Introductory Notes on Quantum Information and Computation

Aidan Chatwin-Davies[*]

Institute for Theoretical Physics, KU Leuven

Nov 20, 2019

Abstract

This set of notes constitutes roughly two introductory lectures on quantum information and quantum computation. I have mostly drawn on John Preskill's lecture notes [1], as well as my own experiences studying quantum information as a student. The first lecture consists of a broad overview of quantum information science, as well as an introduction to quantum algorithms. The second lecture covers classical and quantum computational complexity classes, culminating with a discussion of Simon's problem and a proof that $BPP^\theta \neq BQP^\theta$.

1 Overview of quantum information science

Quantum information science, or QIS for short, is the intersection of three disciplines: quantum mechanics, information theory, and computer science. As with so many disciplines which bear the prefix "quantum," QIS has its roots in a classical theory. Classical information science concerns itself with the information stored in, and the manipulation of bit strings. In other words, the fundamental entity is the bit,

$$\text{one bit:} \;\; x \in \{0, 1\}, \tag{1}$$

from which strings are formed and used to encode information,

$$n \text{ bits:} \;\; 0110100 \cdots 0. \tag{2}$$

Computation is essentially the processing of bit strings as a means to process the information that they encode.

Analogously, QIS is concerned with the information stored in, and the manipulation of quantum states. While in principle we can compute with states in any Hilbert space (in the same way that we are free to use any object which encodes classical information to compute), as a matter of convenience, we take the qubit as our basic entity,

$$\begin{aligned} \text{one qubit:} \;\; &|\psi\rangle \in \mathcal{H}_{\text{qubit}} \cong \mathbb{C}^2 \\ &|\psi\rangle \in \text{span}\{|0\rangle, |1\rangle\}, \end{aligned} \tag{3}$$

from which strings of qubits are formed,

$$\begin{aligned} n \text{ qubits:} \;\; &|\psi\rangle \in \mathcal{H} = (\mathcal{H}_{\text{qubit}})^{\otimes n} \cong \mathbb{C}^{2^n} \\ &|\psi\rangle \in \text{span}\{|x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle \mid x_i \in \{0, 1\}\}. \end{aligned} \tag{4}$$

[*]aidan.chatwindavies@kuleuven.be

Written out in the computational basis, the most general pure state of n qubits is therefore

$$|\psi\rangle = \sum_{x_1=0,1}\sum_{x_2=0,1}\cdots\sum_{x_n=0,1} c_{x_1x_2\cdots x_n}|x_1x_2\cdots x_n\rangle. \tag{5}$$

Note that I will often omit the tensor product symbol and also concatenate strings of bits within a single ket, as written above. I will also often abbreviate a n-bit string by just the symbol x, i.e.

$$x \equiv x_1x_2\cdots x_n, \tag{6}$$

and hence also write n-qubit states as

$$|\psi\rangle = \sum_{x=0}^{2^n-1} c_x|x\rangle. \tag{7}$$

QIS is thus the study of the computational capabilities afforded to us by quantum systems.

Quantum information is quite different from classical information. If we are capable of accurately manipulating a quantum system, we can leverage these differences to do interesting things that a classical information processing system cannot do efficiently. The goal of these two short lectures is to illustrate this claim.

1.1 A bit of historical background

People started to get interested in quantum computing in the 1990's for a variety of reasons. Some of these reasons include:

- People realized that we were eventually going to reach the limit of classical architectures (i.e., due to Moore's law).
- It was becoming apparent that simulating quantum systems is extraordinarily difficult.
- People were ever searching for better cryptographic systems (and quantum computing changed the game drastically).

All of these motivations helped to spur the development of quantum computers; however, the singular finding which is perhaps the most responsible for thrusting QIS into the mainstream was Peter Shor's discovery in 1994 of a quantum algorithm for factoring a number that is a product of two primes. That is, given a number N, which you are promised satisfies

$$N = p\cdot q, \qquad p, q \text{ prime}, \tag{8}$$

the problem is to find the prime factors p and q. While it is not proven that this is a hard problem, it certainly seems to be a hard problem—so much that it constitutes the foundation of the RSA crypto-system. We will be more careful about the meaning of the word "hard" later, but for now, let's take hardness to mean "the amount of time a computer takes to solve the problem." For N which has on the order of 190 digits, a typical supercomputer circa 2014 could find p and q on the order of 30 years. For 500 digits, this time increases to 10^{12} years.

