Lecture 18, Tues March 28: Bernstein-Vazirani, Simon

We ended last time with the Deutsch-Jozsa problem. Today we'll start with another black-box problem for which quantum algorithms provide an advantage:

The Bernstein-Vazirani Problem

We're given access to a black box function $f : \{0,1\}^n \to \{0,1\}$

We're promised that $f(x) = s \cdot x \pmod 2$ for some secret string $s \in \{0,1\}^n$

The problem is to find s.

Classically, you could get an answer one bit at a time by querying all the strings of Hamming weight one: for example, $f(1000) = s_1$

$f(0100) = s_2$

$f(0010) = s_3$

$f(0001) = s_4$

But no classical algorithm can do better than this, since each query can only provide one bit of information about s.

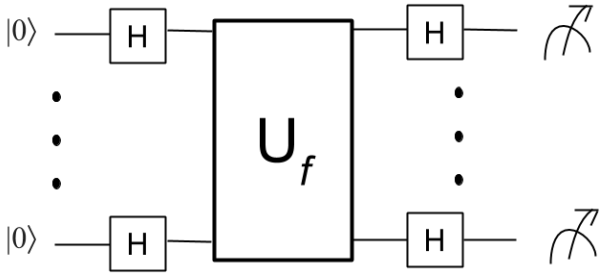
The Bernstein-Vazirani algorithm, however, solves the problem quantumly using only one query (!).

The Bernstein-Vazirani Algorithm

We start with n qubits all in the $|0\rangle$ state.

We then Hadamard them all (what else?).

Next we query f (using the 'phase' type of query).



The question is:

How do we measure the resulting state in a way that gives us information about the secret string s?

We can reuse work from the last lecture…

We know that Hadamard gate maps $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$ (and vice versa)

i.e. it does $|b\rangle \leftrightarrow \dfrac{|0\rangle + (-1)^b|1\rangle}{\sqrt{2}}$

and so Hadamarding a string of bits gets us

$$|s_1, \ldots, s_n\rangle \leftrightarrow \frac{|0\rangle + (-1)^{s_1}|1\rangle}{\sqrt{2}} \bigotimes \cdots \bigotimes \frac{|0\rangle + (-1)^{s_n}|1\rangle}{\sqrt{2}}$$

We pick up a –1 factor only when s_i and x_i are both 1.

Now, note that we can write the current state of the Bernstein-Vazirani algorithm as

$$\frac{1}{\sqrt{2^n}} \sum_{x \epsilon \{0,1\}} (-1)^{s \cdot x} |x\rangle = \frac{|0\rangle + (-1)^{s_1}|1\rangle}{\sqrt{2}} \bigotimes \cdots \bigotimes \frac{|0\rangle + (-1)^{s_n}|1\rangle}{\sqrt{2}}$$

But this means that if we Hadamard all the qubits again, we'll change:

- The qubits that picked up a phase (i.e., for which $s_i = 1$) from $|-\rangle$ to $|1\rangle$.
- The qubits that *didn't* pick up a global phase (i.e., for which $s_i = 0$) from $|+\rangle$ to $|0\rangle$.

So from here we can simply measure the qubits in the $\{|0\rangle, |1\rangle\}$ basis to retrieve s.

You can see that Bernstein and Vazirani designed their problem around what a quantum computer would be able to do!

Don't tell anyone, but: this is actually pretty common in this field

So, only for $|s\rangle$ are all 2^n contributions to the amplitude of a measurement outcome pointing the same way. For all the other outcomes, the contributions interfere destructively, with equally many positive and negative terms (all of the same magnitude), so the total amplitude for each of those outcomes is 0.

With pretty much *every* quantum algorithm, a similar story can be told, about the contributions to the amplitude reinforcing each other only for the outcomes that we want.

On that note, let's next see…

Simon's Problem (1994)

The story goes that Simon looked at the quantum algorithms coming out, and he didn't believe that any of them would give a *real* speedup. Even the Bernstein-Vazirani problem is easy classically: a classical computer can find the n bits of s with n queries. Sure, the quantum algorithm needs only one query, but it also requires $O(n)$ gates, so maybe it's not that impressive.

Simon believed there was a limit that would prevent you from getting a "true" exponential speedup from a quantum computer, and he set out to prove it. What he ended up finding, instead, was that there *is* a true exponential speedup, at least for an artificial black-box problem. As we'll see, this then played a central role in subsequent progress in quantum algorithms: particularly Shor's algorithm, which came shortly afterward.

