## CSE439 Week 6: Small-Scale Applications

Last part of HW3, Problem (1b): I may have confused the fact that

$$(-[a,b]) \otimes [c,d] \;=\; [a,b] \otimes (-[c,d]) \;=\; -([a,b] \otimes [c,d])$$

with having $|\mathbf{u}\rangle = [a,b]$ become $|\mathbf{u}'\rangle = [a,-b]$. Note $|+\rangle = \frac{1}{\sqrt{2}}[1,1]$ while $|-\rangle = \frac{1}{\sqrt{2}}[1,-1]$.
Clarifying hint on (b): try $|v\rangle = |-\rangle$, see what you get, and then try to tell---is it the only possibility?

At last we get into some famous instances of quantum applications. The first one was cooked up just to show that quantum computations could meet a goal that classical algorithms cannot. Whether the goal is compared *fairly* is open to debate, however. Here is some back-story:

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 yet. 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** $f : \{0,1\}^n \to \{0,1\}^r$ given in **reversible form** as the function $F : \{0,1\}^{n+r} \to \{0,1\}^{n+r}$ 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 $F(x,z) = (x, x \oplus z) =$ **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 two-qubit 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 **CNOT** gives the permutation that swaps the coordinates 3 and 4, that is, **CNOT** $= (3\ 4)$ in swap notation. In full, we have:

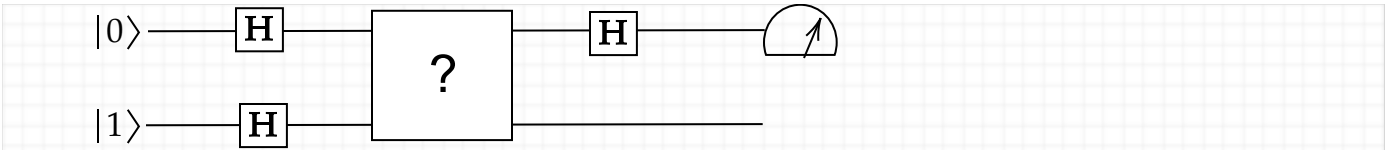
$$F_{id} = -), \quad F_{\neg} = (1\ 2), \quad F_0 = (), \quad F_1 = (1\ 2)(3\ 4).$$

The functions $f(x) = 0$ and $f(x) = 1$ are *constant*. The identity and ¬ 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?

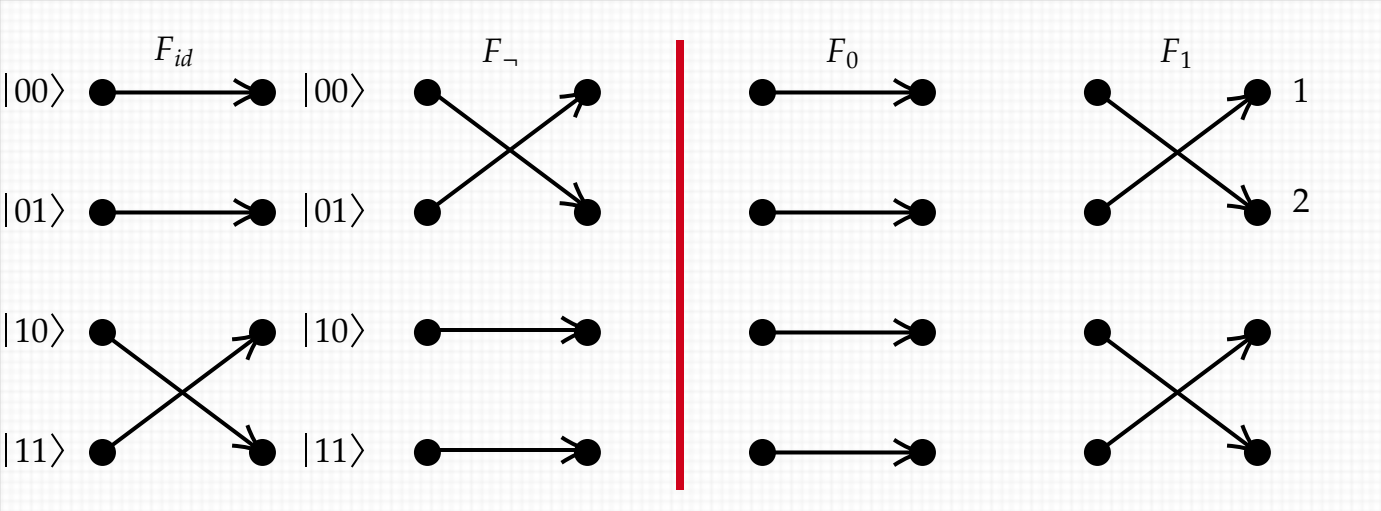
If we just think of $f$, suppose we try the query $y = 0$ and ask for $f(y)$. If we get the answer "$f(0) = 0$" then it $f$ could be constant-false, but $f$ could also be the balanced identity function. The answer $f(0) = 1$ would leave both constant-true and negation as possibilities. Likewise if we 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* $F_1$)?

