Early Quantum Algorithms

Lecture Notes for CS294-6 (Quantum Computing)

Lecturer: Umesh Vazirani Scribe: Scott Aaronson

September 15, 2000

1 Summary

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 covers Fourier sampling and briefly sketches recursive Fourier sampling, and Section 5 presents Simon's algorithm. Finally, Section 6 records a fascinating digression on the modified Church-Turing Thesis, classical physics, and cellular automata.

2 Preliminaries

Last week we saw two primitives for quantum circuits that will be used extensively in the results to follow. Let's briefly review them.

2.1 Reversible Computation

Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a Boolean function. $C(f)$ is the minimum number of gates in a classical circuit computing f. We saw last week that there exists a quantum circuit of size $O(C(f))$ which, for each input x to f, computes the following unitary transformation U_f on m qubits:

$$U_f : |x\rangle |y\rangle \rightarrow |x\rangle |y \oplus f(x)\rangle$$

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,

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

At first sight it might seem that we've computed $f(x)$ simultaneously for each basis state $|x\rangle$ in the superposition. However, were we to make a measurement, we would observe $f(x)$ for only one value of x.

2.2 Hadamard Transform

The second primitive is the Hadamard transform H_{2^n}, which corresponds to the Fourier transform over the abelian group Z_2^n. The correspondence is based on identifying $\{0,1\}^n$ with Z_2^n, where the group operation is bitwise addition modulo 2. One way to define H_{2^n} is as the $2^n\times 2^n$ matrix in which the (x,y) entry is $2^{-n/2}(-1)^{x\cdot y}$. An equivalent way is as follows: let H be the unitary transformation on one qubit defined by the matrix

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

Then $H_{2^n}=H^{\otimes n}$, or H tensored with itself n times. (Incidentally, it is an exercise to show that if U and V are unitary then the tensor product $U\otimes V$ is also unitary.)

3 The Deutsch-Jozsa Algorithm

Suppose we're given a Boolean function $f:\{0,1\}^n\rightarrow\{0,1\}$. We're promised that f is either

(i) identically zero, meaning that $f(x)=0$ for all inputs $x\in\{0,1\}^n$, or else

(ii) 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 whether (i) or (ii) is the case. Our goal is to minimize the number of times we need to invoke f.

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.

What Deutsch and Jozsa showed is that a quantum computer can decide whether f is balanced, with certainty, using only two queries to f. A quantum circuit to accomplish this is shown in Figure 1.

Before explaining the Deutsch-Jozsa algorithm, let us note that 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, it has a nonzero probability of error, equal to 2^{-k}, or slightly less if we choose x values without replacement.

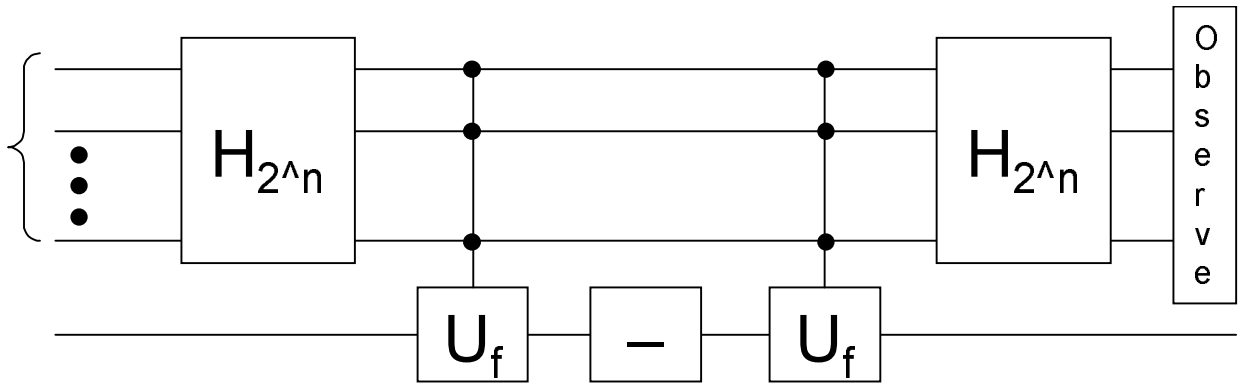


Figure 1: Quantum circuit for Deutsch-Jozsa algorithm, consisting of Hadamard transform H_{2^n}, computation of f, conditional phase flip, erasure of f, another H_{2^n}, and observation.

This is because, even if f is balanced, the randomized algorithm has a nonzero probability of never seeing an x such that $f(x) = 1$.

