Introduction to Quantum Information Science
Homework 8

Due Wednesday, November 3 at 11:59 PM

1. Oracle Queries [6 Points] Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, recall that an "XOR query" maps basis states of the form $|x\rangle|a\rangle$ to $|x\rangle|a \oplus f(x)\rangle$, while a "phase query" maps basis states of the form $|x\rangle|a\rangle$ to $(-1)^{a \cdot f(x)}|x\rangle|a\rangle$, where $x \in \{0,1\}^n$ and $a \in \{0,1\}$. Show that given a phase query we can simulate an XOR query and vice versa.
Hint: Consider the most common gates we've seen in class.

Solution: Given an XOR query oracle $U_X : |x\rangle|a\rangle \to |x\rangle|a \oplus f(x)\rangle$, implement a phase oracle $U_P : |x\rangle|a\rangle \to (-1)^{a \cdot f(x)}|x\rangle|a\rangle$ by Hadamarding the a register, applying U_f, then Hadamarding the a register again. For any basis state x, we observe that $|x\rangle|+\rangle$ is always a $+1$ eigenstate of U_X. Moreover, $|x\rangle|-\rangle$ is a $+1$ eigenstate of U_X if $f(x) = 0$, and is a -1 eigenstate if $f(x) = 1$. Because H applied to the a register interchanges the $|0\rangle, |1\rangle$ and $|+\rangle, |-\rangle$ bases for $a \in \{0,1\}$, this conjugation of U_X by Hadamards on the last qubit maps $|x\rangle|a\rangle$ to $(-1)^{a \cdot f(x)}|x\rangle|a\rangle$, implementing a phase oracle. In other words, we have shown that $H_a U_X H_a = U_P$.

Because Hadamard is self-inverse, we also have the identity $U_X = H_a U_P H_a$, which also gives us a way to implement the XOR oracle using a phase oracle.

2. Generalized Deustch-Jozsa Problem Recall that in the Deustch-Jozsa problem we are given oracle access to an unknown function $f\colon \{0,1\} \to \{0,1\}$ and wish to determine if it is constant or balanced. In the generalized Deustch-Jozsa problem we are given oracle access to a function $f\colon \{0,1\}^n \to \{0,1\}$, a function that maps n-bit strings to a single bit. We're promised that the function is either constant or balanced, where balanced means that it has an equal number of 0 and 1 outputs (2^{n-1} of each).

a) [6 Points] Design a quantum algorithm that can determine whether f is constant or balanced using only a single query to f and prove that your algorithm works.

Solution: Apply a Hadamard gate to every input bit, then apply f as a phase query, then Hadamard every qubit again, finally measure and check if the output is back in the state $|0\ldots0\rangle$ — the input state.

The first round of Hadamards puts our state in a uniform superposition over all strings. And applying f then turns this into

$$\frac{1}{2^{n/2}} \sum_x (-1)^{f(x)} |x\rangle \ .$$

Now, if f happens to be constant, then it's easy to see that up to global phase our state will be unaffected. So the second round of Hadamards will put us back in the $|0\ldots0\rangle$ state.

On the other hand, if f is balanced, then following the second round of Hadamards we get

$$\frac{1}{2^n} \sum_x (-1)^{f(x)} \sum_y (-1)^{x \cdot y} |y\rangle$$

whose amplitude for the all zero result is

$$\frac{1}{2^n}\sum_x (-1)^{f(x)}\,.$$

And that's 0 if f is balanced.

b) [Extra credit, 3 Points] Show that any classical deterministic algorithm for this problem requires $\Omega(2^n)$ queries. Therefore, quantum computers give an exponential speedup over deterministic classical computers for this problem.

Solution: We give a relatively informal proof. Suppose a classical algorithm solves this problem using k queries. So, the algorithm only knows k values of the function. Consider all functions which are constant on those k values: some are constant everywhere and some are balanced. To be precise, restricting a Boolean function to be a particular constant output on k inputs leaves many possible functions, of which one is constant and *at least one* is balanced as long as $k \leq 2^n/2 = 2^{n-1}$. If the algorithm makes $k \leq 2^{n-1}$ queries and is correct on some function, guessing it is balanced or constant, then we can always substitute in another function which will appear identical to the algorithm, answering those queries identically, but which is in the other category, so the algorithm is incorrect. Therefore, the algorithm requires at least $\frac{2^n}{2} + 1 = \Omega(2^n)$ queries.

c) [3 Points] Explain why, nevertheless, this speedup is not very impressive, in the sense that it is not an exponential quantum speedup over all classical computing for this problem. Give a full explanation and analysis of the any runtime or likelihood of success involved in your explanation.

Solution: There is an $O(1)$ query randomized algorithm. In particular, have the algorithm check some constant $k > 1$ random inputs to the function, and if it sees all the same answer then output CONSTANT, and if it sees different answers then output BALANCED.

If the function is constant, then the algorithm is always right. On the other hand, if the algorithm is balanced, the probability of failure is the probability that k random inputs all lead to the same output, which is $\frac{1}{2^k}$, which is constant when k is constant.

