Lecture 17, Thurs March 23: Quantum Query Complexity, Deutsch-Jozsa

People often want to know where the true power of quantum computing comes from.

- Is it the ability of amplitudes to interfere with one another?
- Is it the huge size of Hilbert space (the space of possible quantum states)?
- Is it that entanglement gives us 2^n amplitudes to work with?

But that's sort of like dropping your keys and asking "what made them fall?"

- Is it their proximity to the Earth?
- Is it the curvature of spacetime?
- Is it the fact that you dropped them?

There can be many complementary explanations for the same fact, all of them valid. And that's the case here. If there weren't a huge number of amplitudes, quantum mechanics would be easy to simulate classically. If the "amplitudes" were probabilities, rather than complex numbers that could interfere with each other, QM would also be easy to simulate classically. If no entanglement were allowed, then at least all pure states would be product states, once again easy to represent and simulate classically. If we were restricted to stabilizer operations, QM would be easy to simulate classically. But as far as we know, full QM---involving interference among exponentially many amplitudes in complicated, non-stabilizer entangled states---is hard to simulate classically, and that's what opens up the possibility of getting exponential speedups using a quantum computer.

Quantum Complexity

There are two major ways we look at the complexity of quantum algorithms

The circuit complexity of a unitary transformation U is the size (i.e., number of gates) of the smallest circuit that implements U. We like unitaries with polynomial circuit complexity. Alas, typically it's *extremely* hard to determine the circuit complexity of a unitary; the best we can do is to prove upper bounds, and conjecture lower bounds on the basis of hardness assumptions and reduction arguments. Note that the reasons why this sort of problem is insanely hard have nothing to do with quantum mechanics.

"What's the smallest circuit that solves Boolean satisfiability?"
is a similarly hard problem, indeed closely related to P vs NP.

So if we want a more precise picture of what's going on, albeit in a more limited model, we instead often use…

Query complexity, which is the number of calls the algorithm makes to an oracle (or black box function). The idea is that your oracle takes an input *x* and produces an output $f(x)$, where say

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

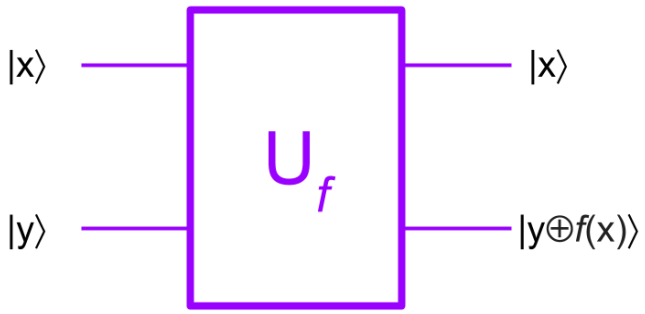
is a Boolean function. In quantum mechanics, our first guess for what this would mean might be: we map the input state $|x\rangle$ to the output state $|f(x)\rangle$, or maybe the output state $|x\rangle|f(x)\rangle$.

Here, though, we run into a bit of trouble because such a transformation is not unitary. To make the transformation unitary, we need a so-called *answer* or *target* register.

So we give the black box two inputs: x, which is unchanged, and y, which has the answer $f(x)$ written into it in a reversible way, typically exclusive-ORing:

$$|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$$

If we took care to ensure that $y = 0$ initially, then this would map $|x, 0\rangle$ to $|x, f(x)\rangle$, exactly like in the classical case.



A lot of times we ignore the answer qubit by moving the phases around. So let's say we prepare the answer qubit as $|-\rangle$.

One important note for later: if f happens to be a Boolean function, then often it's most convenient to consider quantum queries that map each basis state $|x\rangle$ to $(-1)^{f(x)}|x\rangle$: in other words, that *write the function value into the phase of the amplitude*, rather than storing it explicitly in memory. Or more precisely, we consider queries that map each basis state $|x, b\rangle$ to $(-1)^{f(x)\cdot b}|x, b\rangle$, where b is a bit that controls whether the query should even take place or not. This sort of "phase oracle" doesn't really have any classical counterpart, but is extremely useful for setting up desired interference patterns.