We now turn to the Deutsch-Jozsa algorithm. 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 $f(x)$ using U_f and store the result in the second register. 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 $|x\rangle |f(x)\rangle$ unchanged, 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. (What goes wrong if we don't erase $f(x)$ is left as an exercise.) We can't do this directly, since erasure is not a unitary operation. However, applying U_f again does this, since U_f has the property of XORing f onto its output bit. So we get

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

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

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

Let $|\Phi_0\rangle = H_{2^n}(|0\rangle)$, and let $|\Phi_1\rangle = H_{2^n}(|y\rangle)$ for any $y \neq 0$. Then $|\Phi_0\rangle \perp |\Phi_1\rangle$, their inner product being 0. Therefore applying H_{2^n} must leave thee two states orthogonal, and since it maps $|\Phi_0\rangle$ onto $|0 \cdots 0\rangle$, it must map $|\Phi_1\rangle$ onto a state orthogonal to $|0 \cdots 0\rangle$.

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 accomplished this using two queries to f and $\Theta(n)$ steps of computation.

4 Fourier Sampling and Recursive Fourier Sampling

Bernstein and Vazirani realized that computing a Fourier transform followed by measuring all the bits of the resulting state is a powerful primitive—called Fourier sampling.

To see this consider a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, with the promise 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. Clearly any classical algorithm needs n invocations of f to accomplish this, yet there is a quantum algorithm than invokes f only twice.

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

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

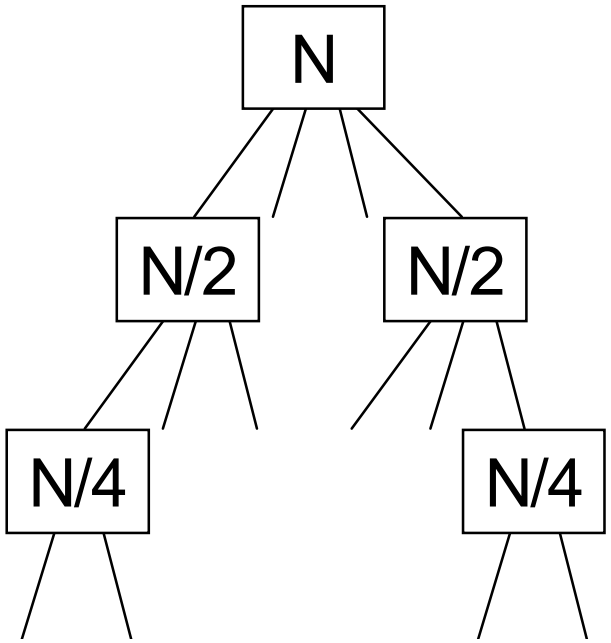


Figure 2: Each query to a k-bit Fourier sampling problem itself involves solving a $k/2$-bit Fourier sampling problem.

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

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

and

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

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 $\Theta(n)$ steps of computation.

Although this gives a task that quantum computers can do $n/2$ times faster than classical (2 queries as opposed to n), we haven't yet demonstrated that quantum computers are superpolynomially faster than classical computers. The idea behind increasing the gap is to make each query itself be the answer to a Fourier sampling problem—thus each query itself is much harder for the classical algorithm than for the quantum algorithm, as shown in Figure 2.

Associate with each input x a function $g_x : \{0,1\}^{n/2} \rightarrow \{0,1\}$. Let's say 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 superpolynomial 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? On the one hand, not even BPP, which is contained in BQP, is known to be contained in NP. (Although recent results suggest that P=BPP, in which case certainly BPP$\subseteq$NP.) Indeed, BQP is not even known to be contained in the polynomial hierarchy. But on the other hand, the most famous problems known to be in BQP, factorization and discrete logarithm, are also in NP. So the question is: are there any candidates for a problem in BQP$-$NP? Currently, recursive Fourier sampling seems to be the only one. Bernstein and Vazirani showed it to be in BQP, yet there is no evidence that it lies in NP. (Strictly speaking, the statement is that relative to an oracle, recursive Fourier sampling is in BQP and not in MA, a generaliation of NP.)

5 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 $f : \{0,1\}^n \rightarrow \{0,1\}^n$, 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.

Claim 1 *Any classical algorithm, deterministic or randomized, needs $\Omega\left(2^{n/2}\right)$ invocations of f to solve this problem.*

Proof. 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,

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

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

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

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 move on to Simon's algorithm. The algorithm uses two registers, both with n qubits, initialized to the basis state $|0\cdots0\rangle|0\cdots0\rangle$. A quantum circuit for a single iteration of the algorithm is shown in Figure 3. Rather than describing the algorithm step by step, we will be concerned to show, first, that each iteration returns nontrivial information about a, and second, that $O(n)$ iterations suffice to determine a.