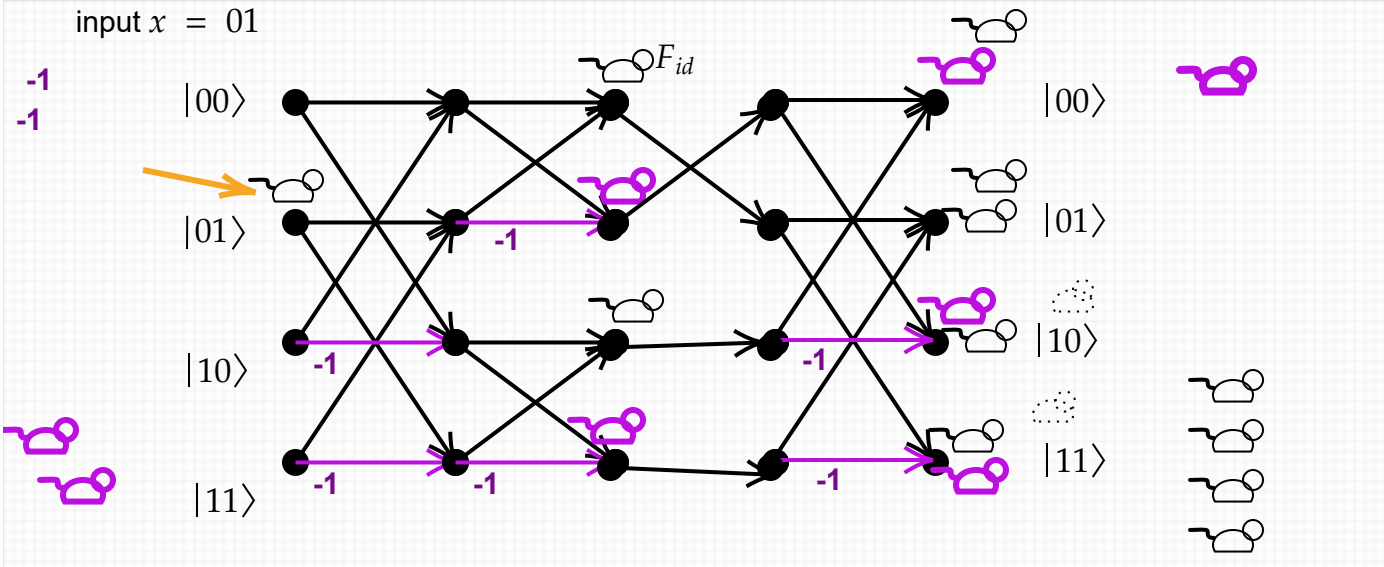



It seems like we have more of a chance because now we can query two things: $00$, $01$, $10$, or $11$. Or in the permutation view, we can query $y = 1, 2, 3$, or $4$. The problem is that the range of answers we

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:




