QuIC Seminar 9

Introduction to the Query Model of Quantum Computing

Contents

Last semester we introduced the *gate model* or *circuit model* of quantum computing[1] and saw several algorithms including quantum teleportation and phase estimation. These algorithms are completely specified by the set of quantum gates acting on them. We'll first recap this model of quantum computing and then introduce the *query model*, which is a superset of the gate model that allows for one additional operation called a *query*. When you first see the query model, it looks like a formal excuse for saying we have no idea how to design quantum algorithms[2] so we'll then spend some time justifying why we even care about this model.

The biggest reason we care is this is what historically led to Shor's algorithm for polynomial time factoring, the most famous result in quantum computing and one of the most famous, important results in all of computer science, physics, mathematics, history, philosophy, music[3]... It was so impactful that it led some famous scientists to change fields[4]. We'll be building up to Shor's algorithm the same way it happened historically, first looking at query model algorithms that are interesting but not necessarily useful, since they're written in terms of a black box. Peter Shor built off these algorithms to come up with his own black box algorithm, but then finished the work by *instantiating the black box*, leading to a polynomial time factoring algorithm that could break essentially all of moden public key cryptography if a good quantum computer were built.

[1] It's called a *circuit* because classical computers use circuits, but then it was realized there's no concept of a "loop" in a quantum "circuit" like there is in a classical circuit. So, people started calling it the *gate* model instead, since it works by throwing in a bunch of quantum gates.

[2] It kind of is...

[3] Okay maybe I got carried away with music. Yet again there's a soundcloud page that posts music corresponding to (and inspired from) quantum circuits. See https://soundcloud.com/quantum-circuit-songs. Don't ask me how I know.

[4] John Preskill was one. See https://blog.ycombinator.com/john-preskill-on-quantum-computing/.

9.1 The Gate (Circuit) Model

The gate model of quantum computing is really just a formal definition of what we mean by a quantum computer.

Definition 9.1 (Gate model of quantum computing.). The **gate model** of quantum computing consists of (1) a set of qubits, sometimes broken up into *registers* (subsets); (2) an ordered set of quantum gates acting on qubits; and (3) a set of measurements on one or more qubits.

This is exactly what we mean by a quantum computer. It's something that implements operations known as quantum gates on a set of qubits, and then measures some or all of the qubits to get a classical bit string as output. The set of operations are ordered in that the order of the gates matters as it determines the algorithm.

9.2 The Query Model

The query model is the same as the gate model with one key addition: access to a "black box" or *oracle* that queries some function f that we want to learn about. Let's make this more precise.

First, let f be a function on bit strings of length m to bit strings of length n, i.e.

$$f : \{0,1\}^m \to \{0,1\}^n. \tag{9.1}$$

This function is going to be of interest to us in that we want to determine something about it. For example, in the Deustch-Jozsa algorithm we'll see later, we'll consider a function on bit strings of length 1 (i.e., just a single bit) and ask if it's constant or not. Other algorithms always put some special structure into the function f that's "easy to see quantumly but hard to see classically." This will become clear when we look at particular examples.

What is this mysterious *oracle* anyhow? How does it tell us information about f? It does so in one of two ways, defined below.

Definition 9.2 (Qubit Query). Let $f : \{0,1\}^m \to \{0,1\}^n$. A **qubit query** is a transformation Q_f that acts on an input register of m qubits, labeled $|x\rangle$, and writes the value of $f(x)$ into an *answer* register of n qubits labeled $|a\rangle$ via the following:

$$Q_f|x, a, w\rangle = |x, a \oplus f(x), w\rangle. \tag{9.2}$$

Here, $\oplus$ denotes addition modulo two, and the register $|w\rangle$ denotes any/all qubits not involved in the query but may be doing some other *work.*

Exercise 41: Prove that the qubit query in (9.2) is unitary—this is quantum computing after all! (*Hint: prove it's self inverse.*)

The other type of query, known as a *phase query*, is defined below.

Definition 9.3. Let $f : \{0,1\}^m \to \{0,1\}$. A **phase query** is a transformation Q_f that acts on an input register of m qubits, labeled $|x\rangle$, and writes the value of $f(x)$ into the *phase* of the resulting state via

$$Q_f|x, c, w\rangle = (-1)^{f(x)\cdot c}|x, c, w\rangle. \tag{9.3}$$

Here, the register of one qubit labeled $|c\rangle$ is a *control register* that controls whether the query happens[a]. As above, the register $|w\rangle$ is any/all qubits doing *work* that are not involved in the query.

