CSE491/596 Lecture Mon. 12/04/23: Small-Scale Quantum Applications

David Deutsch, drawing on two papers by Feynman and other sources, introduced quantum computing while I was a graduate student and he was a postdoc at Oxford in the mid-1980s. At first, he claimed quantum computers could solve the Halting Problem in finite time. Fellows of Oxford's Mathematical Institute refuted the claim. But it was not crazy: a year ago it was proved that a binary quantum system of "interactive provers" ***can*** (kind-of-)solve the Halting Problem in finite time. (My review of the paper is at https://rjlipton.wordpress.com/2020/01/15/halting-is-poly-time-quantum-provable/) Per my memory of observing some meetings about it, the gap in Deutsch's argument had to do with properties of probability measures based on infinite binary sequences.

So Deutsch fell back on something less ambitious: demonstrating that there was a "very finite" task that quantum computers can do and classical ones cannot. (Well, unless the playing field is leveled for them...but before we argue about it, let's see the task.)

Deutsch's Algorithm

The task is a **learning problem**, a kind of interaction we haven't covered until this last day. Instead of "input x, compute $y = f(x)$", a learning problem is to determine facts about an initially-unknown entity f that you can **query**.

1. **Oracle Turing machines** give a classic way to define this kind of problem. For oracle functions f or languages A drawn from a limited class---such as subclasses of the regular languages---can we design an OTM M that on input 0^n (for large enough n) can distinguish what A is in time (say) polynomial in n? The computation $M^A(0^n)$ can learn about A by making queries y on selected strings y and observing the answers $A(y)$.
2. One can also define **oracle circuits** that have special **oracle gates** with some number m of input wires and enough output wires to give the answer $f(y)$ on any $y \in \{0,1\}^m$.
3. An ordinary electrical test kit behaves that way. It is a circuit with a place(s) for you to insert one or more (possibly-defective) electrical components A. The test results should diagnose electrical facts about A.
4. Quantum circuits for all of the Deutsch, Deutsch-Jozsa, Simon, Shor, and Grover algorithms work this way. They involve an **oracle function** $f : \{0,1\}^n \rightarrow \{0,1\}^r$ given in **reversible form** as the function $F : \{0,1\}^{n+r} \rightarrow \{0,1\}^{n+r}$ defined by:

$$F(x,z) = (x, f(x) \oplus z).$$

Usually z is 0^r and the comma is just concatenation (i.e., tensor product) so the output is just $xf(x)$. In the simplest case $n = r = 1$, F is a two-(qu)bit function. Some examples:

- If f is the identity function, $f(x) = x$, then $F(x,z) = (x, x \oplus z) = \mathrm{CNOT}(x,z)$.

- If $f(x) = \neg x$, then $F(x,z) = (x, x \leftrightarrow z)$: $F(00) = 01$, $F(01) = 00$, $F(10) = 10$, $F(11) = 11$.
- If f is always false, i.e., $f(x) = 0$, then F is the identity function.
- If $f(x) = 1$, then $F(x,z) = (x, \neg z)$, so $F(00) = 01$, $F(01) = 00$, $F(10) = 11$, $F(11) = 10$.

These are all deterministic as functions of two-qubit basis states, so they permute the quantum coordinates $1 = 00$, $2 = 01$, $3 = 10$, and $4 = 11$. Recall that $\mathbf{CNOT}$ gives the permutation that swaps the coordinates 3 and 4, that is, $\mathbf{CNOT} = (3\ 4)$ in swap notation. In full, we have:

$$F_{id} = (3\ 4), \quad F_{\neg} = (1\ 2), \quad F_0 = (), \quad F_1 = (1\ 2)(3\ 4).$$