### Interlude: Is the comparison fair?

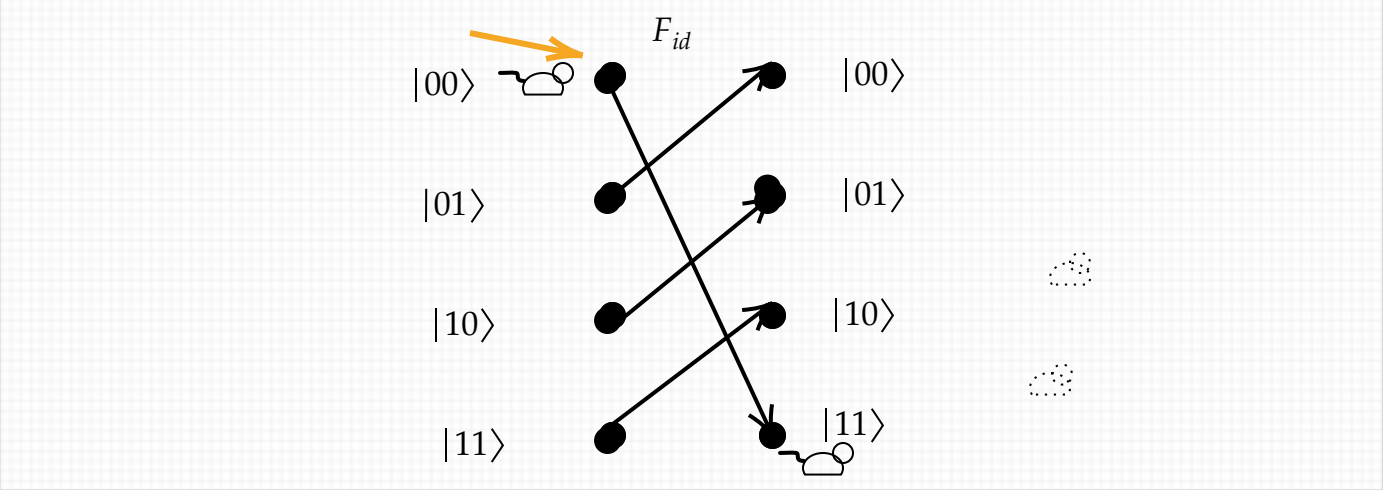
The unfair aspect (IMHO) is that the classical algorithm is being allowed to evaluate the oracle only at basis vectors. The quantum algorithm gets to evaluate it at a linear combination. We can represent this state using the Dirac notation from Chapter 14 as

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

Well, suppose we allow evaluating ordinary Boolean functions at linear combinations of 0 and 1, such as **0.25**. We are really talking about the algebraic equivalents $f'$ of these functions:

- If $f$ is the identity function, $f(x) = x$, then $f'(x) = x$ too, but as algebra.
- If $f(x) = \neg x$, then $f'(x) = 1 - x$. Maybe the only non-obvious choice?
- If $f$ is always false, i.e., $f(x) = 0$, then $f'(x) = 0$. i.e., $f'$ is always zero too.
- If $f(x) = 1$, then $f'(x) = 1$ too.

If we evaluate the unknown $f$ at $0.25$, then we get four different answers that distinguish the four possibilities entirely, not just telling "balanced" apart from "constant." So the classical algorithm does even better---and still with just one "query."
FYI: https://rjlipton.wordpress.com/2011/10/26/quantum-chocolate-boxes/




## Superdense Coding

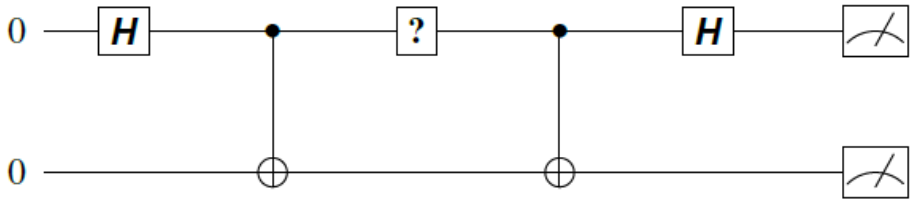
It is easy to rig cases $F_0, F_1, F_2, F_3$ where you can distinguish them exactly by asking one query and measuring both qubits. Just define $F_i(00) = i$, for instance, where $i$ ranges over $\{00, 01, 10, 11\}$---or if you prefer, $i$ ranges over the permutation elements $1, 2, 3, 4$ as used above---and have the other values $F(01), F(10), F(11)$ go in cycle after that. See the above diagram for $F(11)$.

"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 *after* a certain point in time. 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.