So it behooves us to ask: how do the "phase oracle" and the "XOR oracle" compare? Could one be more powerful than the other? Happily, it turns out that they're *equivalent*, in the sense that either can be used to simulate the other, with no cost in the number of queries. This is a result that we'll need later in this lecture.

To see the equivalence, all we need to do is consider what happens when the second register—the one containing y or b—is placed in the $|-\rangle$ state before a query. You can check that this converts a XOR oracle into a phase oracle: we get

$$\frac{1}{\sqrt{2}}\left(|x, 0\rangle - |x, 1\rangle\right) \rightarrow \frac{1}{\sqrt{2}}\left(|x, 0 \oplus f(x)\rangle - |x, 1 \oplus f(x)\rangle\right)$$

which equals $\frac{1}{\sqrt{2}}\left(|x, 0\rangle - |x, 1\rangle\right)$ if $f(x) = 0$

$\frac{1}{\sqrt{2}}\left(|x, 0\rangle - |x, 1\rangle\right)$ $\quad f(x) = 1$

which we can rewrite as just $(-1)^{f(x)}|x, -\rangle$. Meanwhile, if the second register is placed in the $|+\rangle$ state, then nothing happens, so we really do get the $|x, b\rangle \rightarrow (-1)^{f(x)\cdot b}|x, b\rangle$ behavior .

Conveniently, the converse is also true! That is, if we know that a phase oracle will be acting, then by placing the b register in one of the states $|+\rangle$ or $|-\rangle$, we can simulate the effect of a XOR oracle, with the phase oracle causing $|+\rangle$ and $|-\rangle$ to be swapped if and only if $f(x) = 1$.

Taking a step back, though, what are we really doing when we design a quantum algorithm in the query complexity model? We're abstracting away part of the problem, by saying:

"Our problem involves some function, say, $f : \{0,1\}^n \rightarrow \{0,1\}$, and we're trying to learn some property of f, in a way that only requires evaluating f on various inputs, not looking at the internals of how f is computed."

So for example, you might want to learn:

Is there some input x such that $f(x) = 1$?

Does $f(x) = 1$ for the majority of x's?

Does f satisfy some global symmetry, such as periodic, or is it far from any function satisfying the symmetry?

Etc.

We then want to know how many queries to f are needed to learn this property.

In this model, we abstract out the cost of any quantum gates that are needed before or after the queries: those are treated as "free."

Before we delve deeper, it's worth asking:
"Why do we care about the black-box model? You're debating how you'd phrase your wishes if you found a magical genie. Who cares?"

The truth is more prosaic, though. You can think of a black box as basically a huge input string. From that standpoint, querying $f(i)$ just means looking up the i^{th} element in the string.

Another way to think about it:

Imagine I'm writing code, and I have a subroutine that computes $f(x)$. How many times do I need to call the subroutine to find some information about f? Assuming, in the quantum case, that I even get to call the subroutine on a superposition of different x values, and get back a superposition of answers?

But also assuming that we're not going to "violate abstraction boundaries" by examining the code of the subroutine.

To justify the quantum black-box model, there's one technical question we need to answer…
Suppose we did have a small circuit to compute a function f. Could we then implement the quantum black-box behavior that we described above—without loss of generality, the XOR behavior, $|x, a\rangle \rightarrow |x, a \oplus f(x)\rangle$?

The reason this isn't entirely obvious, is that if you'll recall, quantum circuits have to be *reversible*. So just because there's a small circuit to compute $f(x)$, doesn't immediately imply that there's a small circuit that maps the basis state $|x, a\rangle$ to the basis state $|x, a \oplus f(x)\rangle$ *with nothing else lying around*.

Indeed, let's step back, and think about the constraints on computation that are imposed by reversibility. To start with the obvious: if we had a reversible circuit that maps $|x\rangle$ to $|g(x)\rangle$, then g must be an injective function. But now for the subtle part: even if g is both injective *and* efficiently

computable, that still doesn't imply (at least, as far as we know) that the map $|x\rangle \rightarrow |g(x)\rangle$ is efficiently computable.

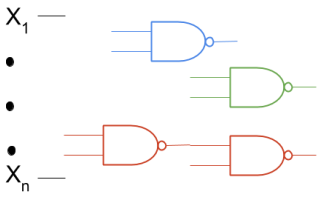
Why not? Well, imagine that g were an injective one-way function: that is, a function that's easy to compute but hard to invert. Such functions are the basic building blocks of cryptography, and are strongly conjectured to exist, even in a world with quantum computers.

