Simon's algorithm

Simon's algorithm is a quantum query algorithm for a problem known as *Simon's problem*. This is a promise problem with a flavor similar to the Deutsch-Jozsa and Bernstein-Vazirani problems, but the specifics are different.

Simon's algorithm is significant because it provides an *exponential* advantage of quantum over classical (including probabilistic) algorithms, and the technique it uses inspired Peter Shor's discovery of an efficient quantum algorithm for integer factorization.

Simon's problem

The input function for Simon's problem takes the form

$$f : \Sigma^n \to \Sigma^m$$

for positive integers n and m. We could restrict our attention to the case $m = n$ in the interest of simplicity, but there's little to be gained in making this assumption — Simon's algorithm and its analysis are basically the same either way.

> **Simon's problem**
> Input: a function $f : \Sigma^n \to \Sigma^m$
> Promise: there exists a string $s \in \Sigma^n$ such that $[f(x) = f(y)] \Leftrightarrow [(x = y) \vee (x \oplus s = y)]$ for all $x, y \in \Sigma^n$
> Output: the string s

We'll unpack the promise to better understand what it says momentarily, but first let's be clear that it requires that f has a very special structure — so most functions won't satisfy this promise. It's also fitting to acknowledge that this problem isn't intended to have practical importance. Rather, it's a somewhat artificial problem tailor-made to be easy for quantum computers and hard for classical computers.

There are two main cases: the first case is that s is the all-zero string 0^n, and the second case is that s is not the all-zero string.

- Case 1: $s = 0^n$. If s is the all-zero string, then we can simplify the if and only if statement in the promise so that it reads $[f(x) = f(y)] \Leftrightarrow [x = y]$. This is equivalent to f being a one-to-one function.
- Case 2: $s \neq 0^n$. If s is not the all-zero string, then the promise being satisfied for this string implies that f is *two-to-one*, meaning that for every possible output string of f, there are exactly two input strings that cause f to output that string. Moreover, these two input strings must take the form w and $w \oplus s$ for some string w.

It's important to recognize that there can only be one string s that works if the promise is met, so there's always a unique correct answer for functions that satisfy the promise.

Here's an example of a function taking the form $f : \Sigma^3 \rightarrow \Sigma^5$ that satisfies the promise for the string $s = 011$.

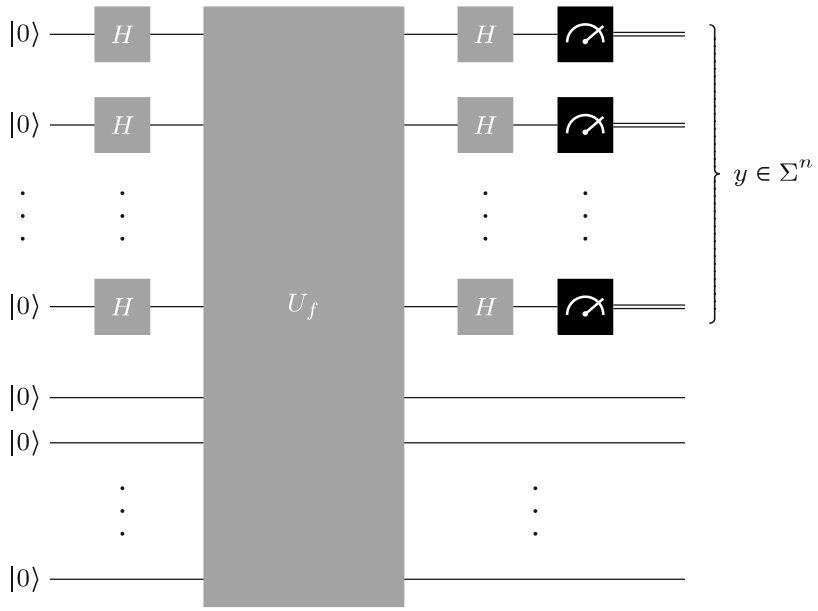
$$
\begin{aligned}
f(000) &= 10011 \\
f(001) &= 00101 \\
f(010) &= 00101 \\
f(011) &= 10011 \\
f(100) &= 11010 \\
f(101) &= 00001 \\
f(110) &= 00001 \\
f(111) &= 11010
\end{aligned}
$$