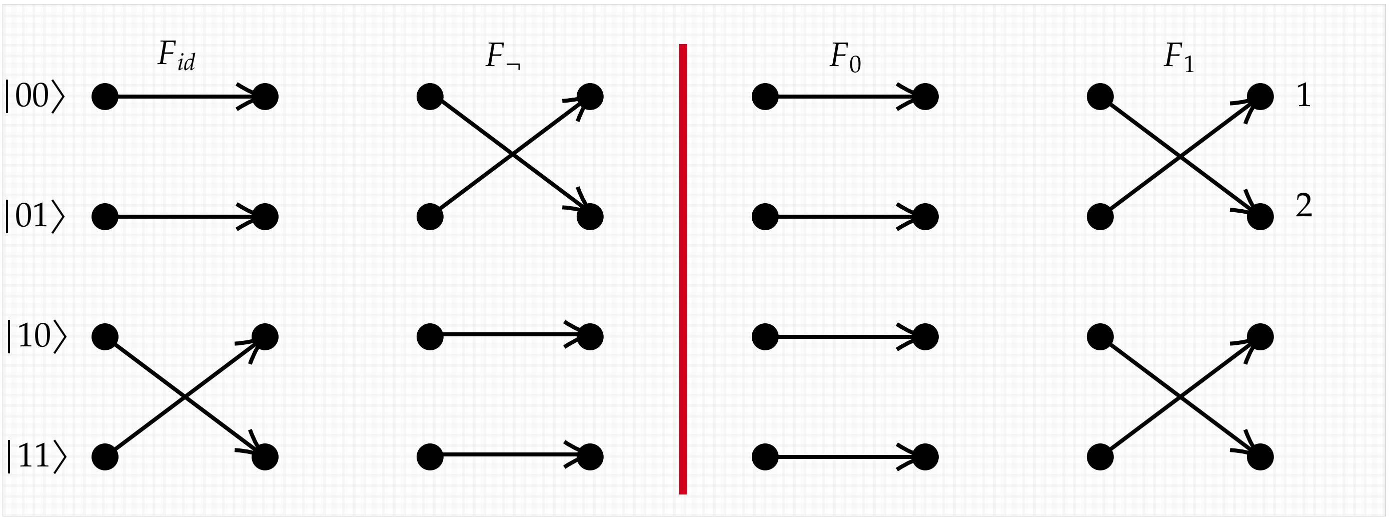
The functions $f(x) = 0$ and $f(x) = 1$ are *constant*. The identity and $\neg$ functions have one true and one false value each, so they *balance* values of 0 and 1. The question posed by Deutsch is:

How many queries are needed to tell whether f is constant from whether f is balanced?

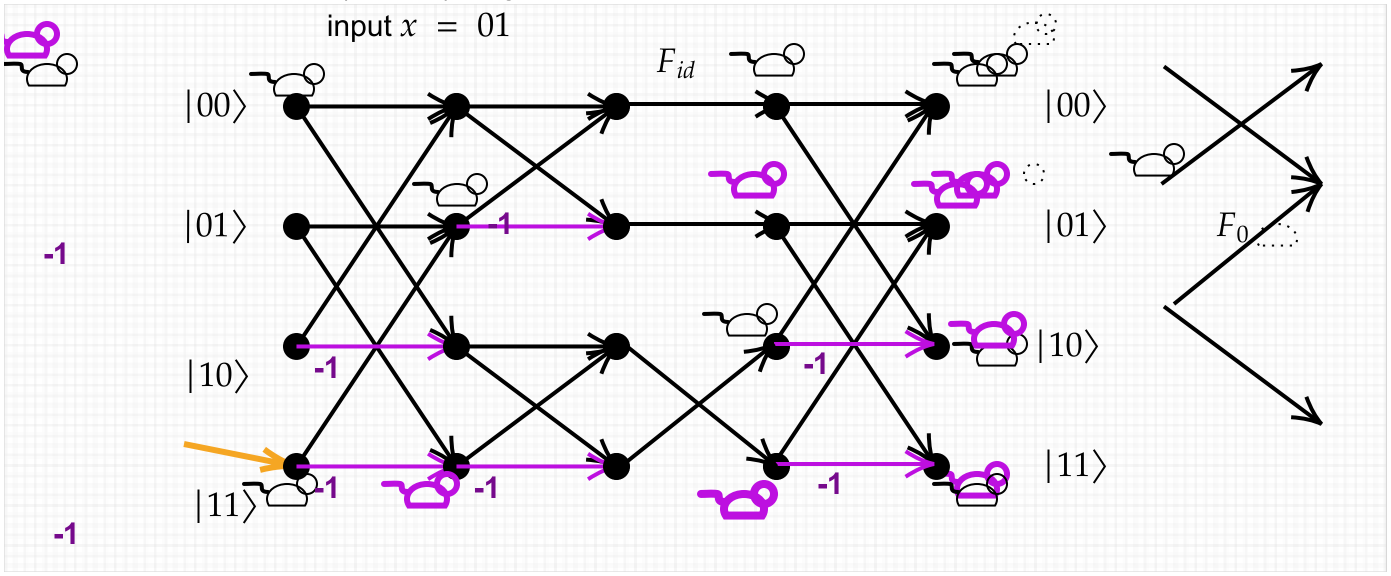
If we just think of f, suppose we try the query $y = 0$ and ask for $f(y)$. If we get the answer " $f(0) = 0$" then it f could be constant-false, but f could also be the balanced identity function. The answer $f(0) = 1$ would leave both constant-true and negation as possibilities. Likewise if we try $y = 1$. The first point is that this impossibility of hitting things with one query carries forward to the way we have to modify the problem for quantum:

