CSE 599d - Quantum Computing
Simon's Algorithm

Dave Bacon
Department of Computer Science & Engineering, University of Washington

Bernstein and Vazirani showed the first superpolynomial separation (relative to an oracle) between quantum and bounded error classical query complexity. Following this result, Simon was able to come up with a problem for which there is an exponential separation. In this lecture we will study this problem.

I. SIMON'S PROBLEM

In Simon's problem we are given a function from n bit strings to n bit strings, $f : \{0,1\}^n \to \{0,1\}^n$ which is guaranteed to satisfy $f(x) = f(y)$ iff $x = y \oplus s$. Simon's problem is then, by querying $f(x)$ to determine whether the function belongs to (i) $s = 0^n$ or to (ii) $s \neq 0^n$. Actually sometimes Simon's problem is stated as the problem of being given a function which satisfies the promise and then to identify s. We will continue with the tradition of blurring the distinction between these two problems and call both of the Simon's problem.

A. Yao's Principle and a Classical Query Lower Bound for Simon's Algorithm

Here we will introduce a useful method for proving lower bounds of algorithms which utilize randomness which is known as Yao's principle. Yao's principle makes a connection between algorithms which fail with a certain probability and distributions over inputs of deterministic algorithms. Suppose we have an algorithm which computes some function F. Let $R_\epsilon(F)$ denote the minimal complexity *over all algorithms* of the algorithm successfully computing F for all inputs with the probability of failing at most ϵ on any of these inputs. Let $D^\mu_\epsilon(F)$ denote the minimal complexity of computing F deterministically on at least $1-\epsilon$ of the inputs given the probability distribution μ on these inputs. Then the Yao principle states that

$$R_\epsilon(F) = \max_\mu D^\mu_\epsilon(F). \tag{1}$$

In other words if you look at deterministic algorithms and take their performance averaged over their inputs with the worst possible distribution (for the complexity) over these inputs, then this is equal to the complexity of the best algorithm (in terms of complexity) for computing a function which fails with probability ϵ over all inputs. Why is this important? Well mostly because it means that we can bound algorithms with error by placing bounds on deterministic algorithms over a particular distribution σ:

$$R_\epsilon(F) \geq D^\sigma_\epsilon(F). \tag{2}$$

We won't prove Yao's principle, but some of you may be interested to know that its proof comes from von Neumann minmax theorem. I point this out because this will make computer scientists happy. And it will also make physicists and mathematicians happy. von Neumann is neutral ground in the turf wars between the scientific disciplines.

Since we won't prove Yao's principle, you're pretty sure that we're going to use it. Indeed, let's use it on Simon's problem!

We begin by picking μ. With fifty percent probability the function we are querying has $s = 0^n$ and further that f is a random permutation (selected equally from all such permutations.) With fifty percent probability the function has $s \neq 0^n$ and we randomly selection one of these s strings with equal probability. Further we randomly select the values of the function in this case, consistent with the constraint. Now we want to consider deterministic algorithms over this distribution μ which succeeds with error $< 1/4$ (under μ).

In the case that $s = 0^n$, then queries to the function will yield random strings of outputs and each of these outputs will occur with equal probability (under μ.) What about in the case where $s \neq 0^n$? Well if we query k times and get outcomes $f(x_1), f(x_2), \ldots, f(x_k)$ then if we find a $f(x_i) = f(x_j)$, $i \neq j$, we can use it to find s (we assume the queries are all different) via $x_i \oplus x_j = s$. If, however, there is no such matching, then the sequence of f's we get will completely random: just like the case where $s = 0^n$. Thus no deterministic algorithm can distinguish between these two cases with probability greater than fifty percent. Thus if we can establish that the probability of obtaining a sequence (over μ) which has a matching is $1/4$ only after a large number of queries, then we will have established (by

Yao's principle) that randomized algorithm which fail with probability at most 1/4 also requires a large number of queries.

