Building and Running Grover's Algorithm on a Quantum Computer

Thelonious Omori

Abstract

In this paper, Grover's search algorithm will be implemented from scratch using Qiskit to solve the unique search problem on a real IBM quantum computer. First, the important background details governing the mathematics behind the quantum computer will be discussed. This is to give the reader an in-depth understanding of the theory and notation needed for quantum algorithms. This will be followed by an in depth explanation of the Grover algorithm, demonstrating why it provides such a theoretical advantage compared to classical search algorithms. Following this, Grover's algorithm is going to be implemented from scratch using Qiskit. This will first be run on a simulator of a perfect quantum computer to demonstrate its theoretical effectiveness. This is followed by running the algorithm on a real IBM quantum computer with different levels of optimization and database size to demonstrate the effectiveness of contemporary quantum computers. Finally, this we be compared to a classical search algorithm to showcase the practicality of using quantum algorithms in 2024. Showcasing the current limitations that halt the widespread adoption of quantum computing solutions.

1 Introduction

The idea of quantum computation was first introduced by Richard Feynmann in 1982 presenting the idea that simulations of quantum systems could be done more efficiently at the scale of an atom, utilizing its quantum mechanical effects[1]. Since then, there has been a plethora of research and innovation that has led to the rapid development of quantum computing. In 2022 alone quantum computing startups in hardware and software had $2.35 billion invested into them[2]. This resulted in a very rapidly growing industry.

To accurately describe a quantum computer, it is helpful to start with the principles behind classical computing. In classical computers, information is represented using bits, the smallest units of data. A bit indicates the status of the current flow in a circuit, with '1' representing the current flow and '0' represents no current. Transistors are used to construct logical gates, these are configured to perform computational operations within this binary system. These foundational elements enable the representation and manipulation of data in electronic devices, namely computers[3].

Quantum computers work in some ways similarly to classical computers however they can exploit mechanisms that are exclusively quantum mechanical such as superposition or entanglement. In a quantum computer, the classical bit is instead a qubit (short for quantum bit) which obeys quantum properties. In practice, the qubit can be any quantum mechanical system that is two-level (has two measurable states). Examples of this are the polarization of a photon or the spin of an electron[4]. Any quantum mechanical system is described by a unique wave function. This wave function comes from solving the Schrodinger

equation for that particular system (a qubit in this case) and it describes everything there is to know about the system before it is measured. The wave function describes the qubit as a superposition of all its states at once with each state having its own probability of being measured. Once the actual state of the qubit is measured/observed it causes the wave function to collapse resulting in just one of the different possible states being measured. This is why in quantum mechanics it is said that before the state of the qubit is measured it is in both states at once. So if one of the qubit states is labeled 0 and the other labeled 1 the qubit can be both 0 and 1 at the same time (a superposition of states) whereas the classical bit can only be in the state 0 or 1. This is why quantum computers allow for parallel computing and increased processing power compared to a classical bit. By not measuring the state of the qubit, it can be kept in a superposition. Therefore, allowing the computer to perform calculations with all the states at the same time. The mathematical details of this will be explained in the next section.

Due to the large-scale interest in this technology, there are now many different quantum computers that currently exist. Since this technology is constantly being innovated these quantum computers also rely on different methods to work. For example, some quantum computers developed by Alpine Quantum and IonQ use trapped ions as qubits. This works by trapping the ions and using electromagnetic fields to carefully manipulate them. On the other hand companies like IBM and Google use superconducting materials as qubits[5]. These quantum computers are a huge leap forward in terms of technical sophistication however they're far from perfect and prone to many technical problems.

One of the biggest problems currently halting the progress of quantum computing is quantum decoherence. Decoherence is when the quantum system loses a definite phase relation between states (quantum coherence), meaning that the laws that govern that specific system become better explained by the classical understanding of physics as opposed to the quantum mechanical view. If a quantum system is completely isolated from the surroundings it is said to be coherent, however, any interaction this system has with the external world decreases its coherence[6]. There are many sources of decoherence in quantum computers depending on the way it is engineered such as ionizing radiation and tunneling. These decoherence sources ultimately cause errors in the computation and result in inaccurate quantum computers. The amount of time that a quantum system takes to become decoherent is known as its coherence time. This is what many quantum computer scientists attempt to increase to allow for longer and more robust quantum computing. One of the potential solutions to increase coherence time is making use of clock transitions[7]. If coherence time is increased enough it can lead to a completely fault-tolerant quantum computer. However, there is still much more to figure out. As well as engineering solutions for new quantum computers there is a large area of research into error-correcting algorithms to run alongside quantum computers. These algorithms are effective and widely adopted for most quantum computers. This paper will

also demonstrate the use of error-correcting quantum algorithms when running Grover's algorithm. The major drawback of error correction is that it requires more steps in the algorithm so that the error correcting part can be run too. This decreases the overall efficiency of the quantum algorithm.

To demonstrate the effectiveness of quantum computers this paper is going to focus on Grover's algorithm. Grover's algorithm is a quantum database search algorithm discovered by Lov Grover in 1996[8]. The algorithm solves the unstructured search problem with a database of size N in $O(\sqrt{N})$ evaluations. While there are more efficient classical search algorithms, these are all for a structured database. So the best known classical search algorithm for unstructured database is the linear search[9]. This can solve the unstructured search in $O(N)$ evaluations. This means Grover's algorithm offers a quadratic speedup. Hence, it is an example of a quantum algorithm with a real speed advantage compared to its classical counterpart. In this project Grover's algorithm will be used to solve the unique search problem which searches for one item in a list of items. This is a very wide-reaching problem and is helpful for many algorithms in optimization. This will be implemented using Qiskit. Qiskit is an open-source software development kit (SDK) developed by IBM[10]. It is run via Python and easy to install with a wide range of functionality that enables the development of quantum circuits and algorithms.

2 Quantum Information

This section will deal with how quantum information can be represented mathematically. Qubits are quantum mechanical and hence can be in a superposition of different states governed by its wave function[11]. So in the case of a two-level system like the spin of an electron which is either spin up or down these states can be represented as the two possible states of the qubit. One state e.g. spin down will be called $|0\rangle$ and the other state (spin up) will conversely be called $|1\rangle$. This can be demonstrated using Dirac notation in (equations 1 & 2).

