Early Quantum Algorithms

Lecture Notes for CS294-4 (Quantum Computation)

Lecturer: Umesh Vazirani Scribes: Xylar Asay-Davis and Tim Robertson

September 20, 2002

1 Summary

Last week, we covered the basics of quantum gates and quantum circuits, and showed that quantum circuits are at least as powerful as their classical counterparts. Applying what we learned, this week we study problems that can be solved *more* efficiently with a quantum circuit than with a classical one. In contrast to factorization and search, though, these problems are "promise problems," meaning that the input has to satisfy some (possibly contrived) condition for the algorithm to work. However, the problems are important for two reasons. First, historically they provided early evidence of the power of quantum computation. Second, their solutions illustrate general techniques that will recur in the course—for example, the Fourier transform on an Abelian group.

These notes are organized as follows. Section 2 reviews material on reversible simulations, Hadamard transforms, and tensor products. Section 3 gives our first example of a quantum algorithm, the Deutsch-Jozsa algorithm. Section 4 reviews the principle of deferred measurement, and Section 5 covers Fourier sampling and briefly sketches recursive Fourier sampling. Finally, Section 6 presents Simon's algorithm.

2 Preliminaries

A quantum circuit accepts a superposition $|\phi\rangle$ as input, performs a unitary transformation U on $|\phi\rangle$ by composing elementary gates, and returns a new superposition $U(|\phi\rangle)$ as output. Last week we saw two primitives for quantum circuits that will be used extensively in the results to follow. Let's briefly review them.

The first primitive is a quantum circuit U_f that 'pretends' to do classical computation, computing a classical function f defined by a Boolean circuit, so that $f : \{0,1\}^n \rightarrow \{0,1\}$. U_f is constructed so that whenever the input $|\phi\rangle$ is a basis state, the output $U_f(|\phi\rangle)$ is also a basis state. A classical circuit for computing f would have n inputs and only 1 output. Thus $n-1$ bits of data are erased by the classical circuit. However, a quantum circuit must perform a unitary transformation (meaning that the circuit must be reversible, and hence no erasure may occur). Since U_f is unitary, it follows that U_f induces a permutation of basis states.

Now f is not, in general, a permutation. So if we want to compute f quantum-mechanically, we have to do it in a special way. As we saw, U_f can accept inputs of the form $|x\rangle|y\rangle$, where the first register contains an input x to f and the second register contains blank space on which to write the output. (Recall that $|x\rangle|y\rangle$ is shorthand for the tensor product $|x\rangle \otimes |y\rangle$.) The output of U_f is then $|x\rangle|y \oplus f(x)\rangle$, where $\oplus$

denotes bitwise XOR, and one can check that this indeed defines a permutation. In particular, when $y = 0$, $|x\rangle |0\rangle$ is mapped onto $|x\rangle |f(x)\rangle$.

In general, though, we don't need to feed U_f a classical state $|x\rangle$. If we feed U_f a superposition

$$\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle |0\rangle$$

then, by linearity,

$$U_f \left(\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle |0\rangle \right) = \sum_{x \in \{0,1\}^n} \alpha_x U_f (|x\rangle |0\rangle) = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle |f(x)\rangle$$

and we've computed $f(x)$ simultaneously for each basis state $|x\rangle$ in the superposition.

The second primitive is the Hadamard transform H, also called the Fourier transform. H is the unitary transformation on one qubit defined by the matrix

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

In other words, H maps $|0\rangle$ onto $\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$, and $|1\rangle$ onto $\frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle$. Given an n-qubit system, we'll often want to apply H separately to each of the qubits in turn. We call the resulting transformation $H_{2^n} = H^{\otimes n}$; it equals H tensored with itself n times. For all $n > 0$, $H^{\otimes n}$ can also be defined recursively by the matrix

$$\begin{pmatrix} \frac{1}{\sqrt{2}} H^{\otimes n-1} & \frac{1}{\sqrt{2}} H^{\otimes n-1} \\ \frac{1}{\sqrt{2}} H^{\otimes n-1} & -\frac{1}{\sqrt{2}} H^{\otimes n-1} \end{pmatrix}.$$

where $H^{\otimes 0} = \begin{pmatrix} 1 \end{pmatrix}$. (As we showed in our homework last week, for or all $n \times n$ matrices U and V, if U and V are unitary then the tensor product $U \otimes V$ is also unitary.)

