Lecture 15: Thurs March 9

Until recently, the Bell Inequality was taught exclusively for being historically important, without having any practical applications. Sure, it establishes that you can't get away with a local hidden variable theory, but practically speaking, no one *actually* wants to play the CHSH game. In the last 10 years, however, it's found applications in…

Generating Guaranteed Random Numbers

This is one of the most basic important tasks in computing (or at least in cryptography), and you might think the solution is trivial. After all, you can get a random bit by measuring $|+\rangle$ in the $|0\rangle,|1\rangle$ basis. Easy, right? But this solution isn't good enough for cryptography. Cryptographers are paranoid people, and they want the ability to maintain security, even if the hardware they're on was designed by their worst enemy.

These sorts of assumptions aren't just academic speculation, especially since Snowden. For example, NIST (the National Institute of Standards and Technology) put out a standard for pseudo-random number generation based on elliptic curves to be used for encryption a while back. This standard was later discovered to have a backdoor created by the NSA that would allow them to characterize the output numbers, thus being able to break systems encrypted under this standard.

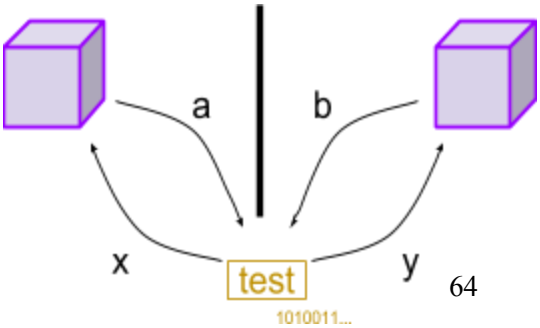
Thus cryptographers want to base their random number generation on the most minimal set of assumptions possible. They want systems that are guaranteed to be truly random, and to be sure that no one had added predictability to the number generation through some sort of backdoor.

You might think that, logically, one can never prove that numbers are truly random, and that the best one can say is that "I can't find any patterns here." After all, you can't prove a negative, and if not the NSA, who's to say that God himself didn't insert a pseudo-random function the workings of quantum mechanics?

Though presumably, if God wanted to read our emails he could do it some other way.

Interestingly, the Bell Inequality lets you certify that numbers are truly random under very weak assumptions, which basically boil down to "No faster-than-light travel is possible." Here's how:

You have two boxes that share quantum entanglement, which presumably were designed by your worst enemy. We'll assume they can't send signals back and forth (say you put them in Faraday Cages).
A referee sends them numbers.
They each return numbers.
If the returned numbers pass a test, we can say that they are truly random.



Vazirani calls this Einstein-Certified Randomness.

The usual way to present the CHSH game is that Alice and Bob prove that they share entanglement, and thus the universe is quantum mechanical. However, winning the game (better than 75% of the time) also establishes that *a* and *b* have some randomness, that there was some amount of entropy generated.

If *a* and *b* were deterministic functions—which is to say that they could be written as $a(x, r)$ and $b(y, r)$, in terms of their input and pure randomness—then you'd have a local hidden variable theory. If *x* and *y* were random, then there must exist some randomness in the outputs.

To put it another way: If Alice has a non-deterministic outcome *and* Alice's state isn't affected by Bob's, then some randomness must be in play.

What is the random result from Alice and Bob? What do you get out?

You can just take the stream of all *b*'s. The measure of entropy is just the Shannon Entropy.

$\{p_x\}_x$ if string x occurs with probability p_x

The total is $\Sigma\, p_x \log_2 1/p_x$

But each output *b* doesn't represent an entire bit of randomness. You'd take these bits and run them through a *randomness extractor* which would crunch them down from many sort-of-random bits to fewer very random bits.

David Zuckerman (here at UT) is an expert on this.

There's a problem here:

We need *x* and *y* to be random (CHSH assumes it), which means we're putting in two random bits and getting out less than one. The entropy we put in is greater than the entropy we get out.

In a 2006 paper, Roger Colbeck addresses this by saying that you don't have to give Alice and Bob randomness every time the game is run. You can just input $x = y = 0$ most of the time, and occasionally stick some purely random *x*'s and *y*'s in to prevent Alice and Bob from using hidden variables. If in the test cases Alice and Bob win with classical probability, then discard the results.

So how much entropy needs to be put in?

There was a race to solve this question, first with upper bounds like $O(\frac{c*n}{\sqrt{n}}$ for some $c< 1)$ and $O(\log^2 n)$ proposed. Eventually someone asked, "Why not just use a constant amount of randomness to jumpstart the randomness generation, and then feed the randomness outputted by Alice and Bob back in as an input?"

It turns out that this doesn't work because randomness generated by Alice and Bob can be exploited by them if you feed it back to them as input, making further outputs not random.

Remember: We're working under the assumption that Alice and Bob were designed by our worst enemy!

What you can do instead, if you don't have a limit on the number of devices used, is to feed Alice and Bob's output to two other machines Charlie and David. Using this technique it's possible to generate randomness with a constant amount of seed randomness and only four machines.

You're essentially using the extra devices as "random laundering machines".