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{1}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2}$$

In Dirac notation each state is represented as a vector in the Hilbert space [11]. Now using this defined expression for the two possible qubit states the wave function of the qubit can be expressed as in (equation 3)

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{3}$$

Where α and β are the complex amplitudes of the quantum states and the probability of each state is given by the modulus square of its amplitude and

hence is subject to the conditions in (equation 4).

$$|\alpha^2| + |\beta^2| = 1 \tag{4}$$

These conditions are given by the Born rule which was postulated by Max Born in his 1926 paper [12]. In quantum mechanics, the square modulus of the amplitudes represents the probability of the state being measured. This is why the Born rule (equation 4) must be true since the qubit has a 100% probability of being in one of its possible states. Since the amplitudes of these state vectors are complex numbers smaller than 1, they can be represented by Euler's formula in (equation 5).

$$\alpha = e^{i\phi} \tag{5}$$

Where ϕ is the angle (in radians) between the positive real axis and the complex number α when plotted in the complex plane. ϕ is known as the relative phase of that specific state. This concept of phase is deeply important to quantum computing since many algorithms involve phase manipulation.

Using Dirac notation allows for a more accurate understanding of what makes qubits in some ways superior to classical bits. Qubits are described by their wave function and this is very useful since it allows for superposition. However, it becomes more useful when introducing a second qubit. When a second qubit is included, the wave functions of both qubits can be described as one total wave function with 4 different states. This is known as a product state. This is done mathematically via the tensor product[13], and it is key to understanding more complicated quantum computing formulas. Suppose two qubits have wave functions $|\phi_1\rangle$ and $|\phi_2\rangle$. These can both be described by (equation 6 & 7) respectively.

$$|\phi_1\rangle = \alpha_1 |0_1\rangle + \beta_1 |1_1\rangle \tag{6}$$

$$|\phi_2\rangle = \alpha_2 |0_2\rangle + \beta_2 |1_2\rangle \tag{7}$$

The subscript is used in this case to denote which qubit is being referred to, but it is the same as the states described in (equation 1 & 2). In a quantum circuit each qubit has two possible states and hence there are 4 different possible outcomes when considering both qubits at once. These are ($|0_1\rangle$ & $|0_2\rangle$, $|0_1\rangle$ & $|1_2\rangle$, $|1_1\rangle$ & $|0_2\rangle$ or $|1_1\rangle$ & $|1_2\rangle$). This means that the two different qubits could be described by the same wave function. This is done mathematically via the tensor product in (equation 8).

$$|\phi_1\rangle \otimes |\phi_2\rangle = \alpha_1\alpha_2 |0_1\rangle |0_2\rangle + \alpha_1\beta_2 |0_1\rangle |1_2\rangle + \beta_1\alpha_2 |1_1\rangle |0_2\rangle + \beta_1\beta_2 |1_1\rangle |1_2\rangle \tag{8}$$

This equation can then be further simplified by dropping the subscripts on the states as it is usually unimportant while still paying attention to the order. Then, combining the product states e.g. $|0\rangle |1\rangle$ into single states $|01\rangle$ for compactness. Hence, the new form of (equation 8) can be seen in (equation 9).

$$|\phi_1\rangle \otimes |\phi_2\rangle = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle \tag{9}$$

As can be seen, the result is a wavefunction with 4 different distinct states and hence represents a 4-d vector in Hilbert space as expected. The normalisation factor is also accounted for within the mathematics. This concept of the tensor product can then be generalised to n qubits resulting in a total of 2^n different distinct states by repeating the process in (equation 9). This notation is also really helpful once you get into a high number of qubits. For example, if there are 3 qubits and the goal was to find the state where qubit 1 is in state $|0\rangle$, qubit 2 is in state $|1\rangle$ and qubit 3 is in state $|0\rangle$, the state can be written as $|010\rangle$. The convention of qubit ordering is important when using a quantum computer and will be covered when discussing quantum circuits. Also, when n qubits are each in the state $|x\rangle$ where x is either 1 or 0, the total state can simply be written as $|x^n\rangle$. Therefore, quantum computers can allow computing with 2^n states at the same time which is one of the biggest advantages it potentially has over classical computers.

This is great since product states allow for a rapid increase in possible states by adding more qubits. However since qubits are quantum, using specific quantum gates can entangle two qubits. A quantum entanglement is when the state of a system becomes dependent (entangled) with the state of another, no matter the separation[14]. This allows for different computational operations that are physically impossible on other devices. This is also exactly the property that is exploited in the Grover's algorithm.

3 Quantum Circuits

With the mathematical details describing qubits above, qubits can now be applied and made useful with quantum circuits. Quantum circuits are very analogous to boolean circuits that you would find in a classical computer. In classical computers, the state that is flowing through a wire either on or off can be changed using logic gates as mentioned before. In a quantum computer the qubits rely on a similar concept, the difference is that the gates are physically more complicated since they're quantum mechanical. The quantum gates are described theoretically as operators that act on the qubits wave function. In quantum mechanics it is important that these gates are unitary and hermitian such that the probability is conserved and the eigenvalues correspond to real observable's[11]. The quantum circuit is started by first initializing the states of the n qubits being used. This is known as a quantum register.

With a register defined, quantum gates can be added to the register to build a quantum circuit. The first gate to define is the hadamard gate. This gate is very commonly used in quantum algorithms and notably Grover's algorithm. The hadamard is defined by (equation 10).

$$\hat{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{10}$$

This gate takes in a single qubit wave function with two states as an operator and outputs the two states in an equal superposition of each other. For example the state $|0\rangle$ will return $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ such that upon measurement there is 50% probability that you'll get either state. This hadamard gate can be generalized to higher dimensional wave functions via the tensor product and works exactly the same for any number of states. To give an example of how this tensor product works for matrices, the hadamard gate for two qubits is demonstrated in (equation 11).

$$\hat{H}^{\otimes 2} = \hat{H} \otimes \hat{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} \hat{H} & \hat{H} \\ \hat{H} & -\hat{H} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{11}$$

Notationally the n dimensional hadamard gate is represented as $\hat{H}^{\otimes n}$. Therefore, in order to have a hadarmad gate for more qubits you just repeat the tensor product operation on the $\hat{H}^{\otimes 2}$ matrix such that $\hat{H}^{\otimes n} = \hat{H} \otimes \hat{H}^{\otimes (n-1)}$. This is useful for mathematically analyzing the quantum circuit, but equally it is the same as applying a 2d hadamard gate to each qubit in turn. (figure 1) demonstrates a quantum circuit with a register of 3 qubits with each qubit having a hadamard gate applied.

