

Quantum Computing and Algorithms

Jennifer Ren and Andrew Chang

December 21, 2022

Abstract

We start with a bit of classical computation as a reference point. Then, we go over provide necessary linear algebra definitions, basic quantum mechanics, and quantum gates. We then explain a model of quantum computation, specifically the quantum finite automata. Finally, we conclude with quantum algorithms that allow us to understand how quantum computing can perform tasks substantially more efficiently compared to classical computers.

1 Introduction

Quantum computing is a type of computation that takes advantage of quantum mechanical phenomena, namely superposition, interference, and entanglement. They are believed to be able to carry out massive calculations. Despite being too small for practical applications, they are believed to compute specific problems faster than classical computing, a phenomenon known as “quantum supremacy.”

While classical models of computation may attempt to brute force complex tasks such as sequencing protein sequences and result in failure due to the sheer amount of possibilities, quantum computers create multidimensional space that allows them to find patterns in otherwise unfeasible combinations. The study of quantum computation is a branch of quantum information science.

Different models of quantum computation exist, with circuits being the most popular among them. Other models include the quantum finite automaton, quantum Turing machine, quantum annealing, and adiabatic quantum computation. The unit of quantum computing is known as the qubit, which parallels to the basic unit of classical computation known as the bit. What separates it from its classical counterpart is its ability to represent both 0 and 1 simultaneously.

Because the qubit has so many unique properties and is well suited for tasks difficult to accomplish with traditional computers, these models of computation are used to represent what a quantum computer can do. Furthermore, various algorithms have been designed which are theoretically capable of solving various elusive problems, like efficiently cracking modern encryption, through the use of unique quantum properties. We will give an overview in this paper.

2 Classical Computation

2.1 Computability

A quantum computer can solve any computational issue that can be resolved by a classical computer. In contrast, each issue that a quantum computer can resolve can, in theory, be resolved by a classical computer if given enough time. We will give examples of what we mean by “computable” in the next section.

In order to appreciate quantum computation we must first understand what classical computers cannot do. So far we have two equivalent mathematical models that are equivalent and so, in general we can pick one of the models.

Example 2.1 *Let N be an integer. We want to figure out if N a prime. This is clearly computable, since we can do this through classical computation by brute-force.*

Example 2.2 *The Halting problem asks us to find out, based on your computer program, whether your program terminates or will continue forever hence the word “halting”. Alan Turing actually proved this to be uncomputable in 1936.*

Example 2.3 Given a massive polynomial like $2(wxyz)^{17} - wxyz^{15} + 195wzy^3 + 9$, we can imagine that it is uncomputable, which is backed by the fact that in 1976, it was indeed determined to be uncomputable by proof.

It remains that with quantum computing that the last two examples are uncomputables, as theoretically anything we compute with quantum computation can be computed using a classical computer, so it remains that uncomputable classical problems remain uncomputable. Despite that, quantum computation is still worth looking at especially in the realm of complexity.

Definition 2.1 An *input bit string* is a sequence of bits $x = i_1i_2 \dots i_n$, where each i_k is either 0 or 1. We write B_n for the set of all n -bit strings. The input size is the length n . If we are given a number as an input, we take the logarithm of it to be its size.

2.2 Classical Gates

While the Turing machine is the most famous model for classical computation, we generally use what's called a **circuit model**. [DW19] The idea is quite simple. When we are working with bits, define a map $f : B_m \rightarrow B_n$ to represent our program. The program that is the arrangement of the boolean gates: *AND*, *OR*, and *NOT*, takes generally the time of the number of gates. If we choose a different set of gates, we still only need a fixed number of gates to do it because it still will take a fixed number of gates to construct the program with *AND*, *OR*, and *NOT*. As a result, the time difference between the programs will only be polynomial, and thus won't make a huge difference.

3 Review

3.1 Linear Algebra Definitions/Concepts

Complex numbers are prevalent in the field of Quantum Computing. As a result, we will go over basic definitions that are relevant to understanding the computational models, and more specifically the algorithms.

In Quantum Computing, the probabilities of states and transitions are given in complex numbers, and these **probabilities** are known as amplitudes. Where z is a complex number such that $z = a + bi$ where $a, b \in \mathbb{R}$, we can write the magnitude of z ($|z|$) as to $\sqrt{a^2 + b^2}$. For two real numbers r_1 and r_2 , the sum of the two real numbers is greater or equal to that of r_1 and r_2 [Zec22]. In contrast, for two complex numbers c_1 and c_2 , the squared magnitude of the sum of the two complex numbers does not necessarily have to be greater or equal to the either $|c_1|^2$ or $|c_2|^2$. We can describe any complex number as a pair of real numbers. We use complex numbers in quantum computing due to time evolution and the reversibility condition.

