CS333 - Problem Set 9

1. Quantum Hype Chapter II: Limits of Quantum Computation

   We've seen that superposition in combination with interference due to non-positive phases can result in quantum speed-ups. While it is not true that superposition alone gives quantum computers an advantage, you might still assume that quantum computers can in general give a large speed-up for most problems, as long as you apply the proper phases.

   However, we have query complexity lower bounds for many important problems. A quantum query lower bound means it is impossible to solve the problem using fewer than that many queries, even with a quantum computer. Thus lower bounds put a limit on the speed-up possible with a quantum computer. Here are some problems where only a limited, polynomial quantum speed-up is proven to exist (we use Ω to denote an asymptotic lower bound):

Problem	**Quantum Query Complexity Lower Bound**	**Best Classical Query Complexity**
Parity of length-n binary string	$n/2$	n
Searching unordered length-n list	$\Omega(\sqrt{n})$	$O(n)$
Graph connectivity (n vertices)	$\Omega(n^{3/2})$	$O(n^2)$

   Researchers have found that to get exponential quantum speed-ups, the problem needs to have some structure in it, like the promise of a periodic function, that allows all of the phases to interfere in a systematic, rather than random, way. Either that, or the problem should be natural for quantum computers - namely simulating quantum chemistry, which will likely be helpful for drug and industrial chemical engineering problems.

   Of the problems in the above table, of particular importance is searching an unordered list. People want to apply quantum algorithms to solve NP-Complete optimization problems like 3-SAT, but part of the reason 3-SAT is hard is that there is not a lot of structure to the problem - the best classical algorithms are (more-or-less) searching through exponentially many assignments to find one that works. But we see that when it comes to searching through a list of items, quantum computers only give a moderate advantage - a quadratic speed-up, not an exponential speed-up. Using all of the mathematical techniques we have currently developed to analyze quantum algorithms, there is *no evidence* that quantum computers can use the limited structure present in 3-SAT type problems to get a better than quadratic speed-up. A quadratic speed-up of a problem that requires exponential time to solve is obviously better than nothing, but a quadratic speed-up of an exponential time algorithm still results in an exponential time algorithm.

   Despite this, when people are trying to sell the advantages of quantum computers, they frequently say that it is possible that quantum computers will given an advantage on hard optimization problems. They sometimes argue that with our current techniques, we can't analyze everything that a quantum computer will do, and so until we actually build a quantum computer, we won't know whether we will get an advantage or not.

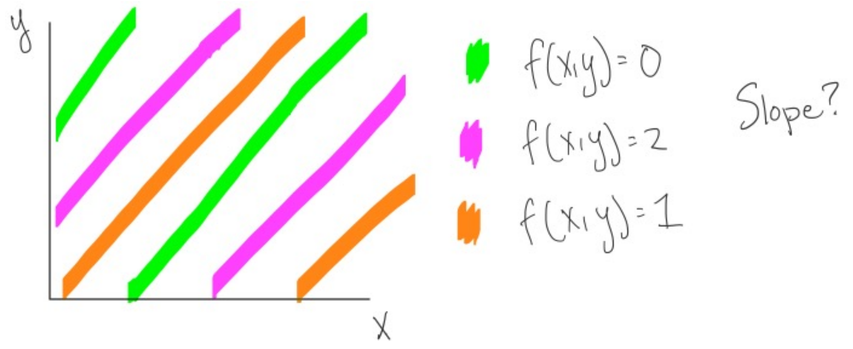
Examples of some of this optimization hype in action:

- See page 27 of IonQ's Investor Presentation. (IonQ is a start-up company that recently raised \$2 billion in public investment.)
- Traffic optimization