There are 8 different input strings and 4 different output strings, each of which occurs twice — so this is a two-to-one function. Moreover, for any two different input strings that produce the same output string, we see that the bitwise XOR of these two input strings is equal to 011, which is equivalent to saying that either one of them equals the other XORed with s.

Notice that the only thing that matters about the actual output strings is whether they're the same or different for different choices of input strings. For instance, in the example above, there are four strings $(10011, 00101, 00001,$ and $11010)$ that appear as outputs of f. We could replace these four strings with different strings, so long as they're all distinct, and the correct solution $s = 011$ would not change.

Algorithm description

Here's a quantum circuit diagram representing Simon's algorithm.



To be clear, there are n qubits on the top that are acted upon by Hadamard gates and m qubits on the bottom that go directly into the query gate. It looks very similar to the algorithms we've already discussed in the lesson, but this time there's no phase kickback; the bottom m qubits all go into the query gate in the state $|0\rangle$.

To solve Simon's problem using this circuit will actually require several independent runs of it followed by a classical post-processing step, which will be described later after the behavior of the circuit is analyzed.

Analysis

The analysis of Simon's algorithm begins along similar lines to the Deutsch-Jozsa algorithm. After the first layer of Hadamard gates is performed on the top n qubits, the state becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \Sigma^n} |0^m\rangle |x\rangle.$$

When the U_f is performed, the output of the function f is XORed onto the all-zero state of the bottom m qubits, so the state becomes

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

When the second layer of Hadamard gates is performed, we obtain the following state by using the same formula for the action of a layer of Hadamard gates as before.

$$\frac{1}{2^n} \sum_{x \in \Sigma^n} \sum_{y \in \Sigma^n} (-1)^{x \cdot y} |f(x)\rangle |y\rangle$$

At this point, the analysis diverges from the ones for the previous algorithms in this lesson.

We're interested in the probability for the measurements to result in each possible string $y \in \Sigma^n$. Through the rules for analyzing measurements described in the *Multiple systems* lesson of the *Basics of quantum information* course, we find that the probability $p(y)$ to obtain the string y is equal to

$$p(y) = \left\| \frac{1}{2^n} \sum_{x \in \Sigma^n} (-1)^{x \cdot y} |f(x)\rangle \right\|^2.$$

To get a better handle on these probabilities, we'll need just a bit more notation and terminology. First, the *range* of the function f is the set containing all of its output strings.

$$\mathrm{range}(f) = \{f(x) : x \in \Sigma^n\}$$

Second, for each string $z \in \mathrm{range}(f)$, we can express the set of all input strings that cause the function to evaluate to this output string z as $f^{-1}(\{z\})$.

$$f^{-1}(\{z\}) = \{x \in \Sigma^n : f(x) = z\}$$

The set $f^{-1}(\{z\})$ is known as the *preimage* of $\{z\}$ under f. We can define the preimage under f of any set in place of $\{z\}$ in an analogous way — it's the set of all elements that f maps to that set. (This notation should not be confused with the *inverse* of the function f, which may not exist. The fact that the argument on the left-hand side is the set $\{z\}$ rather than the element z is the clue that allows us to avoid this confusion.)

Using this notation, we can split up the sum in our expression for the probabilities above to obtain

$$p(y) = \left\| \frac{1}{2^n} \sum_{z \in \text{range}(f)} \left(\sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right) |z\rangle \right\|^2 .$$

Every string $x \in \Sigma^n$ is represented exactly once by the two summations — we're basically just putting these strings into separate buckets depending on which output string $z = f(x)$ they produce when we evaluate the function f, and then summing separately over all the buckets.

