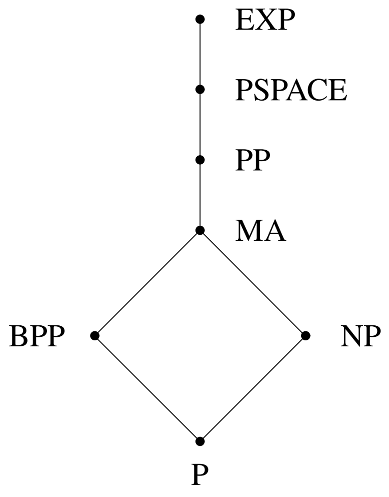

Figure 1: Relations among some classical complexity classes

Figure 1 describes known inclusions among the classes defined above. This is a typical type of figure for showing inclusions—one class is contained in another if you can get from the smaller class to the larger one by following lines upward. For example, BPP and NP are both contained in MA, and both contain P. The diagram gives no information about the relationship between NP and BPP, reflecting the fact that it is not known whether BPP $\subseteq$ NP, NP $\subseteq$ BPP, or if they are incomparable. The diagram also does not indicate proper inclusions, which are often hard to come by in complexity theory. For instance, there is only one known proper inclusion among the classes in the diagram: P $\subsetneq$ EXP. It is not known if P $=$ PSPACE or not, or if NP $=$ EXP or not, for instance. If you can prove P $\neq$ NP, which most people conjecture is the case, then you will have solved a major open problem and will receive a \$1 million prise from the Clay Mathematics Institute. Of course, the \$ 1 million prise would be nothing in comparison to having proved P $\neq$ NP—you can't buy mathematical immortality for a mere \$ 1 million.

Quantum complexity classes

Uniform quantum circuits

We can define quantum complexity classes in a similar way to the classes defined above, except based on quantum computations instead of classical (deterministic or probabilistic) computations. Although it is possible to define quantum Turing machines for the purpose of doing this, it turns out to be easier and equivalent to use the quantum circuit model. Let us briefly discuss some assumptions that we need to make concerning quantum circuits in order to have a good model for complexity theory.

One technical concern that must be addressed when we discuss quantum circuits for complexity classes is the notion of *uniformity*. Suppose that we have a collection of quantum circuits

$$\{Q_x : x \in \{0,1\}^*\},$$

one circuit Q_x for each possible string x. Then we say that this collection is *polynomial-time uniformly generated* (or sometimes people say *polynomial-time uniform* or *polynomial-time generated*) if there exists a deterministic polynomial-time algorithm that, on input x, outputs a classical description of Q_x.

Secondly, we need to fix a set of gates that are permitted. Although you could make different (but equivalent) choices, the following collection of gates works out nicely:

1. Toffoli gates,
2. Hadamard gates,
3. i-phase shift gates (which induce the transformations $|0\rangle \mapsto |0\rangle$ and $|1\rangle \mapsto i\,|1\rangle$ on a single qubit),
4. initialization gates (which input nothing and output a qubit in state $|0\rangle$), and
5. trace-out gates (which take one qubit as input and produce no output, effectively tracing out that qubit).

This collection is *universal* in the sense that any admissible operation from n qubits to m qubits can be approximated with arbitrarily good accuracy with a circuit composed of these gates. We will assume all circuits are composed of these gates for the rest of the lecture. A circuit will be said to be of type (n, m) if it has n input qubits and m output qubits.

BQP

Now we are ready to define our first quantum complexity class: BQP. It is intended that this class abstracts the notion of a promise problem that could be solved efficiently with a quantum computer. Formally, a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in BQP if there exists a polynomial-time generated family $\{Q_x : x \in \{0,1\}^*\}$ of type $(0,1)$ quantum circuits such that:

1. If $x \in A_{\text{yes}}$, then $\langle 1|Q_x|1\rangle \geq 99/100$, and
2. if $x \in A_{\text{no}}$, then $\langle 1|Q_x|1\rangle \leq 1/100$.