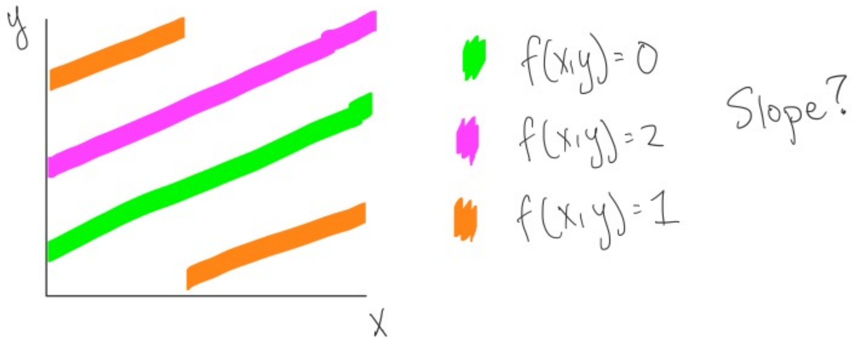
Please write a brief reflection on some or all of the following questions: Based on what you know about quantum computers (which, though still limited, is likely more than what most investors in quantum start-ups know) do you find this build-it-and-see argument compelling? What might be the negative and/or positive consequences of this type of hype? Is the investment in quantum computers worth it if the only advantage for optimization ends up being a quadratic speed-up, and the uses of quantum computers are limited primarily to chemistry simulations?

2. Suppose we have a function f that has a hidden slope m. By this we mean that f takes as input a pair of non-negative integers x and y, that we interpret as x and y coordinates. Then we can imagine plotting a series of parallel lines in the $x - y$ plane, all with slope m. Then $f(x, y)$ takes the same value for all pairs (x, y) on the same line. Given such a function f, our goal is to find the slope, m, using as few queries to f as possible.

   Below are two examples of what f looks like for different values of m, where the value of f is shown using colors, since both the x and y axes correspond to input values (we have periodic boundary conditions, so the lines that extend off of the top of the plot come back up from the bottom):



vs.



More precisely, given a prime number p, let $f\colon \{0,1,\dots,p-1\} \times \{0,1,\dots,p-1\} \to \{0,1,\dots,p-1\}$ such that $f(x,y) = f(x',y')$ if and only if

$$y' - y \equiv m(x'-x) \bmod p \tag{1}$$

for some fixed but unknown integer m. Or equivalently, all pairs (x,y) such that

$$y \equiv mx + b \bmod p \tag{2}$$

have the same output value of f – in particular, they all have output $f(0,b)$, since $(0,b)$ is one point on the line. For each $b \in \{0,1,\dots,p-1\}$, we get a different line, with a different output value of f.

You are given access to U_f which acts on standard basis states as $U_f|x\rangle_A|y\rangle_B|z\rangle_C = |x\rangle_A|y\rangle_B|(z+f(x,y)) \bmod p\rangle_C$ for all $x,y,z \in \{0,1,\dots,p-1\}$. (Note that each of the three registers A, B, and C are p-dimensional states.)

(If you are not familiar with modular arithmetic through basic addition and multiplication, or need a refresher, this article seems to cover the topic clearly and concisely. There are also a wealth of videos online. If you find a good resource, please post to Teams!)

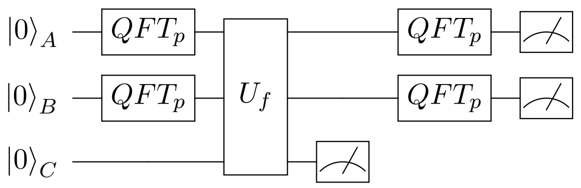
(a) Consider the case that $p = 3$. What is m for the function f with the following truth table?

x	y	$f(x,y)$
0	0	1
0	1	2
0	2	0
1	0	2
1	1	0
1	2	1
2	0	0
2	1	1
2	2	2

(3)

(b) What is the classical probabilistic query complexity of this problem?

(c) Consider the following circuit (which is very similar to the period finding circuit!!)