Definition 3.1 *Time evolution* refers to how states can be changed in the passage of time.

Definition 3.2 *Reversibility condition* refers to how the transformations between quantum states are reversible.

These complex numbers can then be written as polar coordinates (ρ, θ) . The ρ is called the magnitude, and the θ is called the phase. From the original complex number, we can write it in polar such that,

$$\rho = \sqrt{a^2 + b^2}, \theta = \tan^{-1}\left(\frac{b}{a}\right).$$

Vectors are **orthogonal** if the inner product between the two vectors is equal to 0. A set of vectors is **orthonormal** if each pair in the set is orthogonal and each vector is a unit vector or has a norm of 1:

$$\|V\| = \sqrt{\langle v, v \rangle}$$

In Quantum Computing, a particular space that is of interest is **Hilbert Space** [Hol03]. Hilbert Space is relevant to us as it is the space where the wave function (ψ) lives. More specifically, the inner product of two vectors in a Hilbert Space gives the distance between the two vectors. Hilbert spaces can be both finite and infinite. For a matrix A , A^\dagger is equal to the conjugate of transposed A . An identity matrix is an $n \times n$ matrix that consists of 1s and 0s and can be represented as I . A matrix A is unitary if **unitary** if $AA^\dagger = A^\dagger A = I$.

3.2 Basic Units of Computation: Bits and Qubits

The building block of classical computing, otherwise known as binary computing, is the bit. In classical computing, information is stored in bits, with a discrete number of possible states.

Compared to bits of classical computation, qubits have an infinite, continuous number of states due to the principle of superposition. Generally, a qubit can be represented as:

$$\alpha|0\rangle + \beta|1\rangle$$

where, α and β are complex numbers[Zec22]. The qubits is a unit vector in \mathbb{C}^2 with basis vectors

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

When we have multiple qubits, we write them as $\langle a \rangle \langle b \rangle$, which is a shorthand for $\langle a \rangle \otimes \langle b \rangle$. This \otimes is referred to as a tensor product, which we will go over later.

Now any classical bit string $x = i_1 i_2 \dots i_n$ can be encoded as a qubit.

$$|i_1\rangle|i_2\rangle \dots |i_n\rangle|0\rangle \dots |0\rangle \in \bigotimes_{i=0}^{n+k} \mathbb{C}^2 \cong \mathbb{C}^{2^{n+k}},$$

Often, we will have to pad extra zeros at the end because quantum computers generally have fixed input size. In contrast, in classical computing, we could be free to ditch all the extra zeros. However, we cannot do that in quantum computing as all our operators have to be invertible. Thus, to make this map invertible, we have to supply the zeros in the beginning of the string.

3.3 Tensor Products and Bra-ket notation

Quantum states are generally represented as "kets" by physicists. This notation is known as bra-ket notation. Where $\langle a \rangle = v_0 \langle 0 \rangle + v_1 \langle 1 \rangle$. Where v_0 and v_1 are the probability would be v_0^2 and v_1^2 of measuring a 0 or a 1[Zec22]. As a result, they must sum to 1. $|v_0|^2 + |v_1|^2 = 1$. The tensor product is a way we combine quantum states. Essentially, if we have two particles a and b then we can represent

their tensor product as a vector. $|ab\rangle = |a\rangle \otimes |b\rangle = v_{00}|00\rangle + v_{01}|01\rangle + v_{10}|10\rangle + v_{11}|11\rangle \rightarrow \begin{bmatrix} v_{00} \\ v_{01} \\ v_{10} \\ v_{11} \end{bmatrix}$

In math, we define the Tensor Product to be $V \otimes W$ where V and W are vector spaces over the same field. It is a bilinear map which means it's linear in V and W . $V \times W \rightarrow V \otimes W$ that maps a pair (v, w) , $v \in V, w \in W$ to an element of $V \otimes W$ denoted $v \otimes w$

Definition 3.3 We say that an element in $v \otimes w$ is called the **tensor product** of v and w .

Definition 3.4 An element of $V \otimes W$ is called **tensor**.

Sometimes we call tensors that can be represented in this form as an elementary tensor. The elementary tensors span $V \otimes W$, where we can write each element as a linear combination of elementary tensors. Like in linear algebra if we are given bases given for V and W , the AA basis for $V \otimes W$ is formed by taking all possible combinations of tensor products of a basis element in V and W .