Shor's algorithm, however, has much better scaling. If the 193-digit problem runs in 1 second, then the 500-digit problem would run on the order of a few 10's of seconds. So, not only do you solve the problem faster with a quantum computer, you solve the problem *exponentially* faster... and you break RSA encryption along the way. This was enough evidence to convince people that quantum computers might be capable of some pretty interesting things.

1.2 Quantum information

Let us begin our investigations with the following question: how is quantum information different from classical information? Well...

- *Randomness* — While we can engineer pseudo-randomness in classical computers to simulate randomized computation, quantum systems are inherently and inexorably random.
- *Uncertainty* — Many quantum observables fail to commute. This ultimately results in a trade-off between information retrieval and disturbance of stored information.
- *No cloning* — Classical information can be copied to the extent that the laws of thermodynamics allow (so, essentially totally freely). The operation $|\psi\rangle|0\rangle \to |\psi\rangle|\psi\rangle$, however, if generally impossible.
- *Entanglement* — Quantum systems can store information nonlocally. An analogy is as follows: if classical, local information is the content of the pages in a book, nonlocal information would be information stored in correlations among the pages. In particular, you need all of the pages in order to access the nonlocal information. What's more, this quantum book's pages disappear after you read them, so if you read a single page, you generally end up ruining the nonlocal information!
- *Parallelism* — A common platitude is that "quantum computers are so powerful because they have exponentially many states." While this is true—the dimension of the Hilbert space of n qubits is 2^n—it is also true that one can form 2^n different strings out of n bits. Rather, what makes quantum computers special is that their computational states can be superpositions. This makes possible a phenomenon known as quantum parallelism, which will feature prominently in these two lectures.

1.3 Quantum computation

The essential elements of a quantum computation are

1. A state space — e.g., n qubits
2. Initialization — the starting state, usually $|00\cdots 0\rangle$
3. Computation — a unitary transformation to $|\psi\rangle = U|00\cdots 0\rangle$
4. Readout — measure; the n-bit string x occurs with probability $Pr(x) = |\langle x|\psi\rangle|^2$

Hopefully, you get the correct answer, x, to an interesting question with high probability!

State space - implementations

It's fine and all to talk about abstract qubits, but how does one actually implement a qubit in a laboratory? Implementations and building quantum computers is a whole field of QIS, and while we won't spend much time discussing the subject, I'll just give a couple of examples here.

One technology is trapped ion quantum computers. Here, a collection of n charged particles are electromagnetically confined and are used to implement qubits by identifying $|0\rangle$ with the ground state, $|g\rangle$, and $|1\rangle$ with some long-lived excited state, $|e\rangle$. The qubits are thus electromagnetic atomic systems. The state of the trapped ions can be manipulated by laser pulses. For example, readout can be achieved by shining a laser that is resonant with a transition between

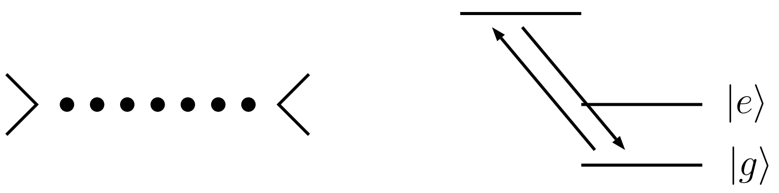

Figure 1: Trapped ions as qubits. Checking for fluorescence by driving a transition from g to a state besides e is a way to read out the state.

$|g\rangle$ and some other excited state besides $|e\rangle$. If the atom fluoresces, then it was in the state $|g\rangle$; otherwise, it was in the state $|e\rangle$.

Trapped ions are good test beds for quantum computing because they offer good control over the qubits and are a relatively clean system. However, they are unsuitable for large-scale computing because they are short lived, and difficult to scale up in number. Other examples of quantum architectures include:

- cavity QED systems
- superconducting flux qubits (e.g. Google's machine)
- NMR (nuclear magnetic resonance) systems (still a young technology)

Computation

Universal gate sets

So, assuming that you have a collection of qubits, which you can initialize into a starting state $|00\cdots0\rangle$, the next thing you want to do is manipulate them to carry out some interesting operation. In other words, you want to be able to apply some unitary, U, to the qubits, to bring them into some final state $U|00\cdots0\rangle$.