In Simon's problem, we're once again given an oracle function, this time mapping n bits to n bits:

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

We're promised that there's a secret string $s \neq 0^n$, such that

$$f(x) = f(y) \Leftrightarrow y = x \oplus s$$

for all inputs x, y, where the symbol $\oplus$ denotes bitwise XOR. The problem is to find the secret string *s*, by querying *f* as few times as possible.

Compared to Bernstein-Vazirani, here there's more freedom in the choice of function f. In Simon's problem, all we require is that f has a "hidden XOR mask": that is, a subset of bits such that when you flip the bits in that subset (but *only* when you do so), the output is unaffected.

What does this mean?

Let's do an example with 3-bit inputs, and with secret string $s = 110$.

Let's say we query f a few times and get…

$f(000) = 5$
$f(110) = 0$
$f(001) = 6$
$f(111) = 6$

We're given no information about how to interpret the outputs themselves, so it doesn't really matter whether we think of them as strings, integers, or whatever. The only thing that *does* matter is whether two inputs map to the same output.

Since $f(001) = f(111) = 6$, we know that $s = 001 \oplus 111 = 110$.

This is simple enough with 3 bits, but we're more interested in f's with, say, 1000-bit inputs. In that case, we claim that finding the secret string s is prohibitively expensive classically. How expensive? Well, it's not hard to show that any deterministic algorithm needs to query f at least $2^{n-1} + 1$ times, by an argument similar to the one that we used for Deutsch-Jozsa. But once again, the more relevant question is how many queries are needed for a *randomized* algorithm.

We claim that we can do a little bit better in that case, getting down to $O(2^{n/2})$ queries. This is related to the famous **Birthday Paradox**, which isn't really a paradox, it's more of a "birthday fact."

It says that, if you gather merely 23 people in a room, that's enough to have a ~50% probability of getting at least one pair of people who share a birthday. More generally, if there were n days in the year, then you'd need $\sqrt{n}$ about people in the room for a likely birthday collision. (At least, assuming birthdays are uniformly distributed, which they're not exactly: e.g., there are clusters of them about 9 months after major holidays.)

The takeaway here is that the number of pairs of people is what's important, and that scales quadratically with the number of people. Similarly, with Simon's problem, the number of pairs of inputs that could collide is what's important, and that grows quadratically with the number of inputs we check.

The Birthday Paradox is useful in cryptanalysis. For example, cryptographic hash functions need to make it intractable to find any two inputs x, y with the same hash value, $f(x) = f(y)$. But by using a "birthday attack"—i.e., repeatedly choosing a random input x, then comparing $f(x)$ against $f(y)$ for *every* previously queried input y, looking for a match—we can find a collision pair using a number of queries to f that scales only like the square root of the size of f's range. This is quadratically faster than one might have expected naïvely.

Whatever other structure it has, Simon's problem involves a two-to-one function in which we're looking for a collision pair—so it also admits a birthday attack. Roughly speaking, given two randomly-chosen inputs x and y, we'll observe $f(x) = f(y)$ with probability $p = \frac{1}{2^n - 1}$, and while these events aren't quite independent between the various (x, y) pairs, they're nearly so. Doing the calculation carefully, we find that we'll observe a collision with high probability after querying $\frac{1}{\sqrt{p}} \sim 2^{n/2}$ values of f.

Is there a better classical algorithm?

Let's prove that the answer is no. We'll use an **Adversary Argument**: basically,

"If my worst enemy got to choose f, what would he do?"

Presumably, he'd choose a secret string $s \neq 0^n$ uniformly at random among all possible s's, to make it as hard as possible to find an underlying structure in f. And then, perhaps, choose a random f among all those consistent with that s.

Now, once we fix such a strategy of the adversary, we can assume without loss of generality that the algorithm is deterministic. This is because a randomized algorithm is just a probabilistic mixture of deterministic algorithms, and there must be at least one algorithm in the mixture that does at least as well as the average! (This observation—together with the complementary observation that *all* randomized lower bounds can be proved in this way—is sometimes referred to as *Yao's minimax principle*.)

So the upshot is that we can view an optimal strategy as just a deterministic sequence of queries. Let the queried inputs be x_1, x_2, etc. Then the question is:

What information can we derive about *s* after the first *t* queries?

