※ Complexity and Circuits

"Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."

- Richard P. Feynman

10.1 Classical Circuits and Complexity

Consider a classical function f that takes n input bits and returns m output bits. The signature of this function would be

$$f : \{0,1\}^n \to \{0,1\}^m. \tag{63}$$

You may have learned that a **universal gate set** can be used to compute any Boolean function in theory.

Question 87. What are some universal gate sets for classical circuits?

Question 88. The following set is NOT universal: $\{OR, AND\}$. Can you explain why?

In complexity theory, we are primary interested in what are called **decision problems**, where there is always just a single output bit. We can express the signature of decision problems as

$$f : \{0,1\}^* \to \{0,1\}. \tag{64}$$

A decision problem can also be defined as a **language** L, which just means some subset of binary strings.

$$L \subseteq \{0,1\}^* \quad f(x) = 1 \leftrightarrow x \in L. \tag{65}$$

Question 89. A natural computational problem is searching for the minimum value of a list. How could we capture this problem as a decision problem?

Let's look at some classical complexity classes to get a sense of how to classify functions.

Definition 10.1 (P (Deterministic Polynomial Time)). A language $L \subseteq \{0,1\}^*$ is in P if there exists a turing machine M and a fixed polynomial $r_L : \mathbb{N} \to \mathbb{R}^+$, such that for any input $x \in \{0,1\}^n$, M halts in at most $O(r_L(n))$ steps, and:

- (Completeness/YES case) If $x \in L$, M accepts.
- (Soundness/NO case) If $x \notin L$, M rejects.

Definition 10.2 (NP (Non-Deterministic Polynomial Time)). A language $L \subseteq \{0,1\}^*$ is in NP if there exists a turing machine M and fixed polynomials $p_L, r_L : \mathbb{N} \to \mathbb{R}^+$, such that for any input $x \in \{0,1\}^n$, M takes in a "proof" $y \in \{0,1\}^{p_L(n)}$, halts in at most $O(r_L(n))$ steps, and:

- (Completeness/YES case) If $x \in L$, then there exists a proof $y \in \{0,1\}^{p_L(n)}$ causing M to accept.
- (Soundness/NO case) If $x \notin L$, then for all proofs $y \in \{0,1\}^{p_L(n)}$, M rejects.

Definition 10.3 (co-NP (Non-Deterministic Polynomial Time)). A language $L \subseteq \{0,1\}^*$ is in NP if there exists a turing machine M and fixed polynomials $p_L, r_L : \mathbb{N} \to \mathbb{R}^+$, such that for any input $x \in \{0,1\}^n$, M takes in a "proof" $y \in \{0,1\}^{p_L(n)}$, halts in at most $O(r_L(n))$ steps, and:

- (Completeness/YES case) If $x \in L$, then for all proofs $y \in \{0,1\}^{p_L(n)}$, M rejects.
- (Soundness/NO case) If $x \notin L$, then there exists a proof $y \in \{0,1\}^{p_L(n)}$ causing M to accept.

Question 90. In words, what is the difference between P and NP?

Question 91. Consider the decision problem MULTIPLY: Given as input $x, y \in \mathbb{Z}$ and a threshold t, is the product $x, y \leq t$?

The reverse problem is FACTOR: Given $z \in \mathbb{Z}^{+}$ and threshold t, does z have a nontrivial factor $x \leq t$?

What class does MULTIPLY belong to? What about FACTOR? Why?

Question 92. Let L be the set of bit strings that describes a graph. What class does this language belong to? Why?

One question we may want to ask is if it is possible for every possible Boolean function

$$f : \{0,1\}^n \to \{0,1\} \tag{66}$$

is computable using a circuit of size poly(n). Claude Shannon showed in 1949 that the answer is no, and in fact, *most* functions with n input variables require about $2^n/n$ gates when using a combination of AND, OR, and NOT gates.

Question 93. How many different Boolean functions f that take n inputs are there?

A key difference between classical and quantum computing is that the outputs of quantum algorithms are probabilistic. To handle this, quantum complexity classes need to have some room for error. To prepare for this, let's look at the probabilistic version of P.

Classically, we simulate randomness by allowing the circuit to take random bits as input. That is, the randomness is in the inputs, not in the implementation of the circuit. On the other hand, the randomness of a quantum circuit is in the final measurement process.

