Bernstein-Vazirani Algorithm

Azka Rafey Khan | Bismah Rizwan | Fariha Hassan
Roll numbers: 23100077 | 23100029 | 23100071

PHY 318: Final Report

Sunday, April, 30, 2023

Contents

1 Abstract

This report aims to explore and build the intuition behind the Bernstein-Vazirani algorithm, a quantum algorithm designed to efficiently determine an unknown bit string hidden in an oracle. It has a relatively simple mathematical structure and circuit implementation, which makes it an ideal candidate for practical applications in quantum computing. In this report, we discuss the algorithm's mathematical structure, focusing on the quantum properties that allow it a significant advantage over the classical algorithm. We show both the classical algorithm and explain in detail the working of quantum algorithm, including the logic behind the creation of the circuit.

2 Introduction

Developed in 1992 by Ethan Bernstein and Umesh Vazirani, the BV algorithm solves a problem known as the hidden shift problem, which has important applications in cryptography and error-correcting codes. In the hidden shift problem, we are given a function $f(x)$ that is guaranteed to have some hidden shift a, meaning that some unknown bit string a exists such that $f(x) = f(a \oplus x)$ for all inputs x. The goal of the hidden shift problem is to find the hidden shift a.

The BV algorithm uses the phase kickback technique (discussed in the next section) to determine the n-bit hidden shift (or the hidden string) on a quantum computer using a black box function. In other words, given a function $f(x)$ that takes an n-bit string x as input and returns a single bit, the algorithm aims to find the unknown n-bit string a that defines the function as

$$f(x) = a \cdot x \ (\text{mod} 2).$$

The intuition behind the Bernstein-Vazirani algorithm is to exploit the superposition and entanglement properties of quantum mechanics to efficiently determine the unknown string. The algorithm uses a quantum oracle that applies the function $f(x)$ to a superposition of all possible inputs in parallel, and the resulting quantum state carries information about a. By measuring the quantum state, the algorithm can extract the bits of a one by one with high probability.

3 Bernstein-Vazirani Algorithm

3.1 Motivation

Our approach towards the mathematical formulation of the BV algorithm assumes the reader's familiarity with basic quantum algorithms. That being said, it is not the intention to take a shorter route towards diving into the mathematics of

this algorithm, but like all things rational, this must be approached systematically as well. Hence, it is assumed that the reader will be acquainted with how the Deutsch-Josza algorithm works on a fundamental level.

The Deutsch-Josza (DJ) algorithm ensures that a problem can be solved with certainty in polynomial time; the intuition behind the DJ algorithm is that for a balanced function, we know that half of the inputs output zero and the other half output one. This is analogous to having two buckets, one with zeroes and the other with ones. If we query the oracle on a random set of points, that is, if we choose the inputs randomly, the probability of inputs coming from the same bucket reduces exponentially. Classical computers can also solve the DJ algorithm in polynomial time if a negligible probability of wrong answers is allowed. However, in the BV algorithm, we define our problem in a way that quantum computers can solve with a high probability but classical computers cannot solve it with a probability greater than half within the same amount of time.

3.2 Classical Solution

At this point, the reader has already been exposed to the oracle function in the introduction. An n-bit input provides the output zero or one. Our guarantee in the Deutsch-Josza function was that it would either be a balanced function or a biased one; here, our function is guaranteed to be of a type $a \cdot x$. This essentially means that we are given a string embedded in the oracle function, such that for any input given to the oracle function, the output is $a \cdot x \ (\mathrm{mod} 2)$. The oracle will output the n-bit string a embedded in the function for any input.

Classically, we can perform n possible queries on the oracle where the inputs are in the sequence of $100\ldots0$, $010\ldots0$, all the way until $000\ldots1$ (all possible n-bit strings). When these inputs are fed into the function $f(x) = a \cdot x \ (\mathrm{mod} 2)$, one bit of the string a is revealed with each query. The first query will give the first bit, the second query will give the second bit, and so on for n bits, after which the string is revealed.

