Chapter 5

Early Quantum Algorithms

Quantum algorithms look nothing like classical algorithms, and we will need to start from scratch in understanding how to program a quantum computer. Let us recall some primitives we introduced in the last lecture and introduce some more:

Hadamard Transform

Consider a quantum circuit on n qubits where each qubit is acted on by a Hadamard gate. The quantum circuit is denoted by H_n, and it carries out the unitary transform $H^{\otimes n} = H \otimes H \otimes \cdots \otimes H$.

This is a very powerful primitive. To begin with let us just observe that if the input is $|0\rangle^{\otimes n}$, i.e. n qubits each in the state $|0\rangle$, then the output is $|+\rangle^{\otimes n} = (1/\sqrt{2}\,|0\rangle + 1/\sqrt{2}\,|01\rangle)^{\otimes n} = \sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}} |x\rangle$.

Phase State

Suppose we are given a classical circuit C_f for computing a boolean function (a function whose output is always 0 or 1). i.e. $f : \{0,1\}^n \to \{0,1\}$. We wish to create the superposition $\sum_{x\in\{0,1\}^n} \frac{-1^{f(x)}}{2^{n/2}} |x\rangle$. Here is a quantum circuit to create the superposition:
Start with n qubits each in the state $|0\rangle$.
Apply the Hadamard transform to get the superposition $\sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}} |x\rangle$.
We now wish to apply a phase depending upon $f(x)$. Let us input this superposition (together with a number of scratch qubits initialized to $|0\rangle$) to the quantum circuit U_f:
Apply U_f to get the superposition $\sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}} |x\rangle |f(x)\rangle$
Now since we have a qubit in the state $|f(x)\rangle$, if we apply the phase flip gate to it, we will get the desired phase in the superposition:
Apply Z to the $f(x)$-qubit to get the superposition $\sum_{x\in\{0,1\}^n} \frac{-1^{f(x)}}{2^{n/2}} |x\rangle |f(x)\rangle$
This is pretty close to what we want, except that there is an extra qubit $|f(x)\rangle$. Unfortunately it is entangled with the first register containing $|x\rangle$. We must uncompute it. But this is easily done by applying $U_f^\dagger$ – note that since U_f just xors its answer into the last bit, we can simply apply U_f to undo the action of U_f). Note also that this does not undo the application of the phase.
Apply U_f to get the superposition $\sum_{x\in\{0,1\}^n} \frac{-1^{f(x)}}{2^{n/2}} |x\rangle$.

Black-box Algorithms

The algorithms we will consider here are black-box algorithms — we are given a program for computing some function $f : \{0,1\}^n \to \{0,1\}^k$. We will regard the program as a black-box in the sense that we can run the program on any input, but cannot look at the code. We would like to answer some question about the function f by running the program on as few inputs as possible. Moreover our choice of the function f and the question about it will be guided by the objective of demonstrating the power of quantum algorithms. The important thing to note is that if we are answering the question using a quantum algorithm, we are naturally allowed to invoke the program in superposition, so we can let the input be a superposition $\sum_x \alpha_x |x\rangle$ and the output would be the superposition $\sum_x \alpha_x |x\rangle |f(x)\rangle$.

Deutsch-Jozsa Algorithm

Suppose we are given a circuit C_f for computing a boolean function $f : \{0,1\}^n \to \{0,1\}$, where either
Case 1: f is constant. i.e. $f(x) = 0$ for all x. or
Case 2: f is balanced. i.e. $f(x) = 0$ for exactly half the 2^n choices for x.
Challenge: determine which: case 1 or case 2.

Deutsch-Jozsa Algorithm: Set up phase state: $\sum_{x\in\{0,1\}^n} \frac{-1^{f(x)}}{2^{n/2}} |x\rangle$
Apply the Hadamard transform and measure.
If the output is 0^n output Case 1, else Case 2.