3. Bernstein-Vazirani In the Bernstein-Vazirani problem, recall that we're given oracle access to a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$. We're promised that there exists a "secret string" $s \in \{0,1\}^n$ such that $f(x) = s\cdot x \bmod 2$ for all x. The problem is to recover s. The Bernstein-Vazirani algorithm solves this problem with just a single quantum query to f. Consider a variant of this problem where we are promised that $f(x) = s \cdot x \bmod 2$ for *at most* a $(1-\epsilon)$ fraction of the inputs x, and that $f(x) = (s \cdot x + 1) \bmod 2$ for the remaining ϵ fraction of inputs.

a) [4 Points] Calculate an upper bound on the probability that a single run of the Bernstein-Vazirani algorithm nevertheless succeeds in recovering s. Show your work.

Solution: We can model the noisy function with $f(x) = (s \cdot x + b_x) \bmod 2$ where $b_x \in \{0,1\}$. The final state output by the Bernstein-Vazirani algorithm will look like

Then the state output by the Bernstein-Vazirani algorithm will be

$$H^{\otimes n} U_f H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{2^n} \sum_{y\in\{0,1\}^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot y}(-1)^{s\cdot x + b_x} |y\rangle\,.$$

Then, we need to calculate the probability that when we measure this state that we see $|s\rangle$.

$$\begin{aligned}
\langle s| H^{\otimes n} U_f H^{\otimes n} |0\rangle^{\otimes n} &= \frac{1}{2^n} \langle s| \sum_{y\in\{0,1\}^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot y}(-1)^{s\cdot x + b_x} |y\rangle \\
&= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{2(s\cdot x)+b_x} \\
&= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{b_x}. \\
&= \frac{1}{2^n} \left[\sum_{x:b_x=0} 1 - \sum_{x:b_x=1} 1 \right]
\end{aligned}$$

By assumption, there are at least $2^n\epsilon$ many strings where $(-1)^{b_x} = -1$ and at most $2^n(1-\epsilon)$ many where $(-1)^{b_x} = 1$. Therefore, the above expression is at most $2^{-n}\left[2^n(1-\epsilon) - 2^n\epsilon\right] = 1 - 2\epsilon$. Squaring this, we get a success probability of at most $1 - 4\epsilon + 4\epsilon^2$.

b) [3 Points] Explain what happens when $\epsilon = 1/2$. Is there a reason why, in some sense, no algorithm can possibly succeed at recovering s in that case?

Solution: When $\epsilon = 1/2$, the probability of observing $|s\rangle$ is at most 0. The reason no algorithm can recover s is that many other secret strings can produce the same output, so it's unclear which of the potential strings was used.

Let $f_1(x) = s \cdot x \bmod 2$. Note that $s \cdot x \bmod 2 \neq (s \cdot x + 1) \bmod 2$ always. So when $\epsilon = 1/2$, f will equal f_1 half of the time and $f \neq f_1$ half of the time. Call the set of x where $f(x) = f_1(x)$ by A and the complement B.

Let $f_2(x) = t \cdot x \bmod 2$ equal $(s \cdot x + 1) \bmod 2$ on exactly the "bad" inputs to f. Since we're only specifying half of the inputs, there are many such t. Since $f_2 \neq f_1$ on many inputs, $s \neq t$.

Given our definition of f_2, let's see how else it relates to f_1. Observe that for any x, the function $f_1(x)+f_2(x) = (s+t)\cdot x \bmod 2$ is 0 mod 2 if and only if $f_1(x) = f_2(x)$. Also, recall that for any nonzero binary string b, $b \cdot x \bmod 2 = 0$ for exactly half of all x. Since $s + t \bmod 2$ is some nonzero binary string, this holds for $s + t$. Therefore, $f_1(x) + f_2(x) = 0 \bmod 2$ for exactly of all x, so $f_1(x) = f_2(x)$ for exactly half of all x and $f_1(x) \neq f_2(x)$ for exactly half — whereas our definition of f_2 only gave a lower bound on the fraction of all x on which they'd disagree.

As we mentioned, many different strings t could be used in f_2. But, for any of those functions, we could define $f_1' = f_2$ and $f_2' = f_1' + 1 = (t \cdot x + 1) \bmod 2$. Further, define a function $g(x)$ which equals $f_1'(x) = f_2(x)$ if $x \in B$ and equals $f_2'(x) = f_2(x) + 1 = f_1(x)$ if $x \in A$. We see that $g(x)$ would satisfy the promises we were given about the function input to our algorithm — that it equals some function of the form $b \cdot x \bmod 2$ half of the time and the complement the other half — and that $g(x) = f(x)$ for all x!

We conclude that the output of f, given the promise we have on its behavior, could have been produced using many different secret strings t. The string s is just one option. There is no way to distinguish them, *even if we could observe $f(x)$ for all x*, since the outputs would all be the same no matter which of the many possible secret strings t were used.