The idea here is that the input string x is "hard coded" into the circuit Q_x. When you run Q_x, you start with no qubits, and end up with a 1 qubit mixed state (which we are simply denoting Q_x, because it is produced by the circuit Q_x). If you measure this qubit with respect to the standard basis, you get the outcome 1, which we interpret as "yes", with probability $\langle 1|Q_x|1\rangle$.

A decision problem version of factoring is is BQP because of Shor's algorithm:

Integer Factoring

Input:	Positive integers (N, k) with $2 \leq k \leq N$.
Yes:	There exists a proper factor of N in the range $\{2, \ldots, k\}$.
No:	There is no proper factor of N in the range $\{2, \ldots, k\}$.

It also happens to be the case that both Integer Factoring and its complement (where we switch the yes and no conditions) are in NP. We would typically state this as *Integer Factoring* $\in$ NP$\cap$co-NP.

A natural question to ask is: how does BQP compare with the classical complexity classes defined above? It is fairly easy to prove that BPP $\subseteq$ BQP, based on the observation that quantum circuits can simulate coin-flips and perform deterministic computations efficiently. It is also easy to prove BQP $\subseteq$ EXP: if all of the gates of Q_x are replaced by matrices and all of the matrix arithmetic is done explicitly, then at most exponential time (and space) is required to calculate $\langle 1|Q_x|1\rangle$ given a description of the circuit Q_x. This method can be improved to show BQP $\subseteq$ PSPACE by performing the same computations in a very space-efficient manner. It may not be obvious how this can be done, but there are well-known methods that can be used to accomplish it. In essence, if you are willing to repeatedly calculate certain numbers over and over, once each time you need them, you don't need to actually store much information to perform matrix arithmetic.

We can do somewhat better than PSPACE as an upper bound on BQP, though; we can prove BQP $\subseteq$ PP. To establish this, we will consider a classical probabilistic algorithm for simulating a quantum circuit. There is a catch, of course, which is that the probabilistic algorithm will have *unbounded error*: although it will work correctly a majority of the time, its probability of outputting yes and no will be so close to 1/2 that it will be useless from a practical point of view. This, however, is the way that the class PP is defined, and it should not be viewed as representing a practical notion of probabilistic computation. Instead, the class PP represents a different type of problem that can in some sense be solved if one has the ability to count an exponential number of things quickly.

Before describing this algorithm, it will be helpful to get rid of and change some gates in a given circuit Q_x. The first thing to notice is that the initialization and trace-out gates are useless for computation—they are only there so that we can say certain things more efficiently when we state theorems or discuss admissible operations. Therefore, you might as well imagine that Q_x has type (n, n) for some integer n, and only contains unitary (Toffoli, Hadamard, and i-phase shift) gates. You start the circuit on the state $|0^n\rangle$ and ignore all of the qubits except one of them (say the first one) after all of the gates are applied. Lastly, each of the i-phase shift gates can be replaced with a two qubit gate that induces the transformation

$$\begin{aligned}|00\rangle &\mapsto |00\rangle\\ |01\rangle &\mapsto |01\rangle\\ |10\rangle &\mapsto |11\rangle\\ |11\rangle &\mapsto -|10\rangle\,.\end{aligned}$$

Here, the first qubit is the qubit on which the phase shift originally acted and the second is a single new qubit that is added to the circuit. This new qubit can be thought of as the "real/imaginary qubit", and starts in the state $|0\rangle$ like all of the other qubits. You simply ignore it when the circuit is done. Effectively, this qubit is doubling the dimension of the space we are working with so that we can identify $\mathbb{C}$ with $\mathbb{R}^2$. (You don't actually even need the above two qubit gate if you are willing to do some fancy footwork with Toffoli and Hadamard gates alone, but it will not make the proof that BQP $\subseteq$ PP any harder to include it.)