Why does this work? If we are in Case 1, then the phase state is $\sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}} |x\rangle$, exactly the state that results from applying the Hadamard transform to $|0^n\rangle$, as noted above. Since the Hadamard transform is its own inverse, in Case 1 the output is $|0^n\rangle$.
In Case 2, since the function is balanced, the phase state is orthogonal to $\sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}} |x\rangle$, and therefore the state after the Hadamard transform must be orthogonal to $|0^n\rangle$. So it must be some superposition over n-bit strings other than 0^n.

Bernstein-Vazirani Algorithm

Suppose we are given a circuit C_f for computing a boolean function $f : \{0,1\}^n \to \{0,1\}$, where $f(x) = u \cdot x$, for some secret n bit string u. The challenge is to find u.

Before solving this problem let us understand the concept of Fourier Sampling, which plays a central role in quantum algorithms.

Fourier Sampling

A central building block for quantum algorithms is Fourier Sampling. This primitive consists of preparing some interesting superposition $|\psi\rangle = \sum_u \alpha_u |u\rangle$, feeding that as input to the Hadamard transform circuit H_n and measuring the output in the standard basis. Write the output superposition by $\sum_x \beta_x |x\rangle$. Then the result of the measurement is x with probability $|\beta_x|^2$. Here $\beta_x = \sum_u \frac{-1^{u\cdot x}\alpha_u}{2^{n/2}}$.

To understand the action of H_n, let us feed in a computational basis state $|u\rangle$, where u is an n bit string. The output is the superposition $\psi_u = \sum_{x\in\{0,1\}^n} \frac{-1^{u\cdot x}}{2^{n/2}} |x\rangle$. Here $u \cdot x$ is the inner product $u_1x_1 + u_2x_2 + \cdots + u_nx_n$.

Consider the case $n = 1$. Then $H_1 |u_1\rangle = \sum_{x_1\in\{0,1\}} \frac{-1^{u_1x_1}}{\sqrt{2}} |x_1\rangle$. This expresses the fact that the

Hadamard gate applies a negative phase only in the case that the input qubit and the output qubit are in the state $|1\rangle$.

What is the output of the Hadamard transform H_n when the input is a general superposition $\sum_u \alpha_u |u\rangle$? Invoking linearity, the output should be $\sum_u \alpha_u |\phi_u\rangle = \sum_u \alpha_u \sum_{x\in\{0,1\}^n} \frac{-1^{u\cdot x}}{2^{n/2}} |x\rangle$.

As an example, let $n = 2$, and suppose the input state is $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.
Note that $H^{\otimes 2}|00\rangle = 1/2(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$ and $H^{\otimes 2}|11\rangle = 1/2(|00\rangle - |01\rangle - |10\rangle + |11\rangle)$.
Taking the suitable linear combination of the two, we get
$H^{\otimes 2}\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle) = 1/2\sqrt{2}((|00\rangle+|01\rangle+|10\rangle+|11\rangle)+(|00\rangle-|01\rangle-|10\rangle+|11\rangle)) = 1/\sqrt{2}(|00\rangle+|11\rangle)$.
Pay special attention to the constructive and destructive interference here. This is a critical feature that gives the Hadamard transform great computational power.

Recall our next task:

Suppose we are given a circuit C_f for computing a boolean function $f : \{0,1\}^n \to \{0,1\}$, where $f(x) = u \cdot x$, for some secret n bit string u. The challenge is to find u.

Bernstein-Vazirani Algorithm:

Construct the corresponding quantum circuit U_f and use it to set up the phase state $\sum_{x\in\{0,1\}^n} \frac{-1^{s\cdot x}}{2^{n/2}} |x\rangle$. Apply the Hadamard transform and measure and output the result.

This works because the phase state is exactly the state that would result if we applied the Hadamard transform to $|u\rangle$. Since the Hadamard transform is its own inverse, applying the Hadamard transform to this phase state will result in the output state $|u\rangle$, whose measurement reveals u with probability 1 .

How difficult is it to solve this problem classically?

It is easy to see how to determine the j-th bit of u. Simply choose x which has j-th bit 1 and all others 0, and evaluate $f(x)$ (which takes on value u_j). Repeat this n times for all n values of j.