We won't really go into how else it's really derived but we will state a couple properties for completeness: If V and W are vector spaces of finite dimension, then $V \otimes W$ is also finite dimensional. This follows from the fact that the bases of V and W are finite.

$$(rv) \otimes w = r(v \otimes w)$$

$$r \otimes (rw) = r(v \otimes w)$$

$$(v_1 + v_2) \otimes w = v_1 \otimes w + v_2 \otimes w$$

$$v \otimes (w_1 + w_2) = v \otimes w_1 + v \otimes w_2,$$

where r is over the field of our vector spaces.

We will also show how to take tensor products of matrices because often times it might be easier to understand our linear maps that way. Let us have two linear maps S and T . We want to find $S \otimes T$. Let's also say that we know we can represent S and T as 2 by 2 matrices A and B respectively:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, \quad B = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix},$$

Then our tensor product of $A \otimes B$ is

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}.$$

If something is not an elementary tensor, then there's some entanglement going on. This will be all we need to know to understand the theory that we need to understand before moving on to gates.

3.4 Quantum Gates

Quantum gates are not just binary operators, but are in fact unitary operators. As a result, they come with all the nice properties of being a unitary operator. A gate that acts on n qubits is represented as a $2^n \times 2^n$ matrix. Our gates act on one over two bits but we chain them together.

Our first gate is the identity gate denoted as I denoted for a single qubit as one might guess the identity matrix [DW19].

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

It doesn't change the quantum state of the particle we apply it to.

Another very common gate is called the Pauli gate, (X, Y, Z) are the three Pauli matrices $(\sigma_x, \sigma_y, \sigma_z)$ [Hol03]. What the gates X, Y, Z do is correspond to π rotations around x, y, z planes respectively in the Bloch Sphere. X often called the bit-flip $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. The Pauli-Y gate maps $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $i|0\rangle$. Pauli-Z does leaves $|0\rangle$ unchanged and will map $|1\rangle$ to $|-1\rangle$. These matrices are represented as

$$= \sigma_x = \text{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The Pauli matrices are inverse revolutions of themselves and satisfy $I^2 = X^2 = Y^2 = Z^2 = -iXYZ = I$

Controlled gates act on two or more qubits. The most famous example is the CNOT gate! It acts on two qubits and it functions and performs the not operation on the 2nd qubit if the 1st qubit basis state is $|1\rangle$. With respect to the basis $|11\rangle |00\rangle |01\rangle |10\rangle$. The matrix is

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We can also generalize this idea: Let U be a matrix that acts on 1 qubit, Then U can be written as $U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}$

We define the controlled U-gate to be a gate that operates on two qubits in which the 1st qubit will always serve as the control.

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |1\rangle \otimes U|0\rangle = |1\rangle \otimes (u_{00}|0\rangle + u_{10}|1\rangle) \\ |11\rangle &\mapsto |1\rangle \otimes U|1\rangle = |1\rangle \otimes (u_{01}|0\rangle + u_{11}|1\rangle) \end{aligned}$$

In matrix form this is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$

Most commonly, the U represents the Pauli matrices

The next gate is called the Hadamard Gate named after Jacques Hadamard. It acts on a single qubit but it sends $|0\rangle \mapsto \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle \mapsto \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ The matrix representation of this is $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Furthermore, there is also the Swap gate where it swaps two qubits with respect to the basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ It is decomposed as

$$\frac{I \otimes I + X \otimes X + Y \otimes Y + Z \otimes Z}{2}$$

We now move onto the Toffoli gate which acts on 3 qubits. If we only accept qubits of the form $|0\rangle$ and $|1\rangle$. If the first two bits are $|1\rangle$ then we apply the Pauli-X gate on the third bit. Otherwise we do nothing. This is the matrix representation.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The Toffoli gate is related to *AND* and *XOR* gate because it maps $|a, b, c\rangle \mapsto |a, b, c \oplus a \wedge b\rangle$

4 Quantum Computation Models

4.1 Quantum Finite Automata

Quantum finite automata, like their classical counterparts, are an abstraction of quantum computers. There are two main types: measure-many automata and measure-once.