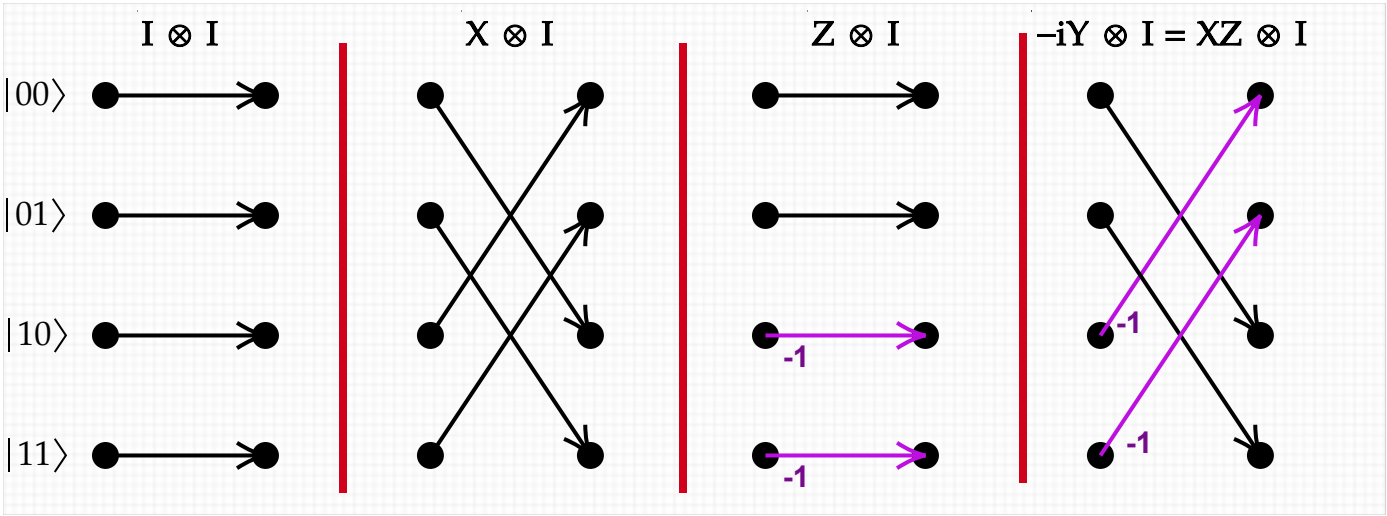
The most important case where this "bites" IMHO is with graph states: You can input $\sim \frac{1}{2}n^2$ bits of information by choosing the $\mathbf{CZ}$ gates for edges of an undirected $n$-vertex graph $G$ in a graph-state circuit $C_G$ on $n$ qubits, one for each vertex. But you can only get $n$ 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:

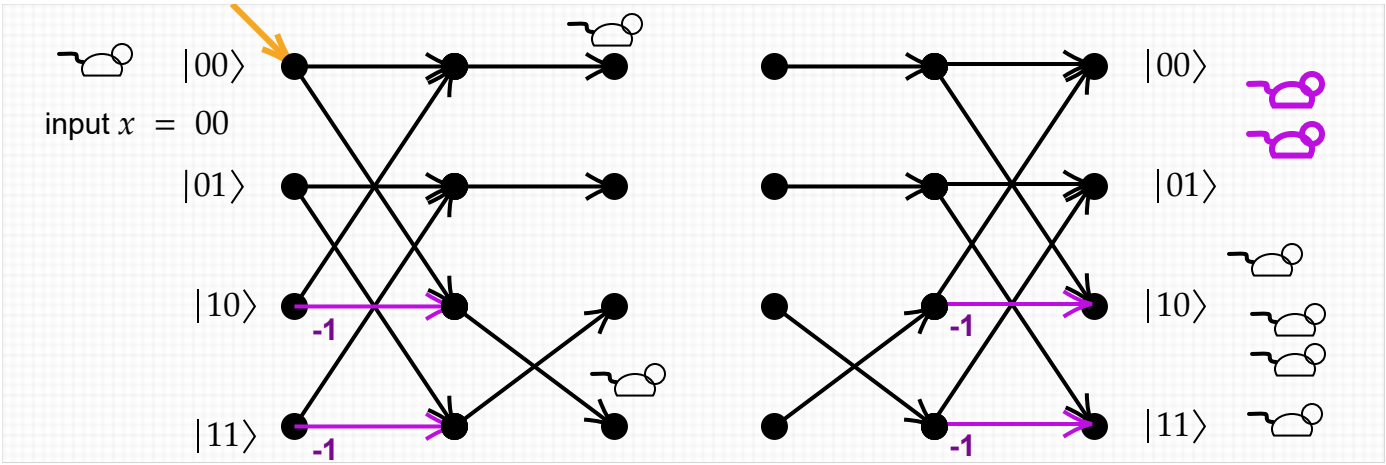



The opening Hadamard and CNOT set up the entangled pair. Alice then chooses one of the four Pauli operators for the unknown operation in the middle. After the second CNOT, she applies Hadamard to her qubit, measures, and sends the result to Bob. Bob then measures his qubit, and is able to infer which 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.



