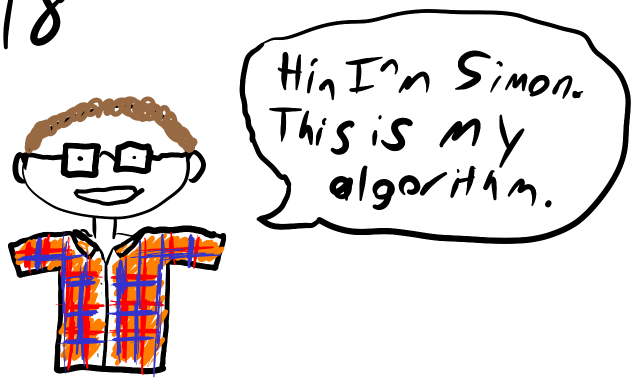
CMPT 476 Lecture 18

Simon's algorithm

(Not actually Simon, probably)



So far the algorithms we've seen have been pretty trivial. On the one hand, it seems the "problems" were reverse-engineered based on the interference patterns generated from the hadamard transform. On the other hand, we didn't even really get a non-trivial advantage over classical algorithms on those problems.

Today we'll look at the first non-trivial quantum algorithm which gives an exponential speed-up in the black-box model over classical computations, albeit for another synthetic problem.

(Simon's algorithm)

While the Bernstein–Vazirani algorithm gives a query complexity speed-up compared to probabilistic algorithms, in the end it's only linear since we have an $O(n)$ classical algorithm

$$f(100\cdots0) = s_1$$
$$f(010\cdots0) = s_2$$
$$f(001\cdots0) = s_3$$
$$\vdots$$
$$f(000\cdots1) = s_n$$

Moreover, BV itself takes $O(n)$ total work so it's not a "real" speed-up. In 1994, Simon set out to show that there could be no "real" speed-up and instead found the first "truly" exponential speed-up.

not really...

Simon's Problem

Input: A function $f: \{0,1\}^n \longrightarrow \{0,1\}^n$

Promise: $\exists s \in \{0,1\}^n$ such that $s \neq 0$ and $f(x) = f(y)$ if & only if

$$x = y \quad \text{or} \quad x = y \oplus s$$

Goal: Find s.

Simon's problem is artificial (like all problems we've seen so far), but it's closely related to and based on the hardness of finding collisions

$$x, y \text{ such that } f(x) = f(y)$$

Cryptographic hash functions like SHA are based on the hardness of finding collisions too.

First: how do we solve Simon classically?

Ex.

Suppose we have $f: \{0,1\}^3 \to \{0,1\}^3$ with the truth table

x	$f(x)$
000	101
001	011
010	111
011	010
100	011
101	111
110	011
111	101

Note- the values of f don't matter, only collisions

If we sample $f(001) = 011$ and $f(110) = 011$, then we've found $x = 001$ & $y = 110$ such that $f(x) = f(y)$. Since $f(x) = f(y)$ iff $x = y \oplus s$, then

$$s = x \oplus y = 001 \oplus 110 = 111$$

If however $f(x) \neq f(y)$, we've learned nothing about s. We know there are collisions since $s \neq 0$. So, the question is how many queries do we need to find one with high probability?

Fact

Any probabilistic algorithm solving Simon's problem with probability $\frac{1}{2}$ must use $\Omega(\sqrt{2^n})$ queries.
(at least)

This number comes from the birthday paradox:

The probability one pair out of 23 people share a birthday is $> 50\%$

In general, to find a collision out of N uniformly distributed possibilities with $>50\%$ probability, you need

$$\approx \sqrt{N} \text{ samples.}$$

(A quantum algorithm)

Simon's algorithm is our first one which is meaningfully different from Deutsch's. It uses interference on collisions

$$|x\rangle, \quad |y = x \oplus s\rangle \text{ where } f(x) = f(y)$$

to repeatedly sample vectors orthogonal to s.

We first recall some facts from linear algebra.
Note that $\{0,1\}^n = \mathbb{Z}_2^n$ is a vector space.

(Orthogonal subspace)

Given a subspace S of a vector space V, denote by $S^\perp$ the orthogonal complement of S in V.

$$S^\perp = \{v \in V \mid s \cdot v = 0 \ \forall s \in S\}$$

The orthogonal complement of S is a subspace of V, and

$$\dim(V) = \dim(S) + \dim(S^\perp)$$

(Finding a complement)

Let $s \in \mathbb{Z}_2^n$. Then $s^\perp = (\operatorname{span}\{s\})^\perp$. Suppose we can randomly sample from $s^\perp$ — how could we find s?