Like Probabilistic Finite Automata, or PFAs, Quantum Finite Automata (QFAs) have weighted transitions except instead of probabilities, the weight is given by amplitudes. The measure-many QFA, also known as (1-QFA), has a transition function δ defined as:

$$\delta : Q \times \Gamma \times Q \longrightarrow \mathbb{C}_{[0,1]}$$

The QFA uses left and right end markers for the inputs. This is represented as:

$$\Gamma = \Sigma \cup \{\#, \$\},$$

where the $\#$ is the left and the $\$$ is the right respectively. The QFA rejects or accepts a string with probability between 0 and 1. The measure once automata has N possible internal states which is represented by the N qubit $|\psi\rangle$. The transitions, analogous to those of PFAs and classical DFAs, can be represented by $N \times N$ unitary matrices: each unitary matrix for a letter of the alphabet, Σ . [Daw04] Transitioning from $|\psi t_0\rangle$ to $|\psi t\rangle$, we assign the probability of this transition as:

$$P(|\psi t_0\rangle, |\psi t\rangle) = |\langle \psi t_0 | \psi t \rangle|^2.$$

As mentioned before, this probability is a value between 1 and 0.

We have the measure once-automata as an example.

Let's do some computation based on this state-machine. The quantum state is The quantum state is a vector, in bra-ket notation

$$|\psi\rangle = a_1|S_1\rangle + a_2|S_2\rangle = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$











Operator	Gate(s)	Matrix
Pauli-X (X)	 \oplus	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Figure 1: Gates and there matrix representations

State Transition Table

Input State	1	0
S ₁	S ₁	S ₂
S ₂	S ₂	S ₁

State Diagram

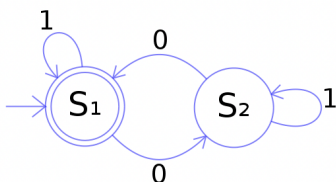
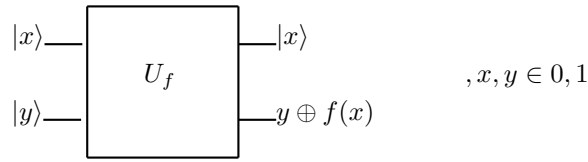


Figure 2: State machine

Figure 3: Oracle Gate



with the complex number a_1, a_2 normalized so that

$$\begin{bmatrix} a_1^* & a_2^* \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = a_1^* a_1 + a_2^* a_2 = 1$$

The unitary transition matrices are

$$U_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Taking S_1 to be the accept state, the projection matrix is

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

As should be readily apparent, if the initial state is the pure state $|S_1\rangle$ or $|S_2\rangle$, It's going to be identical to the classical DFA so our expression is $(1^*(01^*0)^*)^*$.

5 Algorithms

5.1 Deutsch's Algorithm

Deutsch's Algorithm is a unique case among the Deutsch-Jozsa algorithm. It is designed to determine whether Boolean function $f : \{0, 1\} \rightarrow \{0, 1\}$ is constant or balanced. In other words, the algorithm is designed to determine if $f(0) = f(1)$ (constant) or both possible outputs happen once (balanced).

While a classical approach to this problem results in the evaluation of $f(0)$ and $f(1)$, so 2 evaluations, the algorithm allows us to evaluate the function in a single quantum measurement. Answering the question if the function is constant or balanced is the same as calculating $f(0) \oplus f(1)$ because if the sum is 0, then the function is constant, and if the sum is 1, then the function is balanced. Deutsch's circuit uses the "oracle" gate, which is a two-qubit gate. Referred to as the "oracle" gate, the gate is essentially a "black box" that outputs a value for a supplied input.

In the oracle gate, the upper qubit outputs in the same state, while the lower qubit can change or be unchanged depending on $f(x) = 1$. If $f(x) = 1$, the lower qubit is flipped. The gate represents a unitary transformation.

We observe the Deutsch circuit in the following figure. First, we start with two qubits, starting with one in basis state 0 and the other in state 1. Then, we apply the Hadamard gate. The Hadamard gate maps the basis state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. These are the plus state and minus state, respectively, so $|\psi_1\rangle = |+-\rangle$.