So let's show this. How do we do this? Suppose that we have queried $j-1$ times and found no matching. What is the probability that we obtain a good matching in the jth query? We have excluded $\binom{j-1}{2}$ possible values for s. There are thus $2^n - 1 - \binom{j-1}{2}$ equally possible values for s possible. But we can only have matched one of the previous queries, $f(x_i), i < j$. Thus the probability that we do not obtain a matching in the jth query having not obtained one in the previous $j-1$ queries is

$$1 - \frac{j-1}{2^n - 1 - \binom{j-1}{2}} \tag{3}$$

Using Bayes rule the total probability of not obtaining a match after k queries is therefore

$$Pr = \prod_{j=2}^{k} \left[1 - \frac{j-1}{2^n - 1 - \binom{j-1}{2}} \right] \tag{4}$$

where we define $\binom{1}{2}$ as zero. There are lots of ways to bound this probability. One way to do it is to use the highest order terms to bound

$$Pr \geq 1 - \sum_{j=2}^{k} \left[\frac{j-1}{2^n - 1 - \binom{j-1}{2}} \right] \tag{5}$$

and then we bound this by

$$Pr \geq 1 - \sum_{j=2}^{k} \left[\frac{j-1}{2^n - 1 - \binom{k-1}{2}} \right]. \tag{6}$$

The second of these terms can be explicitly summed,

$$Pr \geq 1 - \frac{\frac{(k-1)k}{2}}{2^n - 1 - \binom{k-1}{2}}. \tag{7}$$

Thus to get this probability small enough, we see that we require

$$k \geq \Omega(2^{n/2}) \tag{8}$$

We have thus shown that a classical algorithm to solve Simon's problem which can fail with at most probability 1/4 requires an exponential number of queries to solve Simon's problem.

B. Quantum Algorithm for Simon's Problem

Well of course having just shown that Simon's problem is hard classically, you can bet what we're going to do next. Yep, we're going to show how Simon's algorithm can be solved efficiently on a quantum computer.

The first difficulty we face when constructing a quantum algorithm for Simon's problem is that we can no longer use our favorite phase kickback trick (well not in the standard way!) Why? A good reason is that we now we have a function from n bits to n bits instead of n bits to 1 bit. Let's just charge ahead anyways!

As in our previous algorithms, we have a unitary which evaluates the function in a reversible manner:

$$U_f = \sum_{x,y \in \{0,1\}^n} |x\rangle\langle x| \otimes |y \oplus f(x)\rangle\langle y| \tag{9}$$

If we input a superposition over all bit strings into the first register of this unitary and $|0^n\rangle$ into the second register, we obtain the state

$$U_f \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |f(x)\rangle \tag{10}$$

Next measure the second register. Now I have to make a little bit of a detour here because I haven't told you how to make a measurement onto one part of a multipartite quantum system.

1. Measurement Rule Re Redux

Recall that we have discussed measurements in a particular basis $\{|\psi_i\rangle\}$. So far we have been working with measurements that we perform on the entire part of our quantum system (or at least in cases where the state is separable between the part of the system we were measuring and the rest of the system.) But what if we want to measure one of two qubits, for instance? How do we calculate the probability of getting a particular outcome and how does the state change?

To answer this we slightly rewrite our measurement description. Instead of describing a measurement by a basis, we describe a measurement by a set of projection operators. What is a projection operator? Recall that it is an operator which squares to identity: $P^2 = P$. Projection operators P and Q are called orthogonal if $PQ = 0$. Then we describe a measurement by a set of orthogonal projection operators $\{P_i\}$ which sum to the identity operator I (on the whole system.) Given such a set of projection operators, the probability of getting out come i associated with projection operator P_i given that the state of the quantum system is $|\psi\rangle$ is given by

$$Pr(i) = \langle\psi|P_i|\psi\rangle \tag{11}$$

Further the quantum state of the system after we make this measurement is given by

$$|\psi'\rangle = \frac{P_i|\psi\rangle}{\sqrt{Pr(i)}} \tag{12}$$

It is easy to check that for a measurement over a complete basis for a system this will reproduce our previous approach. But this is also a more general method because it allows us to measure part of a quantum system.