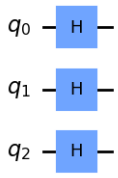

Figure 1: Quantum circuit with a register of 3 qubits all going through a hadamard gate

Where each branch denoted q_i represents a different qubit and by convention this qubit goes from left to right. In a quantum circuit with n qubits each state is denoted by $|x\rangle$ where x is a string of size n composed of 1s and 0s. The first number on the left corresponds to the bottom most qubit and following along the state the final number on the right corresponds the top most qubit. So in this example the state $|100\rangle$ would mean that q_0 is in the state $|0\rangle$, q_1 is in the state $|0\rangle$ and q_2 is in the state $|1\rangle$.

There are lots of different gates that are described as matrices and are useful for many of different operations. However, only some of these gates and their symbols are necessary for Grover's algorithm. One of these gates is the X gate

($\hat{X}$). This gate simply flips the state $|0\rangle$ to the state $|1\rangle$ and vice versa. This gate can also be represented as a matrix (equation 12).

$$\hat{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{12}$$

As before this gate can be generalized to higher dimensional inputs with the tensor product. Another really important gate is the z gate ($\hat{Z}$). The Z gate flips the state $|1\rangle$ to $-|1\rangle$ and doesn't do anything to the state $|0\rangle$. The matrix representation for this gate can be seen in (equation 13).

$$\hat{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{13}$$

Equally important for all quantum circuits is the measurement gate. As suggested by the name, the measurement gate measures the state of the qubit causing the wave function to collapse. It then maps the measured state from the qubit onto a classical bit to provide a computer output. It is perhaps the most important gate since it is used in all quantum circuits, otherwise there is no output. The circuit symbols for both the Z and X gate acting on different qubits then measured can be seen in (figure 2).

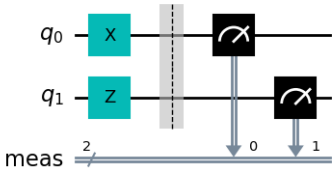

Figure 2: Quantum circuit with the qubit q_0 going through an X gate and q_1 going through the Z gate. Then both qubits are measured

As can be seen from (figure 2) the circuit must contain a classical register with 2 bits to measure the output. The classical register is represented via a double line and is initialized from the start so that the resulting output from the circuit can be stored.

Finally, the last gate necessary for implementing Grover's algorithm is the controlled Z gate ($\hat{CZ}$). This gate requires both controlled and target qubits to be implemented. It works such that if the control qubits are each in the state $|1\rangle$, $\hat{Z}$ is performed on the target qubits. However, if any of the controlled qubits are in the state $|0\rangle$, $\hat{Z}$ is not applied to the targets. As mentioned previously the $\hat{CZ}$ gate actually uses the phenomena of quantum entanglement since the

state of the target qubits are depended on the state of the controlled hence they are entangled. The circuit symbol for the $\hat{CZ}$ gate can be seen in (figure 3).

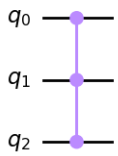

Figure 3: Quantum circuit with 3 qubits with going through the $\hat{CZ}$ gate with q_0 and q_1 as controlled qubits and q_3 is the target

4 Grover's algorithm

In this algorithm the different states of all the qubits represent the database of items to search through. From this database, the states that the Grover algorithm is searching for is selected by the user. These states are known as the 'good states'. Grover's algorithm will then try to find the good states with high probability. The algorithm is classified as a solution to an 'amplification problem' because at its most fundamental level the algorithm will amplify the good states and therefore increase the probability of finding any of these states. This algorithm is stochastic and therefore doesn't guarantee that it will find a good state with 100% probability but as will be explained it is optimized via the number of iterations such that the amplitude of the good states is maximal.

The first part of understanding Grover's algorithm is understanding the 'oracle function' $f(x)$. This function takes in as its input a quantum state $|x\rangle$ and gives an output of 1 if the state is a good state and a 0 if the state is not a good state (a bad state). This allows delineation between the set of all good and bad states. η will be used to represent all the states in the quantum computer. This can then be used to define two subsets g and b to represent the good and bad states respectively (equations 14 & 15).

$$g = \{x \in \eta : f(x) = 1\} \tag{14}$$

$$b = \{x \in \eta : f(x) = 0\} \tag{15}$$

Immediately by this definition it is implied that $g \cup b = \eta$. These two sets can

also be written as state vectors in (equations 16 & 17).

$$|g\rangle = \sum_{x \in g} |x\rangle \tag{16}$$

$$|b\rangle = \sum_{x \in b} |x\rangle \tag{17}$$

This new form of writing different states as state vectors will become very useful later. The total number of states must be given as the sum of the number of good and bad states (equation 18).

$$N = |g| + |b| \tag{18}$$

These states are also orthonormal and hence subject to the conditions in (equation 19).

$$\langle g|b\rangle = \delta_{g,b} \tag{19}$$

Where N is the total number of states. The next key component is the different gates involved in this algorithm. Firstly the 'phase query gate' $\hat{Z}_f$ which uses the oracle function to flip the sign in front of the state vectors inputed to it. This gate can be defined in (equation 20).

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

As can be seen when considering the definitions of $|g\rangle$ and $|b\rangle$ if the input state $x \in g$ then $\hat{Z}_f |x\rangle = -|x\rangle$ alternatively when $x \in b$ then $\hat{Z}_f |x\rangle = |x\rangle$. When using the amplitude definition from (equation 5), flipping the sign of the amplitude is equivalent to rotating the complex amplitude π radians through the complex plane. So this gate allows Grover's algorithm to distinguish between the good and bad states since it makes there amplitudes π out of phase with each other. Now finally the $\hat{Z}_{or}$ gate is the last gate necessary to define and can be defined in (equation 21).

$$\hat{Z}_{or} |x\rangle = \begin{cases} |x\rangle & x = 0^n \\ -|x\rangle & x \neq 0^n \end{cases} \tag{21}$$

This gate performs a phase flip on all the non-zero states and can be thought of as a reflection about the zero state. Similarly to the $\hat{Z}_f$ gate the $\hat{Z}_{or}$ gate creates a phase difference of π between the state $|0^n\rangle$ and the other the states. This gate can be written in a more convenient Dirac notation in (equation 22).

$$\hat{Z}_{or} = 2 |0^n\rangle \langle 0^n| - I \tag{22}$$

Where I is the identity matrix. These two expressions are completely equivalent and this is the form that will be used throughout the rest of the paper as it's more useful to see what it is doing. With these gates defined the Grover operator $\hat{G}$ can be constructed (equation 23).

$$\hat{G} \equiv \hat{S}^{\otimes n} \hat{Z_{or}} \hat{S}^{\otimes n\dagger} \hat{Z}_f \tag{23}$$

(Note: since the operator components are matrices this operator acts from right to left). $\hat{S}$ is known as the input state preparation. There are many different gates that can be chosen for $\hat{S}$ but for this example the most basic to chose is the hadamard $\hat{H}$. Since this hadamard gate is hermitian ($\hat{H} = \hat{H}^{\dagger}$) the Grover operator can be rewritten as (equation 24).

$$\hat{G} \equiv \hat{H}^{\otimes n} \hat{Z_{or}} \hat{H}^{\otimes n} \hat{Z}_f \tag{24}$$

With these mathematical definitions Grover's algorithm can be explained in 4 steps.