Claim 2 *After each iteration we learn an equation of the form $z_1 a_1 + \cdots + z_n a_n \equiv 0 \pmod 2$, with $z = (z_1, \ldots, z_n)$ chosen uniformly at random from all z having this property.*

Proof. After U_f we're in the state

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

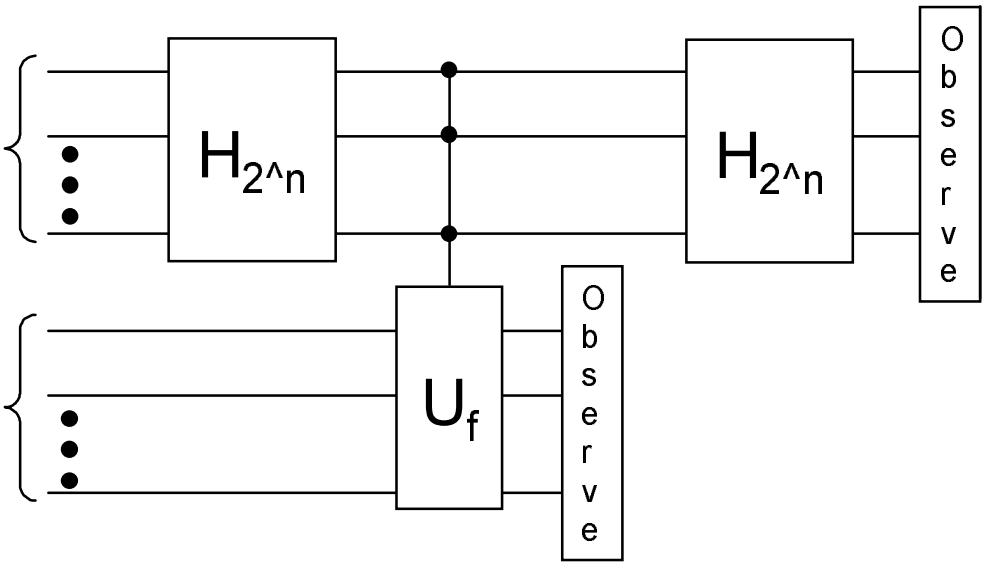


Figure 3: Quantum circuit for one iteration of Simon's algorithm, consisting of Hadamard transform H_{2^n} on first register, computation of f in second register, partial measurement of second register, another H_{2^n} on first register, and measurement of first register.

and the partial measurement reduces this to the state

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

for some input x.

Actually the partial measurement 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. However, partial measurements can sometimes make quantum algorithms easier to understand pedagogically.

In any case, the first register now has state

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

After applying H_{2^n} again, we obtain the superposition

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

where

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

There are now two cases. For each z, if $(-1)^{z \cdot a} = -1$, then $\alpha_z = 0$, whereas if $(-1)^{z \cdot a} = 1$, 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 $(-1)^{z \cdot a} = 1$, 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 among all z having the property. ■

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?

Claim 3 *We can determine a with constant probability of error after repeating the algorithm $O(n)$ times.*

Proof. From linear algebra, we know that a is uniquely determined once we have n linearly independent equations—in other words, n equations

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

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 $0 \cdot a \equiv 0 \pmod 2$ 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

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

To lower-bound this expression, we just 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 has constant lower bound

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

The claim follows. ■

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? 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 the question is, where is the killer app for Simon's algorithm?

6 The Modified Church-Turing Thesis and Classical Physics

In previous sections, we saw three examples of how quantum physics permits seemingly impossible tasks, such as computing an interesting property of n bits using only a constant number of queries to the bits. In this section, we take a step back and consider a more philosophical issue: what, if anything, prevents classical physics from exhibiting comparable computational power?

Underlying classical complexity theory is what is sometimes called the "Modified Church-Turing Thesis." This thesis states that all physically reasonable models of computation can be simulated with polynomial overhead by a probabilistic Turing machine. (The original Church-Turing Thesis has no stipulation of polynomial overhead.) For this discussion, we'll sidestep the difficult issues of what "reasonable" means and of what constitutes a simulation—for example, how the inputs and outputs of a physical process are defined.

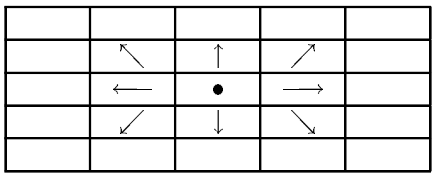
Quantum computing represents a challenge to the Modified Church-Turing Thesis, perhaps the *only* credible challenge. But here we want to ask: where did the thesis originally come from? Before quantum computing was investigated, why might people have considered it plausible?

