Quantum Information and Computing
Simple Quantum Algorithms
Topic 10 : Simon Problem

Dipan Kumar Ghosh
Physics Department,
Indian Institute of Technology Powai, Mumbai 400076

March 16, 2017

1 Introduction

In the last lecture we Talked about Bernstein Vazirani problem in which given an n qubit input string, we need to find an unknown n qubit string such that the sum of the bitwise product of these two strings modulo 2 is a given 1 qubit result. Simon problem is a straightforward extension of Bernstein-Vazirani problem in which the input is an n qubit string which has the property that there exists anunknown n qubit string such that the bitwise XOR of this unknown string with a pair of input strings gives the same n qubit output. The unknown string is unique and satisfies the above properties for distinct pairs of inputs.

2 Simon Problem

Simon's problem is a generalization of Bernstein Vazirani problem to the case where the oracle can evaluate a function

$$f : \{0,1\}^n \to \{0,1\}^n$$

i.e., both the input and the output are n bit strings, whereas in Bernstein Vazirani problem, the output is a single bit. The function f has the property that for all inputs x and y,

$$f(x) = f(y)$$

if, and only if, $x = y \oplus \xi$ where $\oplus$ denotes addition modulo 2 and ξ is a non-zero string which is to be determined.

Example 1:
Consider $n = 3$. Let $f(x)$ be a function given by the following table:

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

Note that the pairs of strings which satisfy $f(x) = f(y)$ are delated by $x = y \oplus \xi$ where $\xi = 011$. (e.g. the strings 100 and 111 both evaluate to the same value of $f(x) = 111$ and it is seen that $100 \oplus 011 = 111$, likewise for the other three pairs.)

2.1 Classical Complexity:

Classically, the problem is difficult. We need to take the inputs x sequentially and compute $f(x)$ for each of the inputs and compare it with each of the previously computed value of the function for different strings. The process stops when we have been lucky enough to find a pair x and y such that $f(x) = f(y)$. We can then compute $x \oplus y$ to evaluate ξ.
Consider a deterministic algorithm which takes in a random element x_1 independent of (yet unknown) ξ. The algorithm evaluates $f(x_1)$ and returns an output y_1. This, by itself, does not give any information about ξ. Now take a second element x_2 and evaluate $y_2 = f(x_2)$. There are two possibilities, either $y_1 = y_2$, in which case the string ξ is determined by $\xi = x_1 \oplus x_2$ or $y_1 \neq y_2$. In the former case, the problem is solved while in the latter case, it is still open. The probability of getting $y_1 = y_2$ is $\dfrac{1}{2^n - 1}$ because other than the specific string y_1 there are $2^n - 1$ different possible results. The algorithm fails with a probability $1 - \dfrac{1}{2^n - 1}$ and we would have simply ruled out the possibility of ξ being equal to $x_1 \oplus x_2$.
We now continue the process of choosing a third input x_3 and calculate $y_3 = f(x_3)$. We have to compare y_3 with the two previously calculated outputs y_1 and y_2. If there is no match, we have ruled out $\xi = x_3 \oplus x_1$ and $x_3 \oplus x_2$, which, along with the previously ruled out string $x_1 \oplus x_2$ has so far ruled out 3 possibilities. Continuing like this, suppose, we have already computed $f(x)$ for k different inputs $x_1, x_2, \ldots x_k$ and have still not found a match. This means we have ruled out ${}^kC_2 = k(k-1)/2$ possibilities for the string ξ. We now take in the $k+1$-th input x_{k+1}, compute the value y_{k+1} and ask ourselves what is the probability that we will find a match now.
Out of $2^n - 1$ strings (leaving aside the string $\{0^{\otimes n}\}$, which is trivial string)we have already ruled out $k(k-1)/2$ possibilities for ξ. We are now looking at whether the present result

matches with any of the previous k strings. The probability that we will find a match is

$$P_k = \frac{k}{2^n - 1 - \dfrac{k(k-1)}{2}} \leq \frac{2k}{2^{n+1} - k^2}$$

where the last inequality is obtained by reducing the denominator of the previous expression. This expression holds for every attempt. Hence the probability that there will be a match in the first m attempts is given by adding the probability of success for $k = 2$ to $k = m$,

$$P(\text{success in m attempts}) = \sum_{k=1}^{m} \frac{2k}{2^{n+1} - k^2} \leq \sum_{k=2}^{m} \frac{2m}{2^{n+1} - m^2} \leq \frac{2m^2}{2^{n+1} - m^2}$$

Suppose we decide that the classical algorithm is good if the above probability is at least 3/4, then we must have

$$\frac{2m^2}{2^{n+1} - m^2} \geq \frac{3}{4}$$

which gives

$$m \geq \sqrt{\frac{6}{11} 2^n}$$

which shows that classical algorithm requires exponential number of queries. A quantum computer, on the other hand, can speed up the process dramatically.