We can now evaluate the Euclidean norm squared to obtain

$$p(y) = \frac{1}{2^{2n}} \sum_{z \in \text{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 .$$

To simplify these probabilities further, let's take a look at the value

$$\left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 \tag{1}$$

for an arbitrary selection of $z \in \text{range}(f)$.

If it happens to be the case that $s = 0^n$, then f is a one-to-one function and there's always just a single element $x \in f^{-1}(\{z\})$, for every $z \in \text{range}(f)$. The value of the expression (1) is 1 in this case.

If, on the other hand, $s \neq 0^n$, then there are exactly two strings in the set $f^{-1}(\{z\})$. To be precise, if we choose $w \in f^{-1}(\{z\})$ to be any one of these two strings, then the other string must be $w \oplus s$ by the promise in Simon's problem. Using this observation we can simplify (1) as follows.

$$\begin{aligned}
\left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 &= \left| (-1)^{w \cdot y} + (-1)^{(w \oplus s) \cdot y} \right|^2 \\
&= \left| (-1)^{w \cdot y} \Big(1 + (-1)^{s \cdot y} \Big) \right|^2 \\
&= \left| 1 + (-1)^{y \cdot s} \right|^2 \\
&= \begin{cases} 4 & y \cdot s = 0 \\ 0 & y \cdot s = 1 \end{cases}
\end{aligned}$$

So, it turns out that the value (1) is independent of the specific choice of $z \in \mathrm{range}(f)$ in both cases.

We can now finish off the analysis by looking at the same two cases as before separately.

- Case 1: $s = 0^n$. In this case the function f is one-to-one, so there are 2^n strings $z \in \mathrm{range}(f)$, and we obtain

  $$p(y) = \frac{1}{2^{2n}} \cdot 2^n = \frac{1}{2^n}.$$

  In words, the measurements result in a string $y \in \Sigma^n$ chosen uniformly at random.

- Case 2: $s \neq 0^n$. In this case f is two-to-one, so there are 2^{n-1} elements in $\mathrm{range}(f)$. Using the formula from above we conclude that the probability to measure each $y \in \Sigma^n$ is

  $$p(y) = \frac{1}{2^{2n}} \sum_{z \in \mathrm{range}(f)} \left| \sum_{x \in f^{-1}(\{z\})} (-1)^{x \cdot y} \right|^2 = \begin{cases} \frac{1}{2^{n-1}} & y \cdot s = 0 \\ 0 & y \cdot s = 1 \end{cases}$$

  In words, we obtain a string chosen uniformly at random from the set $\{y \in \Sigma^n : y \cdot s = 0\}$, which contains 2^{n-1} strings. (Because $s \neq 0^n$, exactly half of the binary strings of length n have binary dot product 1 with s and the other have binary dot product 0 with s, as we already observed in the analysis of the Deutsch-Jozsa algorithm for the Bernstein-Vazirani problem.)

Classical post-processing

We now know what the probabilities are for the possible measurement outcomes when we run the quantum circuit for Simon's algorithm. Is this enough information to determine s?

The answer is yes, provided that we're willing to repeat the process several times and accept that it could fail with some probability, which we can make very small by running the circuit enough times. The essential idea is that each execution of the circuit provides us with statistical evidence concerning s, and we can use that evidence to find s with very high probability if we run the circuit sufficiently many times.

Let's suppose that we run the circuit independently k times, for $k = n + 10$. There's nothing special about this particular number of iterations — we could take k to be larger (or smaller) depending on the

probability of failure we're willing to tolerate, as we will see. Choosing $k = n + 10$ will ensure that we have greater than a 99.9% chance of recovering s.