Now we are ready to state the algorithm that probabilistically simulates a given quantum circuit Q_x. This algorithm will output a single bit, and we just need that the probability to output a 1 is greater than the probability to output 0 if and only if the same is true of Q_x.

1. Let $B \leftarrow 0$ be a single bit register that we will think of as a sign bit. Let $Z \leftarrow 0^n$ represent the initial state of the n qubits of Q_x. Simulate the effect of each gate of Q_x on Z as follows:
   - If the gate is a Toffoli gate, simply modify Z accordingly.
   - If the gate is a Hadamard gate, flip a coin to determine the new state of the corresponding bit of Z. If the induced transformation was $1 \mapsto 1$, toggle the sign bit.
   - If the gate is the two-qubit gate we used to replace the i-phase shift gate, modify Z appropriately and toggle the sign bit if the transformation induced was $11 \mapsto 10$.
2. Repeat step 1 a second time, using variables B' and Z' in place of B and Z.
3. If $Z = Z'$ and the first bit of Z and Z' is a 0, then output $B \oplus B'$.

   If $Z = Z'$ and the first bit of Z and Z' is a 1, then output $\neg B \oplus B'$.

   If $Z \neq Z'$, then "give up". This means: flip a fair coin and output 0 or 1 accordingly.

You might take a look at this algorithm and wonder how it could possibly work. The idea is that the probability that the algorithm outputs 1 minus the probability that it outputs 0 is proportional to $\langle 1|Q_x|1\rangle - \langle 0|Q_x|0\rangle$. The constant of proportionality happens to be tiny—exponentially small in the number of Hadamard gates—but that is okay. All that we care about is whether the algorithm outputs 1 with probability greater than or less than $1/2$.

Quantum proofs: QMA

Finally, I would like to just mention an interesting quantum complexity class that gives a quantum analogue of NP (or really MA because the verification is probabilistic). The class is called QMA. Formally, a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in QMA if there exists (i) a polynomial p, and (ii) a polynomial-time generated family $\{Q_x : x \in \{0,1\}^*\}$ of circuits, where each Q_x is of type $(p(|x|), 1)$, such that these two properties hold:

Completeness: If $x \in A_{\text{yes}}$, then there exists a state ρ on $p(|x|)$ qubits such that

$$\langle 1|Q_x(\rho)|1\rangle \geq \frac{2}{3}.$$

The state ρ is essentially a "quantum proof" or "quantum certificate" that establishes that $x \in A_{\text{yes}}$.

Soundness: If $x \in A_{\text{no}}$, then for every state ρ on $p(|x|)$ qubits it holds that

$$\langle 1|Q_x(\rho)|1\rangle \leq \frac{1}{3}.$$

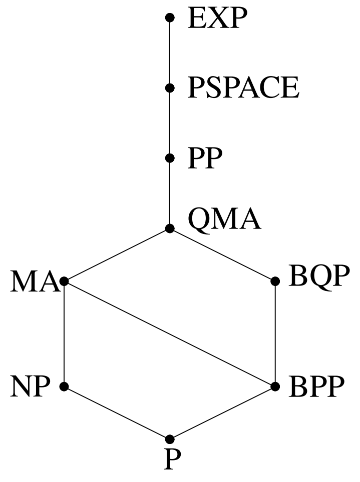

Figure 2: Containments among classes discussed in this lecture.

It can be shown that even this class is contained in PP. The proof is more difficult than the proof that BQP $\subseteq$ PP, so I won't have time to discuss it.

An interesting question is whether quantum certificates can be more powerful than classical ones. An example of a problem that is known to be in QMA but not known to be in MA (or even QMA but restricting the proof to be classical) is *Group Non-membership*: given a group and an element of some common super-group, answer "yes" if the element is not contained in the group, and "no" otherwise. This is another example of how the structure of groups can be exploited by quantum computational models.

Figure 2 contains a diagram of the classes we have discussed in this lecture.