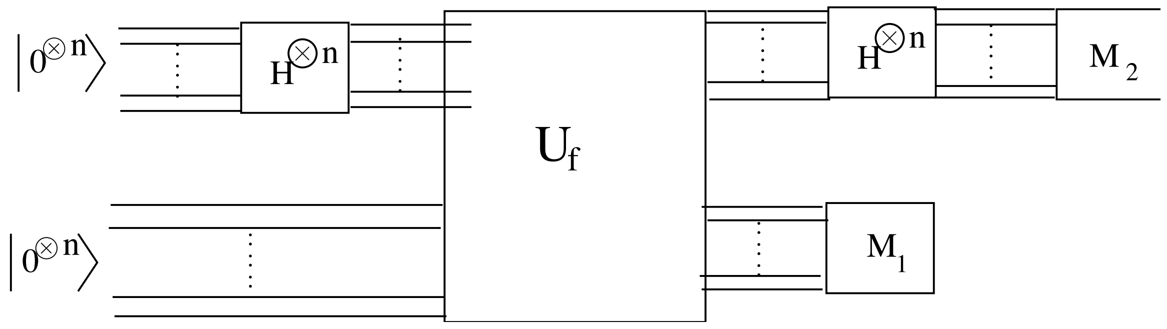
2.2 Quantum Circuit for Simon Problem

The strategy is to use three registers, one input register of n bits, an output register of $m \geq n-1$ bits and a third ancilla register. The oracle U_f that evaluates the function will generate an entangled state of the input and output bits $U_f \mid x\rangle \mid y\rangle \rightarrow \mid x\rangle \mid y \oplus f(x)\rangle$. If the input is prepared as an equal superposition of all possible n-bit basis states (prepared by an initial null string $\mid 0^{\otimes n}\rangle$ being passed through Hadamard gates), the oracle gives the following:

$$U_f \left[\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} \mid x\rangle \mid 0\rangle \right] \rightarrow \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} \mid x\rangle \mid f(x)\rangle \tag{1}$$

At this point, if we query the second register, we would measure a value of $f(x)$. If the function is one-one, i.e. if the string $\xi = \{0^{\otimes n}\}$, each value of $f(x)$ will be measured with a probability of $1/2^n$. If, on the other hand, the function is 2-1, i.e. if $\xi \neq \{0^{\otimes n}\}$, the probability of measuring a particular value of $f(x)$ is $\dfrac{1}{2^{n-1}}$. There are two values of x_0, viz., x and $x_0 + \oplus\xi$ which correspond to the measured value f_0 in the second register. The state in the first register is then a linear combination $\dfrac{1}{\sqrt{2}}[\mid x_0\rangle + \mid x_0 + \xi\rangle]$, so that on measuring the first register, $\mid x_0\rangle$ and $\mid x_0+\xi\rangle$ are obtained with equal probability. A measurement of the first register, therefore, does not give us information about ξ.

Instead of measuring the first register, we pass the n qubits of the register through Hadamard gates and then measure the first register in a computational basis, as shown in the figure. Recall that the second register was measured first (M_1) as a result of which the two registers have become entangled before being passed through the Hadamard gates.



Recalling that the Hadamard transform of $\mid x\rangle$ gives $\dfrac{1}{\sqrt{2}}\sum_{y\in 0,1}(-1)^{x\cdot y}\mid y\rangle$, we get, in the case where $x=\mid 0^{\otimes n}\rangle$, the Hadamard transform of the first register to give

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

When we measure the second register (M_1), we measure a random string with equal probability. Suppose we measure a string a particular string $\mid z\rangle$. Since the function is 2 to 1, $\mid z\rangle = f(\mid x_0\rangle)$ or $f(\mid x_0+\xi\rangle)$. The state in the first register has thus collapsed to a superposition of these two states. Instead of measuring the first register (which would have given us with equal probability, one of these two states), we have passed the collapsed state through Hadamard gates, which gives us for the first register,

$$\frac{1}{2^{(n+1)/2}}\sum_{y\in\{0,1\}^n}[(-1)^{x_0\cdot y}+(-1)^{(x_0+\xi)\cdot y}]\mid y\rangle = \frac{1}{2^{(n+1)/2}}\sum_{y\in\{0,1\}^n}(-1)^{x_0\cdot y}[1+(-1)^{\xi\cdot y}]\mid y\rangle$$

Since $(-1)^{\xi\cdot y}$ can be either $+1$ or -1, depending on whether $\xi\cdot y \pmod 2$ is zero or 1, we have two cases. If $\xi\cdot y=1$, the coefficient in front of $\mid y\rangle$ is zero. If, on the other hand, $\xi\cdot y=0$, since the factor in front becomes 2, we get the state as $\dfrac{1}{2^{(n-1)/2}}\sum_{y\in\{0,1\}^n}(-1)^{x_0\cdot y}\mid y\rangle$, so that the probability of any state $\mid y\rangle$ is $\dfrac{1}{2^{n-1}}$. Thus when we measure the register 1, we obtain a state $\mid y\rangle$ which satisfies $\xi\cdot y=0$, i.e. a vector $\mid y\rangle$ which is orthogonal to ξ, i.e. a state which satisfies

$$\xi_1 y_1+\xi_2 y_2+\cdots+\xi_n y_n = 0 \mod 2 \tag{2}$$

If we assume $y\neq 0$, the number of possible strings satisfying (**??**) reduces by half.