Note that even though quantum computers can break a few supposedly one-way functions, like those based on factoring and discrete log, there are many, many more "generic," less "structured" one-way functions that don't seem threatened by quantum computers. We'll have more to say about such issues later.

Anyway, now suppose we had a small circuit C such that $C|x\rangle = |g(x)\rangle$. Then simply by running that circuit backwards—that is, inverting all the gates and reversing their order—we could get $C^{-1}|g(x)\rangle = |x\rangle$, thereby inverting the supposed one-way function!

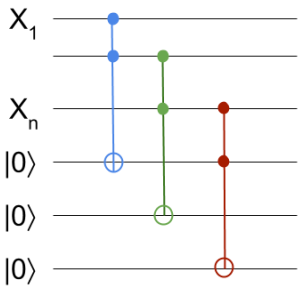
But why doesn't this contradict our starting assumption that g was easy to compute? Here's why: because a reversible circuit for g would at best give us a mapping like $|x\rangle|0\rangle \rightarrow |x\rangle|g(x)\rangle$, a mapping that leaves x around afterward. And inverting *that* mapping will only take us from $g(x)$ to x if we know x already, so it's no help in breaking the one-way function.

OK, but this still leaves the question: how do we even efficiently implement the mapping $|x\rangle|0\rangle \rightarrow |x\rangle|g(x)\rangle$, if given a small *non*-reversible circuit for g?



In the last lecture, we saw how it's possible to take any non-reversible circuit and simulate it by a reversible one, by using Toffoli gates to simulate NAND gates.

The trouble with that is that, along the way to the final answer, the construction also produces all sorts of undesired results in the intermediate bits—the technical name for this is *garbage*. Yes, really.



Why is garbage such a problem for quantum computing? Because garbage can prevent the desired interference patterns from showing up—and the whole point of quantum algorithms is to create those interference patterns.

For example, what's the difference between having $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and having $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$, where we treat the second qubit as unwanted garbage?

The garbage is entangled with the first qubit, the qubit we *do* want. So, in the second case, when we look at the first qubit only, we see the maximally mixed state rather than the $|+\rangle$ state that we wanted.

So to return to the question: suppose you have a circuit to compute f. How you we get a circuit that maps $\sum_x \alpha_x |x, 0\rangle \rightarrow \sum_x \alpha_x |x, f(x)\rangle$ without all the garbage? Back in the 1970s, Charles Bennett invented a trick for this called…

Uncomputing

It's simple, though also strange when you first see it. The trick to getting rid of garbage is to run computations *forward and then in reverse*.

Let's say I have some circuit C such that

$$C|x, 0, \ldots, 0\rangle = |x, \mathrm{gar}(x), f(x)\rangle,$$

where $\mathrm{gar}(x)$ is a generic term for all the garbage: that is, byproducts of the computation that I don't want. Then I do the following:

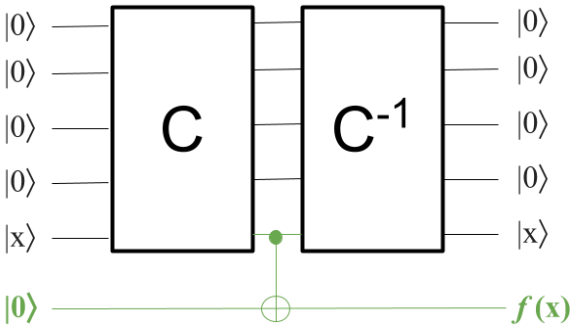
First run the circuit C, to get $|x, \mathrm{gar}(x), f(x)\rangle$.

Then cNOT the answer $f(x)$ into a register initialized to 0, to get $|x, \mathrm{gar(x)}, f(x), f(x)\rangle$

(in other words, make a copy of $f(x)$ in a safe place)

Finally, run the inverse circuit, C^{-1}, to get $|x, 0, \ldots, 0, f(x)\rangle$ or just $|x, f(x)\rangle$ if we ignore the 0 qubits.