Coudron and Yuen did a student project to figure out the number of seed bits necessary, and were able to establish an upper bound of ~200,000 random bits. That's likely still far from the truth: it may be possible with as few as 50.

It's a pretty amazing conceptual fact that playing the CHSH game can create certified randomness, and it's worth mentioning that you can currently go out and buy a quantum RNG from the internet. So you may ask…

What else could you certify about the boxes playing the CHSH game?

It turns out: an *enormous* amount of things.

You can certify that Alice and Bob did a specific sequence of local quantum transformations (up to a change in bases). So just by making them play the CHSH game, you can guarantee they do *any unitary transformation* of your choice. Reichardt and Vazirani describe this as a "classical leash for a quantum system."

One of the main current ideas for how a classical skeptic could verify a quantum solution also appears here. For prime factoring, we can easily verify the solution of a quantum algorithm, but this isn't the case for all problems. Sometimes the only way to verify the solution to a quantum algorithm is by testing the solution on a quantum computer. With this application of a CHSH game, you can guarantee that the quantum computer is behaving as expected.

This brings us nicely to…

Quantum Computation

Having seen all these protocols, we're finally ready to address the holy grail of the field: a programmable quantum computer that can do any short series of operations.

Quantum computation has two distinct intellectual origins:

One comes from Deutsch, who was thinking about experimentally testing Many Worlds (of which he was a firm believer) during his time as a postdoc here at UT. He imagined creating an equal superposition of a brain in configurations where it measured a qubit as $|0\rangle$, and where it measured it as $|1\rangle$. If you measured several times, and always got out the $|1\rangle$ possibility, then we'd have to discard the Copenhagen Interpretation.

But how could we test this? Step 1 would have to be to take a complete description of a human in quantum mechanical terms, and upload it to a computer.

You could presumably instead make an AI that's able to perceive the qubit, but what would a computer made of quantum mechanics even look like?

The other, less crazy, path to the same association came from Feynman, who gave a famous lecture in 1982 concerned with the question, "how do you simulate quantum mechanics on a classical computer?"

Chemists and Physicists had known for decades that this is hard, because the number of things you need to keep track of increases exponentially with the number of particles. This is the case because, as we know, an n-qubit state can be maximally entangled.

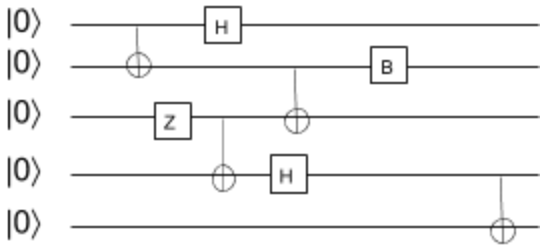
$(\alpha_{00\ldots.0})$

The state $|\Psi\rangle = \sum_{x \in \{0,1\}^n} \alpha x_n |x\rangle$ must be described by the vector $(\alpha_{00\ldots.1})$ of length 2^n

:

Even to solve for the energy of the system, or for the state of some particular qubit, there's no shortcut for reasoning with this enormous vector. So he raised the question, "Why don't we build computers *out of qubits* to simulate qubits?" No one knew if this would be useful for classical tasks as well.

We already have all the tools we need to discuss quantum computers.



The basic picture of a quantum computer is that it's just a quantum circuit, but we're jumping from working with 1, 2, or 3 qubits at a time to *n* (where *n* could equal a million). You apply a sequence of gates on these qubits, each gate acting on a few qubits, then measure some or all of them.

We have a few conceptual points to address:

1. Can we solve anything on a quantum computer that can't be solved on a classical computer?

   No. Anything that can be done on a quantum computer can be done on a classical computer too by storing the exponential number of variables that arise when working with qubits.

   Quantum computing "only" may violate the Extended Church-Turing Thesis.

2. Why does each gate act only on a few qubits? Where is this assumption coming from?

   It's similar to how classical computers don't have gates act on arbitrarily large quantities of bits, and instead use small gates like AND, NOT to build up complex circuitry.

   For the quantum case, you could imagine a giant unitary U, which takes qubits, encodes on them the decision version of Travelling Salesman, which is then cNOT'ed to another qubit to get an answer. But given such a definition, how would you go about building U?

   Difficulty arises because there exists a staggeringly large amount of possible unitary matrices. You can decompose any U, but it might result in an exponential number of small gates (just like deconstructing an arbitrary Boolean string may require an exponential number of classical gates). We can sort of circumvent this with the…

Accounting Argument

which says that we don't need to consider all unitary matrices, just all of the diagonal ones where the diagonal entries are either 1 or -1. And that's great, because it means you don't need to keep track of 2^(2^n) variables to keep track of U.

Shannon proved that the number of bits it takes to describe a circuit is roughly linear to the number of gates. So almost every unitary matrix would take exponential gates to build.

Interestingly enough, we don't know any examples of such unitary matrices.
But we do know that they're out there!

This tells us something important. In quantum computing, we're not interested in all unitary matrices, only the ones that can be encoded in small circuits requiring a polynomial number of gates.