We repeat the algorithm so as to obtain, say $n-1$ different values of y which satisfies (??). Denoting the $r-$ th value of y by y_r, we get a set of r homogeneous equations:

$$\begin{aligned}\xi \cdot y_1 &= 0\\ \xi \cdot y_2 &= 0\\ &\cdots\\ &\cdots\\ \xi \cdot y_{n-1} &= 0\end{aligned}$$

If the $(n-1)$ vectors are linearly independent, we can solve for the vector ξ which is orthonormal to them.

Example 2:

As an illustration, consider the case of 2 to 1 function discussed in Example 1. The table is repeated here for ready reference.

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

Starting with $\mid 000\rangle$ in both the input and the ancilla registers, the Hadamard transform on the input register gives a uniform linear combination of 8 basis states, i.e. the input to the oracle is

$$\frac{1}{\sqrt{8}} \sum_{x\in\{0,1\}^3} \mid x\rangle \mid 000\rangle$$

Since the second register is $\mid 000\rangle$, on application of the oracle, the second register will contain $f(x)$ corresponding to each x. The content of the two registers is given by

$$\begin{aligned}\frac{1}{\sqrt{8}}[&\mid 000\rangle \mid 011\rangle + \mid 001\rangle \mid 010\rangle + \mid 010\rangle \mid 010\rangle + \mid 011\rangle \mid 011\rangle\\ &+ \mid 100\rangle \mid 111\rangle + \mid 101\rangle \mid 110\rangle + \mid 110\rangle \mid 110\rangle + \mid 111\rangle \mid 111\rangle]\end{aligned} \tag{3}$$

Suppose now, the measurement of the second register gives $\mid 111\rangle$. The state of the register is then $\frac{1}{\sqrt{2}}[\mid 100\rangle + \mid 111\rangle]$. When the three bits of the first register is passed through Hadamard gates, we get

$$\frac{1}{\sqrt{2}}\left[\frac{1}{\sqrt{8}}\sum_{y\in\{0,1\}^3}(-1)^{100.y} \mid y\rangle + \frac{1}{\sqrt{8}}\sum_{y\in\{0,1\}^3}(-1)^{111.y} \mid y\rangle\right]$$

We are given that there exists a string ξ such that $100 \oplus \xi = 111$. Thus we may write the above expression as

$$\frac{1}{4}\left[\sum_{y\in\{0,1\}^3} (-1)^{100.y}\left(1+(-1)^{\xi\cdot y}\right) \mid y\rangle\right]$$

The non-zero coefficients of $\mid y\rangle$ in the above equation are those for which $(-1)^{\xi\cdot y} = 1$. For the values of ξ applicable in this case, the content of the first register will be

$$\frac{1}{2}\left[\mid 000\rangle + \mid 011\rangle - \mid 100\rangle - \mid 111\rangle\right]$$

the signs in the above expression are determined by the pre-factor $(-1)^{100\cdot y}$. If a measurement of the first register is made, the four states in the above expression occur with equall probability. Suppose the measurement yields $\mid 100\rangle$. Since ξ is orthogonall to this, we conclude that the left most bit of ξ is 0. If a second measurement gives $\mid 111\rangle$, we can conclude that the last two bits are equal. Since the left most bit is a zero and the function being 2 to 1, the string is not 000, the only alternative for the other two bits is to be 11. Thus $\xi = 011$.

Probability of Success

The probability of success of the algorithm depends on our ability to ensure that the measurement of the first register will generate n independent states. Since each run of the algorithm is independent, the problem is similar to calculation of probability with replacement. Note that the null state being orthogonal to all others, does not yield a two to one function. When we measure the first register, the probability that we *do not* choose the null state is $\frac{2^n-1}{2^n}$, as only one of the 2^n states is the null state. Let the first state picked by y_1. We now run the algorithm once more and draw a second vector y_2. The probability that we do not draw either a null state or the state y_1 is

$$P(y_1, y_2) = \frac{2^n-1}{2^n}\frac{2^n-2}{2^n}$$

Thus the probability of choosing n independent vectors is

$$P_n = \frac{2^n-1}{2^n}\frac{2^n-2}{2^n}\cdots\frac{2^n-2^{n-1}}{2^n}$$

which can be written as

$$P_n = \prod_{k=1}^{n}\left(1-\frac{1}{2^k}\right) \geq \prod_{k=1}^{\infty}\left(1-\frac{1}{2^k}\right)$$

The product above has an approximate value 0.288788 which is greater than 1/4. Thus if we repeat the algorithm $4m$ times the probability of failure is

$$P_{failure} < \left(1-\frac{1}{4}\right)^{4m} < e^{-m}$$

, which is very small even for $4m = 10$.