Simon's Algorithm

Peter Young

(Dated: October 31, 2019)

So far we have studied Deutsch's algorithm , `https://young.physics.ucsc.edu/150/deutsch.pdf` which gave a quantum speedup of a factor of 2, and the Bernstein-Vazirani algorithm `https://young.physics.ucsc.edu/150/bv.pdf` which gave a speedup of n, where n is the size of the problem. Next we consider a problem, due to Daniel Simon, which gives an *exponential* speedup in n. Like the previous algorithms it has an artificial character and is not of practical use, but it has features in common with the vastly more useful algorithm of Shor for factoring integers which we shall spend quite some time on next. Like Shor's algorithm it is of a probabilistic nature.

In Simon's problem we are given a black box function which takes an n-bit input and has the property that

$$f(x \oplus a) = f(x), \tag{1}$$

where a is a non-zero n-bit integer and $\oplus$ means bitwise addition modulo 2. Adding a twice to x (modulo 2) gives back x, i.e.

$$x \oplus a \oplus a = x \tag{2}$$

since adding a bit to itself gives $0\,(\mod 2)$ irrespective of whether that bit is 0 or 1. Hence

$$f(x) = f(x \oplus a) = f(x \oplus a \oplus a) \tag{3}$$

and so on, so $f(x)$ is *periodic* under bitwise mod 2 addition.

For every x there is only one other input to the function, $x \oplus a$, which gives the same output, so there are 2^{n-1} distinct values of f. Hence we assume that we can represent f by $n-1$ qubits.

If we input different values of x and find a repeated output, i.e. if $f(x_i) = f(x_j)$, then $x_j = x_i \oplus a$. If we add x_i to both sides (bitwise addition modulo 2) we get

$$a = x_i \oplus x_j\,. \tag{4}$$

so we obtain a if we can find two values of x which give the same function value.

The problem is to determine a with as few calls to the function as possible.

An example is shown in Table I.

x	000	001	010	011	100	101	110	111
$f(x)$	3	2	2	3	0	1	1	0

TABLE I: An example with $n = 3$ bits of the type of function that is considered in Simon's algorithm. The function satisfies $f(x) = f(x \oplus a)$ for some non-zero a. To determine a we look for repetitions. The first occurs for $f(1) = f(2) = 2$. Hence, according to Eq. (4), $a = 001 \oplus 010 = 011 = 3$. The other repetitions satisfy this same condition as you can check.

Classically this problem is hard, by which we mean that number of function calls grows *exponentially* with n. All one can do is call the function with successive values of x until one finds a repeated output, i.e. $f(x_i) = f(x_j)$, which gives us a from Eq. (4). After m calls to the function we have compared $m(m-1)/2$ pairs. For a reasonable chance of success we need $\frac{1}{2}m(m-1) \sim 2^n$, so $m = O(2^{n/2})$, i.e. exponential in the number of bits n.

The circuit to solve this problem **quantum mechanically** is similar to that in the Bernstein-Vazirani algorithm except that the lower register has enough qubits to contain the function values, i.e. $n-1$. Also the phase kickback is not used, so the lower register is initialized to $|0\rangle_{n-1}$ rather than $|1\rangle$ and we do not have Hadamards on the lower register. A final difference is that we measure first on the lower register rather than the upper one. The circuit diagram is shown in Fig. 1.

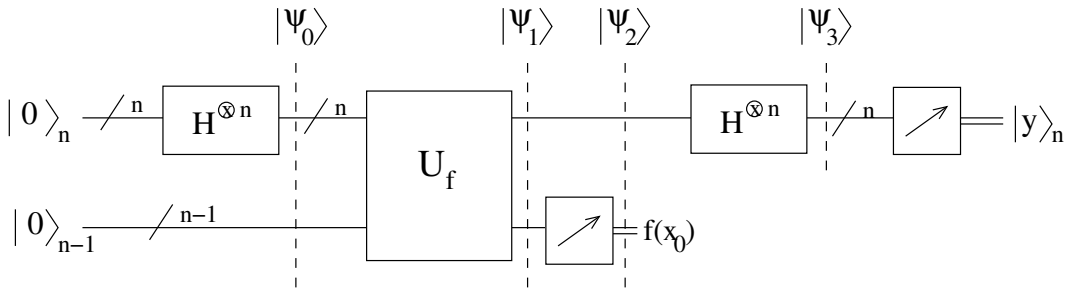

FIG. 1: Circuit diagram for Simon's algorithm. The upper register has n qubits and contains the x values, while the lower register has $n-1$ qubits and contains the values of the function $f(x)$.

After the first Hadamards in the upper register the state of the system is

$$|\psi_0\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle_n \otimes |0\rangle_{n-1} \,. \tag{5}$$

The function call makes the transformation $|x\rangle_n \otimes |y\rangle_{n-1} \to |x\rangle_n \otimes |y \oplus f(x)\rangle_{n-1}$, see Fig. 1 in the Bernstein-Vazirani handout, `https://young.physics.ucsc.edu/150/bv.pdf`. Here $y = 0$ so,

after the function call the state becomes

$$|\psi_1\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle_n \otimes |\, f(x)\,\rangle_{n-1}\,. \tag{6}$$

A measurement is then done on the lower register which will record some value of the function, f_0 say, for which there are two values of x which we denote by x_0 and $x_0 \oplus a$. Hence, immediately after the measurement, the state of the system is

$$|\psi_2\rangle = \frac{|x_0\rangle_n + |x_0 \oplus a\rangle_n}{\sqrt{2}} \otimes |\, f(x_0)\,\rangle_{n-1}\,. \tag{7}$$

If we were now to measure the upper register, we would get *either* x_0 or $x_0 \oplus a$. At first glance, this might seem like progress since we appear to be halfway there. If we could just get the other number, we would have a. However there is no way to get both. If we could clone the state several times and measure each clone then, with high probability, we would be able to determine both of them. However, the no-cloning theorem says that we can't clone an arbitrary, unknown state. Also, repeating the whole procedure doesn't help because, with high probability, one would get a different function value, f_0', and hence a different pair x_0' and $x_0' \oplus a$, from which again one would not be able to extract a.