3 The Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm was published in 1992, and was the first non-trivial quantum algorithm.

Suppose we're given a Boolean function $f : \{0,1\}^n \to \{0,1\}$. We're promised that f is either identically zero (meaning that $f(x) = 0$ for all inputs $x \in \{0,1\}^n$), or else balanced (meaning that $f(x) = 0$ for precisely half of all inputs x, and $f(x) = 1$ for the other half). The challenge is to decide which is the case. To do this, we can make queries of the form "Is $f(x)$ equal to 1?" for particular values of x. Our goal is to minimize the number of queries that have to be made.

Clearly, a deterministic classical algorithm requires $2^{n-1} + 1$ queries in the worst case: if 2^{n-1} bits have been queried and all turned out to be 0, we still need to query one more bit to decide whether the function is zero or balanced.

On the other hand, the problem admits an efficient randomized algorithm, as follows. Choose a value of x uniformly at random; if $f(x) = 1$ then conclude that f is balanced, otherwise repeat. If, after k iterations (for some constant k), we still haven't found an x such that $f(x) = 1$, then we halt and conclude that f is zero. This algorithm uses only $O(1)$ queries. However, has a nonzero probability of error, equal to

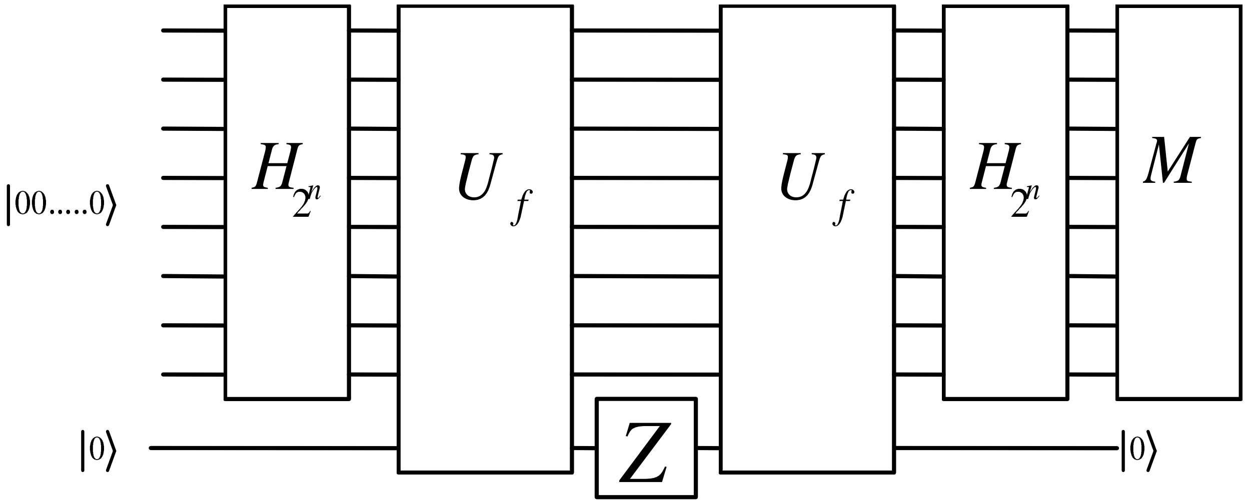

Figure 1: Quantum circuit for the Deutsch-Jozsa algorithm

2^{-k}, or slightly less if we choose x values without replacement. This is because, even if f is balanced, the randomized algorithm has a nonzero probability of never seeing an x such that $f(x) = 1$.

What Deutsch and Jozsa showed is that a quantum computer can decide whether f is balanced, with certainty, using only two queries to f. What follows is an algorithm to accomplish this.