1. Initialize the N qubit register Q into the state $|0^n\rangle$.
2. Apply a hadamard gate to each qubit in Q to create an equal superposition of all states.
3. Apply the operator $\hat{G}$ (equation 24) τ times to the register Q
4. Measure the qubits of Q and output the resulting string

Using the steps above, the total wave function of the qubit register will be denoted $|\psi\rangle$. The wave-function is set to be in the initial state in step 1 such that $|\psi\rangle = |0^n\rangle$. All qubits are then sent through the hadamard gate in step 2 this can be seen in (equation 25).

$$|\psi\rangle = \hat{H}^{\otimes n} |0^n\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \eta} |x\rangle \tag{25}$$

Where N is the total number of states $|\eta| = N = 2^n$ since n is the number of qubits as previously shown. Continuing with this equation and using the previous definition of good and bad state vectors (equations 16 & 17). $|\psi\rangle$ can be rewritten in (equation 26).

$$|\psi\rangle = \sqrt{\frac{|g|}{N}} |g\rangle + \sqrt{\frac{|b|}{N}} |b\rangle \tag{26}$$

This expression the wave function $|\psi\rangle$ gives all the information one can know about the system after step 2 without needing to specify $|b|$ or $|g|$. Also, each state within the state set η is equally probable for now.

Now it is time to send the states through the Grover operator. This is done by applying each operator within $\hat{G}$ to the wave function $|\psi\rangle$ chronologically. However, to give more detail on how Grover's algorithm works it's more helpful to apply $\hat{G}$ to $|g\rangle$ and $|b\rangle$ to see what happens. Firstly these states are sent through the phase query gate $\hat{Z}_f$ defined in (equation 20). The result is shown in (equation 27).

$$\begin{aligned} \hat{Z}_f |g\rangle &= -|g\rangle \\ \hat{Z}_f |b\rangle &= |b\rangle \end{aligned} \tag{27}$$

This is an unsurprising result and just gives the algorithm room to differentiate between the good and bad states. Now focusing on the rest of the Grover operator $\hat{H}^{\otimes n}\hat{Z_{or}}\hat{H}^{\otimes n}$. This is commonly referred to as the diffusion operator $\hat{D}$. By using the expressions in (equations 22 and 25) to further simplify the expression, recalling orthonormality (equation 19) and that $\hat{H}$ is unitary and hermitian. $\hat{D}$ can be expressed in (equation 28).

$$\hat{D} \equiv \hat{H}^{\otimes n}\hat{Z_{or}}\hat{H}^{\otimes n} = 2\left|\psi\right\rangle\left\langle\psi\right| - I \tag{28}$$

This makes the whole expression much simpler and therefore the Grover operator can be rewritten in (equation 29).

$$\hat{G} = \hat{D}\hat{Z}_f = (2\left|\psi\right\rangle\left\langle\psi\right| - I)\hat{Z}_f \tag{29}$$

With this final expression for $\hat{G}$ it's very easy to see through algebra what happens to $|g\rangle$ and $|b\rangle$. Following the algebra and making the right substitutions along the way using (equations 16 & 17), the final expression of the Grover operator acting on the good and bad states can be seen in (equation 30).

$$\begin{aligned}\hat{G}\left|g\right\rangle &= \frac{|b|-|g|}{N}\left|g\right\rangle - 2\frac{\sqrt{|g||b|}}{N}\left|b\right\rangle \\ \hat{G}\left|b\right\rangle &= 2\frac{\sqrt{|g||b|}}{N}\left|g\right\rangle + \frac{|b|-|g|}{N}\left|b\right\rangle\end{aligned} \tag{30}$$

This is a really important result and allows me to write $\hat{G}$ in matrix form (equation 31).

$$\hat{G} = \begin{pmatrix} \frac{|b|-|g|}{N} & -2\frac{\sqrt{|g||b|}}{N} \\ 2\frac{\sqrt{|g||b|}}{N} & \frac{|b|-|g|}{N} \end{pmatrix} \tag{31}$$

Following on from this definition upon analysis this matrix can be written in the same form as a 2d rotation matrix squared (equation 32).

$$\hat{G} = \begin{pmatrix} \sqrt{\frac{|b|}{N}} & -\sqrt{\frac{|g|}{N}} \\ \sqrt{\frac{|g|}{N}} & \sqrt{\frac{|b|}{N}} \end{pmatrix}^2 = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}^2 \tag{32}$$

Where θ is given by (equation 33).

$$\theta = \sin^{-1}\sqrt{\frac{|g|}{N}} = \cos^{-1}\sqrt{\frac{|b|}{N}} \tag{33}$$

This leads to the new way of writing the wave function from (step 2) as a vector in terms of trigonometric functions where the first row represents the amplitude of $|g\rangle$ and the second represents the amplitude of $|b\rangle$ (equation 34).

$$\left|\psi\right\rangle = \sin\theta\left|g\right\rangle + \cos\theta\left|b\right\rangle = \begin{pmatrix}\cos\theta \\ \sin\theta\end{pmatrix} \tag{34}$$

This picture gives a completely new way of visualizing the algorithm. Applying the Grover operator can simply be thought of as rotating the amplitude of the good states around the complex plane. This new expression will be used to calculate the optimal number of iterations for the Grover operator (τ) in step 3. Luckily, (equation 32) makes that a relatively straightforward process. Applying a rotation matrix squared τ times is equivalent to a single rotation matrix of $2\tau\theta$ and therefore can be written in (equation 35).

$$\hat{G}^{\tau} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}^{2\tau} = \begin{pmatrix} \cos(2\tau\theta) & -\sin(2\tau\theta) \\ \sin(2\tau\theta) & \cos(2\tau\theta) \end{pmatrix} \tag{35}$$