A basic principle of classical physics is that "every point in space minds its own business." That is, a physical system is described by a set of local differential equations, which might have the form, say,

$$\frac{\partial f(x,y,z)}{\partial t} = R\left[f(x,y,z), \frac{\partial f(x,y,z)}{\partial x}, \frac{\partial f(x,y,z)}{\partial y}, \frac{\partial f(x,y,z)}{\partial z}\right]$$

for some physical quantity f and function R, where (x,y,z) is a point in space. What's important is that the equation makes no reference to any point (x',y',z') separated from (x,y,z) by a distance bounded away from zero. (The partial derivatives, though, make implicit reference to points at distance ε from (x,y,z).)

What does the principle mean in terms of models of computation? Being computer scientists, we like to think of reality as discrete rather than continuous, and we prefer iterative transition rules to differential equations. So consider a "grid world," which for simplicity we'll take to be two-dimensional:



In this world, space is divided into tiny cells, each of which can hold only a finite amount of information at any given time. Time is also discrete: the state of the world at time $t+1$, which we'll call S_{t+1}, depends only on the state S_t during the previous time step. Let U be the transition function mapping from S_t to S_{t+1}. U can be either deterministic or randomized; in the latter case, U depends not only on S_t but also on random bits from some unspecified external source.

Now here is the crucial point. By analogy with local differential equations, U can't be just any function; it has to be one that respects locality constraints. Let $S_t(x,y)$ be the information contained in cell (x,y) at time t. Then $S_{t+1}(x,y)$ can depend only on the information in cell (x,y) and its immediate neighbors at time t. We might define the neighbors, for example, to be

$$I = ((x+a, y+b) : a, b \in \{-1, 0, 1\}).$$

The type of world we've defined is called a cellular automaton. Cellular automata have been extensively studied; the most famous example of them is Conway's Game of Life, invented in 1971. So how are cellular automata relevant to the Modified Church-Turing Thesis? Suppose that the grid world, rather than being infinite, is of size $n \times n$. (Perhaps it wraps around at the edges, or perhaps cells at the edges simply have fewer neighbors than other cells.) The amount of information stored in each cell, as well as the transition function, is fixed and doesn't depend on n. Suppose also that we're given the initial state S_0 of the system. We're then asked a question of the form,

"What is the value of $S_{p(n)}(x,y)$?"

where p is a polynomial. It's easy to see that, if U is deterministic, then we can answer this question using time $O\left(n^2 p(n)\right)$ and space $O\left(n^2\right)$. Furthermore, if U is randomized, then in probabilistic polynomial time we can estimate the probability distribution on $S_{p(n)}(x,y)$ to any desired precision. It follows that, if a physical system were a finite cellular automaton, then given its initial state, we could answer any basic question about how it evolves using time polynomial in its size. In this restricted sense, the Modified Church-Turing Thesis would hold.

On the other hand, even setting quantum effects aside, we don't know that all physical systems are cellular automata. The most accurate measurements in modern physics are to perhaps fifteen digits of precision. But isn't it conceivable that, if only we had better measuring equipment, we could make certain measurements to *unbounded* precision? Even if we couldn't, though, we can imagine a physical system whose "inputs" and "outputs" (whatever that means) belong to some finite set, but which *internally* is able to manipulate quantities having infinite precision:

$$\text{Finite input} \equiv \boxed{\text{Infinite internal precision}} \equiv \text{Finite output}$$

There are many results saying that unbounded-precision systems could be used to perform extraordinary computational tasks. Such tasks could include, for example, factoring, solving PSPACE-complete problems in polynomial time, or even solving non-recursive problems. (The exact amount of computational power depends on how the system is defined.) It isn't surprising that such results plagued early efforts to define computational models with unit-cost arithmetic. Without bounds on precision, a unit-cost model leads to apparent computational absurdities.

The unbounded-precision results also remind one of "analog computers" involving soap films, pulleys and levers, and the like, which superficially seem able to solve NP-complete problems in a single step. The analog computers don't work because they're predicated on a false assumption, that nature always finds the minimum-energy configuration of a system. If the unbounded-precision computers fail, on the other hand, it has to be for a different reason: because regardless of whether nature uses unbounded precision, it doesn't let us exploit it computationally.

But all of this, of course, is speculation. Can one formulate the notion of an unbounded-precision computer more precisely with respect to classical physics? That seems like a worthwhile question to think about.