Definition 10.4 (BPP (Bounded-Error Probabilistic Polynomial Time)). A language $L \subseteq \{0,1\}^*$ is in BPP if there exists a turing machine M and fixed polynomials $s_L, r_L : \mathbb{N} \to \mathbb{R}^+$, such that for any input $x \in \{0,1\}^n$, M takes in a "proof" $y \in \{0,1\}^{s_L(n)}$, halts in at most $O(r_L(n))$ steps, and:

- (Completeness/YES case) If $x \in L$, then for at least 2/3 of the choices of $y \in \{0,1\}^{s_L(n)}$, M accepts.
- (Soundness/NO case) If $x \notin L$, then for at most 1/3 of the choices of $y \in \{0,1\}^{s_L(n)}$, M accepts.

Question 94. How does the definition of BPP differ from that of NP?

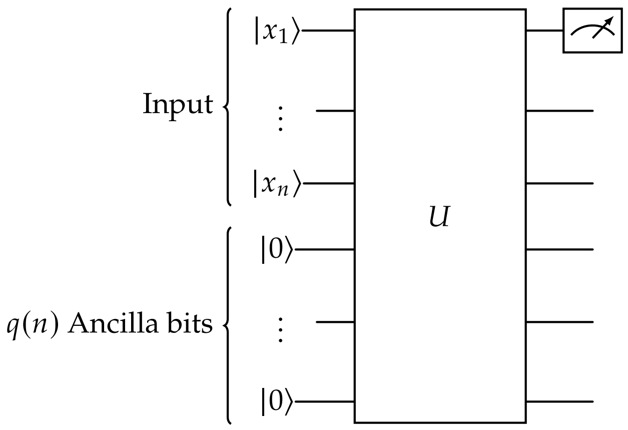
Question 95. We've picked some constants 2/3 and 1/3 for the completeness and soundness conditions. Suppose we have three copies of the turing machine M, and run the machine three times on the same input. We say that we accept if the majority of turing machines accept. What is the probability that we accept a string $x \in L$? What about a string $x \notin L$?

10.2 Quantum Complexity

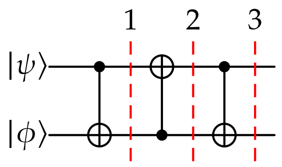
One critical difference between classical and quantum computing was seen through the No Cloning Theorem. Since we can copy data in classical computing, we only have to think about the input size, and make sure that the computation doesn't use more than say, polynomial space, throughout the process.

For analyzing quantum algorithms, we need to be explicit about the number of extra qubits we will need for our "work space". We call these **ancillary** qubits, and our input is appended with m ancillary qubits initialized to a known state, for example $|0\rangle$.

One final thing to note, is that we are still going to be interested in decision problems. To make this precise, we will require the algorithm to place the output bit on the first wire. This is not a big issue, as we can add SWAP gates at the end of the circuit to place the output bit on the first qubit.



Question 96. The swap gate switches the state of two qubits. Show that it can be modeled as three CNOT gates like below.



Now that we have a formal model for a quantum circuit, let's see our first quantum complexity class.

Definition 10.5 (BQP (Bounded-error quantum polynomial time))**.** A language L is in BQP if there exist polynomials $p(n)$ and $q(n)$ and a family of circuits $C_1, C_2, C_3 \dots$ such that the following hold:

- (polynomial space) On an n bit input x, the circuit C_n takes $|x\rangle |0\rangle^{\otimes q(n)}$ as input.
- (fixed output location) Output is the measurement of the first qubit in the standard basis.
- (polynomial size circuit) $|C_n| \leq p(n)$.
- (uniformity) There is a polynomial time Turing Machine that on input 1^n outputs a description of C_n.
- (completeness) If $x \in L$,
$$\text{Prob}\left[C_n(|x\rangle |0\rangle^{\otimes q(n)}) = 1\right] \geq \frac{2}{3} \tag{67}$$
- (soundness) If $x \notin L$,
$$\text{Prob}\left[C_n(|x\rangle |0\rangle^{\otimes q(n)}) = 1\right] \leq \frac{1}{3} \tag{68}$$

Note that the inputs and outputs are classical bits, making the problem inherently classical. So for BQP, we are solving a classical problem using a quantum computer.

How do the classical classes P and BPP compare to BQP? From what we hear from the news, we would expect that P, BPP $\subseteq$ BQP. That is, any function that can be computed efficiently with a classical circuit can also be computed efficiently with a quantum circuit.

To prove this rigorously, we can just show that every classical family of P and BPP circuits can be simulated using a quantum computer. However, to show this there are some issues that must be resolved related to some of the weird properties of quantum computing not seen in classical computers.

