Quantum Programming
Algorithms: Simon

Jens Palsberg

Apr 22, 2019

Outline

Hook: Our next quantum algorithm is Simon's algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will look at how Simon's algorithm mixes quantum and classical computing.
3. We will cover full details of how Simon's algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.
[Most of you have done the homework on solving the problem on a classical computer]
[We can represent the black-box function in multiple ways]
[We can program the solution in any classical language]

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will look at how Simon's algorithm mixes quantum and classical computing.
[First quantum produces a set of equations, and then classical solves them]
[We can solve the equations with an off-the-shelf classical solver]
[Simon's algorithm is an approximation algorithm]

Transition to MP3: Now let us look at the quantum side of Simon's algorithm.

Main Point 3: We will cover full details of Simon's algorithm works.
[The quantum circuit is a variation of what we have seen before]
[The number of helper bits is equal to the number of input bits]
[The reasoning about correctness relies on some known lemmas]

Transition to Close: So there you have it.

Review: Simon's algorithm combines classical and quantum computing.

Strong finish: Simon's algorithm is a powerful indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.

Detailed presentation

Hook: Our next quantum algorithm is Simon's algorithm. It solves a natural problem and does so faster on a quantum computer than we can ever do on a classical computer.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will look at how Simon's algorithm mixes quantum and classical computing.
3. We will cover full details of how Simon's algorithm works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.

[Most of you have done the homework on solving the problem on a classical computer] Here is the homework problem.

Simon's problem:
Input: a function $f : \{0,1\}^n \to \{0,1\}^n$.
Assumption: there exists $s \in \{0,1\}^n$ such that

$$\forall x, y : \; [f(x) = f(y)] \text{ iff } [(x + y) \in \{0^n, s\}].$$

Output: s.
Notation: $\{0,1\}^n$ is the set of bit strings of length n, s is an unknown bit string of length n, $=$ is comparison of bit strings of length n, $+$ is pointwise addition mod 2 of bit strings of length n, and 0^n is a bit string of length n with all 0.

The assignment:
On a classical computer, in a classical language of your choice (such as C, Java, Python, etc), program a solution to Simon's problem. Treat the input function f as black box that you can call but cannot inspect in any way at all. Each solution will be code that includes one or more calls of f.

Let us consider the assumption in more detail. Specifically, let us exhibit a particular f that satisfies the assumption. Suppose $n = 3$ and let f be the function given by the following table:

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

For the above f, the assumption is correct and $s = 110$. In particular, notice that every output of f occurs twice. Indeed, f has four different outputs: 000, 010, 101, 110. We can check that $s = 110$ is what we need by considering each of the four outputs in turn.

For 000, the two inputs are 010 and 100 and we see that $010 \oplus 100 = 110 = s$.
For 010, the two inputs are 001 and 111 and we see that $001 \oplus 111 = 110 = s$.
For 101, the two inputs are 000 and 110 and we see that $000 \oplus 110 = 110 = s$.
For 110, the two inputs are 011 and 101 and we see that $011 \oplus 101 = 110 = s$.
We conclude that f satisfies the assumption.

Any f that satisfies the assumption deserves to be called a 2-to-1 function. As a special case, we may have an f that satisfies the assumption with $s = 0^n$, in which case f is a 1-to-1 function.

[We can represent the black-box function in multiple ways]
How do we keep the black-box function at an arms length such that we cannot get tempted to inspect it? In C we can use a function pointer. In Java we can use a lambda expression. In Python we can use an anonymous function. Other languages have similar constructs that will enable us to call the function, while giving us no way to inspect it.

[We can program the solution in any classical language]
You wrote the solutions to the homework in several languages. We need to make many calls to f: the reason is that we need to find two different inputs x, y such that $f(x) = f(y)$. We need to try $\Omega(\sqrt{2^n})$ inputs before we have a reasonable chance of finding such x, y.

Transition to MP2: Now let us look at how to solve the problem.

Main Point 2: We will look at how Simon's algorithm mixes quantum and classical computing.