Now is as good a time as any to introduce circuit notation. This is just a pictorial way to represent operations on a collection of qubits. For example, the operation $U|00\cdots0\rangle$ is denoted by the following diagram:

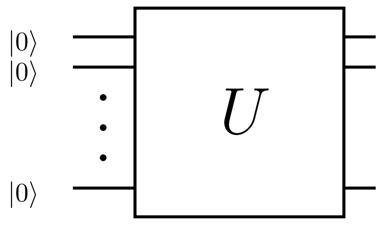

Figure 2: A quantum circuit, read left to right.

Each line represents one qubit, and a square represents a unitary operator which acts on the qubits whose lines feed into the square. Altogether, such a diagram is called a *circuit.*

Of course, it's not practical to expect a quantum computer to be able to carry out arbitrary unitaries by acting on all qubits at once, in the same way that a classical computer is not build to be able to carry out every n-bit function in its hardware in a single shot. Instead, functions of n bits are build up out of simpler logical operations, like *AND*, *OR*, *NOT*, *XOR*, and so on. Similarly, we build up complicated and interesting unitaries through circuits consisting of simpler unitary operations acting on smaller numbers of qubits, which we often call *gates.*

For example, some simple gates are the Pauli X gate, the Pauli Z gate, or the controlled not (CNOT) gate.

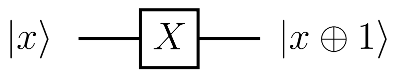

Figure 3: Pauli X operator.

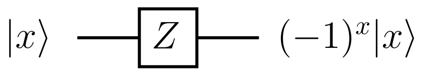

Figure 4: Pauli Z operator

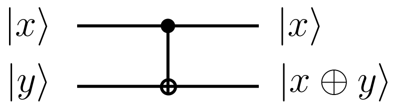

Figure 5: Controlled not, or CNOT operator.

If a collection of gates is sufficient to implement *any* possible unitary operation, then we say that it is a *universal gate set*. Stated a bit more carefully, its definition is the following:

Definition 1.1 *Let $\mathcal{H}$ be a finite-dimensional Hilbert space with $\dim \mathcal{H} = n$. A collection of unitary operators $\{U_1, U_2, \ldots, U_m\}$ is a* universal gate set *if, for every unitary operator U acting on $\mathcal{H}$ and for every $\epsilon > 0$, there exists a finite sequence $U_{i_1}, U_{i_2}, \ldots, U_{i_k}$ such that*

$$\left\| U_{i_k} \cdots U_{i_2} U_{i_1} |0\rangle - U|0\rangle \right\|_1 < \epsilon \tag{9}$$

It turns out that almost any single-qubit gate, together with a single 2-qubit entangling gate, is a universal gate set. This fact is quite difficult to prove, but in the exercises, you will be pointed toward a pretty tractable proof that a single-qubit gate called the Hadamard gate, together with a 2-qubit gate called a controlled phase gate, together constitute a universal gate set.

Error Correction

So, supposing you have some physical system which implements a collection of qubits, which you can prepare in a simple initial state, and you possess the capability to manipulate your qubits and read out their state. One final obstruction to performing quantum computations is the fact that the real world is not mathematically perfect nor ideal. Every operation that you perform will introduce some amount of error into your computation. Furthermore, qubits are physical systems which interact with their surrounding environment. These uncontrolled and unmonitored interactions lead to decoherence of your qubits, which, if your goal is to carry out some coherent manipulation of a pure quantum state, is a huge obstruction to computation.

All of these considerations thus necessitate *quantum error correction*. Schematically, this is done by encoding some fixed, logical state in a much larger Hilbert space, in a way that protects it from environmental interactions and that allows us to detect and correct any errors. The task is surprisingly subtle, however, because you cannot just *measure* the intermediate state of a computation to see if any error occurred. Recall the interplay between information and disturbance—such checks will generally ruin your computation! As such, quantum error

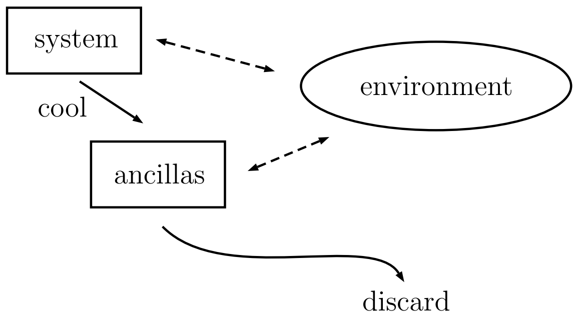

Figure 6: Quantum error correction, in a schematic nutshell.

correction consists of keeping track of errors and correcting them, all the while protecting the encoded information from the environment... and from ourselves! We won't go into any more details about error correction here, but there are several great introductory resources that you can consult if you want to delve into quantum error correction, such as Preskill's notes [1] or Gottesman's notes [2] to name a couple.