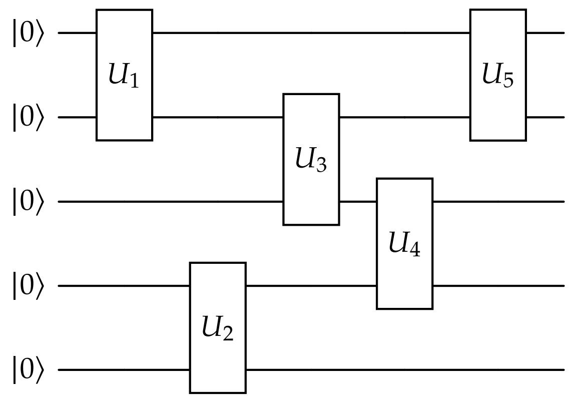
The first is No Cloning, since classical circuits often copy information around. This turns out to not be an issue though, since we can still copy classical bits.

Question 97. Make a reversible COPY gate.

10.3 Quantum Circuits

Let's first take a moment to study the computational model of quantum circuits. Since any function computed by a quantum circuit has to be unitary, we will always have circuits that have the same number of input and output qubits.

We define a quantum circuit as a sequence of primitive gates, each of which acts on a small (usually 1-3) number of qubits. We've seen some diagrams already that look like the following.



Here, each of the 2 qubit gates are actually unitaries acting on all n qubits, but we saw that the tensor product notation helps abstract this away. For example, the third layer as a matrix is

$$I_1 \otimes U_3 \otimes I_4 \otimes I_5 \tag{69}$$

Here we will explore some questions that arise when discussing quantum circuits.

- **Is it possible that quantum computers can solve any computational problem?**

No. A classical computer can always simulate an n-qubit circuit by explicitly storing the state as a 2^n-length vector, and performing matrix multiplication with the unitaries to update the state after each gate.

Since there are problems that classical computers cannot solve, a quantum computer is not able to compute any computational problem. The best we can hope for is an exponential improvement over classical computers in space and time.

- **Why does each gate act on only a few qubits?**

Similar to classical computing, when physically implementing these circuits, we often rely on small local interactions which are combined in clever ways by algorithms designers (you!) to do something interesting or useful. Thus it is natural for our computational model to have similar restrictions. We will see that this is not an issue in a similar way that it isn't in classical computing.

- **What is the role of interference in quantum computing?**

This is the most important feature of quantum circuits! An ideal quantum algorithm will elaborately plan the interference so that

- Amplitudes of incorrect answers cancel out and become small (destructive interference)
- Amplitudes of correct answers overlap and become large (constructive interference)

- **What is the role of entanglement in quantum computing?**

We don't usually talk about intentionally creating entanglement but it has to be there if we're doing anything interesting. Suppose we have a product state $|\phi\rangle$ on n-qubits:

$$|\phi\rangle := |\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle \tag{70}$$

Question 98. How many distinct amplitudes have to be stored in order to completely describe $|\phi\rangle$?

The consequence of the above exercise is that any algorithm that only works with product states and never entangles any qubits will never need more than a linear number of amplitudes. This makes it completely feasible to simulate such an algorithm on a classical computer, meaning we wouldn't need a quantum computer. Entanglement blows up this number of amplitudes, making it very difficult to simulate classically.

- **Is the restriction to local gates significant?**

We previously discussed that our computational model will be restricted to just using local gates. Will this lead to us not being able to solve certain problems? It turns out that this is not the case, and nature justifies our usage of local gates.

It can be proven that any unitary U ($2^n \times 2^n$ matrix) on n qubits can be decomposed as a product of 1-qubit and 2-qubit gates:

$$U = U_1 U_2 \cdots U_i \cdots U_n. \tag{71}$$

Furthermore, this decomposition is exact! The problem is, however, that *most* decompositions will need an order of 4^n 1- and 2-qubit gates to synthesize. This can be shown in a similar way to Shannon's counting argument.

As algorithm designers, we are mostly interested in *efficient* quantum circuits, which as computer scientists means the number of gates required grows polynomially with the input. The hope is that among these efficient circuits, there are interesting classes that allow us to solve problems faster than any classical algorithm could.

Another thing to note, is that the small gates we listed above are arbitrary 1- or 2-qubit gates. If we restricted to a finite set of gate types (H, CNOT, Z, etc.), would we be able to say the same thing? This leads us to another question we have alluded to earlier during the discussion of classical circuits.