The algorithm uses two quantum registers, the first having n qubits and the second having only one qubit. We initialize the system to the basis state $|0 \cdots 0\rangle |0\rangle$. Then we apply the Hadamard transform H_{2^n} to the first register (or equivalently, the Hadamard transform H to each qubit of the first register separately). This results in the superposition

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

We then compute $f(x)$ and store the result in the second register. How is this done? If f is a black-box oracle, then (by assumption) we don't need to worry about how it's done. If, on the other hand, f is given explicitly (say, as a Boolean circuit), then we've seen that we can compute f reversibly using Fredkin gates. In either case, the resultant superposition is

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

Next we apply the one-qubit transformation

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

to the second register. This transformation, which is easily seen to be unitary, is a "phase flip": if $f(x) = 0$ then it leaves the amplitude of $|x\rangle |f(x)\rangle$ alone, whereas if $f(x) = 1$ then it inverts the amplitude. So we get

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

At this point we want to erase $f(x)$ from the second register, to allow for proper interference among the states. As we show in Section 4, we cannot ignore the value of this qubit. If we did so, the results of our computation would be as if $f(x) = 0$ with probability $\frac{1}{2}$ and as if $f(x) = 1$ with probability $\frac{1}{2}$. It is as if we have measured $f(x)$, and this is precisely what we do not want because we wish to keep $f(x)$ as a superposition of states. Nonetheless, since the $f(x)$ register is part of an entangled quantum state, we cannot erase this register directly because erasure is not a unitary operation. But since we have an oracle for f, we can simply compute f a second time and CNOT the result into the second register, so that the bit in that register is $f(x) \oplus f(x) = 0$. Doing so, we obtain

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

The final step is to perform another Hadamard transform H_{2^n} on the first register. We could work out the result of this algebraically, but would rather reason about it to obtain more insight. We've seen that the Hadamard transform is its own inverse, and that

$$H_{2^n}(|y\rangle) = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{y \cdot x} |x\rangle .$$

From these it follows that if f is identically zero, then applying H_{2^n} brings us back to $|0 \cdots 0\rangle |0\rangle$. If, on the other hand, f is balanced, then the state

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

is a linear combination of $H_{2^n}(|y\rangle)$ for various values of y. However, none of these values can be $y = 0$, since the state is orthogonal to $H_{2^n}(|0\rangle)$ in the vector space Z_2^n (the two having inner product $2^{n-1} - 2^{n-1} = 0$).

Therefore, if f is zero, then $x = 0$ at the end of the computation, whereas if f is balanced, then $x \neq 0$. So by observing the $|x\rangle$ register, we can decide with certainty whether f is zero or balanced. We've done this using two queries to f and $O(n)$ steps of computation.

4 The Principle of Deferred Measurement

At this point, it is useful to review the principle of deferred measurement. We imagine three different quantum circuits as shown below:

In the first circuit the high qubit is never entangled with the low qubit, and therefore the second Hadamard transform undoes the first. For purposes of comparison with circuit 3, we will write the final state as

$$\frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) + \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) = |0\rangle$$

