Simon's Algorithm

Suppose we are given function 2-1 $f:\{0,1\}^n \to \{0,1\}^n$, specified by a black box, with the promise that there is an $a \in \{0,1\}^n$ with $a \neq 0^n$ such that

- For all x f(x+a) = f(x).
- If f(x) = f(y) then either x = y or y = x + a.

The challenge is to determine a. It should be intuitively clear that this is a difficult task for a classical (probabilistic) computer. This is because the algorithm cannot determine a until it finds two inputs x and y such that f(x) = f(y). And the best an algorithm can do is try random inputs x until it finds a match. By the birthday paradox (actually its converse), the chance of this is negligible if f is probed on many fewer than $2^{n/2}$ inputs. By contrast, we will show an efficient quantum algorithm.

1. Use f to set up random pre-image state

$$\phi = 1/\sqrt{2}|z\rangle + 1/\sqrt{2}|z+a\rangle$$

where z is a random n-bit string.

2. Perform a Hadamard transform $H^{\otimes n}$.

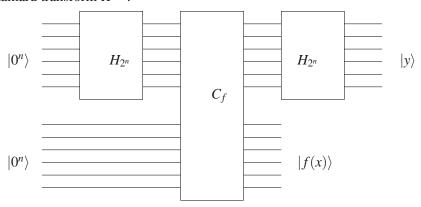


Figure 1: Simon's algorithm

1 Setting up a random pre-image state

Suppose we're given a classical circuit for a k-1 function $f: \{0,1\}^n \to \{0,1\}^n$.

We will show how to set up the quantum state $|\phi\rangle=1/\sqrt{k}\sum_{x:f(x)=a}|x\rangle$. Here a is uniformly random among all a in the image of f.

The algorithm uses two registers, both with n qubits. The registers are initialized to the basis state $|0\cdots 0\rangle |0\cdots 0\rangle$. We then perform the Hadamard transform H_{2^n} on the first register, producing the superposition

$$\frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}|x\rangle\,|0\cdots0\rangle\,.$$

C 191, Fall 2010,

Then, we compute f(x) through the oracle C_f and store the result in the second register, obtaining the state

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

The second register is not modified after this step. Thus we may invoke the principle of safe storage and assume that the second register is measured at this point.

Let a be the result of measuring of the second register. Then a is a random element in the range of f, and according to rules of partial measurement, the state of the first register is a superposition over exactly those values of x that are consistent with those contents for the second register. i.e.

$$\left|\phi\right\rangle = 1/\sqrt{k} \sum_{x:f(x)=a} \left|x\right\rangle$$

2 Obtaining a linear equation

After step 2 we obtain a superposition

$$\sum_{y \in \{0,1\}^n} \alpha_y |y\rangle$$

where

$$\alpha_{y} = \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} (-1)^{y \cdot z} + \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} (-1)^{y \cdot (z \oplus a)} = \frac{1}{2^{(n+1)/2}} (-1)^{y \cdot z} [1 + (-1)^{y \cdot a}].$$

There are now two cases. For each y, if $y \cdot a = 1$, then $\alpha_y = 0$, whereas if $y \cdot a = 0$, then

$$\alpha_{y} = \frac{\pm 1}{2^{(n-1)/2}}.$$

So when we observe the first register, with certainty we'll see a y such that $y \cdot a = 0$. Hence, the output of the measurement is a random y such that $y \cdot a = 0$. Furthermore, each y such that $y \cdot a = 0$ has an equal probability of occurring. Therefore what we've managed to learn is an equation

$$y_1 a_1 \oplus \dots \oplus y_n a_n = 0 \tag{1}$$

where $y = (y_1, ..., y_n)$ is chosen uniformly at random from $\{0,1\}^n$. Now, that isn't enough information to determine a, but assuming that $y \neq 0$, it reduces the number of possibilities for a by half.

It should now be clear how to proceed. We run the algorithm over and over, accumulating more and more equations of the form in (1). Then, once we have enough of these equations, we solve them using Gaussian elimination to obtain a unique value of a. But how many equations is enough? From linear algebra, we know that a is uniquely determined once we have n-1 linearly independent equations—in other words, n-1 equations

$$y^{(1)} \cdot a \equiv 0 \pmod{2}$$

$$\vdots$$

$$y^{(n-1)} \cdot a \equiv 0 \pmod{2}$$

C 191, Fall 2010, 2

such that the set $\{y^{(1)}, \dots, y^{(n-1)}\}$ is linearly independent in the vector space \mathbb{Z}_2^n . Thus, our strategy will be to lower-bound the probability that any n-1 equations returned by the algorithm are independent.

Suppose we already have k linearly independent equations, with associated vectors $y^{(1)}, \dots, y^{(k)}$. The vectors then span a subspace $S \subseteq \mathbb{Z}_2^n$ of size 2^k , consisting of all vectors of the form

$$b_1 y^{(1)} + \dots + b_k y^{(k)}$$

with $b_1, \ldots, b_k \in \{0,1\}$. Now suppose we learn a new equation with associated vector $y^{(k+1)}$. This equation will be independent of all the previous equations provided that $y^{(k+1)}$ lies *outside* of S, which in turn has probability at least $(2^n - 2^k)/2^n = 1 - 2^{k-n}$ of occurring. So the probability that any n equations are independent is exactly the product of those probabilities.

$$\left(1-\frac{1}{2^n}\right)\times\left(1-\frac{1}{2^{n-1}}\right)\times\cdots\times\left(1-\frac{1}{4}\right)\times\left(1-\frac{1}{2}\right).$$

Can we lower-bound this expression? Trivially, it's at least

$$\prod_{k=1}^{\infty} \left(1 - \frac{1}{2^k} \right) \approx 0.28879;$$

the infinite product here is related to something in analysis called a q-series. Another way to look at the constant 0.28879... is this: it is the limit, as n goes to infinity, of the probability that an $n \times n$ random matrix over Z_2 is invertible.

But we don't need heavy-duty analysis to show that the product has a constant lower bound. We use the inequality (1-a)(1-b) = 1-a-b+ab > 1-(a+b), if $a,b \in (0,1)$. We just need to multiply the product out, ignore monomials involving two or more $\frac{1}{2^k}$ terms multiplied together (which only increase the product), and observe that the product is lower-bounded by

$$\left[1 - \left(\frac{1}{2^n} + \frac{1}{2^{n-1}} + \dots + \frac{1}{4}\right)\right] \cdot \frac{1}{2} \ge \frac{1}{4}.$$

We conclude that we can determine a with constant probability of error after repeating the algorithm O(n) times. So the number of queries to f used by Simon's algorithm is O(n). The number of computation steps, though, is dominated by the number of steps needed to solve a system of linear equations. This can be done by Gaussian elimination, which takes $O(n^3)$ steps.