- **Is there a universal gate set for quantum circuits?**

We need to formalize what exactly we mean when asking this question. Remember, unitaries are **continuous** in nature. If the question were asking whether there exists a finite set of gates that can be combined to compute *any* unitary on n qubits, the answer would be no. A finite set of gates could never capture the full continuous space.

What if we allowed for a little bit of error? Since the measurement process is probabilistic anyways, it seems quite reasonable to achieve *approximate* synthesis of unitaries without affecting the measurement statistics too much.

When thinking about designing a universal quantum gate set, we need to keep in mind the critical features of quantum computing that we need to support.

1. Must be able to create interference/superpositions.
2. Must be able to create entanglement.
3. Must be able to create states with imaginary amplitudes.

An important example of creating superpositions is the H gate. However, since it only acts on a single qubit, it isn't able to create entanglement. An entangled state we've seen already is the bell pair, and we were able to create that by combining the H gate and the CNOT gate. Now we only need a gate to satisfy the third point.

We could use something like the P gate, whose action can be described using the matrix

$$P = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}. \tag{72}$$

These gates seem to do what we wanted in terms of the three requirements, but they still do not form a universal set, as they only cover a discrete subset of all unitary operations. Furthermore, in a surprising result, it was shown that any circuit that starts in $|0 \cdots 0\rangle$ and uses only these gates can be simulated efficiently using a classical computer!

Let

$$R_\theta := \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{73}$$

Replacing the Hadamard gate with $R_{\pi/8}$ makes the aforementioned set a universal gate set, where universal means that it can approximate any unitary we want.

There are also other universal gate sets, a popular example being $\{\text{Toffoli}, H, P\}$. Depending on the hardware, some gates are easier to implement than others, so you may choose a different universal gate set based on what you have available. A fascinating result by Yaoyun Shi (https://arxiv.org/abs/quant-ph/0205115) states that CNOT and Toffoli gates do almost all of the heavy lifting towards making a gate set universal.

10.3.1 Approximating Unitaries and Quantum Universal Gate Sets

Now that we have some candidates, let's try to be more precise about what we mean when we say we can approximate a unitary U using another unitary U'. Intuitively, we would want to conclude that two matrices are *similar* if something like the following holds: "For every input, the outputs from U and U' are close." The mathematical tool we will be using to reason about this is the **operator norm**.

Definition 10.6 (Operator Norm)**.** Let U be an $n \times n$ matrix and $\{|\phi\rangle\}$ be the set of unit vectors of size n. Then, the **operator norm** of U is defined as

$$||U|| := \max_{|\phi\rangle} \left|U|\phi\rangle\right|. \tag{74}$$

We can use the norm to define a distance between two unitaries. We will say that U' **approximates** U if

$$||U - U'|| \leq \epsilon. \tag{75}$$

By the definition, this is equivalent to saying

$$||U - U'|| := \max_{|\phi\rangle} |U|\phi\rangle - U'|\phi\rangle| \leq \epsilon. \tag{76}$$

We can now formally define what a universal quantum gate set is.

Definition 10.7 (Universal Gate Set)**.** A gate set G is **universal** if for every

- n,
- unitary U on n qubits,
- $\epsilon > 0$,

there exists a sequence of gates $g_1, \ldots g_l$ such that each $g_i \in G$ and

$$||U - U_{g_1} U_{g_2} \cdots U_{g_l}|| \leq \epsilon \tag{77}$$

As you might expect, the more precise we want the approximation to be (small ϵ), the larger l will become. Can we calculate what this overhead will be exactly?

One natural way to start doing this from what we already know is to first decompose U into a product of (arbitrary) 1- and 2-qubit gates (which we know we can do):

$$U = U_1 U_2 \dots U_l \tag{78}$$

Recall that l *could* be as large as 4^n here. We are not expecting most unitaries to be approximate-able efficiently, so we won't worry about this for now. Then we can try to analyze how many gates from our universal gate set we will need to approximate each U_i. The following is a landmark result in quantum computing, giving us a good idea of what the overhead is.

Theorem 10.8 (Solovay-Kitaev Theorem)**.** If the universal gate set G is

- closed under inverse, and
$$g \in G \Leftrightarrow g^{-1} \in G \tag{79}$$
- generates a dense subset for 2-qubit gates. (You can approximate any 2-qubit gate to within ϵ using gates from G,

then only $(\log(1/\epsilon))^2$ gates are required to simulate any 2-qubit gate to within ϵ.