Let us consider an example of this. We assume that the secret string embedded in the function, a, is 1101. The oracle is going to be queried with $n = 4$ inputs then; the 1 in each input will move a place ahead, the bit-wise dot will determine what lies at each position,

$$1000 \cdot 1101 = 1 \tag{1}$$
$$0100 \cdot 1101 = 1 \tag{2}$$
$$0010 \cdot 1101 = 0 \tag{3}$$
$$0001 \cdot 1101 = 1, \tag{4}$$

revealing the full string, a, to be 1011. Classically, we cannot do this in less than n calls. We can prove this lower bound. Assuming that we can find a solution by performing $n - 1$ queries, we can represent our oracle function using a system of $n - 1$ linear equations,

$$f(x_1) = x_1a_1 + x_1a_2 + \ldots + x_1a_n \tag{5}$$
$$f(x_2) = x_2a_1 + x_2a_2 + \ldots + x_2a_n \tag{6}$$
$$\vdots$$
$$f(x_{n-1}) = x_{n-1}a_1 + x_{n-1}a_2 + \ldots + x_{n-1}a_n. \tag{7}$$

Since the system has $n-1$ equations and n variables, it is underdetermined, which means that there are less equations than there are variables. There is no unique solution to this system then. Therefore, we would need n queries as the minimum lower bound.

However, using a method rooted in quantum mechanics can help us solve this problem with only one query to the function, and this is what we will explore next.

3.3 Quantum Solution

This section aims to give the reader an intuitive understanding of the circuit that solves the Bernstein-Vazirani problem. This circuit uses two important properties: quantum superposition and phase kickback. To use quantum superposition, we use Hadamard gates to generate multiple states that can query the oracle. We also add an ancillary qubit to the circuit so that for a n-bit query string we would require $n+1$ qubits. The interaction between the query and ancillary bits is crucial to the algorithm. Phase kickback is used in such a way that while a controlled phase gate may be applied in such a way that the control is at the top qubit, the phase change also effects the top qubit. This can be seen in the following example [1].

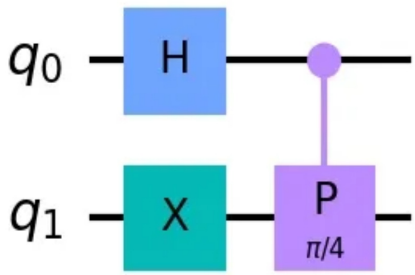
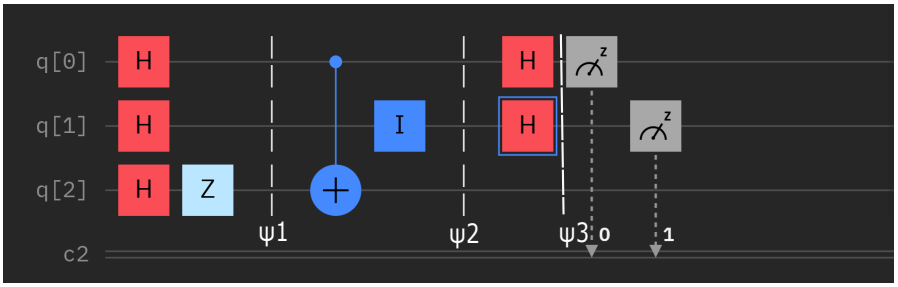


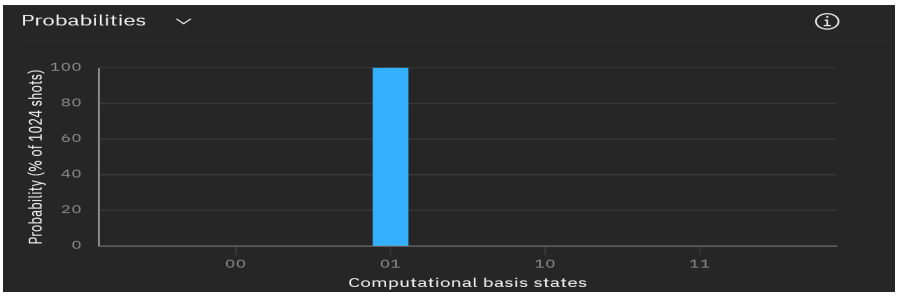
Figure 1: A schematic to explain phase kickback.