Therefore computing the result of step 3 can be done by using the operator in (equation 35), acting it upon the new expression for $|\psi\rangle$ in (equation 34). Using the appropriate trigonometric identities this leads to the expression given in (equation 36).

$$\hat{G}^{\tau}|\psi\rangle = \cos(\theta(2\tau+1))|b\rangle + \sin(\theta(2\tau+1))|g\rangle \tag{36}$$

This expression is used figure out how big τ must be in order for Grover's algorithm to be effective. Since the probability of measuring a state in this wave function is given by the square of the amplitude it would be ideal to make the amplitude of the good state $|g\rangle$ as large as possible while inversely making the amplitude of $|b\rangle$ as small as possible. Luckily from basic inspection it is clear that this condition would be exactly satisfied if $(2\tau+1) = \frac{\pi}{2}$. This would result in a zero probability of measuring $|b\rangle$ and a 100% probability of measuring $|g\rangle$. Hence, solving for τ gives an expression for the optimal number of iterations for the Grover operator (equation 37).

$$\tau = \frac{\pi}{4\theta} - \frac{1}{2} \approx \lfloor \frac{\pi}{4\theta} \rfloor \tag{37}$$

τ must be an integer, since it is the number of times to apply the Grover operator. Hence, τ is approximated in the above expression with the floor function. This means τ will always be rounded down to the nearest integer. This is ideal as it minimizes the number of steps while optimizing the success rate. For this project $|g| = 1$ always, since the unique search problem looks only for one item from the unstructured database. Using (equation 37), (equation 33) and the fact that $\sin^{-1}(\beta) \geq \beta$, the upper bound for τ can be calculated as (equation 38).

$$\tau \leq \frac{\pi}{4}\sqrt{N} \tag{38}$$

From this expression it is clear that Grover's algorithm has a computational complexity of $O(\sqrt{N})$ as mentioned before. This is why in essence Grover's algorithm is theoretically superior to its classical counterpart. Following from this, an expression for the probability of success is given by the amplitude squared of $|g\rangle$ from (equation 36). This expression can be seen in (equation 39).

$$P(succsess) = \langle g|\hat{G}^{\tau}|\phi\rangle = \sin^2(\theta(2\tau+1)) \tag{39}$$

Then since $N = 2^n$ where n is the number of qubits, the success rate can be investigated by plotting it as a function of n. This will demonstrate how the number of qubits effects the chance of success for the unique search problem $|g| = 1$ (figure 4).

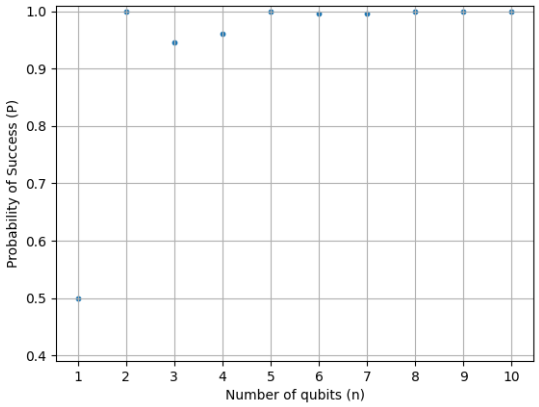

Figure 4: $P(success)$ against the number of qubits

As can be seen from this figure the probability of success is 50% for 1 qubit and 100% for 2 qubits. It drops off after this point for 3 and 4 but after that the probability of success gets extremely close to 100% and stays within this range as the number of qubits are scaled up. This demonstrates how Grover's algorithm gets more accurate for larger databases. For the code behind (figure 4) see appendix A.

5 Implementing Grover's algorithm

Taking a step back from the very theoretical side of this algorithm, implementing it is quite different. This is because the previously defined gates can't just simply be imported through a Qiskit package since they don't physically exist as gates within quantum computers with the exception of the hadamard gate. So these gates instead have to be constructed by a series of other gates such as the $\hat{Z}$, $\hat{CZ}$ and the $\hat{X}$ gate as mentioned before.

The first step of implementing the Grover algorithm is implementing the oracle circuit which in this case is done by building the $\hat{Z}_f$ gate with a system of $\hat{X}$ and $\hat{Z}$ gates. This oracle function is designed to adapt for the different state you search for as it would be different depending on your choice of good state. The code involves defining a function such that you input your chosen good state as a string e.g. '0001' and then returns the bespoke $\hat{Z}_f$ gate as a

quantum circuit. This works by first counting the number of digits in the given number so that it can start a quantum circuit with the appropriate number of qubits. Then an $\hat{X}$ gate is applied for each qubit in the good state that is 0 by following the previously mentioned ordering convention. This essentially converts the good state $|x\rangle$ into the state $|1^n\rangle$. Then all the qubits are sent through the multi controlled $\hat{CZ}$ gate which causes a sign flip only on the state $|1^n\rangle$. Finally, the $\hat{X}$ gate is reapplied to the qubits that where previously sent though the $\hat{X}$ gate causing the $|1^n\rangle$ state to return to flip back to $|x\rangle$ with a negative amplitude. Hence, feeding the input wave function $\hat{H}^{\otimes n}|0^n\rangle$ into the circuit returns the same wave function but with the good state having a '-' in front. As an example the oracle function for the state $|011\rangle$ can be seen in (figure 5).

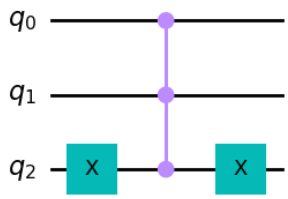

Figure 5: $\hat{Z}_f$ gate as a quantum circuit for the state $|011\rangle$

Following from this the next step is to construct the diffusion operator $\hat{D}$ this can be constructed via a series of $\hat{H}$, $\hat{X}$ and $\hat{CZ}$ gates. From the definition of the diffusion operator in (equation 28), the hadamard gate is first applied to each qubit. Then the goal is to perform a phase flip on every state except the state $|0^n\rangle$. However, as was discussed before, the point of this operator is to make the state $|0^n\rangle$ π out of phase with every other state. Since the overall effect on the probabilities is the same. So instead the circuit will flip the state $|0^n\rangle$ to $-|0^n\rangle$ while keeping the rest the same. This way the phase difference between the states is still π and this is also easier to implement. It can be done by first applying $\hat{X}$ gates to all qubits such that the state $|0^n\rangle$ becomes the state $|1^n\rangle$. Then by applying a $\hat{CZ}$ gate to the whole circuit only the state $|1^n\rangle$ flips sign while the rest remain unchanged. Then by applying the $\hat{X}$ gates again to all qubits the state $-|1^n\rangle$ becomes $-|0^n\rangle$ as was intended while the rest remain unchanged. Following from this, adding this diffusion operator to the oracle creates the total Grover operator for the state $|011\rangle$ which can be seen in (figure 6).