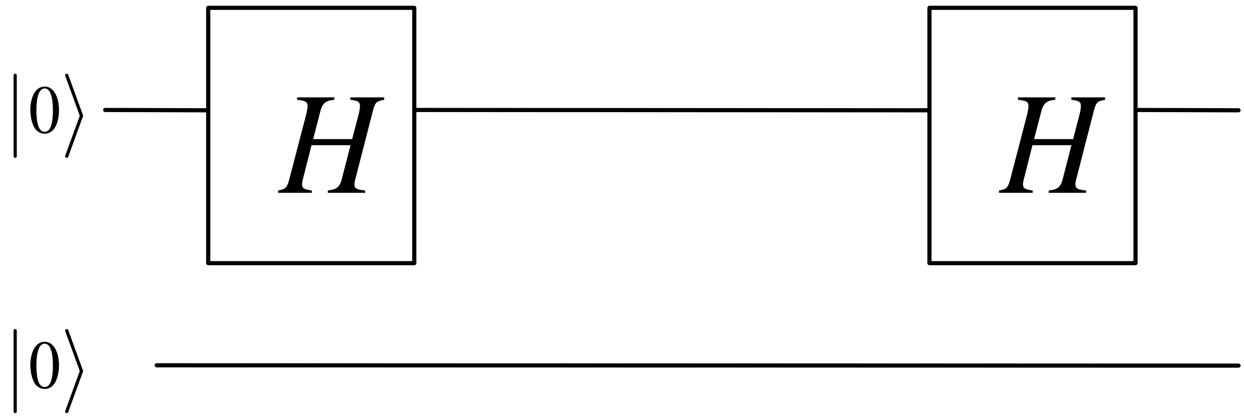

Figure 2: Circuit 1

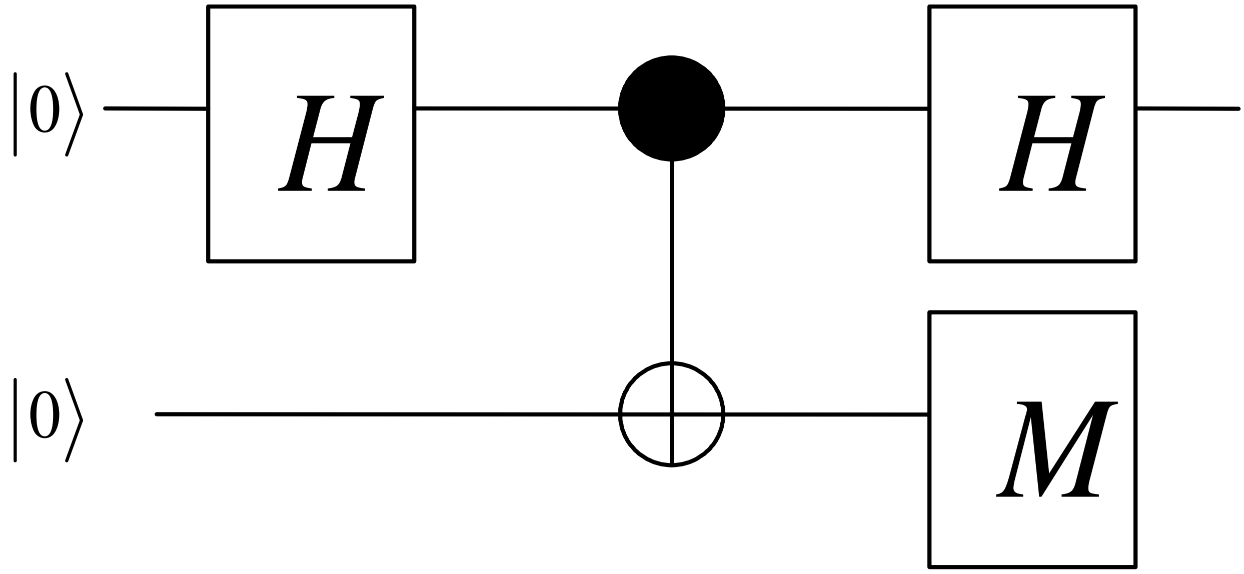

Figure 3: Circuit 2

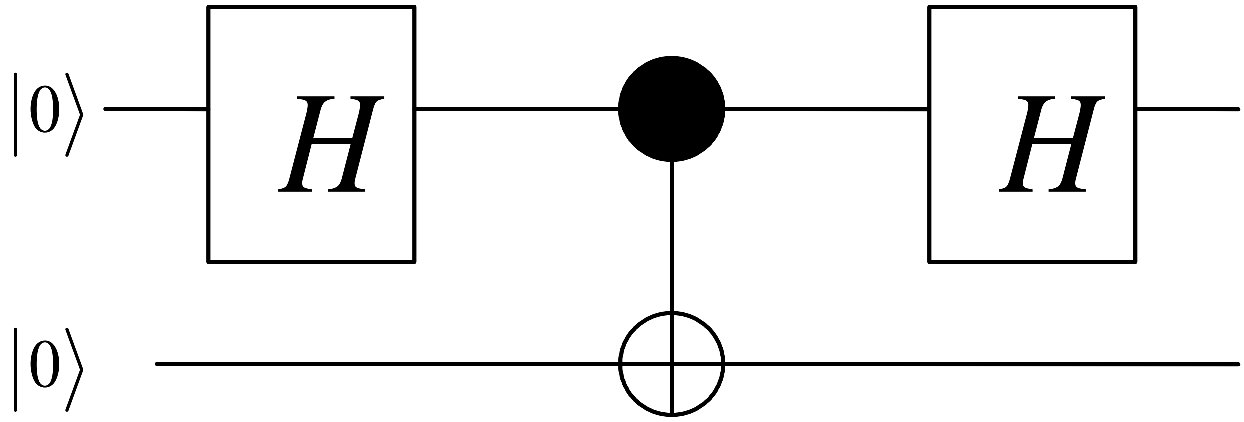

Figure 4: Circuit 3

In the second circuit the CNOT gate places the system in the state

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Measurement of the second qubit places the system in the state $|00\rangle$ with probability $\frac{1}{2}$ and in the state $|11\rangle$ with probability $\frac{1}{2}$. The final state of the first qubit will be $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ with probability $\frac{1}{2}$ and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ with probability $\frac{1}{2}$.

In the third circuit we do not explicitly measure the second qubit. Nonetheless, the fact that the two qubits are entangled means that the effect on the output is the same as that of circuit 2 in which the second qubit is explicitly measured. To see that, let us look at the final state of the circuit. After the CNOT gate, the state of the system is

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

as it was in circuit 2. Since we do not measure the second qubit, the final state of the system is

$$\frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle + \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|1\rangle$$

which cannot be simplified to a single base state as in the first circuit. The result of a measurement on the first qubit would be $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ with probability $\frac{1}{2}$ and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ with probability $\frac{1}{2}$, just as it was when we measured the second qubit. Performing the CNOT operation and subsequently ignoring the bottom qubit has collapsed the space of possible results into two equally probable subspaces in the same way as a measurement would have.