1.4 Deutsch's Problem

We have spent a lot of time talking about quantum computing, how it differs from classical computing, and some challenges (both practical and conceptual) surrounding quantum computations, and so let's get our hands dirty and study a simple quantum algorithm, which solves Deutsch's problem.

The problem is the following. Suppose that $f : \{0, 1\} \to \{0, 1\}$ is some function that is unknown to us, but we are allowed to query it. In other words, we possess some black box which can tell us the values of $f(0)$ and $f(1)$. The question we must answer is the following: Is f constant ($f(0) = f(1)$) or balanced ($f(0) \neq f(1)$)?

Classically, we can of course answer the question with two queries of the black box. We just ask for the values $f(0)$ and $f(1)$, and then we compare them. On the other hand, we cannot do any better than that. We need at least two queries of the black box; otherwise, we can only make a guess, with a 50% chance of being correct.

Quantum-ly, however, we can solve the problem with a single query. In this case, suppose that we have two qubits, and that now, our black box is a unitary operation U_f which implements

$$U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle. \tag{10}$$

In the expression above, x and y can take the values 0 or 1, and the symbol $\oplus$ denotes modular addition, i.e.

$$\begin{aligned} &0 \oplus 0 = 0 \\ &0 \oplus 1 = 1 \oplus 0 = 1 \\ &1 \oplus 1 = 0. \end{aligned} \tag{11}$$

My claim is that the following circuit lets you solve the problem with a single query of U_f: Note that H denotes the Hadamard gate, which acts on a single qubit in the computational basis as

$$H \equiv \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{12}$$

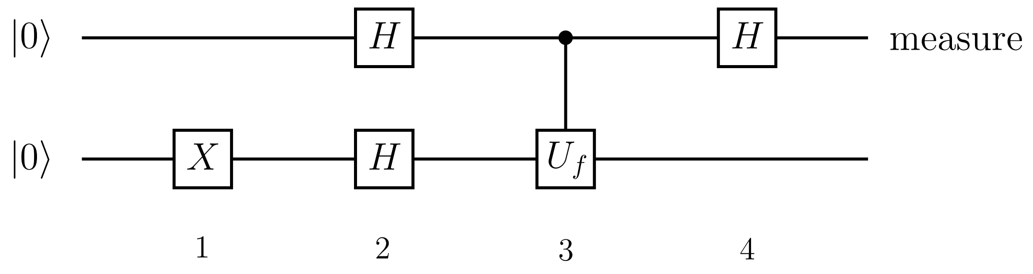

Figure 7: A quantum circuit for solving Deutsch's problem.

Or, in other words,

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \equiv |+\rangle \tag{13}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \equiv |-\rangle. \tag{14}$$

X is of course the Pauli X operator, $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$.

So, what does the circuit do? Let's step through it:

$$|0\rangle|0\rangle \xrightarrow{1} |0\rangle|1\rangle \tag{15}$$

$$\xrightarrow{2} |+\rangle|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |-\rangle \tag{16}$$

But now, notice that the action of U_f on a state $|x\rangle|-\rangle$ is the following:

$$U_f|x\rangle|-\rangle = \frac{1}{\sqrt{2}}(U_f|x\rangle|0\rangle - U_f|x\rangle|1\rangle) \tag{17}$$

$$= \frac{1}{\sqrt{2}}(|x\rangle|f(x)\rangle - |x\rangle|f(x) \oplus 1\rangle) \tag{18}$$

$$= \begin{cases} \frac{1}{\sqrt{2}}|x\rangle(|0\rangle - |1\rangle) & \text{if } f(x) = 0 \\ \frac{1}{\sqrt{2}}|x\rangle(|1\rangle - |0\rangle) & \text{if } f(x) = 1 \end{cases} \tag{19}$$

$$= (-1)^{f(x)}|x\rangle|-\rangle \tag{20}$$

Therefore, the next step of the algorithm is

$$|+\rangle|-\rangle \xrightarrow{3} \frac{1}{\sqrt{2}}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right)|-\rangle \tag{21}$$

$$\xrightarrow{4} \frac{1}{\sqrt{2}}\left((-1)^{f(0)}\frac{1}{\sqrt{2}}[|0\rangle + |1\rangle] + (-1)^{f(1)}\frac{1}{\sqrt{2}}[|0\rangle - |1\rangle]\right)|-\rangle \tag{22}$$

$$= \frac{1}{2}\left(\left[(-1)^{f(0)} + (-1)^{f(1)}\right]|0\rangle + \left[(-1)^{f(0)} - (-1)^{f(1)}\right]|1\rangle\right)|-\rangle \tag{23}$$