Consider Figure (1). Assuming that the initial input is 0 for both qubits, after the action of the Hadamard on the first qubit we get,

$$|00\rangle = (\frac{|0\rangle + |1\rangle}{\sqrt{2}}) |0\rangle , \tag{8}$$



(a) Main Circuit representation of the algorithm for a 2-bit query string.



(b) We can see that upon measurement we get the state $|01\rangle$

Figure 2: Implementation of the circuit in IBM Quantum Composer.

Then applying the Pauli-X gate leads us to,

$$\frac{|01\rangle + |11\rangle}{\sqrt{2}}, \tag{9}$$

Finally the application of the controlled phase gate leads to,

$$\frac{|01\rangle + e^{\iota\frac{\pi}{4}|11\rangle}}{\sqrt{2}} \tag{10}$$

$$= (\frac{|0\rangle + e^{\iota\frac{\pi}{4}}|1\rangle}{\sqrt{2}})\,|1\rangle\,. \tag{11}$$

As we can see the phase only effects the first qubit, even though that is where the control is. A similar concept is used for Bernstein-Vazirani. In this case we prepare the ancillary qubit in the state $|-\rangle$ and use controlled not gates to apply a phase of -1.

As can be seen in Figure (2a), q[2], which is the ancillary qubit, is first converted into $|+\rangle$ using the Hadamard gate and then converted to $|-\rangle$ using the Z-gate which applies a negative phase to $|1\rangle$. After this, we apply the oracle. This oracle aims to convert the input into the query string. It does so using the phase kickback as described above. So in Figure (2a), the query is 01, as can be seen from the final probability in Figure (2b). q[0] and q[1] start as $|0\rangle$ and the application of the Hadamard lead them to become $|+\rangle$ then the controlled not gate on q[0] turns it into $|-\rangle$, due to phase kickback. Identity makes no change to q[1]. When the Hadamard gate is applied again $|-\rangle$ in q[0] turns into $|1\rangle$ and $|+\rangle$ in q[1] turns into $|0\rangle$ giving us the string 01 as desired.

3.4 Mathematical Formulation

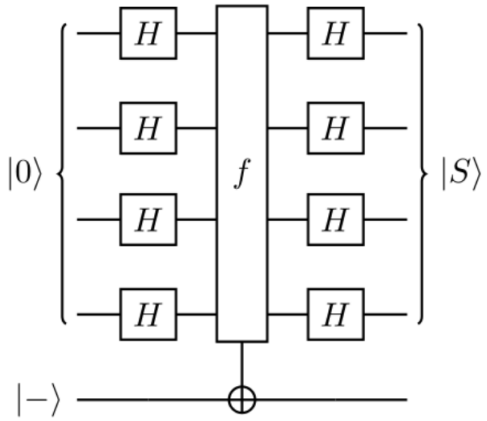


Figure 3: A schematic to ease the visualisation of how the quantum solution is approached [2].

Our approach to the quantum solution is summarised in Figure 3. We will begin by initializing all input qubits in the $|0\rangle$ state and the ancillary output qubit to $|-\rangle$. Like all quantum algorithms, we then surrender our classical knowledge by passing our n qubit state $|a\rangle$ through a series of Hadamard gates that act on all of the qubits. Multiple Hadamard gates are needed since a Hadamard gate is, by definition, a single-qubit gate. To evaluate the action of n Hadamard gates on n qubits, let us first generalize the definition of the Hadamard gate without having to write its action on $|0\rangle$ and $|1\rangle$ separately. The action of the Hadamard gate on $|0\rangle$ and $|1\rangle$ is defined as,