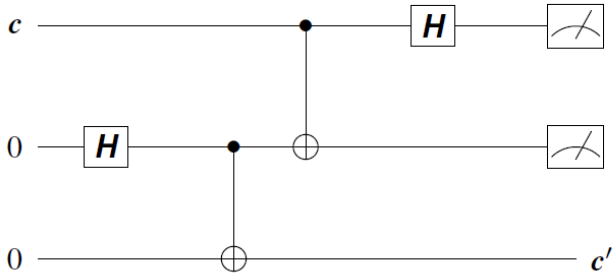

This time the input is $|00\rangle$. To work it out via wavefronts (the figure below is left with $\mathbf{XZ \otimes I}$ in the middle, but all four will be exemplified):




## Example: Quantum Teleportation

Quantum teleportation involves three qubits, two initially owned by Alice and one by Bob. Alice and Bob share entangled qubits as before, whereas Alice's other qubit is in an arbitrary (pure) state $\boldsymbol{c} = a\boldsymbol{e}_0 + b\boldsymbol{e}_1$. Alice has no knowledge of this state and hence cannot tell Bob how to prepare it, yet entirely by means of operations on her side of the lake she can ensure that Bob can possess a qubit in the identical state.

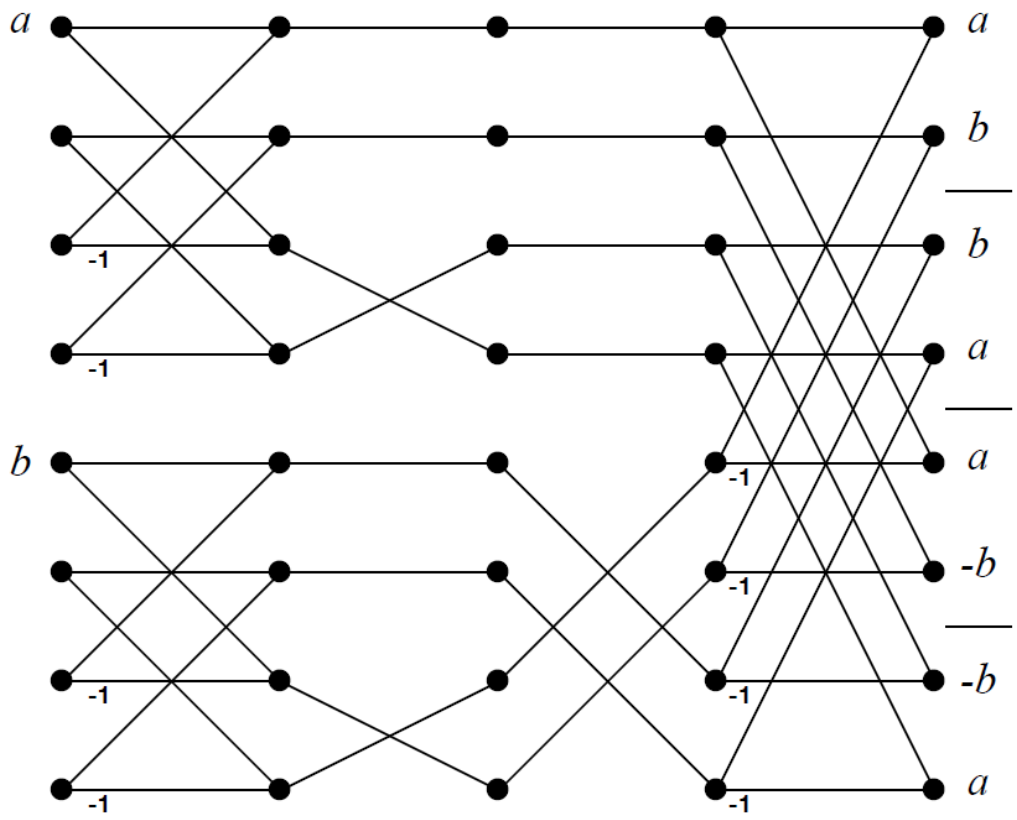
The following quantum circuit shows the operations, with $\boldsymbol{c}$ in the first quantum coordinate, Alice's entangled qubit second, and Bob's last. The circuit includes the Hadamard and ***CNOT*** gates used to entangle the latter two qubits.




With this indexing, the start state is $\boldsymbol{c} \otimes \boldsymbol{e}_{00}$, which equals $a\boldsymbol{e}_{000} + b\boldsymbol{e}_{100}$. After the first two gates, the state is

$$\boldsymbol{c} \otimes \frac{1}{\sqrt{2}} (\boldsymbol{e}_{00} + \boldsymbol{e}_{11}),$$

with Alice still in possession of the first coordinate of the entangled basis vectors. The point is that the rest of the circuit involves operations by Alice alone, including the measurements, all done on her side of the lake. This is different from using a two-qubit swap gate to switch the $\boldsymbol{c}$ part to Bob, which would cross the lake. No quantum interference is involved, so a maze diagram helps visualize the results even with "arbitrary-phase Phils" lined up at the first and fifth rows shown in figure 8.5, which are the entrances for $\boldsymbol{e}_{000}$ and $\boldsymbol{e}_{100}$.