So, when we measure the first qubit, we find $|0\rangle$ with probability 1 if $f(0) = f(1)$, but we find $|1\rangle$ with probability 1 if $f(0) \neq f(1)$. With a single query of U_f and a single measurement, we can therefore determine whether f is constant or balanced. Pretty neat!

2 Classical and quantum algorithmic complexity

Having had an overview of quantum information science, let's now dig a bit deeper into quantum algorithms. In particular, we'll try to argue that a quantum computer can, at least in principle, efficiently perform a task that a classical computer cannot perform efficiently.

How *can* you tell if a quantum computer can outperform a classical computer? This is a surprisingly subtle question, and ultimately, the answer depends on how you define "outperform." Do you just measure the amount of time it takes a classical computer to solve a problem and then compare it to the amount of time taken by a quantum computer by calling some `clock()` function? Do you or do you not take into account how much space it has taken up in memory? Does the answer depend on which machine you use? Does it count if you cannot actually do anything useful with your quantum computer? For example, while factoring is an interesting problem, at least in the near term, quantum computers will be totally incapable of factoring numbers as large as those used in RSA encryption schemes.

The challenge of demonstrating that a quantum computer outperformed any classical machine is known as demonstrating "quantum supremacy," and the race is on to be the first to achieve this. (Although, Google recently claims to have demonstrated quantum supremacy, a claim which will be assessed in the coming weeks and months.)

Here, we will assess the question by framing it in terms of *algorithmic complexity.* In other words, we will characterize how difficult it is to solve a problem of input size n by how many fundamental gate operations it takes, asymptotically and as a function of n, to solve the problem. First we will discuss classical complexity classes, and then we will move on to quantum complexity classes.

2.1 Classical algorithmic complexity

The most general binary function is a deterministic function f from n to m bits,

$$f : \{0,1\}^n \longrightarrow \{0,1\}^m. \tag{24}$$

All computation can be recast in terms of Boolean functions, however, so we need only concern ourselves with the case $m = 1$:

$$f : \{0,1\}^n \longrightarrow \{0,1\} \tag{25}$$

There are 2^{2^n} such functions, so that is still quite a lot of information!

Definition 2.1 *Given a Boolean function f, we will say that f "accepts" an input x if $f(x) = 1$, and f "rejects" an input x if $f(x) = 0$.*

Definition 2.2 *A* language*, L, is a set of bit strings of variable length, $x \in \{0,1\}^*$, which are accepted by a family of functions f^* taking inputs of varying length,*

$$L = \{x \in \{0,1\}^* \mid f^*(x) = 1\}. \tag{26}$$

The question of computation can be reduced to the question of "finding the language" for a given family of functions. For most problems, a naïve determination of the language takes an extraordinary amount of time. Rather, a better way to think of computation is to ask whether there's a better, interesting way which reflects the structure of f, i.e. a *circuit*, which lets you evaluate f more efficiently.

Let us first make a brief aside and note that it's convenient to reformulate all problems as *decision problems*, so that they are implemented by Boolean functions.

Example 2.3 Factoring can be recast as a decision problem: `FACTOR:` "Does x have a divisor less that z and greater than 1?"

$$f(x,z) = \begin{cases} 1 & \text{if } \exists\, y : 1 < y < z,\ y|x \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

In this way, you can "factor" a number by starting with $z = 2$ and incrementing z until you either find a value z_* such that $f(z_*) = 1$ or you reach $z = \sqrt{x}$. Another classic problem in computer science is the Hamiltonian path problem:

Example 2.4 The Hamiltonian path problem: `HAMPATH`: "Given a graph $G = \{v_i, e_a\}$, is there a path that visits each edge e_a exactly once?"

We will quantify "hardness" via the scaling of the resources needed to solve a problem as a function of the input size. For example, for factoring above, the input size is $n = \log_2 z + \log_2 x$ (the number of bits used to represent x and z).