5 Fourier Sampling and Recursive Fourier Sampling

Bernstein and Vazirani realized that what the Deutsch-Jozsa algorithm was really doing was computing a Fourier transform over the Abelian group Z_2^n. This insight led to a *quantum Fourier sampling* algorithm, which accomplishes in only two queries a task requiring n queries for any classical algorithm, deterministic or randomized.

As with the Deutsch-Jozsa algorithm, we're given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, to which we can make oracle queries. This time, the promise is that there exists a "secret string" $a \in \{0,1\}^n$ such that for all inputs $x \in \{0,1\}^n$, $f(x) = a \cdot x$. Here $a \cdot x$ denotes the inner product

$$a_1 x_1 + \cdots + a_n x_n \pmod 2 .$$

The challenge is to determine the secret string a. Another way of saying this is that f is the XOR of some subset of its input bits, and the challenge is to determine the subset.

The string a consists of n bits, and each query yields at most one bit of new information. Therefore any classical algorithm needs at least n queries to determine a. In fact, there exists a classical algorithm using exactly n queries, as well as $O(n)$ steps of computation. The algorithm simply applies the following equations:

$$\begin{aligned} f(100\cdots 0) &= a_0 \\ f(010\cdots 0) &= a_1 \\ &\vdots \\ f(00\cdots 01) &= a_n. \end{aligned}$$

The quantum algorithm for Fourier sampling uses exactly the same operations as does the Deutsch-Jozsa algorithm. Let's see why this is. After we initialize a quantum system to the state $|0\cdots 0\rangle|0\rangle$, apply a Hadamard/Fourier transform to the first register, compute $f(x)$ for each state $|x\rangle$ in the second register, apply a phase flip to those states for which $f(x) = 1$, and finally uncompute $f(x)$ for each state $|x\rangle$, the system is in the state