[a]Setting $c = 0$ guarantees that nothing happens in the query.

Exercise 42: Prove that the phase query (9.3) is unitary.

Exercise 43: Prove that the qubit query (9.2) and the phase query (9.3) are equivalent. That is, prove any algorithm with only qubit queries can be done by an algorithm with only phase queries, and vice versa.

Example 1: Examples of queries.
These definitions might seem a bit abstract by presenting them in the most general way. For concreteness, let f be a function on one bit, $f : \{0,1\} \to \{0,1\}$, and consider a quantum algorithm with only one register, the input register $|x\rangle$ where $x \in \{0,1\}$. (That is, x labels a computational basis state.) Then, the phase query (9.3) is simply

$$Q_f|x\rangle = (-1)^{f(x)}|x\rangle. \tag{9.4}$$

We could extend this to some arbitrary qubit state (not strictly a computational basis state) by acting on the state, for example, $|+\rangle = |0\rangle + |1\rangle$ (up to normalization). Then,

$$Q_f|+\rangle = Q_f(|0\rangle + |1\rangle) = Q_f|0\rangle + Q_f|1\rangle = (-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle. \tag{9.5}$$

This form of "quantum parallelism" is key to getting "rapid solutions of problems by quantum computation."[a]

[a] Reference to the title of the first "fast" quantum algorithm paper by David Deustch and Richard Jozsa. See https://www.isical.ac.in/ rcbose/internship/lectures2016/rt08deutschjozsa.pdf for the original paper, we'll discuss this algorithm shortly.

Now that we've discussed quantum queries, you may be asking what a classical query is. A classical query is even simpler than a quantum one, and it works as follows.

Definition 9.4 (Classical query.). Let $f : \{0,1\}^m \to \{0,1\}^n$. A **classical query** is a transformation[a] that inputs a bit string $z \in \{0,1\}^m$ and outputs $f(z)$:

$$C_f(z) = f(z). \tag{9.6}$$

[a] Pedantic mathematicians might call it a *functional.*

We're going to see examples of query model quantum algorithms that outperform query model classical algorithms, where we're measuring performance by number of queries made. I know what you're thinking: "Well of course the quantum query model is going to perform better. You cheated to allow it to query multiple inputs at once!"

This is a fair point to raise, but it's easily dismissed. At the surface, (9.5) looks like two of (9.6). The key thing here is *superposition.* Quantum states can be in superpositions of basis states labelled by bit strings. Classical states cannot—they're simply one bit string. This model isn't "cheating" by using multiple queries at once, it's *exploiting the laws of quantum physics* by using multiple queries at once. The difference is in the model of computation, not in the type of query[5].

9.3 Why Care about the Query Model?

You still may be thinking: "Okay, so even if I accept these *types* of queries, why on earth should I accept the *query model*? We haven't said anything about how to actually implement these queries in terms of the quantum gates we know and love. What's the point? Why do we care?"

This is a fair point that can be addressed as follows.

1. Query model algorithms can rule out fast quantum algorithms.

[5] "Well then why didn't we define quantum queries and classical queries in the same way? The quantum definition looks all weird with phases and addition modulo two." The answer is that we have to define quantum queries this way to make them reversible. Also, the concept of *phase* makes no sense in classical computing.

- Regardless of how to actually implement a query in terms of quantum gates, we can all agree that a query is *at least* one quantum gate. Therefore, the number of queries is a *lower bound* on the number of total gates. If a particular query model algorithm takes *exponentially many* queries to the oracle, we can be sure the algorithm will take more than polnomially many gates.

2. The query model of classical computing is well studied.

   - In this sense it's natural to consider the same model of computing in the quantum world.

3. It gives insight into how quantum algorithms work. Instantiating the "black box" or query Q_f in terms of quantum gates can lead to fast quantum algorithms.

   - Peter Shor came up with his famous *gate model* algorithm for factoring by building off literature on the query model.