In what sense does the quantum algorithm perform better? Notice that the quantum algorithm requires only two invocations of U_f (to set up the phase state). Since every invocation of C_f yields only one bit of information, it is clear that n invocations are necessary to reconstruct u.

This gap between 2 and n invocations does not give a super-polynomial separation between quantum and classical algorithms. However, a more complex recursive version of this problem, called Recursive Fourier Sampling, does give a super-polynomial separation.

Extended Church-Turing Thesis

One of the fundamental building blocks of the theory of computation is the Extended Church-Turing Thesis (ECTT), which states that all "reasonable" models of computation (or computers) is polynomially equivalent. This means that we cannot speed up a computation by much by changing our instruction set or adding esoteric features to our computer. Our only realistic option is increasing the clock speed. Here "reasonable" means that the computer must be a) programmable and b) digital.

The super-polynomial speed up given by Recursive Fourier Sampling, together with constructions showing quantum computers are programmable and an argument showing that quantum computers are not too sensitive to noise and therefore may be thought of as digital provided an early indication that quantum computers violate the ECTT.

Simon's Algorithm The Problem

Suppose we are given a black box for computing a 2-to-1 function $f : \{0,1\}^n \to \{0,1\}^n$ (from n-bit strings to n-bit strings), with the promise that there is a non-zero string $s \in \{0,1\}^n \setminus \{0\}$ such that

$$\text{for all } x \neq y,\ f(x) = f(y) \text{ if and only if } x \oplus y = s.$$

Here $\oplus$ is the bitwise direct sum modulo 2. For example, $\begin{array}{r} 1\,1\,0\,1 \\ \oplus\,0\,1\,1\,1 \\ \hline 1\,0\,1\,0 \end{array}$ or $\begin{array}{r} 0\,1 \\ \oplus\,0\,1 \\ \hline 0\,0 \end{array}$.

A quick example of such a function with $n = 3$ is

$$f(x) = \begin{cases} 001 & \text{if } x = 000 \text{ or } 011 \\ 010 & \text{if } x = 001 \text{ or } 010 \\ 100 & \text{if } x = 111 \text{ or } 100 \\ 111 & \text{if } x = 110 \text{ or } 101 \end{cases}$$

where $s = 011$.

The *problem* of Simon's algorithm is to determine s.

Classically

A simple way to solve this problem classically would be to randomly input values to the black box until we find two inputs that produce the same output, and compute their direct sum. But there are 2^{n-1} possible outputs, so despite the help from the birthday paradox, we expect it to take $\sqrt{2^{n-1}} = 2^{(n-1)/2}$ attempts to find s: still exponential time.

Furthermore, it can be shown that no classical computer can find s faster than exponential time. We will use the power of quantum computing to find a faster way.

Quantum

To utilize the power of quantum computing, we will access the function in superposition. So suppose instead of a black box we are given the circuit C_f for computing $|f\rangle$, from which we can construct the unitary transformation U_f:

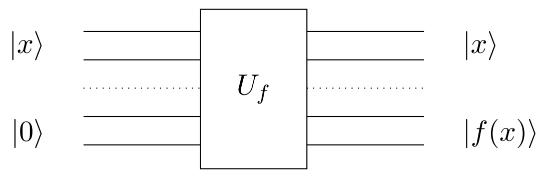

Figure 5.1: Black Box Circuit

The point here is that the input can be a superposition over all n-bit strings $\sum_{x=0}^{N-1} \alpha_x |x\rangle$ ($N = 2^n$), yielding the output $\sum_{x=0}^{N-1} \alpha_x |x\rangle |f(x)\rangle$. This can be thought of as querying f in superposition.

Simon's Algorithm consists of 3 main steps.

Step 1: Prepare the random superposition $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)$

Step 2: Use Fourier sampling to produce a y such that $y \cdot s = 0$

Step 3: Repeat until there are enough such y's that we can classically solve for s.

Now lets see the details on how to do each step.