This diffusion operator circuit is then condensed down into a gate in qiskit.

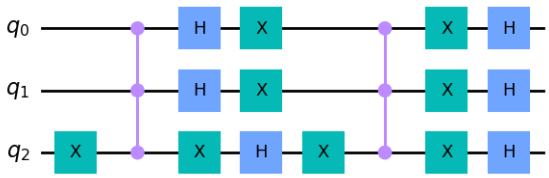

Figure 6: Grover operator as a quantum circuit for the good state $|011\rangle$

Then a new function is defined that hadamards the initial qubits, calculates τ from(equation 37), repeats the Grover operator τ times and measures the result. This final function completes the Grover algorithm and only takes the good state string as its input. The Grover algorithm for this good state $|011\rangle$ can be seen in (figure 7).

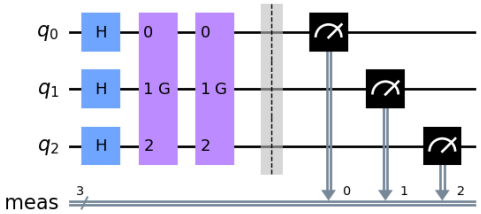

Figure 7: Quantum circuit of Grover's algorithm for the good state $|011\rangle$

For the code behind how this circuit was built see appendix B. This is circuit now ready to be deployed on a quantum computer.

6 Results

To check the effectiveness of this algorithm it will be run this with 3 qubits (6 possible states), looking for the state $|011\rangle$. Firstly the algorithm was run on a quantum computer simulator locally with a classical computer. This simulated an ideal quantum computer that is completely fault tolerant. The simulation was run 10^4 times and measured. The measurement results from the 10^4 runs (shots) was then used to create a bar chart of pseudo probabilities (figure 8). The code for (figure 8) can be found in appendix C.

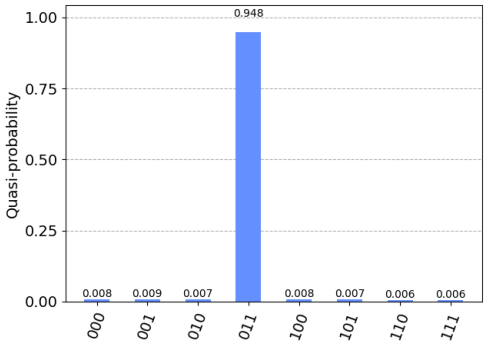

Figure 8: Locally simulated Grover's algorithm results for good state = $|011\rangle$, 10^4 shots results.

This circuit was then run on a real IBM quantum computer. For this implementation when running on a real quantum computer you have the choice of several parameters, notably the optimization parameter. This parameter is given a number from 0 to 3 to decide how optimized you want your algorithm to be using error correcting tools. For this first example the optimization set to 0 such that there is no optimization algorithms being used. The results of this can be seen in (figure 9).

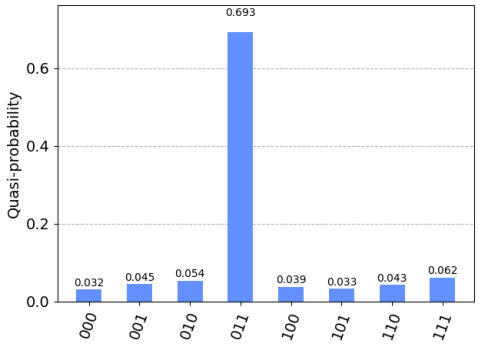

Figure 9: Real quantum computer Grover algorithm good state = $|011\rangle$, shots = 10^4 and optimization = 0 results

This same circuit was then run again with the highest level of optimization (optimization = 3) for contrast. The results for this can be seen in (figure 10). The code for (figures 9 & 10) can be found in appendix D

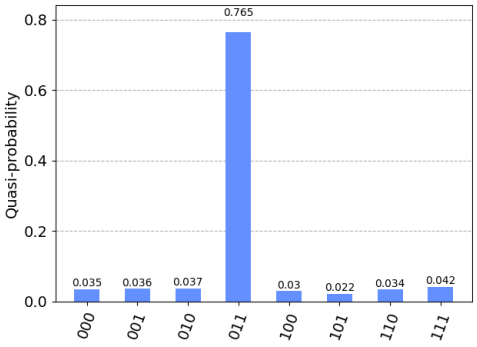

Figure 10: Real quantum computer Grover algorithm good state = $|011\rangle$, shots = 10^4 and optimization = 3 results.

Finally, this same code was used to run Grover's algorithm for the good state $|00010101\rangle$. Since this state requires 8 qubits, the algorithm is searching a database with $2^8 = 256$ states. This was done with 100 shots since qiskit only allows 10 minutes of free use per month. The theoretical expecteded probability for 8 of qubits is approximately 100% success. However, when this was implemented on the quantum computer for the non optimized version the good state was never measured. The state with the highest probability was found to be $|10011011\rangle$ with a probability of 5%. For the optimized version (optimization = 3, shots = 100) the good state was also never measured.

7 Analysis

As can be seen from (figure 8), the expected probability lines up exactly with the predicted value in (figure 4). Since quantum mechanics is inherently stochastic running this simulation multiple times doesn't necessarily yield the same graphs. However, since this circuit used 10^4 shots, the figure approximately gives the true probabilities of each state. This is due to the law of large numbers[15]. This graph will now be used as a benchmark for running on a real quantum computer since this is a perfect simulation.

For the first iteration in (figure 9) with no optimization it can be seen that the algorithm produces the correct output 69% of the time. When this is compared to the ideal quantum computer (figure 8) which is accurate 95% of the time, it means that the real quantum computer has a relative accuracy of 73%. This is a fairly impressive result since there was no optimization running in the background. The quantum computer also took approximately 28.46 seconds to run Grover's algorithm 10^4 times. This means that it took 2.8×10^{-3} seconds to run the Grover algorithm.