The reason why we can copy x, in spite of the No-Cloning Theorem, is that we're assuming that x is a classical answer. This won't work if the output is a general quantum state.



This justifies the quantum query model because if we can compute f at all, then we do have the ability to map $|x, a\rangle$ to $|x, a \oplus f(x)\rangle$. (Note that in the uncomputing process, if the "safe" register was initialized to some arbitrary a rather than to 0, then it would end up in the $a \oplus f(x)$state .)

With that out of the way, we're finally ready to talk about some quantum algorithms.

Deutsch's Algorithm

was, by some definition, the first quantum algorithm proposed, in the mid-1980s. It achieves something unimpressive, except for the fact of being possible at all: namely, it computes the parity of two bits using only one (superposed) query to the bits.

In more detail, we're given two unknown bits, b_0 and b_1.

Given an index $x \in \{0, 1\}$, our oracle returns the bit. i.e. $f(x) = b_x$

What we want to know is, "What is the parity of these bits?"

Parity is whether the bits have different values, so $b_0 + b_1 \pmod 2$ or $b_0 \oplus b_1$

Classically, this would clearly take two queries since we need to know both bits. So using a quantum algorithm, how do we do it in one?

Simple: we start with a qubit at $|0\rangle$, Hadamard it to get $|+\rangle$, then apply a phase query, which applies a phase change to each branch of the superposition depending on the value of the function. (Here

we're using the result from earlier in this lecture, that we can use a single "ordinary" query to simulate the effect of a single phase query.) This yields:

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \rightarrow \frac{1}{\sqrt{2}}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right)$$

Let's factor out $f(0)$ to get

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

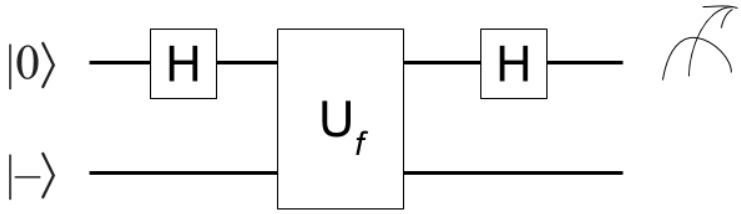
So now if $f(0) = f(1)$ we get $(-1)^{f(0)}\frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$

while if $f(0) \neq f(1)$ we get $(-1)^{f(0)}\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)$

We can ignore the phase out front since global phase doesn't affect measurement, and then Hadamard again to get our quantum states back in the $\{|0\rangle, |1\rangle\}$ basis.

Now in the $f(0)=f(1)$ case we get $|0\rangle$
and in the $f(0)\neq f(1)$ case we get $|1\rangle$.

The complete quantum circuit is shown on the right.



Note that, if we wanted the parity of an n-bit input string x, Deutsch's algorithm would let us get that with $n/2$ queries. We simply need to break x up into $n/2$ blocks of 2 bits each, use Deutsch's algorithm to learn the parity of each block B (using $n/2$ queries in total), then calculate the parity of all the 's. This last step doesn't increase the query complexity, because it doesn't involve making any additional queries to x.

OK, if we understand Deutsch's algorithm, then let's next see a generalization, called…

The Deutsch-Jozsa Algorithm

Suppose we have a black box that computes a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, and suppose we're promised that f is either:

- a constant function — All outputs are 0 or all outputs are 1
- a balanced function — There are the same number of 0 outputs as 1 outputs

The problem is to decide which.

Classically, deterministically, you could solve this problem by examining any $2^{n-1}+1$ values of the function. If all the values match, then the function is constant; otherwise the function is balanced. If you want no possibility of error, then it's not hard to see that this is the best you can do.

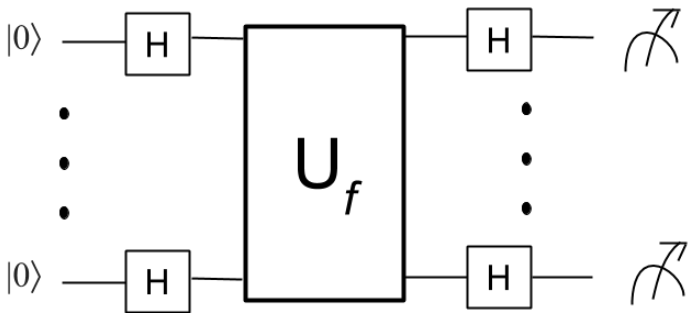
Of course, you can do much better by using random sampling. On average, you'd need maybe 5 or 6 queries—or at any rate, a constant number—to get an answer with small probability of error. If all of your samples match, you can guess that the function is constant; if they don't, you know that it's balanced.