Step 1: Prepare the random superposition $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)$

First query the function with a uniform superposition of the n-bit strings. To prepare this uniform superposition, start with the state $|0\rangle$ then apply the Hadamard transform. With $N = 2^n$, this is written:

$$|0\rangle\,|0\rangle \xrightarrow{H^{\otimes n}} \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle\,|0\rangle$$

Next we will use the unitary transformation U_f to query f in uniform superposition.

$$\sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle\,|0\rangle \xrightarrow{U_f} \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle\,|f(x)\rangle$$

Now what happens if we measure the second register containing $|f(x)\rangle$? It must collapse into $|f(x_0)\rangle$ for some $x_0 \in \mathbf{Z}_2^n$. But this reveals information about the first register, and it will also collapse into the pre-images of $f(x_0)$: x_0 and $x_0 \oplus s$.

$$\sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle\,|f(x)\rangle \xrightarrow{\text{measure } f} \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)\,|f(x_0)\rangle$$

The first register is now the state $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)$ where x_0 is a random n-bit string. The challenge is to read off s from this superposition. We cannot simply measure the state because the superposition will be destroyed, and the result we get will have no information about s.

Step 2: Use Fourier sampling to find a y such that $y \cdot s = 0$.

We now show that $H^{\otimes 2}(\frac{1}{\sqrt{2}}|x_0\rangle + \frac{1}{\sqrt{2}}|x_0 \oplus s\rangle$ is a uniform superposition over all states $|y\rangle$ such that $y \cdot s = 0$. This means that Fourier sampling $(\frac{1}{\sqrt{2}}|x_0\rangle + \frac{1}{\sqrt{2}}|x_0 \oplus s\rangle$ results in a uniform superposition of y such that $y \cdot s = 0$. Recall that

$$H^{\otimes n}|x\rangle = \sqrt{\frac{1}{N}} \sum_y \alpha_y |y\rangle \text{ where } \alpha_y = (-1)^{x \cdot y}$$

So $H^{\otimes 2}\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = \frac{1}{2}\sum_y \alpha_y |y\rangle$ where

$$\begin{aligned}\alpha_y &= \frac{1}{\sqrt{2}}(-1)^{x_0\cdot y} + (-1)^{(x_0\oplus s)\cdot y} \\ &= \frac{1}{\sqrt{2}}(-1)^{x_0\cdot y}(1 + (-1)^{s\cdot y})\end{aligned}$$

Now it is easy to see that $\underline{\text{if } s\cdot y = 0,\ \alpha_y = \pm\frac{1}{\sqrt{2}}, \text{ but if } s\cdot y = 1,\ \alpha_y = 0.}$

Therefore when we measure the first register, we will measure a y such that $y \cdot s = 0$.

Step 3: Repeat until there are enough such y's that we can classically solve for s.

There are exactly n linearly independent values of y such that $y\cdot s = y_1s_1 + y_2s_2 + \cdots y_ns_n = 0$, and one of these is the trivial solution $y = 0$. Therefore, there are $n-1$ non-trivial, linearly independent solutions to $y\cdot s = 0$. But if y_1 and y_2 are linearly independent solutions, $(y_1+y_2)\cdot s = y_1\cdot s + y_2\cdot s = 0$ so linear combinations of solutions are also solutions. This gives us a total of 2^{n-1} y's such that $y \cdot s = 0$. To solve for s, we need to find exactly $n-1$ non-trivial, linearly independent y such that $y \cdot s = 0$.

For example, if $s = 010$, then $y_0 = 000$, $y_1 = 001$, and $y_2 = 100$ are linearly independent solutions to $y \cdot s = 0$. But the linear combination $y_1 + y_2 = 101$ is also a solution. We need only find two of $\{y_1, y_2, y_1 + y_2\}$ in order to classically solve for s.

How long should we expect this to take? The probability that we fail on the first run is the probability that we find $y = 0$, which is one value out of 2^{n-1}. So $P_1 = 1/2^{n-1}$, where P_1 denotes the probability of failing on the first run. Lets call the first nontrivial solution y_1.

We fail when looking for y_2 if we find 0 or y_1, so $P_2 = 2/2^{n-1} = 1/2^{n-2}$. When looking for y_3, we fail if we find any of $\{0, y_1, y_2, y_1 + y_2\}$, so $P_3 = 4/2^{n-1} = 1/2^{n-3}$. Carrying on in this way, the probability of failing to find y_i is $P_i = 1/2^{n-i}$.