If we've found a collision pair, $f(x_i) = f(x_j)$ for some $i \neq j$, then we're done; we now know that $s = x_i \oplus x_j$. So let's assume that hasn't happened yet. Then all we can conclude about s is that $s \neq x_i \oplus x_j$ for every $i \neq j$. At most this rules out t^2 possible values of s, with all the other possibilities remaining equally consistent with what we've seen (i.e., having equal posterior probabilities). It follows that, *unless* we observe a collision, narrowing the possibilities down to a single s requires $\Omega(2^{n/2})$ queries. And the probability that we *do* observe a collision in the t^{th} query is only $\frac{t}{2^n - 1}$—so by the union bound, with high probability we won't observe any collisions at all in the first $\sim 2^{n/2}$ queries.

And that's the adversary argument: examining a "worst-case" distribution over f's gives us a lower bound of $\Omega(2^{n/2})$ queries for any randomized algorithm solving Simon's problem.

i.e. the birthday attack yields a quadratic speedup, but the problem still takes exponential time classically.

Quantumly, by contrast, we can solve Simon's problem with only $O(n)$ queries to f, by using...

Simon's Algorithm!

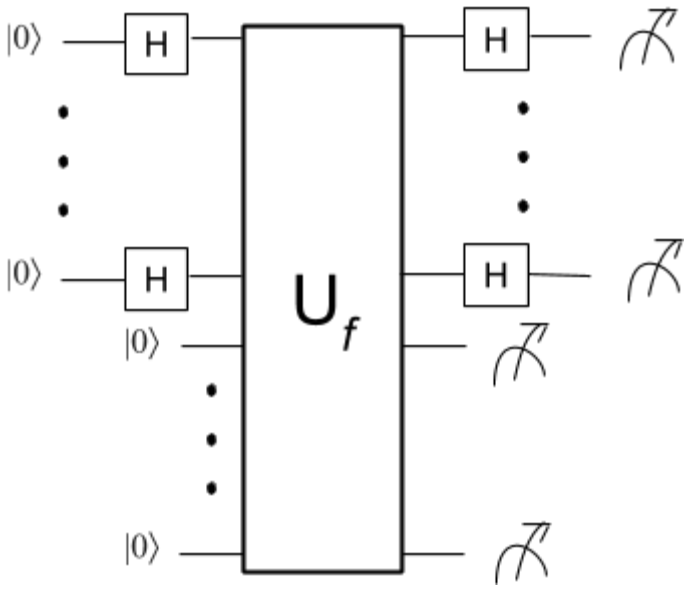
The algorithm follows a now-familiar pattern:

1. Start with n qubits all in the $|0\rangle$ state.
2. Hadamard them all.
3. Query f.

This yields the state

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

This time the function f has a large output, so we need to write out its values in a separate n-qubit



answer register, rather than just encoding it into the phase.

But even though we're giving more space to write out the answers $f(x)$, it's important to note that the answers themselves aren't what we care about!

Instead, we're only writing them out because by doing so, we create a desired interference pattern in the x register. Indeed at this point in the algorithm we could simply discard the $f(x)$ qubits, or do anything else we liked with them.

So for pedagogical simplicity, let's assume we now *measure* the $f(x)$ qubits. And let's assume that the result of the measurement is $|w\rangle$.

Keeping track of all of the w's we could have seen would've resulted in a mixed state.
Instead, we're just conditioning on a particular w.

Now how many different values are superposed in the input register?

By the partial measurement rule, we're left with an equal superposition over all the different x's that are consistent with the $f(x)$ value that we observed, namely w. In Simon's problem, there are necessarily two such x's. In other words, we're left with a superposition $\frac{|x\rangle+|y\rangle}{\sqrt{2}}$ such that $f(x) = f(y) = w$. By the Simon promise, this means in particular that $y = x \oplus s$.

So what is this state good for?

First, observe that if we could just measure $\frac{|x\rangle+|y\rangle}{\sqrt{2}}$ twice, then with high probability we'd first get x and then y—so then bitwise-XORing the two strings would give us the secret string s, and we'd be done! Alas, in quantum mechanics we only get one chance to measure a state, so we'll see either $|x\rangle$ *or* $|y\rangle$, which tells us nothing about s. We could of course repeat the whole algorithm from the beginning. But if we did, then with overwhelming probability we'd get a different w, corresponding to a new pair. (x, y)

So we'll need to be more clever—although not *that* much more! In particular, let's see what happens if we measure the qubits of $\frac{|x\rangle+|y\rangle}{\sqrt{2}}$ in the Hadamard basis.

What's the effect of Hadamarding all the qubits on $\frac{|x\rangle+|y\rangle}{\sqrt{2}}$?