After the Hadamard gate, the qubits then enter the oracle gate. The first qubit goes into the oracle enters as x while the second enters as a minus state which can be written as $\frac{|x0\rangle-|x1\rangle}{\sqrt{2}}$. What exits the oracle gate is

$$\frac{1}{\sqrt{2}}(|x, f(x)\rangle - |x, 1 - f(x)\rangle)$$

The $1 - f(x)$ represents how the lower qubit is flipped based on $f(x)$'s value. If $f(x) = 0$, then our output is

$$\frac{1}{\sqrt{2}}(|x, 0\rangle - |x, 1\rangle)$$

which is equivalent to our input $|x-\rangle$. Vice versa, if $(f(x) = 1$, then our output is

$$\frac{1}{\sqrt{2}}(|x, 1\rangle - |x, 0\rangle)$$

which is equivalent to $-|x, 0\rangle$. These outputs can also be represented as

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

Going back to our result from the Hadamard gate, our $|\psi_1\rangle = |+-\rangle$ can be represented equivalently as $\frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes |1-\rangle$, and then further simplified to $\frac{1}{\sqrt{2}}(|0-\rangle + |1-\rangle)$. Plugging in our output from the oracle gate,

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}((-1)^{f(0)}|0-\rangle + (-1)^{f(1)}|1-\rangle)$$

There are two possible cases as stated above, $f(0) = f(1)$, a constant function and $f(0) \neq f(1)$. For a constant function, we see that since $f(0) = f(1)$

$$|\psi_2\rangle = \pm \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle \otimes |-\rangle) = |+-\rangle$$

. In turn, for a balanced function, we see that

$$|\psi_2\rangle = \pm \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle \otimes |-\rangle) = |--\rangle$$

.
Going back to our original circuit, the third step is the application of the Hadamard gate on our first qubit.

$$\pm \begin{cases} |0-\rangle \\ |1-\rangle \end{cases}$$

After this application, we apply a measurement only on the first qubit to get a basis state of 0 or basis state of 1, depending on, as mentioned above, that if the function is constant or balanced.

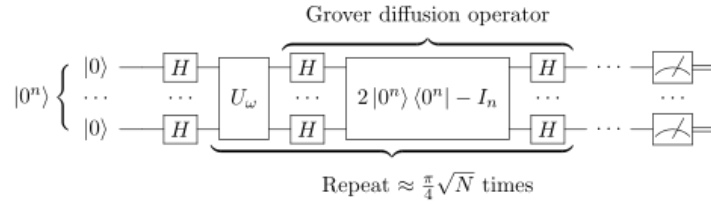
While Deutsch's algorithm may seem to be an unusual method to figure out if a function is constant or balanced, what makes it attractive is that it only needs to be called once. Deutsch-Jozsa's problem demonstrates the superiority of the quantum algorithm over its classical counterpart. Generalizing our previous problem, we now accept values of x from the set $N = 0, 1, 2, \dots, 2^n - 1$. Like in our Deutsch problem above, the function $f(x)$ once again only takes values 0 and 1, and can be either constant or balanced. The worst-case scenario for a classical approach to Deutsch-Jozsa's problem is that it would check more than half of the values before it could declare $f(x)$ to be constant, more specifically $2^{n-1} + 1$ evaluations. On the other hand, Deutsch-Jozsa's algorithm, like Deutsch's algorithm, only needs a single measurement to determine if $f(x)$ is constant or balanced.

This is a substantially better improvement, demonstrating "quantum supremacy". However, reflecting on our classical strategy, we must notice that it is deterministic, while our algorithm is not. This improvement should not be solely evaluated on this comparison as probabilistic classical algorithms can solve the Deutsch-Jozsa problem more efficiently than our classical approach above. Therefore, when comparing classical and quantum algorithms, we must be aware of other approaches that may influence the advantages we see in quantum computing [Zec22].

5.2 Grover's Algorithm

Grover algorithm can perform searches substantially faster than conventional methods. Grover's algorithm is an unstructured search. Grover's algorithm allows us to find the solution in $\sqrt{2^n}$, significantly better than the classical algorithm that has a worst case solution with $2^n - 1$ tests.[EJ96] To demonstrate this difference, let us first define a binary string $x \in 0, 1^n$. Like in Deutsch's Algorithm, we

Figure 4: Grover's algorithm



will define a function $f(x)$ that returns 1 for a positive search result and 0 for everything else. While we are using two qubits for simplicity to demonstrate the algorithm, Grover's Algorithm can use any number of qubits.