Given a family of functions $\{f_n\}$, where f_n takes input of size n, suppose that the corresponding family of circuits $\{C_n\}$ computes the f_n. Computationally "easy" problems are those where the size of the circuits (measured in number of steps, or fundamental gate applications) grows no faster than polynomially in n. This is the definition of the first complexity class that we will encounter.

Definition 2.5 *The complexity class* P *(polynomial time) is the set of languages that can be solve by uniform circuits that grow at most polynomially with the input size. In other words,*

$$\mathrm{P} = \{\text{decision problems solved by poly}(n)\text{ uniform circuit families}\} \tag{28}$$

Although we won't define it carefully, "uniform" basically means that it's easy to go from a circuit accepting input of size n to a circuit that accepts input of size $n + 1$. (More precisely, we mean that the *design* of the circuit itself can be solved in a polynomial amount of time.)

Colloquially speaking, these problems are "easy" and can be solved "quickly." However, in practice, they can still scale pretty poorly. For example, finding Ryu-Takayanagi surfaces for n boundary subregions in holographic 3D gravity scales like $O(n^{12})$. So, it is still an "easy" problem, in the sense that it's in P, but n^{12} is pretty bad scaling.

By contrast, the class NP is what's generally considered to be "hard" problems.

Definition 2.6 *The complexity class* NP *(nondeterministic polynomial) is the set of languages L for which there exists a polynomial-size family of verifier circuits, $V(x, y)$, such that*

- *If $x \in L$, then $\exists\ y$ such that $V(x, y) = 1$ (completeness)*
- *If $x \notin L$, then $V(x, y) = 0\ \forall\ y$ (soundness)*

and where the size of the second input to the verifier is no more than polynomially larger than the first input, i.e.

$$|y| \leq \text{poly}(|x|). \tag{29}$$

Colloquially speaking, NP is the set of functions that are hard to compute, but easy to check.

Example 2.7 `FACTOR` is in NP. Given a an input (x, z), the witness is

$$V(x, z; y) = (y|x) \wedge (y < z) \tag{30}$$

(Note that "$\wedge$" means "`AND`".)

Example 2.8 `CIRCUIT-SAT` is in NP: Given a polynomial-size Boolean circuit C, does there exist an input accepted by C? In this case, the verifier is basically C itself!

$$V(C; y) = C(y) = \begin{cases} 1 & C\text{ accepts } y \\ 0 & C\text{ rejects } y \end{cases} \tag{31}$$

It's easy to see that P $\subseteq$ NP, and it is widely believed that P $\neq$ NP, but this has not been proven.

As another aside, while it's not directly relevant to what we are going to do, it's useful to know about the concept of NP completeness.

Definition 2.9 *A problem B efficiently reduces to A if any machine that solves A can be used to solve B with at most polynomial overhead, i.e.,*

$$B = A \circ R \tag{32}$$

for a polynomial-size circuit R.

Definition 2.10 *A problem A is* NP-complete *if $A \in NP$ and any problem in* NP *is efficiently reducible to A.*

For example, `CIRCUIT-SAT` is NP-complete:

Theorem 2.11 (Cook) `CIRCUIT-SAT` *is NP-complete.*

A sketch of the proof is as follows. Given some language $L \in$ NP, take its verifier V and construct the function

$$f(V(x,\cdot)) = \begin{cases} 1 & \text{if } \exists\, y \ \rightarrow\ V(x,y) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

Determining if $x \in L$ is the same as asking whether there's an input that f accepts. This is precisely `CIRCUIT-SAT`! `HAMPATH` is another NP-complete problem. `FACTOR`, however, is not.

There are many other complexity classes, but the one that we will be particularly interested is BPP.

Definition 2.12 *The complexity class* BPP *(bounded-error probabilistic polynomial time) is the set of languages L solved by uniform, randomized polynomial-sized circuits.*

The new key ingredient here is the randomization. In other words, we will be satisfied if we can find a random circuit (i.e., whose outputs are probabilistic) such that

$$\begin{aligned} x \in L \Rightarrow &\quad Pr(\text{accept}) \geq \tfrac{1}{2} + \delta \\ x \notin L \Rightarrow &\quad Pr(\text{accept}) \leq \tfrac{1}{2} - \delta \end{aligned} \tag{34}$$

where δ is a constant that is independent of input size. In other words while the circuit may not give the right answer all the time, if we repeat the computation enough times, then the probability of getting the wrong answer *on average* decreases to zero. This is made precise by the *Chernoff bound*:

$$Pr(\text{majority is wrong after } N \text{ trials}) \leq e^{-2N\delta^2} \tag{35}$$

The proof is left as an exercise (or alternatively some quick Google-fu).