Well for starters, we saw in the last lecture that Hadamarding all qubits maps the basis state $|x\rangle$ to

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

and maps $|y\rangle$ to

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

By linearity, this means that doing the same thing to an equal superposition of $|x\rangle$ and $|y\rangle$ must give

$$H^{\otimes n}\frac{|x\rangle + |y\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2^{n+1}}}\sum_{z\in\{0,1\}^n}\left[(-1)^{x\cdot z} + (-1)^{y\cdot z}\right]|z\rangle$$

Now let's measure the above state in the standard basis.

Which z's could we get when we do so?

For a given z to be observed, it must have a nonzero amplitude. This means that $(-1)^{x\cdot z}$ and $(-1)^{y\cdot z}$ must be equal, which occurs if and only if

$$x\cdot z = y\cdot z \pmod 2.$$

Or rewriting this equation a bit:

$$x\cdot z + y\cdot z = 0 \pmod 2$$
$$\Rightarrow (x\oplus y)\cdot z = 0 \pmod 2$$
$$\Rightarrow s\cdot z = 0 \pmod 2$$

Note that every z satisfying the above equation will appear with the same probability as every other, namely $1/2^{n-1}$. So, what we get when we measure is an n-bit string z, chosen uniformly at random among all the 2^{n-1} strings whose inner product with s is 0. In other words: we haven't yet learned s itself, but we've learned a bit of information about s, which I hope you'll grant is something!

But what if we really want s itself? In that case, we can just repeat Simon's algorithm over and over, starting from the beginning! This will give us a collection of strings , which are uniformly random and independent of each other, subject to satisfying the equations

$$s\cdot z_1 = 0 \pmod 2$$
$$s\cdot z_2 = 0 \pmod 2$$
$$\vdots$$
$$s\cdot z_k = 0 \pmod 2$$

After we've repeated enough times, what could we tell a classical computer to do with this information?

Well, suppose $k = n + c$, where c is some constant. Then we now have a collection of linear equations in n unknowns, over a finite field with two elements. We can solve this system efficiently using a classical computer.

Through an algorithm called "Run Matlab."

Or Gaussian elimination, taking $O(n^3)$ time.

Or if you want to get all theoretical about it, the fastest known algorithm—whose constant-factor overheads make it useless in practice—takes $O(n^{2.373})$ time.

It's not hard to do a probabilistic analysis showing that, after we've seen slightly more than n equations, with overwhelming probability we'll be left with a system that has exactly two solutions: namely, 0^n and s itself. We can throw away 0^n, because we assumed $s \neq 0^n$. So that leaves us with s.

That's Simon's algorithm, which solved Simon's problem using only $O(n)$ queries to f plus a polynomial amount of side computation, as compared to the $\Omega(2^{n/2})$ queries that are provably needed classically.

Does Simon's algorithm have a deterministic counterpart?

Yes, one can modify the algorithm so that it succeeds with certainty rather than "merely" overwhelming probability. We won't go into the details.

So why doesn't this just prove that quantum algorithms are better?

It's sort of tricky to translate Simon's algorithm into "real-world" consequences. To get a speedup over classical computing in terms of the sheer number of gates, or computational steps, we'd need some small circuit to compute a function f that was actually like our magical Simon function f has (i.e., that satisfied the same promise). For example, $f(x) = Ax$ for some rank-(n–1) Boolean matrix A would do the trick.

But the difficulty in claiming that we're getting a quantum speedup this way is that, once we pin down the details of how we're computing f—so for example, the actual matrix A—we then need to compare against classical algorithms that know those details as well. And as soon as we reveal the innards of the black box, the odds of an efficient classical solution become much higher! So for example, if we knew the matrix A, then we could solve Simon's problem in classical polynomial time just by calculating A's nullspace. More generally, it's not clear whether anyone to this day has found a straightforward "application" of Simon's algorithm, in the sense of a class of efficiently computable functions f that satisfy the Simon promise, *and* for which any classical algorithm plausibly needs exponential time to solve Simon's problem, even if the algorithm is given the code for f.

So the story goes that Simon wrote a paper about this theoretical black-box problem with an exponential quantum speedup, and the paper got rejected. But there was one guy who was like, "Hey, this is interesting." He figured that if you changed a few aspects of what Simon was doing, you could get a quantum algorithm to find the periods of periodic functions, which would in turn let you do all sorts of fun stuff.

That guy was Peter Shor.