The first step for Grover's algorithm is putting the two qubits in Hadamard gates as we did in Deutsche's algorithm. This gate can be represented as:

$$|s\rangle = \frac{1}{\sqrt{n}} \sum_{x=0}^{N-1} |x\rangle$$

The Hadamard gate makes the amplitude of each qubit the same. A more general form of the step for n qubits is

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^N-1} |x\rangle$$

We will now define what is known as the Grover iteration. The Grover iteration is the repetition of the oracle function and what is known as the Grover diffusion operator. For each iteration, the oracle gate (pictured here and above for reference):

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle,$$

flips the amplitude of the solution we are looking for, essentially flagging it as the solution we are searching for in the string. After the oracle function, we apply a basis state change to the flagged qubit and amplify it. This change is known as the aforementioned Grover diffusion operator.[\[Rau21\]](#) This is represented as

$$U_s = 2|s\rangle\langle s| - I$$

for our two qubit example, but can be more generally represented as

$$U_s = 2|s\rangle\langle s| - I$$

The Grover iteration can be recursively written as

$$|\psi_{t+1}\rangle = U_s U_f |\psi_t\rangle$$

Grover final iteration can be represented as

$$|\psi_t\rangle = (U_s U_f)^t |\psi_0\rangle$$

where t is the number of iterations it takes to approach the probability of approaching with the function. Further analysis actually shows that t is $\lceil \frac{\pi}{4} \sqrt{N} \rceil$.

5.3 RSA Encryption Algorithm & Shor's Algorithm

The idea of the RSA Encryption Algorithm is that a key is split into two parts: encryption and decryption. The encryption key is public and allows anyone to encrypt a message, based on modular encryption, while the decryption key is kept private and can unencrypt messages via some basic number theory.

Specifically, we let n be the product of two large prime numbers p and q , such that $\phi(n) = (p-1)(q-1)$. n is public, along with the public exponent e , which is just a number less than n and relatively prime to it. The private key d is $e^{-1} \pmod{\phi(n)}$, meaning de is a multiple of $\phi(n)$ plus one. We can't find d easily even if we know e , because it relies on the value of $\phi(n)$ which is difficult to find (it requires factoring an extremely large number, which traditional computing struggles with).

However, if we encrypt a message m into a ciphertext c by using modular exponentiation:

$$c \equiv m^e \pmod{n},$$

then someone with the private can decrypt this by just taking the exponent again:

$$c^d \equiv m^{de} \equiv m^{k \cdot \phi(n) + 1} \equiv m \pmod{n}.$$

This is based on Euler's theorem, which says that $m^{\phi(n)} \equiv 1 \pmod{n}$ assuming m and n are relatively prime. (The full equation ends up working even if they aren't relatively prime, since both sides would just be $0 \pmod{n}$ or something else depending on what factors they share.)

The crux of the RSA algorithm is because it takes so much time to try and factor n to find $\phi(n)$, rather than starting with p and q , it is impossible for an interceptor to calculate the private key and start reading the encrypted messages. [RSA78] However, quantum computing has the potential to change this with the development of Shor's algorithm.

Essentially, the goal is to factor a large number n by finding a number b such that $b^2 \equiv 1 \pmod{n}$, since then we can factor

$$b^2 - 1 = (b+1)(b-1) = kn$$

to give us new divisors of n . To do this efficiently, we can simply pick a random number less than n , check if it's relatively prime to n and then use a quantum circuit to find the period r of $f(x) = a^x \pmod{n}$. This is the smallest positive integer that satisfies $a^r \pmod{n} = 1$, meaning that if r is even and $a^{r/2} \not\equiv -1 \pmod{n}$, then $a^{r/2}$ is the b mentioned earlier: adding or subtracting 1 will give us a number that shares nontrivial divisors with n .

The circuit to find the period is complex, but the theory is that if it is configured based on the value of n , then converting $f(x)$ into a quantum function and applying it, then applying a Fourier transform, will lead to constructive interference that lines up with the period of f . [Sho94] Of course, this has to be tailored to every new message, and the actual physics of the circuit is difficult to achieve properly. As a result, it will take much future development until this is practical; currently, scientists are working on successfully factoring the number 35. Nevertheless, it is a promising route to developing more advanced mathematical methods and decryption applications.

References

- [Daw04] Anuj Dawar. Quantum computing lecture. *Review of Linear Algebra*, 2004.
- [DW19] Ronald De Wolf. Quantum computing: Lecture notes. *arXiv preprint arXiv:1907.09415*, 2019.
- [EJ96] Artur Ekert and Richard Jozsa. Quantum computation and shor's factoring algorithm. *Rev. Mod. Phys.*, 68:733–753, Jul 1996.
- [Hol03] Alexander S Holevo. *Statistical structure of quantum theory*, volume 67. Springer Science & Business Media, 2003.
- [Rau21] Jochen Rau. *Quantum Theory: An Information Processing Approach*. Oxford University Press, 2021.
- [RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.
- [Zec22] Aleksandar Zecevic. *Quantum and Parallel Computing: A Tale of Two Paradigms*. Independently Published, 2022.