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{12}$$

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

which is just the case when n = 1. In order to generalize the action of the Hadamard gate for n = 1, that is, to write it as acting on some arbitrary one-qubit state $|a\rangle$, we make some observations. Firstly, a could either be 0 or 1 here. The coefficient of $|0\rangle$ in Equations (12) and (13) is always 1, while that of $|1\rangle$ is either 1 or -1. Note that Equation (12) can be expressed as,

$$H|0\rangle = \frac{1}{\sqrt{2}}((-1)^{0\cdot 0}|0\rangle + (-1)^{0\cdot 1}|1\rangle), \tag{14}$$

where the dot product is between the number inside the state on which the Hadamard gate is acting and that inside the ket corresponding to each coefficient. Similarly, we can write Equation (13) as,

$$H|1\rangle = \frac{1}{\sqrt{2}}((-1)^{1\cdot 0}|0\rangle + (-1)^{1\cdot 1}|1\rangle). \tag{15}$$

We can then express the action of the Hadamard gate on a single-qubit state $|a\rangle$ using summation notation as,

$$H|a\rangle = \frac{1}{\sqrt{2}} \sum_{x\in\{0,1\}} (-1)^{a\cdot x}|x\rangle. \tag{16}$$

Since the action of a single Hadamard gate is truly just the action of a 2x2 matrix on a single qubit state, we can use the Kronecker product to denote the action of multiple Hadamard gates. By following the same procedure as for the single qubit case, we can generalize the action of the Hadamard gate on a two-qubit state as well. The action of two Hadamard gates on a two-qubit state is just the product of one Hadamard gate acting on the first qubit and the other acting on the second qubit. This is defined as,

$$H^{\otimes 2}|00\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{17}$$

$$H^{\otimes 2}|01\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \tag{18}$$

$$H^{\otimes 2}|10\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle) \tag{19}$$

$$H^{\otimes 2}|11\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle). \tag{20}$$

Let us apply the coefficient treatment from Equations (14) and (15) to Equation (17). Due to reasons stated above, we can write the left-hand side (LHS) of Equation (17) as $\mathrm{H}|0\rangle\mathrm{H}|0\rangle$. we have already defined the action on a single-qubit state $\mathrm{H}|a\rangle$ in the summation convention in Equation (16). For two qubits, we can then write,

$$H|a\rangle = \frac{1}{2} \sum_{x\in\{0,1\}^2} (-1)^{a\cdot x}|x\rangle. \tag{21}$$

This formalism then entails the following generalization for n-qubit states,

$$H^{\otimes n}|a\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{a\cdot x}|x\rangle. \tag{22}$$

Now, in our algorithm, we initialize the upper register as $|0\rangle$, which is the same as $|000...0\rangle$ or $|0\rangle^{\otimes n}$. We first apply n Hadamard gates to this,

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle, \tag{23}$$

where the $(-1)^{a\cdot x}$ term from Equation (22) disappears because $a = 0$, which makes the overall coefficient equal to 1. The output from Equation (23) is an equal superposition of all binary strings $|x\rangle$ of length n in the first register. By preparing the equally weighted superposition of all possible values of $|x\rangle$ that can be stored in the first register, we open up the quantum interference. We now need to apply the quantum oracle to the outputs of the first n Hadamard gates. We have defined the oracle function $f(x)$ to be 0 or 1 and equal to $a \cdot x$, where a is the secret string that we need to find and x is the input. The phase oracle returns 1 for any input x such that $a \cdot x \bmod 2 = 1$ and returns 0 otherwise [2].