[First quantum produces a set of equations, and then classical solves them]
The idea is to use quantum computing for what it is good at, and then do the rest by classical computing. We will do that by using quantum computing to map the input function f to equations that captures everything we need to know about f. Then we will use classical computing to map the equations to s. The equations is the interface between quantum and classical. The above process may fail so we will repeat it until the chance of success is high. Specifically, the classical part may fail to produce s. Thus, we can illustrate the entire process with the followed diagram.

$$\text{repeat} \left(f \xrightarrow{\text{quantum}} \text{equations} \xrightarrow{\text{classical}} s \right) \text{ until the chance of success is high}$$

The above interface consists of a set of $n-1$ equations:

$$\begin{array}{rcl} y_1 \cdot s & = & 0 \\ y_2 \cdot s & = & 0 \\ & \vdots & \\ y_{n-1} \cdot s & = & 0 \end{array}$$

Above, $y_1, \ldots, y_{n-1}$ are known bit strings, while s is the unknown bit string that is part of the assumption about f.

From one angle, we can view the above as facts about s; the equations are properties that s satisfy.

From another angle, we can view the above as equations that we can solve to get s. Recall that s is a bit string of length n, so we can view each bit in s as an unknown. Then, the above equation system has $n-1$ equations and n unknowns.

[We can solve the equations with an off-the-shelf classical solver]

We have $n-1$ equations and n unknowns, and we have that 0^n is a solution. We can solve the set of equations in polynomial time using Gaussian elimination, as usual.

If $y_1, \ldots, y_{n-1}$ are linearly independent, the equations have two solutions, namely 0^n and a vector s that is orthogonal to the others. Otherwise, the set of equations has more than two solutions.

If $y_1, \ldots, y_{n-1}$ are chosen independently of each other, the probability that they are linearly independent is more than $\frac{1}{4}$.

[Simon's algorithm is an approximation algorithm]

If we run the entire process of quantum followed by classical a single time, we have only a little more than $\frac{1}{4}$ chance of finding s. So, Simon's algorithm runs the entire process a total of $4m$ times, where m is a parameter that we must determine.

$$\begin{array}{ll} & \text{Probability of failing to find } s \text{ after } 4m \text{ iterations} \\ < & \left(1 - \frac{1}{4}\right)^{4m} \\ < & e^{-m} \end{array}$$

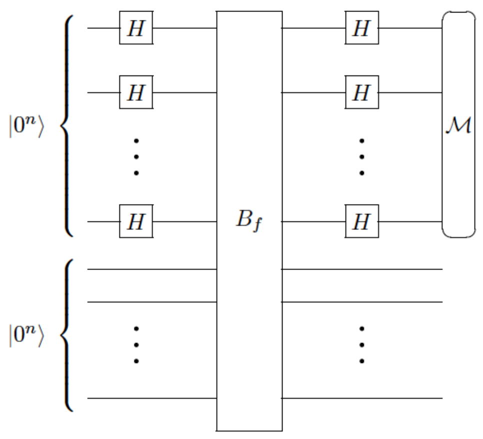
In the first step, we use that the iterations are independent. In the second step, we use one of Bernoulli's formulas.

Notice that $\forall \epsilon > 0 : \ \exists m : \ e^{-m} < \epsilon$. So, for example, if we want Simon's algorithm to succeed with probability at least 99 percent, we choose $m = 5$ and run the process $4 \times 5 = 20$ times.

Transition to MP3: Now let us look at the quantum side of Simon's algorithm.

Main Point 3: We will cover full details of how Simon's algorithm works.

[The quantum circuit is a variation of what we have seen before]



The diagram above uses the notation B_f for what I will call U_f.

[The number of helper bits is equal to the number of input bits]

The main reason is that f produces a bit string of length n. We will represent f as an invertible operation U_f, as usual:

$$\begin{aligned} U_f &: Qubit^{\otimes 2n} \rightarrow Qubit^{\otimes 2n} \\ U_f|x\rangle|b\rangle &= |x\rangle|b \oplus f(x)\rangle \end{aligned}$$

Here, $x, b \in \{0,1\}^n$.