Idea:

Take many samples $x_1 \cdots x_k$ and solve the linear system for $s = s_1 \cdots s_n$

$$\begin{bmatrix} x_1^T \\ \vdots \\ x_k^T \end{bmatrix} \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} x_1 \cdot s \\ \vdots \\ x_k \cdot s \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

How many x's do we need? Well if they're linearly independent then $n-1$ since

$$\dim(s^\perp) = \dim(\mathbb{Z}_2^n) - \dim(\operatorname{span}\{s\})$$
$$= n - 1$$

(Simon's algorithm, high-level)

At a high-level, Simon's algorithm works like this

1. Set $A = []$
2. While $\mathrm{rank}(A) < n-1$ do:
   3. Prepare a uniform superposition $\frac{1}{\sqrt{|S^\perp|}} \sum_{z \in S^\perp} |z\rangle$
   4. Measure to get $z_i \in S^\perp$

   quantum part

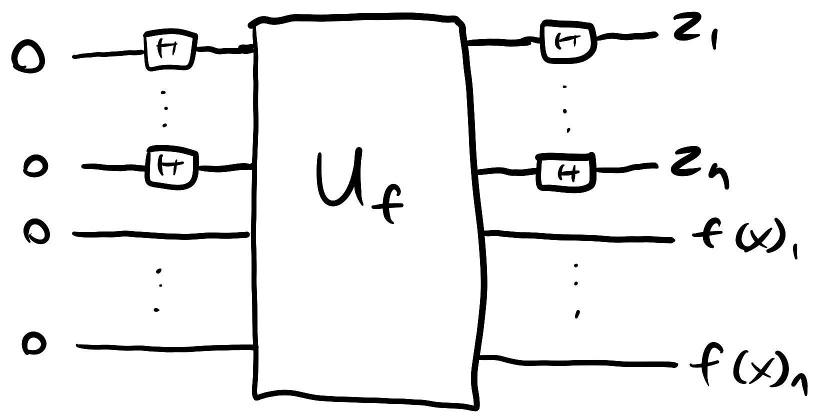
   5. Append row z_i to A
6. Solve $As = 0$ for s

(The quantum part)

The genius of Simon's algorithm is in observing that interference can be used to prepare the state

$$\frac{1}{\sqrt{|S^\perp|}} \sum_{z \in S^\perp} |z\rangle$$

The idea is to generate a uniform superposition of $x \in \{0,1\}^n$, then pair up collisions with a call to f.



Here we use a **State oracle** $U_f|x\rangle|0\rangle = |x\rangle|f(x)\rangle$, so the state after U_f is

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

Now, suppose we measure the second register to get one particular value $w = f(x)$. The resulting state will be a (normalized) sum over all $y \in \{0,1\}^n$ such that $f(y) = w$. Observe that there are exactly two strings x, $x \oplus s$ such that $f(x) = f(y)$.

Why? Suppose $\exists y, z \in \{0,1\}^n$ s.t. $f(x) = f(y) = f(z)$. Then $x = s \oplus y = s \oplus z \Rightarrow y \oplus z = 0 \Rightarrow y = z$. Likewise, since $s \neq 0$, $x \oplus s \neq x$.

So after measuring $f(x)$ the state is

$$\frac{1}{\sqrt{2}} \left(|x\rangle + |x \oplus s\rangle \right) |f(x)\rangle, \quad x \in \{0,1\}^n$$

Now what is the effect of the Hadamard gates? Since $H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{x \cdot z} |z\rangle$, we have

$$\left(H^{\otimes n} \frac{1}{\sqrt{2}} \left(|x\rangle + |x \oplus s\rangle \right) \right) |f(x)\rangle$$

$$= \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2^n}} \sum_z (-1)^{x \cdot z} |z\rangle + \frac{1}{\sqrt{2^n}} \sum_z (-1)^{(x \oplus s) \cdot z} |z\rangle \right) |f(x)\rangle$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_z \left[(-1)^{x \cdot z} + (-1)^{(x \oplus s) \cdot z} \right] |z\rangle |f(x)\rangle$$