We now use the same phase kickback logic that we did in the Deutsch-Josza algorithm where we considered the cases for when the function gave the outputs 0 and 1 to define the action of the oracle as,

$$O_f|x\rangle = (-1)^{a\cdot x}|x\rangle, \tag{24}$$

in which the value of $a \cdot x$ in the exponential (which is just how we defined our classical query function) determines whether the phase factor in front of $|x\rangle$ is +1 or -1. Then the action of O_f on the output from Equation (23) is,

$$O_f\{\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\} = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{a\cdot x}|x\rangle \tag{25}$$

We have the interference pattern now, which we need to close in order to make a measurement on our first register. The second set of Hadamard gates are then applied to the output from the oracle,

$$H^{\otimes n}\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{a\cdot x}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{a\cdot x}H|x\rangle \tag{26}$$

$$= \frac{1}{2^n} \sum_{x,x' \in \{0,1\}^n} (-1)^{a \cdot x} (-1)^{x' \cdot x} |x'\rangle \tag{27}$$

$$= \frac{1}{2^n} \sum_{x,x' \in \{0,1\}^n} (-1)^{a \cdot x} (-1)^{x' \cdot x} |x'\rangle \tag{28}$$

$$= \frac{1}{2^n} \sum_{x',\in \{0,1\}^n} \{ \sum_{\mathbf{x},\in \{\mathbf{0},\mathbf{1}\}^\mathbf{n}} (-\mathbf{1})^{(\mathbf{x'} \oplus \mathbf{a}) \cdot \mathbf{x}} \} |x'\rangle. \tag{29}$$

The exponential of the boldface term inside the curly brackets in Equation (29) is just a bit-wise addition representation of the corresponding exponential in Equation (28). We can consider the boldface term in Equation (29) case-wise. If $a = x'$, that is, if a and x' are two identical binary strings, the bit-wise addition of two identical binary strings would result in a binary string comprising of zeroes only. That would mean that the exponential would be equal to zero, meaning that the phase factor with $|x'\rangle$ would be $+1$. The expression is being added 2^n times, so the boldface term inside the bracket would just be equal to 2^n. The overall output would then just be $|a\rangle$. Summarising the first case where $a = x'$, which implies $a \oplus y = 0$,

$$= \frac{1}{2^n} \sum_{x',\in \{0,1\}^n} \{ \sum_{\mathbf{x},\in \{\mathbf{0},\mathbf{1}\}^\mathbf{n}} (-\mathbf{1})^{\mathbf{0} \cdot \mathbf{x}} \} |x'\rangle \tag{30}$$

$$= \frac{1}{2^n} \sum_{x',\in \{0,1\}^n} \{\mathbf{2^n}\} |x'\rangle = |a\rangle. \tag{31}$$

On the other hand, if a is not equal to x', the bit-wise addition of a and x would be a binary string that would not be equal to zero; there will be at least one 1 in the string. When we run the sum through all possible values of the binary string x or when we take the dot product, an equal number of zeroes and ones is generated, which would mean that the sum of the phase factor would have an equal number of zeroes and ones, equating Equation (29) equal to zero. This makes sense because if we get $x' = a$ at the output with probability 1, the probability of all other possible outcomes should be zero. The output then gives us the value of a with certainty.

From there on, one only has to perform a bit-by-bit measurement on the register in order to read the binary values of a.

3.4.1 Example

In this section we will go through the circuit in Figure (2a), seeing how the state evolves at each step. We start with $\psi_0 = |00\rangle$, we are ignoring the ancillary qubit since its result does not matter, then,

$$\begin{aligned} |\psi_1\rangle &= (\frac{|0\rangle + |1\rangle}{\sqrt{2}})(\frac{|0\rangle + |1\rangle}{\sqrt{2}}) \\ &= \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}. \end{aligned} \tag{32}$$

Now using Equation (24), we can directly see that the result of the oracle is

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{2}((-1)^{00.01}|00\rangle + (-1)^{01.01}|01\rangle + (-1)^{10.01}|10\rangle + (-1)^{11.01}|11\rangle) \\ &= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle). \end{aligned} \tag{33}$$