To better understand the effectiveness of the quantum computer this same problem was compared to a classical search algorithm, again using python. Since Grover's algorithm is for an unstructured database search, the best classical algorithm for that is the linear search. The code for this can be found in appendix E. The result was that even in the worst case, the classical algorithm could find the state in the order of 10^{-5} seconds. This is clearly dramatically faster than Grover's algorithm, although this could be due to many other factors. For example since IBM's quantum computer is cloud based it can be assumed that there is possibly time lost and therefore efficiency lost when being sent through the internet. There is also the issue of compiling the python code into code readable by the quantum computer which will add to the time taken. Comparatively, the classical algorithm is all computed locally, doesn't require internet to work and therefore can be done much faster.

For (figure 10) the quantum computer used the maximum optimization setting. This resulted in an accuracy of 77% and therefore had a relative accuracy of 81% compared to the perfect simulation. This is definitely an improvement from (figure 9), however the optimized version had a runtime of approximately 36.96 seconds. Hence, each execution of Grover's algorithm took on average 3.7×10^{-3} seconds which is almost 1ms longer than the non-optimized version. So while the optimized algorithm has better accuracy than the non-optimized version, it takes significantly longer and hence it's difficult to say whether it is an improvement.

Finally, when the quantum computer used 8 qubits to run Grover's algorithm, the good state was never detected in both cases. From (figure 4) it's clear that these results should theoretically have a higher probability of success compared to the results in (figures 8,9 & 10). The 8 qubit tests only used 100 shots as opposed to 10^4 which could explain this difference. However, since the probability of success is extremely close to 100%, it's statistically unlikely that the computer never measured the good state in both cases. Therefore, not measuring the good state is likely due to errors within the quantum computer. A potential reason for this is that from (equation 37) it's clear that 8 qubits require 12 Grover operations. In comparison, the 3 qubit examples only used 2 Grover operations. This means the quantum computer had to run longer and do a lot more legwork than before. This leads much more room for qubit errors to occur and is almost certainly the reason this experiment didn't produce the results expected.

8 Conclusion

From this paper it is clear that in order for Grover's algorithm to outperform a classical search algorithm there are still many technical challenges in the way. The results from comparing to classical algorithms are quite disappointing. For

Grover's algorithm to completely outperform classical search algorithms it would need to have a much faster run time, roughly 100× faster. Granted as before, the time taken could be due to other factors and theoretically Grover's algorithm becomes a faster alternative given a big enough database. It is clear that runtime is definitely not the only problem. When the algorithm was run using 8 qubits it didn't work at all due to qubit errors. These qubit errors could be due to decoherence as well as ionising radiation and cosmic rays[16]. Hence, the superconducting qubits in IBM's quantum computer need to be much more fault-tolerant to be a viable alternative to classical computers. This means new technologies behind quantum hardware will need to be explored. Alternatively there is the method of error correction but as was demonstrated in this paper the use of error correction for 3 qubits increased the success probability but decreased the efficiency. Furthermore, in the 8 qubit Grover search, the error correcting algorithms did not make the results any better. It is therefore unlikely that error correction is the only thing needed to get quantum computing off the ground to solve problems that classical computers can't.

More optimistically, there are many new methods of creating qubits that are currently being researched. Notably, there is the method of using optical tweezers to trap neutral atoms[17]. Since these atoms are electrically neutral, they are unlikely to interfere with each other and therefore should not be prone to errors. This system is also easily scalable to make more qubits. Since this technology is very new, it is hard to say whether it will fix the problems listed in this paper, but it is still an exciting new method. Theoretically, if the hardware for quantum computers can be engineered to be fault-tolerant then many quantum algorithms including Grover's will be an efficient alternative. With all the research that is currently underway, quantum computing solutions may still have promise in the not so distant future.

References

[1] E. Rieffel and W. Polak, "An introduction to quantum computing for non-physicists," *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 300–335, 2000.

[2] M. Bogobowicz, S. Gao, M. Masiowski, N. Mohr, H. Soller, R. Zemmel, and M. Zesko, "Quantum technology sees record investments, progress on talent gap." `https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/quantum-technology-sees-record-investments-progress-on-talent-gap`, April 2023.

[3] R. White and T. E. Downs, *How Computers Work*. Que Indianapolis, 1998.

[4] T. Hey, "Quantum computing: an introduction," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 105–112, 1999.

[5] J. Dargan, "What types of quantum computers exist in 2024?," June 2023.

[6] I. L. Chuang, R. Laflamme, P. W. Shor, and W. H. Zurek, "Quantum computers, factoring, and decoherence," *Science*, vol. 270, no. 5242, pp. 1633–1635, 1995.

[7] I. Siddiqi, "Engineering high-coherence superconducting qubits," *Nature Reviews Materials*, vol. 6, no. 10, pp. 875–891, 2021.

[8] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

[9] O. P. Contributors, "Opendsa data structures and algorithms modules collection." `https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/BinarySearch.html`, October 2023.

[10] T. Magee, "What is qiskit, ibm's open source quantum computing framework," *Computer World UK*, August 2018.

[11] D. J. Griffiths and D. F. Schroeter, *Introduction to Quantum Mechanics*. Cambridge ; New York, NY: Cambridge University Press, 3rd ed., 2018.

[12] M. Born, "Quantenmechanik der stoßvorgänge," *Zeitschrift für Physik*, vol. 38, no. 11, pp. 803–827, 1926.

[13] D. Cheng, H. Qi, and Y. Zhao, *An Introduction to Semi-tensor Product of Matrices and it's Applications*. World Scientific, 2012.

[14] E. Schrödinger, "Discussion of probability relations between separated systems," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 31, pp. 555–563, Cambridge University Press, 1935.

[15] R. Routledge, "Law of large numbers," *Britannica*, March 2024.

[16] L. Xue-Gang, W. Jun-Hua, J. Yao-Yao, X. Guang-Ming, C. Xiao-Xia, Z. Jun, G. Ming, L. Zhao-Feng, Z. Shuang-Yu, M. Deng-Ke, C. Mo, S. Wei-Jie, Y. Shuang, Y. Fei, J. Yi-Rong, D. Xue-Feng, and Y. Hai-Feng, "Direct evidence for cosmic-ray-induced correlated errors in superconducting qubit array," 2024.

[17] L. Pause, L. Sturm, M. Mittenbühler, S. Amann, T. Preuschoff, D. Schäffner, M. Schlosser, and G. Birkl, "Supercharged two-dimensional tweezer array with more than 1000 atomic qubits," *Optica*, vol. 11, p. 222, Febuary 2024.