What the Deutsch-Jozsa algorithm does, is to solve the problem *perfectly* (that is, with zero probability of error), with only one quantum query. That's something that isn't possible in the classical world.

Truth is, this speedup still isn't all that impressive, because the classical probabilistic algorithm

is nearly as fast, and would be perfectly fine in practice. This helps to explain why, until 1994 or so, most people didn't care about quantum computing. To whatever extent they looked into it at all, they figured all quantum speedups would be in the same vein.

Anyway, here's the quantum circuit for Deutsch-Jozsa:



You'll begin to notice that some patterns appear a lot in quantum algorithms.

- You start by putting everything in superposition
- You then query a function f—in this case, mapping each basis state $|x\rangle$ to $(-1)^{f(x)}|x\rangle$
- You then apply a change a basis (in this case, another round of Hadamards)
- Finally, you measure to get the information you want to know

If you can't figure out what to do next in a quantum algorithm, a round of Hadamards is always a good guess!

So given this circuit (call it C), let's ask: what's the probability of getting back the state $|00\ldots0\rangle$?

In other words, what is $|\langle 00\ldots0|C|00\ldots0\rangle|^2$?

Well, the Hadamard gate maps $|0\rangle \rightarrow |+\rangle$ and $|1\rangle \rightarrow |-\rangle$.

We can summarize this by saying that, for a bit $x\epsilon\{0,1\}$, it maps $|x\rangle \rightarrow \frac{|0\rangle+(-1)^x|1\rangle}{\sqrt{2}}$

So given a string $|x(1),\ldots,x(n)\rangle$, Hadamarding all n of the qubits produces the state

$$\frac{|0\rangle+(-1)^{x(1)}|1\rangle}{\sqrt{2}} \bigotimes \frac{|0\rangle+(-1)^{x(2)}|1\rangle}{\sqrt{2}} \bigotimes \cdots \bigotimes \frac{|0\rangle+(-1)^{x(n)}|1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2^n}} \sum_{y\epsilon\{0,1\}^n} (-1)^{x\cdot y}|y\rangle .$$

This is a fact that we'll also have several occasions to use in the next lecture.

Note: Here $x\cdot y$ denotes the inner product. The formula is saying that we pick up a –1 phase for every i such that $x_i = y_i = 1$.

Now, coming back to the Deutsch-Jozsa algorithm, after we Hadamard all n of the qubits and then query the oracle, we get the state

$$\frac{1}{\sqrt{2^n}} \sum_{x\epsilon\{0,1\}} (-1)^{f(x)}|x\rangle$$

So by our previous result, after the second round of Hadamards we get

$$\frac{1}{\sqrt{2^n}} \sum_{x\epsilon\{0,1\}^n} (-1)^{f(x)} \frac{1}{\sqrt{2^n}} \sum_{y\epsilon\{0,1\}^n} (-1)^{x\cdot y}|y\rangle$$

Rather than simplifying this entire sum, let's take a shortcut by just asking: what is the amplitude for the basis state $y = |00\ldots0\rangle$?

Well, it's $\frac{1}{2^n} \sum_{x \epsilon \{0,1\}} (-1)^{f(x)}$.

Now, what does this amplitude have to do with whether f is constant or balanced?
Well, if f is constant, then the above amplitude is either 1 (if f is identically 0) or –1 (if f is identically 1).
On the other hand, if f is balanced, then the amplitude is 0.
So when we measure, if we see the outcome $|00\ldots0\rangle$ then we know that f is constant, and if we see any other outcome then we know that f is balanced! That was the Deutsch-Jozsa algorithm.

The first problem we'll see in the next lecture is the so-called *Bernstein-Vazirani problem*, for which there's a quantum algorithm that achieves a more impressive speedup than Deutsch-Jozsa does. And the speedups will continue to get more impressive from there.