Now which $|z\rangle$'s have non-zero amplitude? If $x \cdot z \neq (x \oplus s) \cdot z$, then

$$(-1)^{x \cdot z} + (-1)^{(x \oplus s) \cdot z} = 0$$

But $(x \oplus s) \cdot z = x \cdot z \oplus s \cdot z$, so $x \cdot z = (x \oplus s) \cdot z$ implies

$$s \cdot z = 0, \text{ or } z \in s^\perp !$$

So, if we measure the first register, we get some $z \in s^\perp$ as required. Or, in our original language the final quantum state from which we sample is

$$\frac{1}{\sqrt{2^{n-1}}} \sum_{z \in s^\perp} (-1)^{x \cdot z} |z\rangle |f(x)\rangle$$

(The role of measuring $|f(x)\rangle$)

One thing to note is that the measurement of $|f(x)\rangle$ is actually irrelevant — it only serves to simplify the analysis. If we don't measure $f(x)$, the only difference in the final state is that it is a superposition over all possible x's:

$$N \sum_{x\in\{0,1\}^n} \sum_{z\in S^\perp} (-1)^{x\cdot z} |z\rangle|f(x)\rangle$$

normalization factor

Measuring the first register still gives us some $z \in S^\perp$ since for any x, y where $f(x)=f(y)$,

$$|z\rangle|f(x)\rangle \text{ and } |z\rangle|f(y)\rangle$$

have the same phase, since $x\cdot z = (x\oplus s)\cdot z = y\cdot z$.

(Complexity analysis)

Since Simon's algorithm involves a loop where we repeatedly sample from $S^\perp$ until we have a basis, we need to know how many times we need to repeat. This part makes it a probabilistic runtime because we could technically get the sequence of samples

$$z, z, z, z, z, \ldots \quad \text{(i.e. same string each time)}$$

In reality, each time we sample, we get something not in the span of the previous samples with high probability. To see why, think about the extremal cases: when we've seen 1 sample, or $n-2$ samples.

- 1 sample z: $|\text{span}\{z\}| = 1$, $|\mathbb{Z}_2^n - \text{span}\{z\}| = 2^n - 1$
- $n-2$ samples $z_1, \ldots, z_{n-2}$: $|\text{span}\{z_i\}| \leq 2^{n-2}$, $|\mathbb{Z} - \text{span}\{z_i\}| \geq 2^n - 2^{n-2} = 3\cdot 2^{n-2}$

So even when we already have $n-2$ lin. ind. samples, we get the last one with 3/4 probability!

Fact the expected number of samples needed is n

This gives Simon's algorithm an expected quantum query complexity of $O(n)$ with $O(n^3)$ additional classical work to solve the linear system.

Alternatively, we could just sample $m \in O(n)$ times and if we haven't found a basis yet just give up. In practice, $m \approx n+1$ is enough to give a high probability of success. This is a common idea in probabilistic algorithms, where a deterministic algorithm with a probabilistic runtime can be turned into a probabilistic algorithm with a deterministic runtime.

(Recap of algorithms so far)

So far we've seen a series of algorithms with increasing separation between their classical and quantum query complexities. To summarize,

Problem	Classical query complexity	Probabilistic query complexity	Quantum query complexity
Deutsch	2	2	1
Deutsch-Jozsa	$O(2^n)$	2	1
Bernstein-Vazirani	$O(n)$	$O(n)$	1
Simon	$O(2^n)$	$O(2^n)$	$O(n)$

However, this is just the query complexity — we need to ask whether such a speed-up is meaningful in practice.

(Is query complexity meaningful?)

It depends — in particular on whether we can learn the property more efficiently by looking at the implementation of f. In polynomial interpolation, say we have a sub-routine in python which implements f. If this sub-routine is something like

power operator

```
def f(x):
    return a_0 + a_1*x + ... + a_d*(x**d)
```

then we can get the answer in $O(1)$ by opening up the black-box. On the other hand, f may be implemented by some other means — e.g. some financial algorithm — and we want to learn a property which is not apparent from the implementation.

Since we don't have true black boxes in quantum computation, the true complexity is relative to the implementation of the oracle, and to give a speed-up over classical computing it must give a speed-up over any classical algorithm given the same or a comparable implementation of f. That rules out a quantum speed-up for the algorithms we've seen over any implementation that explicitly uses the hidden string. Other options for the function in Simon's algorithm for example are

$$f(x) = Ax, \quad \mathrm{rank}(A) = n-1$$

In this case, the best classical algorithm runs in time $O(n^3)$ by gaussian elimination, so there can be no (non-trivial) quantum speed-up.

Next we'll start to look at algorithms which offer true speed-ups, starting with an algorithm inspired by Simon's...

SHOR!