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

Now, we know the following two properties of the Hadamard/Fourier transform:

$$H_{2^n}(|a\rangle) = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{a \cdot x} |x\rangle |0\rangle$$

and

$$H_{2^n}(H_{2^n}(|\phi\rangle)) = |\phi\rangle$$

for all states $|\phi\rangle$. It follows that, if we apply H_{2^n} to the first register of our system, we will obtain simply $|a\rangle$. This concludes the quantum algorithm for Fourier sampling; it uses two queries and $O(n)$ steps of computation.

But something is unsatisfying. Although switching to a quantum algorithm reduced the query complexity from n to a constant, it did not reduce the computational complexity at all. Yet Bernstein and Vazirani showed that, by nesting Fourier sampling problems one inside another, one can obtain an improvement in computational complexity as well as query complexity.

The idea is as follows. We want each query to f to be expensive, rather than requiring only unit cost. So we associate with each input x a function $g_x : \{0,1\}^{n/2} \to \{0,1\}$. We're promised that, for each x, there exists a secret string $r_x \in \{0,1\}^{n/2}$ such that for all $y \in \{0,1\}^{n/2}$, $g_x(y) = r_x \cdot y$. This means that g_x is a XOR of some secret subset of its input bits, just as f is. Then $f(x)$ is defined to be $c(x, r_x)$ for some known function c. So to evaluate f (classically, at least), we first need to find r_x for some subset of inputs x.

It should be clear how to proceed. In the same way that we attached a function g_x of size $n/2$ to each input of f, for each x we can attach a function $h_{x,y}$ of size $n/4$ to each input of g_x. We then continue the recursion with functions of size $n/8$, $n/16$, and so on for $\log_2 n$ levels. How hard is it to evaluate this nested function?

The quantum Fourier sampling algorithm uses two queries plus $O(n)$ computational steps, so if we apply it recursively, we obtain the recurrence relation

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(n).$$

Here $T(n)$ is the number computational steps needed to evaluate an instance of size n. Solving the recurrence, we obtain that $T(n) = O(n \log n)$. The classical algorithm, on the other hand, uses n queries plus $O(n)$ computational steps, so applying it recursively yields the recurrence relation

$$T(n) \leq nT\left(\frac{n}{2}\right) + O(n).$$

Solving this recurrence yields $T(n) = \Omega\left(n^{\log n}\right)$, which is a super-polynomial growth rate.

Recursive Fourier sampling has particular significance in quantum complexity theory. Recall that BQP is the class of problems solvable in polynomial time on a quantum Turing machine with bounded probability of error. Is BQP contained in NP? Actually the right question to ask is whether BQP is contained in MA (where MA is the probabilistic generalization of NP) (since BPP, which is contained in BQP, is not known to be contained in NP). It can be shown that there is an oracle relative to which Recursive Fourier sampling is not contained in MA, thus showing that there is an oracle relative to which BQP is not contained in MA. It is conjectured that Recursive Fourier sampling is not even in any finite level of the polynomial hierarchy.

6 Simon's Algorithm

As the previous two sections illustrated, the Fourier transform is a powerful tool in the design of quantum algorithms. But the issue is always how to set up the original state $|\phi\rangle$ in such a way that the Fourier transform $H_{2^n}(|\phi\rangle)$ returns useful information. In 1994, Simon found a way to do this for a quite general class of Boolean functions. His algorithm is a predecessor of Shor's algorithm for finding the period of a sequence.

Suppose we're given a function $f : \{0,1\}^n \to \{0,1\}^n$, to which we can make oracle queries. (Note that the outputs of f are n-bit strings, rather than single bits as in the previous examples.) We're promised the following about f: there exists a nonzero secret string $a \in \{0,1\}^n$ such that

- For all inputs $x \in \{0,1\}^n$, $f(x) = f(x \oplus a)$.
- For all inputs $x, y \in \{0,1\}^n$, if $x \neq y \oplus a$, then $f(x) \neq f(y)$.