Now we need to apply the Hadamard gate to both bits,

$$\begin{aligned} |\psi_3\rangle &= (\frac{|0\rangle + |1\rangle}{\sqrt{2}})(\frac{|0\rangle + |1\rangle}{\sqrt{2}}) - (\frac{|0\rangle + |1\rangle}{\sqrt{2}})(\frac{|0\rangle - |1\rangle}{\sqrt{2}}) \\ &\quad + (\frac{|0\rangle - |1\rangle}{\sqrt{2}})(\frac{|0\rangle + |1\rangle}{\sqrt{2}}) - (\frac{|0\rangle - |1\rangle}{\sqrt{2}})(\frac{|0\rangle - |1\rangle}{\sqrt{2}}) \\ &= (|00\rangle + |01\rangle + |10\rangle + |11\rangle - |00\rangle + |01\rangle - |10\rangle + |11\rangle \\ &\quad \underline{+ |00\rangle + |01\rangle - |10\rangle - |11\rangle - |00\rangle + |01\rangle + |10\rangle - |11\rangle)} \\ &\quad 4 \\ &= |01\rangle\,, \end{aligned} \tag{34}$$

which is the query string and the expected output as verified by Figure (2b).

3.5 Complexity

The factor that allows this algorithm its distinguished position can only be understood once we talk about complexity. To define complexity for quantum algorithms and to be able to compare it to the complexity of classical computation complexity theory defines two important classes: BPP and BQP. BPP stands for Bounded-error Probabilistic Polynomial time, this class is reserved for problems that can be solved with a certain probability, usually $\frac{1}{3}$, in polynomial time. This class defines the set of problems that can easily be solved by a classical probabilistic computer. BQP stands for Bounded-error Quantum Polynomial time and defines the class of problems that can be solved by a quantum computer. The Bernstein-Vazirani algorithm was one of the earlier algorithms to make a distinction between these two classes and show that quantum algorithms have an advantage over the classical algorithm[3].

We can analyse the circuit in a myriad of ways. The most common way to discuss the complexity of a quantum algorithm is query complexity, which is the amount of times we need to query the oracle to guess the string. As discussed in previous sections, for the BV algorithm the query complexity is $\emptyset(1)$, where the classical algorithm requires $O(n)$. Another criteria we can look at is the circuit complexity, which is the number of gates required to implement the Oracle. In our case, the worst case is that the string consist entirely of 1s, which leads to a complexity of $O(n)$[4].

Additionally, we can look at time complexity. Traditionally this is defines as the time taken for the algorithm to execute as a function of it's length. For a quantum algorithm if we assume that all the gates at the same depth can be executed in parallel we can then define the time complexity in terms of the depth of the circuit[5]. For our case we have a depth of 3 due to the application of the Hadamard gates at the beginning and end, and the Pauli-Z gate for the auxiliary qubit. In addition to this we can have n gates in the oracle, leading to a time complexity of $O(n+3)$.

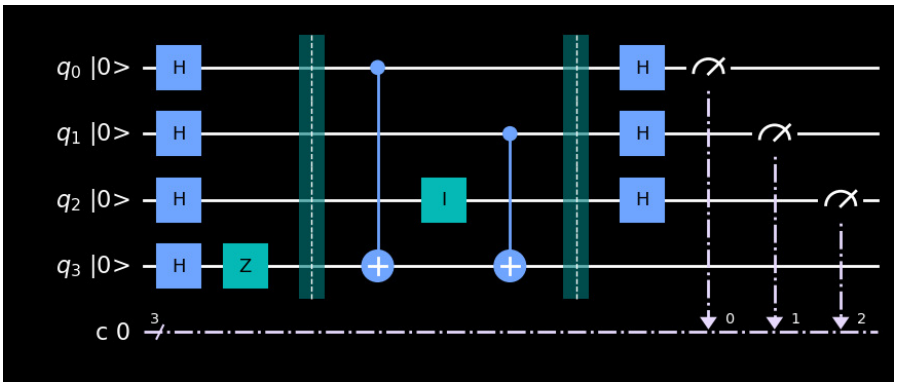
Finally, we can look at the space complexity. Space complexity classes are related to the amount of space taken by quantum and classical Turing machines and measured in terms of number of bits required to encode information. Generally, we can say that at most the space required for a given configuration is bounded by 2^l where l is the length of the binary string we are encoding. In this case, we have $n+1$ bits for a n bit string hence the space complexity can be expressed as $O(2^{n+1})$[6].