[The reasoning about correctness relies on some known lemmas]
Initially, the state of the $2n$ qubits is $|0^n\rangle|0^n\rangle$.

After the first n uses of H, followed by the use of U_f, followed by the n uses of H, the state of the $2n$ qubits is as follows.

$$\begin{aligned} & (H^{\otimes n} \otimes I^{\otimes n}) \circ U_f \circ (H^{\otimes n} \otimes I^{\otimes n})\ |0^n\rangle|0^n\rangle \\ = \ & (H^{\otimes n} \otimes I^{\otimes n}) \circ U_f\ \frac{1}{\sqrt{2^n}}\ \Sigma_{x\in\{0,1\}^n}|x\rangle|0^n\rangle \\ = \ & (H^{\otimes n} \otimes I^{\otimes n})\ \frac{1}{\sqrt{2^n}}\ \Sigma_{x\in\{0,1\}^n}|x\rangle|f(x)\rangle \\ = \ & \frac{1}{2^n}\ \Sigma_{x\in\{0,1\}^n}\Sigma_{y\in\{0,1\}^n}\ (-1)^{x\cdot y}\ |y\rangle|f(x)\rangle \\ = \ & \Sigma_{y\in\{0,1\}^n}\ |y\rangle\ \left(\frac{1}{2^n}\ \Sigma_{x\in\{0,1\}^n}(-1)^{x\cdot y}\ |f(x)\rangle\right) \end{aligned}$$

Given a particular $y \in \{0,1\}^n$, let us calculate the probability of measuring y. We divide the analysis into two cases.

First, consider $s = 0^n$. In this case, f is 1-to-1, so the sum over $x \in \{0,1\}^n$ is a sum of 2^n different vectors $|f(x)\rangle$. For each of those 2^n vectors, the term $(-1)^{x\cdot y}$ is either -1 or $+1$. So, the probability of measuring y is:

$$||\ \frac{1}{2^n}\ \Sigma_{x\in\{0,1\}^n}\ (-1)^{x\cdot y}\ |f(x)\rangle\ ||^2 \ = \ \frac{1}{2^{2n}}\ \Sigma_{x\in\{0,1\}^n}\ 1 \ = \ \frac{2^n}{2^{2n}} \ = \ \frac{1}{2^n}$$

We conclude that each of the 2^n different y are equally likely outcomes of the measurement. Notice that y satisfies

$$y \cdot s \ = \ 0$$

because $s = 0^n$.

Second, consider $s \neq 0^n$. Let A denote the range of f. For each $z \in A$, we have 2 distinct $x_z, x'_z \in \{0,1\}^n$ such that

$$f(x_z) \ = \ f(x'_z) \ = \ z$$

From the above and the assumption about f, we also have:

$$\begin{aligned} x_z \oplus x'_z &= s \quad \text{which is equivalent to} \\ x'_z &= x_z \oplus s \end{aligned}$$

So, the probability of measuring y is:

$$\begin{aligned}
& || \ \frac{1}{2^n} \ \Sigma_{x \in \{0,1\}^n} \ (-1)^{x \cdot y} \ |f(x)\rangle \ ||^2 \\
= \ & || \ \frac{1}{2^n} \ \Sigma_{z \in A} \ \left((-1)^{x_z \cdot y} + (-1)^{x'_z \cdot y} \right) |z\rangle \ ||^2 \\
= \ & || \ \frac{1}{2^n} \ \Sigma_{z \in A} \ \left((-1)^{x_z \cdot y} + (-1)^{(x_z \oplus s) \cdot y} \right) |z\rangle \ ||^2 \\
= \ & || \ \frac{1}{2^n} \ \Sigma_{z \in A} \ (-1)^{x_z \cdot y} \left(1 + (-1)^{s \cdot y}\right) |z\rangle \ ||^2 \\
= \ & \begin{cases} 2^{-(n-1)} & \text{if } s \cdot y = 0 \\ 0 & \text{if } s \cdot y = 1 \end{cases}
\end{aligned}$$

In the third step, we used the property

$$(x_z \oplus s) \cdot y \quad = \quad (x_z \cdot y) \oplus (s \cdot y)$$

We conclude that each of 2^{n-1} different y are equally likely outcomes of the measurement. Additionally, the y that we get as result of the measurement satisfies the equation

$$y \cdot s \quad = \quad 0$$

In summary, the measurement produces a bit string y that satisfies $y \cdot s = 0$, and the distribution is uniform across all such bit strings.