2. Back to Regularly Scheduled Programming

Now lets return back to our quantum algorithm for Simon's algorithm. When we last left our discussion, we had produced the state

$$|\phi\rangle = U_f \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes |f(x)\rangle. \tag{13}$$

Now what happens if we measure the second register. This measurement is described by 2^n projection operators $I \otimes |i\rangle\langle i|$. The probability of each of these outcomes is

$$\langle\phi|(I \otimes |i\rangle\langle i|)|\phi\rangle = \frac{1}{2^n} \sum_{x\in\{0,1\}^n} \langle x| \otimes \langle f(x)|(I \otimes |i\rangle\langle i|) \sum_{y=0\in\{0,1\}^n} |y\rangle \otimes |f(y)\rangle \tag{14}$$

which is equal to

$$\langle\phi|(I \otimes |i\rangle\langle i|)|\phi\rangle = \frac{1}{2^n} \sum_{x,y\in\{0,1\}^n} \langle x|y\rangle\langle f(x)|i\rangle\langle i|f(y)\rangle = \frac{1}{2^n} \sum_{x\in\{0,1\}^n} |\langle f(x)|i\rangle|^2 \tag{15}$$

Now if $s = 0^n$, then this is $\frac{1}{2^n}$ (since each $f(x)$ is distinct) but if $s \neq 0^n$ then this is $\frac{2}{2^n}$ (since $f(x)$ is 2-to-1.) What is the new state? To use this we calculate

$$(I \otimes |i\rangle\langle i|)|\phi\rangle = (I \otimes |i\rangle\langle i|) \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes |f(x)\rangle \tag{16}$$

If $s = 0^n$, then each of the $f(x)$ are distinct, so this will produce the state (properly normalizing)

$$|f^{-1}(i)\rangle \otimes |i\rangle \tag{17}$$

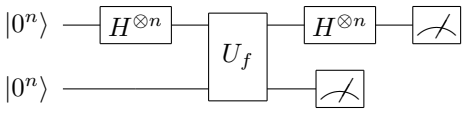
This means that in the first register we will get a random state $|x\rangle$ with equal probability over $x \in \{0,1\}^n$. If, on the other hand, $s \neq 0^n$, there are two values of x such that $f(x) = i$, namely x and $x \oplus s$. Thus we will produce a superposition in the first register over two strings,

$$\frac{1}{\sqrt{2}}(|x\rangle + |x \oplus s\rangle) \tag{18}$$

We obtain this state with equal probability for all such possible strings $x \in \{0,1\}$.

Okay, time to take stock of what we have. We've produced, in the case where $s = 0^n$ a method where in the first register we have a random quantum state, $|x\rangle$, $x \in_U \{0,1\}^n$. In the case where $s \neq 0^n$ we have in the first register a random quantum state, but now it is $\frac{1}{\sqrt{2}}(|x\rangle + |x \oplus s\rangle)$, $x \in_U \{0,1\}^n$. Here we have denoted the uniform bit string at random by $\in_U$. Now if we were to measure this first register we would ge no where. We certainly couldn't find our s: the state in this case would then collapse to either $|x\rangle$ or $|x \oplus s\rangle$ and since x is totally random, these strings will also be totally random.

So what do we do? Well we mimic our previous algorithms. In particular we now perform the n qubit Hadamard on our first register and then measure the result in the computational basis. The full circuit for what we have done so far looks like



(19)

Recall that the n qubit Hadamard is

$$H^{\otimes n} = \sum_{x \in \{0,1\}^n} |\phi_x\rangle\langle x| \tag{20}$$

where

$$|\phi_x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}} (-1)^{x \cdot y} |y\rangle. \tag{21}$$

First lets consider the case where $s = 0^n$. In this case, the first register is in the state $|x\rangle$, $x \in_U \{0,1\}$. If we perform $H^{\otimes n}$ on this state, we produce $|\phi_x\rangle$. If we then measurement $|\phi_x\rangle$ in the computational basis, we obtain a random outcome $|y\rangle$ all with equal probabilities.

Now consider the case where $s \neq 0^n$. In this case, the first register is in the state $\frac{1}{\sqrt{2}}(|x\rangle + |x \oplus s\rangle)$, $x \in_U \{0,1\}$. If we perform $H^{\otimes n}$ on this state, we obtain

$$\frac{1}{\sqrt{2}}(|\phi_x\rangle + |\phi_{x \oplus s}\rangle. \tag{22}$$

We can express this state in the computational basis:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} [(-1)^{x \cdot y} + (-1)^{(x \oplus s) \cdot y}] |y\rangle \tag{23}$$

Since we are working modulo 2, we can factor out the $(-1)^{x \cdot y}$:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} [1 + (-1)^{s \cdot y}] |y\rangle \tag{24}$$

From this expression that we see that the amplitude in front of $|y\rangle$ is non-zero iff $s \cdot y = 0$. Further each of these amplitudes are of equal magnitude ($\frac{1}{\sqrt{2^{n-1}}}$). Thus if we measure this state in the computational basis, we will obtain, uniformly at random a bit string y such that $s \cdot y = 0$.

