CMSC 657: Introduction to Quantum Information Processing Lecture 9

Instructor: Daniel Gottesman

Fall 2024

1 Complexity Theory

1.1 NP

Recall we were discussing the complexity class NP:

Definition 1. *NP (which stands for "nondeterministic polynomial") is the class of languages L such that there exists a "verifier" algorithm $V(x, w)$ such that*

1. *$V(x, w)$ runs in a time polynomial in $|x|$ for all x, w such that $|w| = \mathrm{poly}(|x|)$*
2. *$\forall x \in L$, $\exists w_x$ such that $V(x, w_x) = \mathrm{yes}$ and $|w_x| = \mathrm{poly}(|x|)$*
3. *$\forall x \notin L$, $\forall w$ such that $|w| = \mathrm{poly}(|x|)$, $V(x, w) = \mathrm{no}$.*

w_x is the witness *for a yes instance x.*

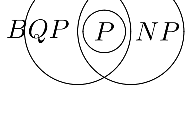
Languages such as 3-SAT are the hardest languages in NP. How can we make this a precise statement?

Definition 2. *A language L* reduces *to language M if there exists a polynomial time function $f : \{0,1\}^* \to \{0,1\}^*$ such that $x \in L$ iff $f(x) \in M$. A language M in NP is NP-complete if for all $L \in \mathrm{NP}$, L reduces to M.*

If L reduces to M that means that, given an algorithm to solve M, we can apply that algorithm to solve L: Take instance x of L and map it to an instance $f(x)$ of M, apply the algorithm to determine if $f(x) \in M$, and return that same answer for whether $x \in L$. This implies that M is at least as hard as L.

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

It is a famous open problem whether P = NP. The general belief is that they are not equal. If P $\neq$ NP, then NP-complete problems are not in P, so to prove that a problem is hard, we are generally satisfied if we can prove it is NP-complete, even though the result is just a conditional one.



We think this is the relationship between P, NP, and BQP. We believe that NP-complete problems cannot be solved efficiently on a quantum computer and that there are problems solvable in BQP that are outside NP (meaning there is no efficient classical verifier for them).

1.2 Oracle Problems

In general, proving that a problem is hard unconditionally, or proving that two complexity classes are not equal, is outside our capabilities (except in extreme cases). Sometimes people look at the *oracle* model. The oracle model is an unrealistic model, but one that still provides guidance towards finding real algorithms, and additionally has the big advantage that it is possible to prove hardness results.

An oracle is a black box function $O : \mathbb{Z}_2^n \to \mathbb{Z}_2$. We know nothing else about how O is computed. We make a *query* to O by giving it an input x and receiving back the output $O(x)$. In the oracle model, instead of counting gates, we determine complexity by counting queries.

Definition 3. *Let $f(O)$ be a property of oracles. The* query complexity *of f is the minimum number of queries to O needed to calculate $f(O)$.*

Example 1. *This is a promise problem. We are promised that the oracle $O : \mathbb{Z}_2^n \to \mathbb{Z}_2$ is either* constant *(i.e., $O(x) = O(y)$ for all x, y) or* balanced *($|\{x \text{ s.t. } O(x) = 0\}| = |\{x \text{ s.t. } O(x) = 1\}|$). Determine which with* 100% *probability.*

To get the right answer with certainty, a classical algorithm needs at least $2^{n-1} + 1$ queries.

Proof. Suppose we have an algorithm that makes only 2^{n-1} queries. Let the output of the first query x_0 be a. Based on that, the algorithm decides to make a second query x_1, and suppose that the output of that query is also a. There are certainly oracles which have $O(x_0) = O(x_1) = a$, so this is a possibility. The algorithm continues deciding on queries x_i for $i < 2^{n-1}$, and suppose that all of the queries return a. This is consistent with both a constant oracle O_1 (for which $O(x) = a$ for all x) and a balanced oracles O_2, for which $O(x_i) = a$ for $i = 0, \ldots, 2^{n-1} - 1$ and $O(y) = a \oplus 1$ for $y \neq x_i$ for any i. This algorithm cannot decide between O_1 and O_2 and therefore will fail with some probability. $\square$

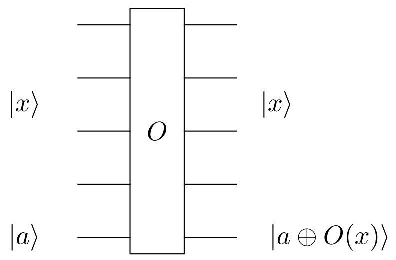
Some points about the oracle model and query complexity:

- Only the number of oracle calls contributes to query complexity, not the number of gates or the amount of memory needed between oracle calls. It is quite possible for an algorithm to have low query complexity but high gate complexity. Nevertheless, it is frequently the case that the gate complexity is similar to the query complexity.
- Given an oracle algorithm, we can replace the black-box function by an explicit function to get an algorithm for a regular problem. However, this is only really useful in cases where the gate complexity is low.
- If we manage to prove a lower bound on the query complexity, that can be an indication that related problems for explicit functions are hard. However, it might not be the case — in particular, there may be a way to take advantage of some structure of the explicit function to solve the problem more quickly. For instance, suppose the oracle in the example above is replaced with the function $O(x) = ax_i + b$, a, b bits and x_i the ith bit of the input x. If $a = 0$, the function is constant and if $a = 1$ it is balanced. Because we know the structure of $O(x)$, we know to try queries that differ in values of x_i for different values of i. E.g., we try the queries $x = 10000, 01000, 00100, 00010, 00001$ for $n = 5$. This is enough to determine a, b, and i and thus the whole function using only n queries.