By running the circuit k times, we obtain strings $y^1, ..., y^k \in \Sigma^n$. To be clear, the superscripts here are part of the names of these strings, not exponents or indexes to their bits, so we have

$$\begin{aligned} y^1 &= y^1_{n-1} \cdots y^1_0 \\ y^2 &= y^2_{n-1} \cdots y^2_0 \\ &\vdots \\ y^k &= y^k_{n-1} \cdots y^k_0 \end{aligned}$$

We now form a matrix M having k rows and n columns by taking the bits of these strings as binary-valued entries.

$$M = \begin{pmatrix} y^1_{n-1} & \cdots & y^1_0 \\ y^2_{n-1} & \cdots & y^2_0 \\ \vdots & \ddots & \vdots \\ y^k_{n-1} & \cdots & y^k_0 \end{pmatrix}$$

Now, we don't know what s is at this point — our goal is to find this string. But imagine for a moment that we do know the string s, and we form a column vector v from the bits of the string $s = s_{n-1} \cdots s_0$ as follows.

$$v = \begin{pmatrix} s_{n-1} \\ \vdots \\ s_0 \end{pmatrix}$$

If we perform the matrix-vector multiplication Mv modulo 2 — meaning that we perform the multiplication as usual and then take the remainder of the entries of the result after dividing by 2 — we obtain the all-zero vector.

$$Mv = \begin{pmatrix} y^1 \cdot s \\ y^2 \cdot s \\ \vdots \\ y^k \cdot s \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

That is, treated as a column vector v as just described, the string s will always be an element of the *null space* of the matrix M, provided that

we do the arithmetic modulo 2. This is true in both the case that $s = 0^n$ and $s \neq 0^n$. To be more precise, the all-zero vector is always in the null space of M, and it's joined by the vector whose entries are the bits of s in case $s \neq 0^n$.

The question remaining is whether there will be any other vectors in the null space of M besides the ones corresponding to 0^n and s. The answer is that this becomes increasingly unlikely as k increases — and if we choose $k = n + 10$, the null space of M will contain no other vectors in addition to those corresponding to 0^n and s with greater than a 99.9% chance. More generally, if we replace $k = n + 10$ with $k = n + r$ for an arbitrary choice of a positive integer r, the probability that the vectors corresponding to 0^n and s are alone in the null space of M is at least $1 - 2^{-r}$.

Using linear algebra, it is possible to efficiently calculate a description of the null space of M modulo 2. Specifically, it can be done using *Gaussian elimination*, which works the same way when arithmetic is done modulo 2 as it does with real or complex numbers. So long as the vectors corresponding to 0^n and s are alone in the null space of M, which happens with high probability, we can deduce s from the results of this computation.

Classical difficulty

How many queries does a *classical* query algorithm need to solve Simon's problem? The answer is: a lot, in general.

There are different precise statements that can be made about the classical difficulty of this problem, and here's just one of them. If we have any probabilistic query algorithm, and that algorithm makes fewer than $2^{n/2-1} - 1$ queries, which is a number of queries that's *exponential* in n, then that algorithm will fail to solve Simon's problem with probability at least $1/2$.

Sometimes, proving impossibility results like this can be very challenging, but this one isn't too difficult to prove through an elementary probabilistic analysis. Here, however, we'll only briefly examine the basic intuition behind it.

We're trying to find the hidden string s, but so long as we don't query the function on two strings having the same output value, we'll get very limited information about s. Intuitively speaking, all we'll learn is that the hidden string s is *not* the exclusive-OR of any two distinct strings we've

queried. And if we query fewer than $2^{n/2-1} - 1$ strings, then there will still be a lot of choices for s that we haven't ruled out because there aren't enough pairs of strings to do this. This isn't a formal proof, it's just the basic idea.

So, in summary, Simon's algorithm provides us with a striking advantage of quantum over classical algorithms within the query model. In particular, Simon's algorithm solves Simon's problem with a number of queries that's *linear* in the number of input bits n of our function, whereas any classical algorithm, even if it's probabilistic, needs to make a number of queries that's *exponential* in n in order to solve Simon's problem with a reasonable probability of success.

Previous page

Start the next lesson