2.2 Quantum algorithmic complexity

We went to the trouble of defining BPP because there is a clear quantum analogue:

Definition 2.13 *The complexity class* BQP *(bounded-error quantum polynomial time) is the set of languages L solved by uniform* quantum *polynomial-size circuits.*

The Giga-€ Question:

$$\boxed{\mathrm{BPP} \overset{?}{\neq} \mathrm{BQP}}$$

This is the question which basically drives the quantum computing industry (or rather, the hope that it is indeed an inequality). If BPP $\neq$ BQP, then quantum computers can efficiently perform certain algorithms that a classical randomized computer cannot, from the perspective of algorithmic complexity.

Unfortunately, this question is probably pretty difficult to answer. Some circumstantial evidence which supports this claim are related implications for other complexity classes. For example:

- BPP $\neq$ BQP would imply that P $\neq$ PSPACE.
- P $\neq$ NP would imply BPP $\neq$ BQP

So, what can we do? One strategy is to come up with examples of algorithms, like Shor's factoring algorithm, that we think solve difficult problems. Although we cannot prove that `FACTOR` is not in P or BPP, it seems sufficiently difficult to inspire *some* confidence that a quantum machine might be useful. We can also construct algorithms which exhibit non-exponential speed-ups, but are much more versatile, such as Grover's algorithm. Grover's algorithm implements an unstructured database search, but achieves it in a time $T_{\text{quantum}} \sim (T_{\text{classical}})^{1/2}$. We won't talk about Grover's algorithm here, but references that discuss it are accessible and widely available.

The strategy that we will pursue here is to consider a different but simpler problem. Namely, we will consider *relativized* computation, where, similarly to Deutsch's problem, we will suppose that we are provided with a black box, or oracle. The oracle computes a function f which is unknown to us, and we assume that we incur no computational cost by calling the oracle. The problem is then to determine f, and we characterize the problem's difficulty by the *query complexity*: How many times must we query the oracle in order to learn f? If f takes inputs consisting of n bits, then of course the largest possible query complexity is 2^n—we just try every input. However, if a problem has an interesting structure, then the query complexity can be much lower.

Denote by $\mathrm{BPP}^{\mathcal{O}}$ the set of problems that can be solved by uniform, randomized polynomial-sized circuits with at most a polynomial number of queries to the oracle, and define $\mathrm{BQP}^{\mathcal{O}}$ analogously.

Let's show that $\mathrm{BPP}^{\mathcal{O}} \neq \mathrm{BQP}^{\mathcal{O}}$.

2.3 Simon's Problem

Let $f\{0,1\}^n \to \{0,1\}^{n-1}$ be some function that is unknown to us. However, we are promised that

- f is two-to-one, and
- if $f(x) = f(y)$ and $x \neq y$, then $x = y \oplus a$, where a is a fixed bit string for all x and y.

Simon's problem is to find a.

First, note that `SIMON` is not in $\mathrm{BPP}^{\mathcal{O}}$. In order to find a, you need to find a "collision;" that is, you need to find a pair (x, y) such that $f(x) = f(y)$. Then, $a = x \oplus y$. Suppose you query the oracle k times with k different inputs, giving you $\binom{k}{2}$ pairs of outputs. Then, the probability that you find a collision is

$$Pr(\text{success}) \leq \frac{\frac{1}{2}k(k-1)}{(2^n - 1)} < \frac{k^2}{2^n}. \tag{36}$$

Therefore, on average, you need to query the oracle $k \sim \exp(n)$ times in order to have a reasonable chance of success. This problem is hard!