3.6 Performance Analysis

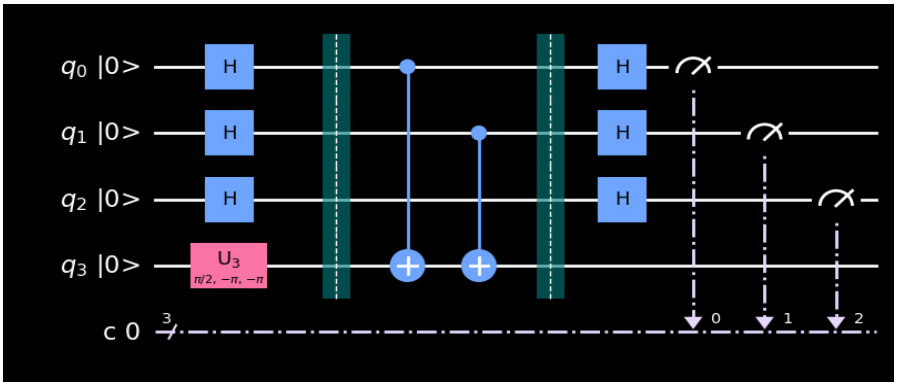
Using the tools provided by IBM we can see this algorithm in action and compare how it performs on a simulator and on a real quantum computer. To do so we used Python to generate a random 3 bit string and created a circuit corresponding to that string (Figure 4a). Then we used the Aer simulator to generate a histogram of results (Figure 4c). In addition to this, we also transpiled the circuit to be able to see how the circuit would be modified once it is run on an actual device. As can be seen in Figure (4b) the gates for the preparation of the auxiliary bit are combined into a unitary operator with three different rotations.

Next we ran the algorithm on actual quantum devices and have compiled the results from those histograms in Figure (6), Figure (8) and Figure (10) beside their corresponding circuits. As can be seen there is a clear distinction between the results of the simulator and the results on the actual device. However, despite the erroneous results we can see that a majority of the times it does produce the correct output and that it is still possible to guess the secret string. There are multiple sources of noise when a circuit is implemented on an actual device. Two of the main ones are gate infidelity and decoherence.

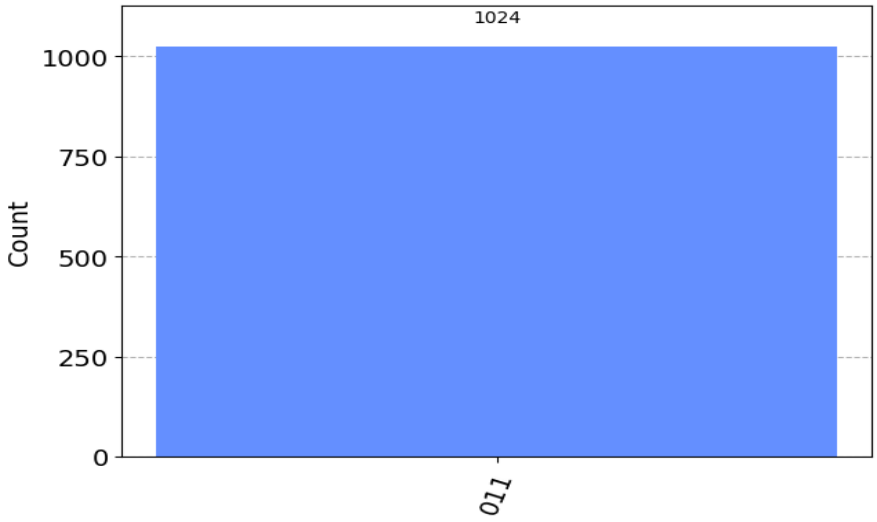
We can try and focus on Figure (6). This circuit was run on the IBM-Perth



(a) The algorithm implemented for the 3 bit string '011'.