How many queries are needed to tell (F_{id} *or* $F_{\neg}$) apart from (F_0 *or* F_1)?

It seems like we have more of a chance because now we can query two things: 00, 01, 10, or 11. Or in the permutation view, we can query $y = 1, 2, 3$, or 4. The problem is that the range of answers we can get is too limited for this to help. $F(1)$ and $F(2)$ can only be 1 ro 2; $F(3)$ and $F(4)$ can only be 3 or 4. So suppose you query $y = 3$ and get the answer 4. Then F could be F_{id} or F could be F_1. The basic problem for a classical algorithm is that every quadrant of the following diagram has both a straight and a cross:



A quantum circuit, however, can make one query to an oracle gate for any of these four functions, and can distinguish a member of the first pair from a member of the second pair by the answer to one qubit after a measurement. The input is not $|00\rangle$ but instead $|01\rangle$; that is, the ancilla is initialized to 1, not to 0. Here is the wavefront ("maze") diagram of how it works:



There is, IMHO, an "unfair" aspect of the comparison. The classical algorithm is being allowed to evaluate the oracle only at basis vectors. The quantum algorithm gets to evaluate it at a linear combination. We can represent this state using the Dirac notation from Chapter 14 as

$$|+-\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

If we do the kind of linear extension of Boolean logic that was covered as the "Binary Linear Equations" presentation option, then we can solve the problem in one shot classically by evaluating at the point $(1,-1,1,-1)$ and seeing where the $-$ signs end up in the resulting vector. FYI: https://rjlipton.wordpress.com/2011/10/26/quantum-chocolate-boxes/

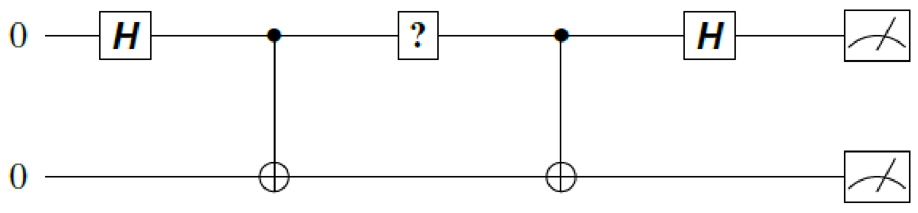
Superdense Coding

It is easy to rig cases F_0, F_1, F_2, F_3 where you can distinguish them exactly by asking one query and measuring both qubits. Just define $F_i(00) = i$, for instance. "Superdense coding" is a case where the rigging has a bit of surprise because it appears to convey 2 bits of information with just 1 qubit of communication. This is impossible by the following theorem:

Holevo's Theorem: It is not possible to extract more than q bits of classical information from any q-qubit quantum state.