Because Bob's qubit is the rightmost index, the measurement of Alice's two qubits selects one of the four pairs of values divided off by the bars at the right. Each pair superposes to yield the value of Bob's qubit *after* the two measurements "collapse" Alice's part of the system. The final step is that Alice sends two *classical* bits across the lake to tell Bob what results she got, that is, which quadrant was selected by nature. The rest is in some sense the inverse of Alice's step in the superdense coding: Bob uses the two bits to select one of the Pauli operations $\boldsymbol{I}, \boldsymbol{X}, \boldsymbol{Z}, i\boldsymbol{Y}$, respectively, and applies it to his qubit $\boldsymbol{c}'$ to restore it to Alice's original value $\boldsymbol{c}$.

### Deutsch-Jozsa Extension (Ch. 9, probably Tue. 10/8)

Getting back to Deutsch's Problem, Richard Jozsa added that if you only care about distinguishing ***constant*** functions $f : \{0,1\}^n \rightarrow \{0,1\}$ from ***balanced*** ones, then you can make the classical algorithms require $2^{n-1} + 1$ queries, while the quantum ones can still do it on one query to a completely separable superposed state. This is a conditional problem, called a **promise problem**, in that it only applies when $f$ is in one of those two cases. If $f$ is neither balanced nor constant, then "all bets are off"---any answer is fine, even $|\ \text{¯\_(ツ)\_/¯}\ \rangle$.

The maze diagrams would get exponentially big, but we can track the computations via linear algebra. It is like Deutsch's setup except with $\mathbf{H}^{\otimes n}$ in place of the first $\mathbf{H}$, input $|0^n 1\rangle$ in place of $|01\rangle$, and targets (ignoring the $\sqrt{2}$ normalizers):

- constant $\mapsto |0^n\rangle\big(|0\rangle + |1\rangle\big)$ (instead of $\big(|00\rangle + |01\rangle\big)$, so that $0^n$ is certainly measured.
- balanced $\mapsto |?\,\rangle$ (instead of $\big(|10\rangle + |11\rangle\big)$, such that $0^n$ is certainly *not* measured.

The key observation is that for any $f$, any argument $x \in \{0,1\}^n$, and $b \in \{0,1\}$, the amplitude in the component $xb$ of the final quantum state $\phi$ is

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{t \in \{0,1\}^n} (-1)^{x \bullet t} (-1)^{f(t) \oplus b}.$$

Here $x \bullet t$ means taking the dot-products $x_i \cdot t_i$ (which is the same as $x_i \wedge t_i$) and adding them up modulo $2$ (which is the same as XOR-ing them). Well, when $x = 0^n$ this is always just zero, so the first term is $(-1)^0$ and just drops out, leaving

$$\phi\big(0^n b\big) = \frac{1}{\sqrt{2^{n+1}}} (-1)^b \sum_{t \in \{0,1\}^n} (-1)^{f(t)}.$$

Note that the $(-1)^b$ term is independent of the sum over $t$, so it comes out of the sum---and this is why we get two equal possibilities in the original Deutsch's algorithm as well. Ths final point is that:

- When $f$ is ***constant***, these terms are all the same, so they ***amplify***---giving $\frac{1}{\sqrt{2}}$ for the constant-false function and $\frac{-1}{\sqrt{2}}$ for constant-true. Both of these amplitudes square to $\frac{1}{2}$ and so together soak up all the output probability, so that $0^n$ is measured with certainty.
- When $f$ is ***balanced***, the big sum has an equal number of $+1$ and $-1$ terms, so they all ***interfere*** and *cancel*. Hence $0^n$ will certainly not be measured.

Added: A *randomized* classical algorithm can efficiently tell with high probability whether $f$ is constant by querying some random strings. If it ever gets different answers $f(y) \neq f(y')$ then definitely $f$ is not constant. (So, under the condition of the "promised problem," it must be balanced.) If it always gets the same answer, then since any balanced function gives 50-50 probability on random strings, it can quickly figure that $f$ is constant. But it is still the case that a deterministic algorithm needs exponentially many queries and hence exponential time.