As in Deutsch's algorithm and the Bernstein-Vazirani algorithm, we must do some processing *before* the final measurement. As we showed in the handout on the Bernstein-Vazirani algorithm `https://young.physics.ucsc.edu/150/bv.pdf` the effect of Hadamards on each qubit of an n-qubit register is given by

$$H^{\otimes n}|x\rangle_n = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} (-1)^{x\cdot y} |y\rangle_n\,, \tag{8}$$

where $x \cdot y$ is the bitwise inner product modulo 2,

$$x \cdot y \equiv x_0 y_0 \oplus x_1 y_1 \oplus \cdots \oplus x_{n-1} y_{n-1} \mod 2. \tag{9}$$

Applying Hadamards to the upper register, the state of that register therefore becomes

$$|\psi_3\rangle_n = \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} \Big[(-1)^{x_0\cdot y} + (-1)^{(x_0\oplus a)\cdot y} \Big] |y\rangle_n\,. \tag{10}$$

Now[1] $(x_0 \oplus a)\cdot y = (x_0 \cdot y) \oplus (a \cdot y)$ and we note that $a \cdot y = 0$ or 1. If $a \cdot y = 1$ then $(-1)^{(x_0\oplus a)\cdot y} = (-1)^{x_0\cdot y}(-1)^{a\cdot y} = -(-1)^{x_0\cdot y}$, so the two terms in Eq. (10) cancel. Hence the only terms with a non-zero amplitude are those with $a \cdot y = 0$.

[1] This is the mod 2 version of the usual associative rule for addition and multiplication: $a \times (b + c) = (a \times b) + (a \times c)$.

A measurement on the upper register then gives, with equal probability, *one* value of y with $a \cdot y = 0$. This is a linear equation for the a_i, the bits of a. If we can find n linearly-independent equations for the a_i, we can obtain the solution. Hence we have to repeat the procedure, each time getting a value for f_0 and y. As discussed in Mermin[1] Appendix G one needs to run the algorithm *a little more* than n times because the equations one gets each time for the a_i are not necessarily linearly independent. The result is that if one runs $n + x$ times, then the probability of getting n linearly dependent equations (and hence the solution for the a_i) is is greater than

$$1 - \frac{1}{2^{x+1}} \,. \tag{11}$$

Hence there is less than one chance in a million of failure if one calls the function $n + 20$ times. A crucial point in this expression is that the number of calls beyond n needed to find a solution with some high probability *does not depend on* n.

The occurrence of probability, and some arcane mathematical arguments to prove that one does get the solution with high probability within the specified number of runs, is characteristic of several quantum algorithms including Shor's.

In the case of Simon's problem, the classical algorithm takes of order $2^{n/2}$ function calls whereas the quantum algorithm finds the answer with high probability with little more than n calls[2]. This is an *exponential* speedup[3].

Finally a few words of anticipation for Shor's algorithm which we will do next. Simon's problem considers a function which is periodic under bitwise modulo 2 addition, see Eq. (3). Shor's algorithm investigates functions which are periodic under *ordinary* addition: $f(x + a) = f(x)$, which is much more useful. In Simon's problem, the action of the n-Hadamards in Eq. (8) can be written

$$H^{\otimes n}|x\rangle_n = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} e^{i\pi x\cdot y}|y\rangle_n \,, \tag{12}$$

Since $x \cdot y$ is the bitwise inner product modulo 2, it only takes values 0 and 1, so the phases in the complex exponential are just 0 and π. The core of Shor's algorithm is a quantum Fourier transform

---

[2] In the interests of full disclosure I should state that one also needs to solve n linear equations on a classical computer, which takes of order n^3 steps. A algorithm which takes a time proportional to a power of the problem size n is said to be *polynomial*. Since classical hardware is cheap it is not clear if one should include this time on the classical computer in the computational cost of Simon's algorithm. However, since n^3 is polynomial, even if does include this time the comparison is still between a polynomial quantum (+classical) algorithm and an exponential purely classical algorithm, which is still an exponential speedup, see footnote 3.

[3] An algorithm which takes a time proportional to a power of the problem size is said to have *polynomial complexity*, while if the time increases exponentially with size (or exponentially with a power of the size) it is said to have *exponential complexity*. If one algorithm has polynomial complexity and another has exponential complexity then the former is said to have an exponential speedup compared with the latter.

(QFT), where an essential difference from Eq. (12) is that the bitwise inner product is replaced by ordinary multiplication. Hence the QFT generates many different phases, with the result that, unlike Simon's algorithm, it cannot, in general, be constructed entirely out of 1-qubit gates. Fortunately, it *can* be constructed entirely out of 1- and 2-qubit gates. All this and more will be discussed in the handout on Shor's algorithm `https://young.physics.ucsc.edu/150/shor.pdf`.

---

[1] N. D. Mermin, *Quantum Computer Science* (Cambridge University Press, Cambridge, 2007).