1.3 Quantum Oracles

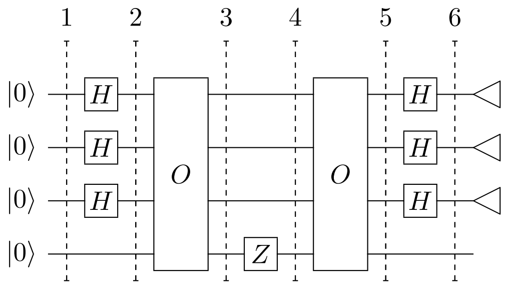
We can convert a classical oracle into a quantum oracle by making it reversible and putting kets around everything:

$$O|x\rangle|a\rangle = |x\rangle|a \oplus O(x)\rangle. \tag{1}$$



1.4 Deutsch-Jozsa algorithm

Deutsch-Jozsa algorithm: Consider the problem in the example. There is a quantum algorithm to solve this problem with just 2 quantum queries:



After the measurement, we do classical post-processing to determine the output: If the measurement result is all 0, output "constant". Otherwise, output "balanced".

Let us follow the state of the system at each step:

1. $|00\ldots0\rangle|0\rangle$
2. $(\sum_x |x\rangle)|0\rangle$
3. $\sum_x |x\rangle|O(x)\rangle$
4. $\sum_x (-1)^{O(x)}|x\rangle|O(x)\rangle$
5. $\sum_x (-1)^{O(x)}|x\rangle|0\rangle$ (uncomputing $O(x)$)

At this point, what happens next depends on if it is constant or balanced.

- **Constant:** The state pre-Hadamards is $\pm\sum_x |x\rangle|0\rangle$, so the Hadamards give $\pm|00\ldots0\rangle|0\rangle$.
- **Balanced:** Let $|\psi\rangle = \sum_x |x\rangle$ and $|\phi\rangle = \sum_x (-1)^{O(x)}|x\rangle$. Then $\langle\psi|\phi\rangle = \sum_x (-1)^{O(x)} = 0$ since there are as many $+1$ terms as -1 terms. But $H^{\otimes n}$ is unitary and $H^{\otimes}n|\psi\rangle = |00\ldots0\rangle$. Thus, $H^{\otimes n}|\phi\rangle$ is orthogonal to $|00\ldots0\rangle$, which means the measurement outcome will always have at least one 1 in it.

Note that if we don't uncompute in step 5, the algorithm does not work. The state for balanced O is only a superposition of half of all x's (conditioned on $O(x)$), and so we get results other than $|00\ldots0\rangle$. This is not orthogonal to the superposition of all x's, the state for constant O still, so it cannot be distinguished with 100% accuracy.

Can we do this with just 1 quantum query? Yes! Look what happens if instead of using $|0\rangle$ for the last qubit, we use $|-\rangle = |0\rangle - |1\rangle$. When we do a NOT on the last qubit, it takes $|-\rangle$ to $|1\rangle - |0\rangle = -|-\rangle$. Therefore, the action of the oracle on the superposition becomes

$$\sum_x |x\rangle|-\rangle \mapsto \sum_x (-1)^{O(x)}|x\rangle|-\rangle. \tag{2}$$

This is where we wanted to get with no need to uncompute $O(x)$.

2 Algorithms

As I discussed last week, oracle algorithms are good toys and sometimes can be models for quantum algorithms for real problems. What we are really interested in, however, are algorithms for actual problems where things are not specified as black boxes. The first quantum algorithm we will discuss is Shor's algorithm for factoring.

Before discussing Shor's algorithm, however, I want to discuss why factoring is an important problem. The answer is that it can break RSA, one of the most widely-used cryptographic system today. So the first thing to do is give a quick overview of crytography in general and RSA specifically. This will also be important later when we talk about quantum cryptography, and will give us an opportunity to discuss some of the mathematics and classical algorithms used in and around Shor's algorithm.

2.1 Cryptography in General

To discuss cryptography, we are going to return to Alice and Bob (who actually worked in cryptography before starting to work on quantum information). Alice wishes to send a message to Bob, who is at some distant point from Alice. Everything is classical now, and Alice has a communications channel to Bob that lets her transmit bits. The problem is that there is an eavesdropper Eve who sometimes has access to the bits being sent, and Alice doesn't want Eve to learn the contents of the message she wants to send to Bob.

In order for Alice to successfully send secret messages to Bob, she and Bob need some sort of advantage over Eve, and usually the advantage that is assumed is a secret key. What kind of secret key and how it works depends on the cryptographic protocol in use.

Encryption is an algorithm which converts the message, the *plaintext*, to an encrypted *ciphertext*, which has the property that if Eve sees it, she does not learn anything about the message. (Actually, she is supposed to not learn anything *new*, since it is possible she already knows something about the message from other sources. For instance, she might know the message is written in English, which tells her something about the frequency of various letters.) Then Bob applies a *decryption* algorithm, which converts the ciphertext back into the plaintext.

A protocol that does this in an *encryption* protocol. There are also cryptographic protocols that protect information in other ways, such as a *digital signature* protocol, which lets Alice transmit a message to Bob in such a way that he can verify that it really came from Alice and not Eve, and moreover, he can prove that fact to a third party Charlie.

The simplest and most secure encryption protocol is the *one-time pad*. The one-time pad is an example of a *private-key cryptosystem*. In a private key cryptosystem, Alice and Bob share a secret key k (the same for both of them) which is unknown to Eve. This is basically the only limitation on Eve for the one-time pad, although we also assume that Alice and Bob have private computers to perform encryption and decryption without Eve seeing. In the one-time pad, k is just a string of uniformly random bits as long as the message (which has also been converted to a string of bits).

- **Encryption:** Given message m, the ciphertext is $e = m \oplus k$, the bitwise XOR of k and m.
- **Decryption:** $e \mapsto e \oplus k = (m \oplus k) \oplus k = m$.