Thus, we can satisfy the interface to the classical part by running the above quantum circuit $n - 1$ times and collecting the equations.

For example, let us run the quantum circuit on the specific f for which we listed a table earlier and for which we have $s = 110$. Just before measurement, the state of the $2 \times 3 = 6$ qubits is:

$$\begin{aligned}
& \Sigma_{y \in \{0,1\}^3} \ |y\rangle \ \left(\frac{1}{2^3} \ \Sigma_{x \in \{0,1\}^3} (-1)^{x \cdot y} \ |f(x)\rangle \right) \\
= \ & \Sigma_{y \in \{0,1\}^3} \ |y\rangle \ (\frac{1}{8} \ (((-1)^{010 \cdot y} + (-1)^{100 \cdot y}) \ |000\rangle \ + \\
& \qquad ((-1)^{001 \cdot y} + (-1)^{111 \cdot y}) \ |010\rangle \ + \\
& \qquad ((-1)^{000 \cdot y} + (-1)^{110 \cdot y}) \ |101\rangle \ + \\
& \qquad ((-1)^{011 \cdot y} + (-1)^{101 \cdot y}) \ |110\rangle)) \\
= \ & \Sigma_{y \in \{0,1\}^3} \ |y\rangle \ (\frac{1}{8} \ (((-1)^{010 \cdot y} + (-1)^{(010 \oplus 110) \cdot y}) \ |000\rangle \ + \\
& \qquad ((-1)^{001 \cdot y} + (-1)^{(001 \oplus 110) \cdot y}) \ |010\rangle \ + \\
& \qquad ((-1)^{000 \cdot y} + (-1)^{(000 \oplus 110) \cdot y}) \ |101\rangle \ + \\
& \qquad ((-1)^{011 \cdot y} + (-1)^{(011 \oplus 110) \cdot y}) \ |110\rangle)) \\
= \ & \Sigma_{y \in \{0,1\}^3} \ |y\rangle \ (\frac{1}{8} \ ((-1)^{010 \cdot y}(1 + (-1)^{110 \cdot y}) \ |000\rangle \ + \\
& \qquad (-1)^{001 \cdot y}(1 + (-1)^{110 \cdot y}) \ |010\rangle \ + \\
& \qquad (-1)^{000 \cdot y}(1 + (-1)^{110 \cdot y}) \ |101\rangle \ + \\
& \qquad (-1)^{011 \cdot y}(1 + (-1)^{110 \cdot y}) \ |110\rangle))
\end{aligned}$$

So, when we measure and get y, we know that $110 \cdot y = 0$ and that all such cases of y are equally likely. We need to run $3 - 1 = 2$ times to produce two such y that will form the interface to the classical part.

We can take the example further and suppose that the two runs produced 000 and 001. Notice that we have $000 \cdot 110 = 0$ and $001 \cdot 110 = 0$.

We see that 000 and 001 are linearly independent. Now we can give the two equations

$$\begin{aligned} 000 \cdot s &= 0 \\ 001 \cdot s &= 0 \end{aligned}$$

to a constraint solver. The constraint solver will tell us that the equations have a single solution that is different from 000, namely 110. Thus, we have succeeded in taking f as input and producing 110 as output.

Notice that Simon's algorithm uses $(n-1) \times 4m$ iterations, which is much fewer than the $\sqrt{2^n}$ that are needed with classical computing. Thus, Simon's algorithm gave an exponential speed-up.

Transition to Close: So there you have it.

Review: Simon's algorithm combines classical and quantum computing.

Strong finish: Simon's algorithm is a powerful indication that a quantum computer can be faster than a classical computer. We will see more examples of that in this course.

Call to action: Look for other natural problems that may be solvable on quantum computers.