(4)

i. What is the state of the system after the U_f is applied?

ii. Rewrite your state from part i so that it is in the proper form to analyze the partial measurement on register C. (You need to group together all terms where register C has the same standard basis state, and normalize the A/B states in each term.)

iii. Now analyze the result of the partial measurement on register C. What is the probability of an outcome $|f(0,b^*)\rangle$ occurring? What is the state of the AB system after getting outcome $|f(0,b^*)\rangle$ in the C system?

iv. If the outcome of the measurement on register C is $|f(0,b^*)\rangle$, show that the final state after the last two QFTs is

$$\frac{1}{\sqrt{p}^3}\sum_{j,l=0}^{p-1} e^{2\pi i l b^*/p}\left(\sum_{x=0}^{p-1} e^{2\pi i x(j+lm)/p}\right)|j\rangle|l\rangle \tag{5}$$

v. Explain why when you measure the remaining state in the standard basis, you will get an outcome $|j\rangle|l\rangle$ where $j \equiv -lm \mod p$.

vi. It is a fact from number theory that every number except 0 has a multiplicative inverse $\bmod p$ when p is prime. In other words, if $j \not\equiv 0 \bmod p$, there exists j^{-1} such that $jj^{-1} \equiv 1 \mod p$. Use this fact to explain how to learn m from the outcome of the measurement on registers A and B.

(d) Comment on the quantum advantage for this problem.

3. When analyzing period finding in class, we assumed $m_b = N/r$, but if N/r is not an integer, then actually $m_b = \lceil N/r \rceil$ or $m_b = \lfloor N/r \rfloor$. In this problem, you'll investigate what happens when N/r is not an integer, and you'll also see what is involved in the period finding classical post-processing step. (For more detail on the reduction from factoring to period finding, and on why the continued fractions approach works, it looks like Quantum Computing, a Gentle Introduction, provides a pretty gentle explanation. Link at bottom of syllabus at go/cs333.)

(a) For this problem, let's assume that we get an outcome $|f(b^*)\rangle$ on the second register, where $m_{b^*} = \lceil N/r \rceil$. Then

$$Pr(|y\rangle) = \frac{1}{Nm_{b^*}}\left|\sum_{m=0}^{m_{b^*}-1} e^{-2\pi i m r y/N}\right|^2. \tag{6}$$

Plot $Pr(|y\rangle)$ (note this function is discrete - it should only take values for integer values of y) for $N = 400$, $r = 7$, and $m_b^* = \lceil N/r \rceil$ using any plotting software you prefer (e.g. matplotlib for python, mathematica, etc). Comment on where the values of $Pr(|y\rangle)$ are largest.

(b) You should find that there are several values of y that are particularly likely. Choose one of these - call it y^*. (Do not choose $y^* = 0$).

To find r from y^* we need to find the convergents of the continued fraction of y. To do this, first find the continued fraction of $y^*/N = y^*/400$. (See Wikipedia on Continued Fractions). Then the convergents of a continued fraction are the fractions you get when you stop the algorithm at an intermediate step. Using the example on the wikipedia page, the first convergent is 3, the second is $3 + \frac{1}{4}$, the third is $3 + \frac{1}{4+\frac{1}{12}}$, and the fourth is $3 + \frac{1}{4+\frac{1}{12+\frac{1}{4}}}$. If we write each of these as a proper fraction, we get the following series of convergents: $3, 13/4, 159/49, 199/65$.

Once you create the series of convergents of the continued fraction of y^*/N, consider all those convergents with denominator less than $\sqrt{N}$. Among those, calculate the difference between each convergent and y^*/N. Take the convergent that has the smallest absolute value difference. Then the denominator of that convergent will be either r or a factor of r.

Do this process for y^* and verify that you get r. (You won't get a factor of r because in our case $r = 7$ is prime!)

[Note - in the case that r is not prime, this process might result in a factor of r. Then if you repeat this twice, you will get two outcomes that might both be factors of r. Then r will, with high probability, be the least common multiple of the two factors. You can find the least common multiple of numbers a and b by taking $ab/gcd(a, b)$, where $gcd(a, b)$ is the greatest common denominator of a and b, which can be found efficiently using the Euclidean Algorithm. If you are interested, you can repeat this problem with an example with an r that is not prime, and go through this process again, including this final part of the analysis.]