Scott Aaronson has compared the query model of quantum computing to perturbation theory in quantum mechanics[6]. Problems in quantum mechanics are hard to solve analytically, so we resort to perturbation theory, which isn't a complete solution, but tells us something about what we want to know. In the same sense, allowing queries to an oracle hasn't completely specified our quantum algorithm, but it's told us something—namely, that for certain problems, quantum computers can solve them in much fewer queries than classical computers.

9.4 The Deustch-Jozsa Algorithm

9.4.1 Defining the Problem

Alright, let's finally get to a query model quantum algorithm. Here's the first one ever: the now-called Deustch-Jozsa algorithm after the authors. Here's the problem we want to solve:

Definition 9.5 (The Deustch-Jozsa Problem.). Let $f : \{0,1\} \to \{0,1\}$. Does $f(0) = f(1)$?

This is often stated in a much more pedantic way for reasons I don't understand[7]: "Is f *constant* or *balanced*? By *constant*, we mean $f(0) = f(1)$, and by *balanced* we mean $f(0) \neq f(1)$. This is to say in much more words than are necessary what we have already said: does $f(0) = f(1)$?

Exercise 44: State the Deustch-Jozsa problem in an even *more* pedantic way, as follows. *Determine the value of the parity*

$$f(0) \oplus f(1) \tag{9.7}$$

where again $\oplus$ denotes addition modulo two. Show that if f is constant then the parity is zero, and if f is balanced then the parity is one.

Now we get to the interesting part: solving the problem.

Exercise 45: How many *classical* queries (9.6) does it take to solve this problem?

That's a pretty easy exercise. It's two. It must be two since the values of $f(0)$ and $f(1)$ are completely independent of each other. We have to ask what both values are, then compare to see if they're the same or different.

Exercise 46: How many *quantum* phase queries (9.3) does it take to solve this problem?

[6]See the three video lectures by Scott Aaronson at the 2018 Boulder School on Quantum Information at https://boulderschool.yale.edu/2018/boulder-school-2018-lecture-notes.

[7]It's to make the problem sound cooler, I guess. By the way, most of these query model algorithms work by writing down some algorithm and *then* defining the problem based on what the algorithm does. Then you present it *backwards* to make it sound more impressive in the standard mathematical way.

To steal Scott Aaronson's joke, if you were to expect a quantum speedup, you'd probably say one! ("It's got to be an integer, it's less than two, can't be zero...") Interestingly, unlike the classical algorithm, we don't learn explicitly what either of the values $f(0)$ or $f(1)$ are, we only learn if $f(0) = f(1)$ or not. (Equivalently, the parity.)

9.4.2 Solving the Problem

Now for the interesting part: the quantum algorithm that solves the DJ problem!

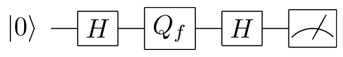

Figure 9.1: The DJ algorithm written in the quantum circuit model. Here, Q_f is a phase query.

How does this work? Let's work through the math, which only takes a few lines. We'll omit normalization coefficents throughout.

$$\begin{aligned}
|0\rangle &\longmapsto |0\rangle + |1\rangle && \text{(First Hadamard)} \\
&\longmapsto (-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle && \text{(Phase query)} \\
&\longmapsto (-1)^{f(0)}[|0\rangle + |1\rangle] + (-1)^{f(1)}[|0\rangle - |1\rangle] =: |\psi\rangle && \text{(Second Hadamard).}
\end{aligned}$$

We can write the resulting state, call it $|\psi\rangle$ in a more insightful way to see what's going on:

$$|\psi\rangle = \left[(-1)^{f(0)} + (-1)^{f(1)}\right] |0\rangle + \left[(-1)^{f(0)} - (-1)^{f(1)}\right] |1\rangle \tag{9.8}$$

How does this solve our problem? Consider the case where $f(0) = f(1)$. Then, the coefficient of the $|1\rangle$ state goes to zero and the coefficient of the $|0\rangle$ state becomes 1 (when properly normalized)[8]. If $f(0) \neq f(1)$, the opposite happens, and the resulting state is $|1\rangle$. Thus, if we measure 0, we know the state must have been $|0\rangle$, meaning that $f(0) = f(1)$. If we measure 1, we know the state must have been $|1\rangle$, meaning that $f(0) \neq f(1)$. Voila! One query.