What these conditions mean is that f is a 2-to-1 function, and that any two inputs mapping to the same output differ in exactly those positions i for which $a_i = 1$. As with Fourier sampling, the challenge is to determine the secret string a.

Let's first understand why this problem is hard to solve classically. Suppose that a is chosen uniformly at random from $\{0,1\}^n - \{0^n\}$. Now consider a classical probabilistic algorithm that's already made k queries, to inputs $x_1, \ldots, x_k$. We want to know how much information the algorithm could have obtained about a. On the one hand, there might be a pair of inputs x_i, x_j (with $1 \leq i, j \leq k$) such that $f(x_i) = f(x_j)$. In this case, the algorithm already has enough information to determine a: a is simply $x_i \oplus x_j$.

On the other hand, suppose no such pair x_i, x_j exists. Then to start, we can eliminate up to $\binom{k}{2}$ possibilities: a is *not* $x_i \oplus x_j$ for any $1 \leq i, j \leq k$. But this leaves at least $2^n - 1 - \binom{k}{2}$ possibilities remaining. We claim that each of these possibilities is equally likely, conditioned on the known information $f(x_1), \ldots, f(x_k)$. To see this, note that by symmetry, we could permute $f(x_1), \ldots, f(x_k)$ to any k strings all of which are unequal. Then applying Bayes' rule,

$$P[a = a^* \mid f(x_1), \ldots, f(x_k) \text{ all unequal}] = \frac{P[f(x_1), \ldots, f(x_k) \text{ all unequal} \mid a = a^*] \, P[a = a^*]}{P[f(x_1), \ldots, f(x_k) \text{ all unequal}]}.$$

The prior probability $P[a = a^*]$ is the same for each a^* that hasn't been eliminated, and

$$P[f(x_1), \ldots, f(x_k) \text{ all unequal} \mid a = a^*] = 1$$

for each of these a^*. Therefore the probability is the same over all a^*.

So if we guess the value of a after step k, our probability of success can be at most one out of $2^n - 1 - \binom{k}{2}$. Clearly we need to wait until

$$\binom{k}{2} \approx 2^n - 1,$$

meaning until $k = \Omega\left(2^{n/2}\right)$.

We now present Simon's quantum algorithm for finding a. The algorithm uses two registers, both with n qubits. We initialize the registers to the basis state $|0 \cdots 0\rangle |0 \cdots 0\rangle$. We then perform the Hadamard/Fourier transform H_{2^n} on the first register, producing the superposition

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

Then, as with the previous two algorithms, we compute $f(x)$ and store the result in the second register, obtaining

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

We then do something new: we observe the second register, even though we're not yet done with the computation. As we'll see in an exercise, this is not strictly necessary: there is a "principle of safe storage" asserting that if we observe bits and then never use them again, then we might as well not have observed them, since the results either way are identical. (And indeed, even if we *were* to use the bits after observing them, it's been shown that any intermediate observations can be deferred until the end of the computation with only a polynomial blowup in resources.) However, intermediate observations can sometimes make quantum algorithms easier to understand pedagogically.

In any case, the superposition after observing is

$$\left(\frac{1}{\sqrt{2}}\left|x\right\rangle + \frac{1}{\sqrt{2}}\left|x \oplus a\right\rangle\right)\left|f\left(x\right)\right\rangle$$

for some input x. We're now done with the second register, so in the discussion to follow, we'll omit it from our notation. The state

$$\frac{1}{\sqrt{2}}\left|x\right\rangle + \frac{1}{\sqrt{2}}\left|x \oplus a\right\rangle$$

clearly contains some information about a—the question is how to extract it. If we observed at this point, we'd obtain a state chosen uniformly at random from $\{0,1\}^n$, obtaining no information at all. Therefore some more computation is required. The key, once again, is to apply the Hadamard/Fourier transform H_{2^n} to the register. Doing so, we obtain a superposition

$$\sum_{z \in \{0,1\}^n} \alpha_z \left|z\right\rangle$$

where

$$\alpha_z = \frac{1}{\sqrt{2}}\frac{1}{2^{n/2}}\left(-1\right)^{z \cdot x} + \frac{1}{\sqrt{2}}\frac{1}{2^{n/2}}\left(-1\right)^{z \cdot (x \oplus a)} = \frac{1}{2^{(n+1)/2}}\left(-1\right)^{z \cdot x}\left[1 + \left(-1\right)^{z \cdot a}\right].$$