How does this help us? We now have a method for extracting uniformly random strings y such that $s \cdot y = 0$ for our unknown s. Notice that if $y \neq 0$, then this cuts in *half* the number of possible s strings consistent with this equation. Thus with all but an exponentially small probability we have eliminated the number of possible strings s could be by one half! This is a pretty darn good start, especially when we compare it to the classical case where we with a single query we could do nothing! (That's an infinite ratio, peoples.)

But how do we use this to find s? Well suppose we perform the same routine $n-1$ times and obtain $y_1, y_2, \ldots, y_{n-1}$ strings. We then have $n-1$ equations $y_1 \cdot s = 0, y_2 \cdot s = 0, \ldots, y_{n-1} \cdot s = 0$. If the y_i are linearly independent, then we can efficiently solve these equations (Note that it is often said that you need n linearly independent vectors to solve for s. But this is incorrect, because we have $s \neq 0^n$ and $s = 0^n$ is clearly always a solution to these equations. Thus we really are finding a two dimensional subspace which corresponds to this solution. Thus we need $n-1$ linearly independent vectors to find this subspace.) So what is the probability that we will obtain $n-1$ linearly independent

vectors? Given that we have m linearly independent vectors $y_1, y_2, \ldots, y_m$, then this specifies a subspace of bit strings $\mathbb{Z}_2^n$ of dimension 2^m. The probability that the next bit string, y_{m+1} is linearly independent is $\frac{2^n - 2^m}{2^n}$. Chaining these conditional probability we obtain the probability of $n-1$ being linearly independent:

$$Pr = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{2}{2^n}\right)\cdots\left(1 - \frac{2^{n-2}}{2^n}\right) \tag{25}$$

To bound this we then use the fact that $(1-a)(1-b) \geq 1 - a - b$ for $0 \leq a, b \leq 1$ to turn the products into a sum

$$Pr \geq \left(1 - \sum_{i=2}^{n} \frac{1}{2^i}\right) \tag{26}$$

or explicitly summing this

$$Pr \geq \frac{1}{2} - \frac{1}{2^n} \tag{27}$$

Thus with probability of greater than almost $\frac{1}{2}$ we obtain $n-1$ linearly independent y_is and can solve for s.

What happens if we don't obtain linearly independent y_i's. Well our calculation of s will fail. We can, of course, check this by querying $f(x) = f(x \oplus s)$ for two values. Thus we can determine when our algorithm has gone wrong even if we haven't checked that the y_i are linearly independent. We can continue to run the algorithm multiple times until we find an s which works. If we find no such s, then we guess that $s = 0^n$. The probability that this will fail if we run this $2k$ times is then at most

$$\left(1 - \frac{1}{2}\right)^{2k} < e^{-k} \tag{28}$$

Thus we have shown that there is a bounded error quantum algorithm for Simon's problem which solves the problem using only a polynomial number of queries. Interestingly it is possible to produce an algorithm for Simon's problem which is exact: i.e. which has a worst case running time which is polynomial with no probability of error (See "An Exact Quantum Polynomial-Time Algorithm for Simon's Problem," by Brassard and Hoyer available at `http://arxiv.org/abs/quant-ph/9704027`)

Thus we have shown that in Simon's problem there is a bounded error quantum algorithm which exponentially beats a classical algorithm in query complexity. Notice also that our algorithm for Simon's algorithm is also polynomial in time and space and not just in the query complexity. Simon's discovery in 1993 added another peg beyond Bernstein-Vazirani in the mounting realization that maybe the quantum model of computation could outperform the classical model. But the problems we've considered seem to be of just academic interest right now: have any of you ever encountered Simon's problem in your coding? But these results are important because they gave hope that indeed there were more natural problems which could be more efficiently solved on a quantum computer. And in the next few lectures we will indeed find that this is true!

Acknowledgments

The diagrams in these notes were typeset using Q-circuit a latex utility written by graduate student extraordinaires Steve Flammia and Bryan Eastin.