On the other hand, $O(n)$ uses of the following quantum circuit lets you solve the problem:

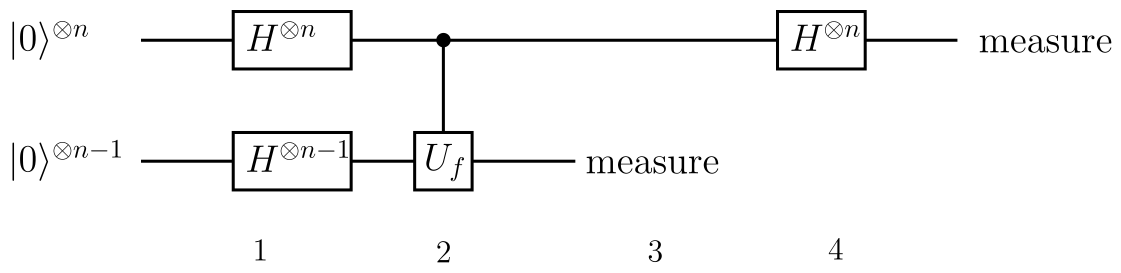

Figure 8: Circuit that solves Simon's problem.

As before, we suppose we have an oracle U_f, but this time it acts on $2n-1$ qubits as follows:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle \tag{37}$$

Let's analyze the circuit. The first n-fold Hadamard gate maps the first n qubits onto the uniform superposition of all n-bit strings,

$$|0\rangle^{\otimes n}|0\rangle^{\otimes n-1} \xrightarrow{1} \frac{1}{2^n}\left(\sum_{x=0}^{2^n-1}|x\rangle\right) \otimes |0\rangle^{\otimes n-1}, \tag{38}$$

which after acting with U_f gives us

$$\xrightarrow{2} \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle. \tag{39}$$

Now we measure the second register. Suppose we get the outcome $f(x_0)$. This means that the first register must get mapped to

$$\xrightarrow{3} \frac{1}{\sqrt{2}}\left(|x_0\rangle + |x_0 \oplus a\rangle\right) \tag{40}$$

In the following calculation, I will use the notation:

$$x \cdot y \equiv (x_1 \wedge y_1) \oplus (x_2 \wedge y_2) \oplus \cdots \oplus (x_n \wedge y_n) \tag{41}$$

We now act with a final $H^{\otimes n}$ on the first register.

$$\xrightarrow{4} \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2^n}} \sum_y (-1)^{x_0 \cdot y} |y\rangle + \frac{1}{\sqrt{2^n}} \sum_y (-1)^{(x_0 \oplus a) \cdot y} |y\rangle \right) \tag{42}$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_y \left[(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus a) \cdot y} \right] |y\rangle \tag{43}$$

$$= \frac{1}{\sqrt{2^{n-1}}} \sum_y (-1)^{x_0 \cdot y} \left[\frac{1 + (-1)^{a \cdot y}}{2} \right] |y\rangle \tag{44}$$

$$= \frac{1}{\sqrt{2^{n-1}}} \sum_{a \cdot y = 0} (-1)^{x_0 \cdot y} |y\rangle \tag{45}$$

Therefore, when we measure the first register, we are uniformly sampling the solution space of the binary linear system $a \cdot y = 0$. After running the algorithm $O(n)$ times, we obtain n linearly independent solutions y_k of $a \cdot y_k = 0$, which allows us to solve the binary linear system for a. We therefore see that `SIMON` is in $\mathrm{BQP}^{\mathcal{O}}$.

2.4 The Solovay-Kitaev Theorem

An important point that we glossed over earlier, but which bears mentioning, is the question of how algorithmic complexity depends on the *gates* that we are allowed to use. In particular, we would be in trouble if the algorithmic complexity of a problem changed drastically if we changed the gate set we use to count the number of "operations" that we perform to solve a problem.

Fortunately, this is not the case, and this is the essential content of the Solovay-Kitaev theorem. Stated very roughly, the Solovay-Kitaev says that:

> If a quantum computer A implements the universal gate set $\mathcal{G}_A$ and solve a given problem in time T, then a quantum computer B that implements a different universal gate set $\mathcal{G}_B$ can solve the same problem in time $O(T\mathrm{poly}(\log T))$.

In other words, up to a possible poly-log overhead, one universal quantum computer can simulate another universal quantum computer, and so polynomial-time algorithms at least are hardware-independent. BQP is a universal notion. (The same holds true for universal gate sets on classical computers; one way to address the issue in the classical case is to instead formulate everything in terms of Turing machines.)

It's also worth noting that we have totally glossed over the question of tolerance. In other words, with a quantum computer, the best we can do is hope to come within trace distance ϵ of some target output state. This typically adds an $O(1/\epsilon)$ scaling to algorithmic complexity, but we will not go into any further detail than that.

References

[1] J. Preskill. Quantum computation. http://www.theory.caltech.edu/people/preskill/ph229/

[2] D. Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. arXiv:0904.2557.