There are now two cases. For each z, if $\left(-1\right)^{z \cdot a} = -1$, then $\alpha_z = 0$, whereas if $\left(-1\right)^{z \cdot a} = 1$, then

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

So when we observe the register, with certainty we'll see a z such that $\left(-1\right)^{z \cdot a} = 1$, or equivalently $z \cdot a = 0$. Furthermore, each z such that $z \cdot a = 0$ has an equal probability of occurring. Therefore what we've managed to learn is an equation

$$z_1 a_1 + \cdots + z_n a_n \equiv 0 \pmod{2}$$

where $z = (z_1, \ldots, z_n)$ is chosen uniformly at random from $\{0,1\}^n$. Now, that isn't enough information to determine a, but assuming that $z \neq 0$, it reduces the number of possibilities for a by half. We've made progress.

It should now be clear how to proceed. We run the algorithm over and over, accumulating more and more equations of the form

$$z_1 a_1 + \cdots + z_n a_n \equiv 0 \pmod{2}.$$

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 linearly independent equations—in other words, n equations

$$z^{(1)} \cdot a \equiv 0 \,(\mathrm{mod}\, 2)$$
$$\vdots$$
$$z^{(n)} \cdot a \equiv 0 \,(\mathrm{mod}\, 2)$$

such that the set $\left\{z^{(1)}, \ldots, z^{(n)}\right\}$ forms a basis for the vector space Z_2^n. Thus, our strategy will be to lower-bound the probability that any n equations returned by the algorithm are independent. (Technically, we only need $n-1$ equations, since we know that $0 \cdot a \equiv 0 \,(\mathrm{mod}\, 2)$ is true before we even begin.)

Suppose we already have k linearly independent equations, with associated vectors $z^{(1)}, \ldots, z^{(k)}$. The vectors then span a subspace $S \subseteq Z_2^n$ of size 2^k, consisting of all vectors of the form

$$b_1 z^{(1)} + \cdots + b_k z^{(k)}$$

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

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

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 $0.28879\ldots$ 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 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

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

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 $O(n)$, compared to $\Omega\left(2^{n/2}\right)$ for any classical algorithm. The number of computation steps, though, is at least the number of steps needed to solve a system of linear equations, and the best known upper bound for this is $O\left(n^{2.376}\right)$, due to Coppersmith and Winograd.

One final question. Suppose that f is not a black-box oracle, but rather is given to us explicitly, say as a Boolean circuit. Is there *then* a more efficient classical algorithm for computing a, perhaps even one that runs in polynomial time, as does the quantum algorithm? If $\mathsf{P}=\mathsf{NP}$, then the answer is certainly yes, for having guessed a value of a, we can verify it efficiently by computing $f(x)$ and $f(x \oplus a)$ for any $x \in \{0,1\}^n$. But assuming that $\mathsf{P}\neq\mathsf{NP}$, the answer hinges on whether one can "smuggle" the value of a into a Boolean circuit without it being efficiently detectable by analyzing the circuit. By analogy, a circuit that computes $y^r \pmod{N}$ has a "smuggled" piece of information in it, namely the order of y in the multiplicative group Z_N^*. Shor's period-finding algorithm is able to ferret out this smuggled information, so a polynomial-size circuit for computing $y^r \pmod{N}$ is what provides the "killer app" for Shor's algorithm (since it's what leads to a factoring algorithm). Thus, a different way to phrase our question is, where is the killer app for Simon's algorithm?