The chance that we fail *up to and including* y_i can be approximated by $P < 1/2^{n-1} + 1/2^{n-2} + \cdots + 1/2^{n-i}$. If we push this approximation all the way to $i = n - 1$, we see that we fail with probability less than 1 (compute the geometric sum). That's not a strong enough approximation, so instead notice that our probability of failure up to and including $i = n - 2$ is less than $1/2$. Then our probability of success up to the $n-2$st run is greater than $1/2$. We find the final linearly independent term on the last run with probability $1/2$ (if you don't believe this, notice that half of the solutions are linear combinations that include y_{n-1}). Finally our total probability of success is $P(\mathit{success}) > 1/2 * 1/2 = 1/4$. Therefore, we expect our process to take $O(n)$ steps (our limit says 4 by n runs of the algorithm should be enough for success).

Simon's algorithm is summed up by the following circuit.

In summary, the above circuit for Simon's algorithm corresponds to the following sequence of

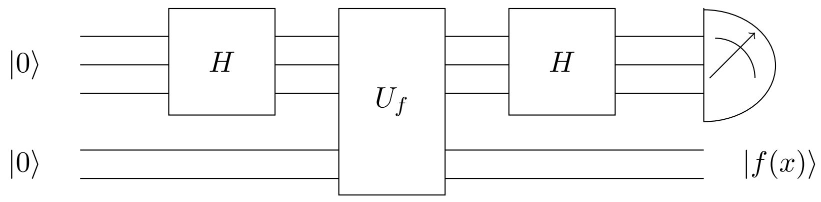

Figure 5.2: Circuit for Simon's Algorithm

transformations.

$$\begin{aligned} |0\rangle\,|0\rangle &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}}\sum_x |x\rangle\,|0\rangle \\ &\xrightarrow{U_f} \frac{1}{\sqrt{2^n}}\sum_x |x\rangle\,|f(x)\rangle \\ &\xrightarrow{\text{measure}} \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) \otimes |f(x_0)\rangle \\ &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}}\sum_y \alpha_y\,|y\rangle\,|f(x_0)\rangle \end{aligned}$$

for some numbers α_y.

As above, for each y, if $s \cdot y = 1$, then $\alpha_y = 0$, whereas if $s \cdot y = 0$, then $\alpha_y = (-1)^{x_0 \cdot y}\sqrt{2}$.

When we observe the first register, we get a uniformly random y such that $s{\cdot}y = s_1y_1+\cdots+s_ny_n = 0$. We repeat to collect more and more equations, and recover s from $n-1$ linearly independent equations.

Example

Let $n = 2$ and $f(x) = \begin{cases} 00 \text{ if } x = 00 \text{ or } 10 \\ 01 \text{ if } x = 01 \text{ or } 11 \end{cases}$ so that $s = 10$.

First, apply the Hadamard transform to prepare $|x\rangle$, then U_f to prepare $|f(x)\rangle$

$$|0\rangle\,|0\rangle \xrightarrow{H^{\otimes 2}} \frac{1}{2}\sum_{x=0}^{3} |x\rangle\,|0\rangle \xrightarrow{U_f} \frac{1}{2}\sum_{x=0}^{3} |x\rangle\,|f(x)\rangle$$

Then measure the second register to finalize the first step of the process. For the purpose of argument, lets suppose we measure $f(x) = 01$, so that $x_0 = 01$ and $x_0 \oplus s = 11$:

$$\frac{1}{2}\sum_{x=0}^{3} |x\rangle\,|f(x)\rangle \xrightarrow{\text{measure}} \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) \otimes |01\rangle$$

We then impose the Hadamard transform to achieve:

$$\frac{1}{2}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{pmatrix}\begin{pmatrix}0\\1\\0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\\0\\0\end{pmatrix}$$

We expect it will take 2 runs through the above process to measure $y = 01$. Then the only nonzero solution for s is $s = 10$.