Appendix

(Python version = 3.12.1, Qikskit version = 0.46.0)

A Code for Figure 4

```
import numpy as np
import matplotlib.pyplot as plt

n_min = 1
n_max = 10
n_step = 1
p_min = 0.4
p_max = 1
p_step = 0.1
#n is a list of different numbers of qubits
n = np.arange(n_min, n_max + n_step, n_step)
N = 2**n
theta = np.arcsin(np.sqrt(1 / N)) #calculatting theta
t = np.floor((np.pi / (4 * theta))) #calculating tau
P = np.sin(theta * (2*t + 1))**2    #calculating the probability
plt.scatter(n,P)
plt.xlabel('Number of qubits (n)')
plt.xticks(np.arange(n_min, n_max + n_step, n_step))
plt.ylabel('Probability of Success (P)')
plt.yticks(np.arange(p_min, p_max + p_step, p_step))
plt.ylim(p_min - 0.01, p_max + 0.01)
plt.grid()
plt.savefig('../../Project/Images/nvsp.png')
plt.show()
```

B Code for Building Grover Circuit

```
#Importing all the nessacary packages
from qiskit import QuantumCircuit
from qiskit.circuit.library import MCMT, ZGate , XGate
import numpy as np
import math

#defining the oracle circuit from the input 'good_state'
def oracle(good_state):

    good_state = list(good_state) #turn the string input into a list
    n = len(good_state) #find the number of qubits for the circuit

    #reversing the list so that it matches qiskit indexing
    good_state.reverse()
```

```
    #initialising the quantum circuit with n qubits
    oracle = QuantumCircuit(n)

    '''adds an x gate for every qubit that
    should be zero for the good state '''
    for i in range(n):
        if good_state[i] == '0':
            oracle.x(i)

    '''applys a controlled z gate to q_{n-1} as the
    target and the rest as controls '''
    oracle.compose(MCMT(ZGate(), n - 1, 1), inplace=True)

    '''adds an x gate for every qubit that
    should be zero for the good state '''
    for i in range(n):
        if good_state[i] == '0':
            oracle.x(i)
    return oracle #returns the oracle as a quantum circuit

def Grover_op(good_state):
    G = oracle(good_state) #renames the oracle
    n = len(list(good_state))

    #creates a list of all qubits for indexing
    list_of_qubits = np.arange(0,n,1).tolist()

    G.h(list_of_qubits) #hadamards the whole circuit
    G.x(list_of_qubits) #applys an x gate to all qubits

    '''applys a controlled z gate to q_{n-1} as the
    target and the rest as controls '''
    G.compose(MCMT(ZGate(), n - 1, 1), inplace=True)

    G.x(list_of_qubits) #applys xgate to all qubits
    G.h(list_of_qubits) #hadamards the whole circuit
    return G #returns the grover operator as a circuit

def Grover(good_state):
    G = oracle(good_state) #renames the oracle
    n = len(list(good_state))

    #create a list of all qubits for indexing
    list_of_qubits = np.arange(0,n,1).tolist()

    qc = QuantumCircuit(n) #create a new quantum circuit qc

    #turn the grover operator into a gate
    G = Grover_op(good_state).to_gate(label='G')
```

```
    theta = math.asin(math.sqrt(1 / 2**n)) #calculate theta
    t = math.floor( (math.pi / (4 * theta))) #calculate tau
    qc.h(list_of_qubits) #hadamard the whole circuit
    for i in range(t): #apply the grover operator tau times
        qc.append(G, list_of_qubits)
    qc.measure_all() #measure all the qubits
    return qc #return the grover circuit as a quantum circuit
```

C Code for Running Grover Circuit Locally

```
#Importing all the nessacary packages
from qiskit.visualization import plot_distribution
from qiskit_ibm_runtime import QiskitRuntimeService
from qiskit.primitives import BackendSampler

good_state ='011' #define the good state

service = QiskitRuntimeService() #choosing a service

#chosing a backend simulator
backend = service.get_backend('ibmq_qasm_simulator')

#setting up the sampler
sampler = BackendSampler(backend)

'''defining the job and runing it with
10000 shots '''
job = sampler.run(Grover(good_state), shots = 10000)

result = job.result() #save the results from the job

#retrieve the quasi probabilities
quasi_dist = result.quasi_dists[0]

#plotting and saving the figure
plot_distribution(quasi_dist.binary_probabilities(),
filename='../../Project/Images/Simulatedhist.png')
```

D Code for Running Grover Circuit on Real Quantum Computer

```
#Importing all the nessacary packages
from qiskit.visualization import plot_distribution
from qiskit_ibm_runtime import QiskitRuntimeService
```

```
good_state ='011' #define the good state
n = len(list(good_state)) #find the number of qubits for the circuit

'''setting optimization level and this
was changed when nessacary'''
optimization_level = 0 #0,1,2,3

service = QiskitRuntimeService() #chosing a service

#retreive the least busy backend
backend = service.least_busy(operational=True, min_num_qubits=n+10,
simulator=False)

#generate a pass manager
pm = generate_preset_pass_manager(optimization_level =
optimization_level, backend=backend)

#set up the circuit to be deployed
isa_circuit = pm.run(Grover(good_state))

#setting up sampler
sampler = Sampler(backend=backend)

'''defining the job and runing it with
10000 shots '''
job = sampler.run(isa_circuit, shots = 10000)

result = job.result() #retreive results
quasi_dist = result.quasi_dists[0] #get quasi probabilities
time = job.usage_estimation #calculate the time taken
print(time) #output the time taken

#plotting and saving the figure
plot_distribution(quasi_dist.binary_probabilities(),
filename=f'../../Project/Images/Qc{optimization_level}.png')
```

E Code for Linear Search

```
#importing packages
import time
import random

'''defining the linear search
function that scans through each
state from a list and returns True once
the target value is found'''
def lsearch(states, good_state):
    for i in range(len(states)):
```

```
        if states[i] == good_state:
            return True

#defining the list of states
states  = ['111', '001', '010', '011', '100','101','110','111']

'''randomonly shuffle the list so
it is an unstructued search'''
random.shuffle(states)

t0 = time.time()#record the time at start
good_state  = '011' #define the good state

'''search for the good state
from the list of states'''
lsearch(states,good_state)

t1 = time.time()#record the time after calculation
print(f'the state {good_state} was found')
t = t1-t0 #calculate the total time
print(f'completed in {t} seconds') #time output
```