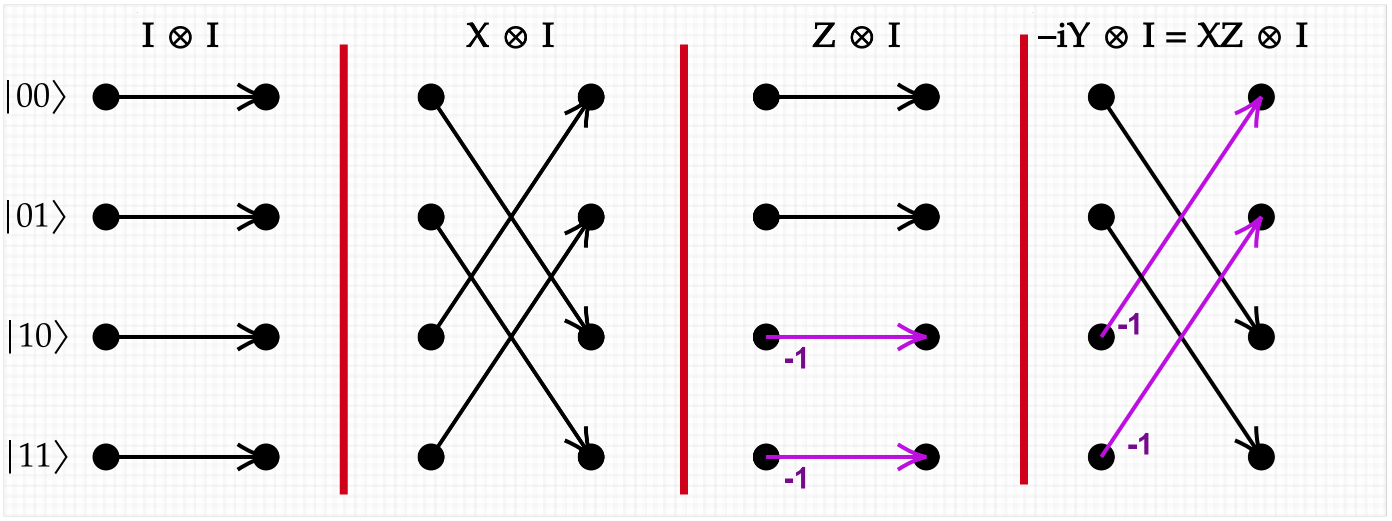
The most important case where this "bites" IMHO is with graph states: You can input $\sim \frac{1}{2}n^2$ bits of information by choosing the $\mathbf{CZ}$ gates for edges of an undirected n-vertex graph G in a graph-state circuit C_G on n qubits, one for each vertex. But you can only get n bits of information out by measuring. Hence graph-state encoding is majorly *lossy* and is often used only for special classes of graphs that already have low information content, such as "grid graphs."

The "cheat" in superdense coding is that the communicating parties "Alice" and "Bob" exchange 1 bit of information beforehand in order to set them up with an entangled qubit pair. Here is their circuit:

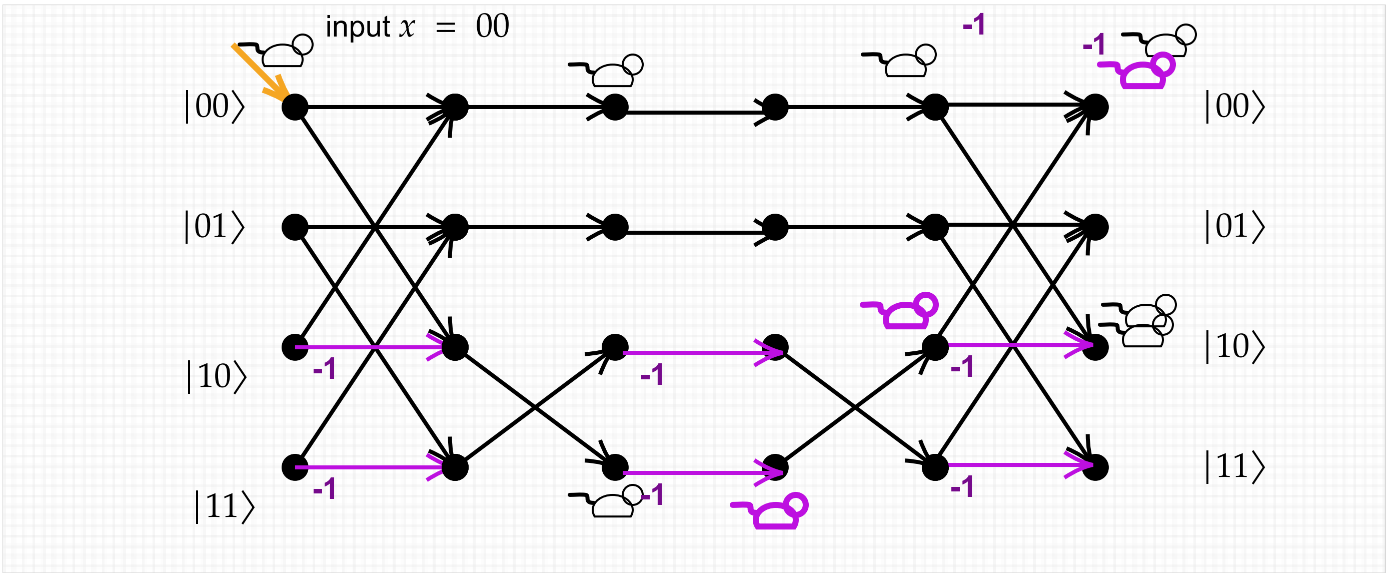


The opening Hadamard and CNOT set up the entangled pair. Alice then chooses one of the four Pauli operators for the unknown operation in the middle. After the second CNOT, she applies Hadamard to her qubit, measures, and sends the result to Bob. Bob then measures his qubit, and is able to infer which of the four operators Alice used. Well, he got a qubit from Alice to begin with, and even though it was before Alice made her 2-bit choice at the "?", it counts as 2 bits of "contact" anyway.

Even after the "magic" is explained away, this remains a nice illustration of a Deutsch-style learning problem using the four Pauli matrices. We want to identify one of the following four possibilities **exactly** by the results of **two** qubits.



This time the input is $|00\rangle$. To work it out via wavefronts (the figure below is left with $\mathbf{XZ} \otimes \mathbf{I}$ in the middle, but all four will be exemplified):



[Lecture stopped short of trying to cover quantum teleportation on Monday and also promised to say more about the setup of "superdense coding" as to why it seems to give something for nothing. Between then and Wednesday I also showed the following big Hadamard tarnsform matrix, so as to set up the identity $\mathbf{H}[x,y] = (-1)^{x\bullet y}$, where $x \bullet y = \sum_i x_i \cdot y_i \bmod 2$ is the "Boolean inner product" of the equal-length binary strings x and y.]

$$\left[\begin{array}{ccccccccccccccccc}
\mathbf{H} & 0000 & 0001 & 0010 & 0011 & 0100 & 0101 & 0110 & 0111 & 1000 & 1001 & 1010 & 1011 & 1100 & 1101 & 1110 & 1111 \\
0000 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0001 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
0010 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0011 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 \\
0100 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
0101 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
0110 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
0111 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
1000 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1001 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\
1010 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\
1011 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\
1100 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\
1101 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\
1110 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\
1111 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1
\end{array}\right]$$

$$\mathbf{H}[u,v] = (-1)^{u\bullet v}$$