9.4.3 Discussion of the DJ Algorithm

What's the key thing happening in the DJ algorithm that allows us to solve the problem in only one query? It's *interference*, both *constructive* and *destructive* interference. (Of course superposition is at play here too, since interference depnds on superposition.) In both cases, the state encoding the answer gets constructive interference and the state with the wrong answer gets destructive interference. This is at the heart of all (good) quantum algorithms. The difficulty is how to choreograph such an interference pattern when *you don't know what the answer is a priori.* That is, you don't know what states should get constructive interference and which states should get destructive interference.

Example 2: Simulating the DJ Algorithm with light.
Figure 9.2 shows how one could implement the DJ algorithm using light and standard optical equipment. A beam of light is sent through a beam splitter and into the "black box," which could be realized by a person who knows the function values, just as the oracle does, and implements π phase shifters appropriately on each beam. Specifically, if $f(0) = 1$, shift the phase of the top beam, else do nothing. Similarly for the bottom beam with $f(1)$. We then reflect both beams[a] using mirrors to combine them at the detector. If we measure any light, we know there must be constructive interference between the two paths, hence the function is constant. If we measure no light, there must be destructive interference, telling us the function is balanced.

Note that this is really a classical simulation of the DJ algorithm, since we're not really using any

[8]Note that global phase doesn't matter, so $-|0\rangle$ and $|0\rangle$ are really the same state as far as measurement statistics are concerned.

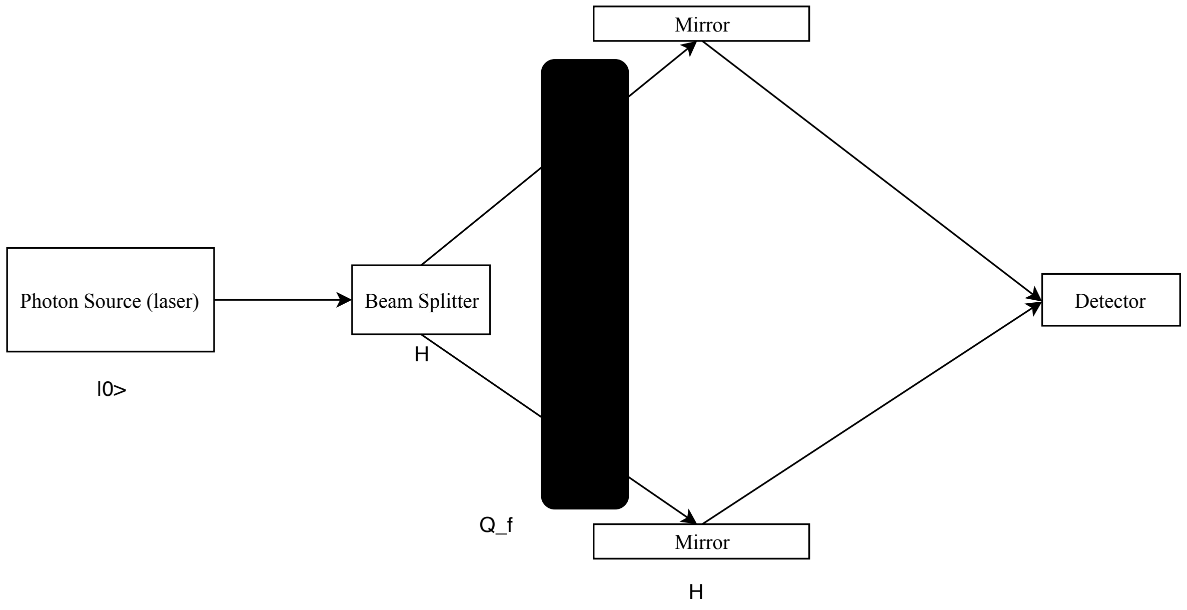

Figure 9.2: A (classical) optical simulation of the DJ algorithm using standard optical equipment. Labels below optical equipment and the black box show the corresponding component in the quantum algorithm shown in Fig. 9.1.

quantum phenomena of the light here. It's just a classical beam. If we wanted to make it quantum, we could restrict to a single photon from the photon source.

[a]Hadmard is self-inverse. What's the inverse of a beam splitter? A mirror!

Exercise 47: You may be unsettled by the fact that the person implementing the black box has to do things separately on each path—i.e., she's doing two things, one on each "beam" or path. Redesign the experiment in Fig. 9.2 so that she only has to implement one operation on one path.