(b) The same circuit once it has been transpiled.



(c) Histogram of results from the Aer simulator.

Figure 4: Implementation of the Bernstein-Vazirani algorithm for a 3 bit string.

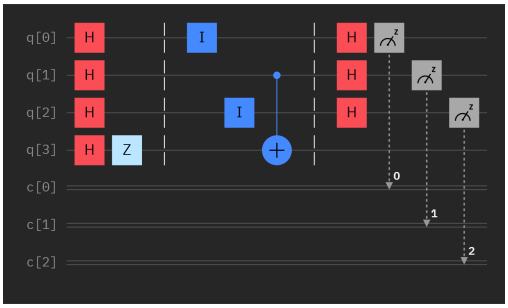


Figure 5: Circuit for the string "010".

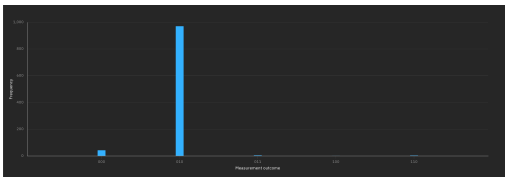


Figure 6: Results of implementation on IBM-Perth.

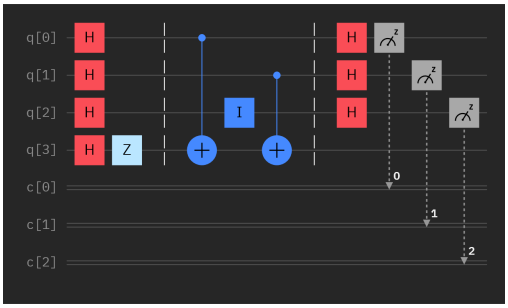


Figure 7: Circuit for the string "011".

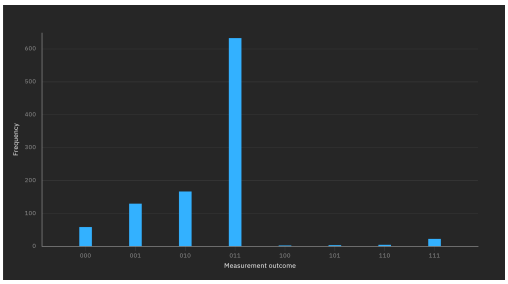


Figure 8: Results of implementation on IBM-Jakarta.

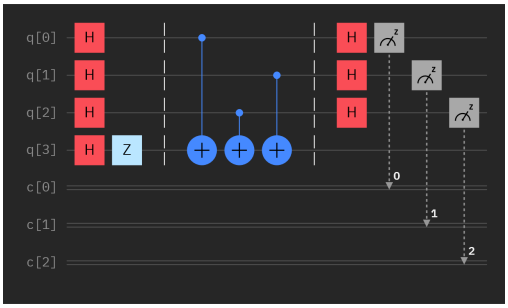


Figure 9: Circuit for the string "111".

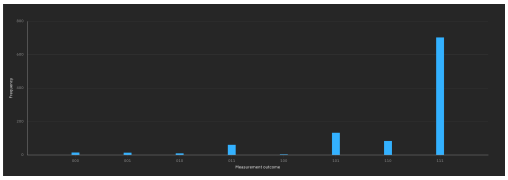


Figure 10: Results of implementation on IBM-Quito.

machine. This machine has the median T1 error of 199.65 μs and a median T2 of 123.09 μs. These values are the decoherence time, where T1 is the qubit relaxation time which is the time taken for an excited state to relax into the ground state, and T2 is the time taken for the superposition to decohere. In other words, for a qubit to turn into a classical bit. The greater the circuit depth, the longer it needs to run and the greater the threat posed by these small values.

Similarly it has the median readout error of 2.090×10^{-2} which is the probability of the measurement returning a wrong value. While this is a small number, as you increase the number of shots the errors become more apparent. There are also values available for errors specific to gates. For the qubit there is Pauli-X error 3.207×10^{-4} and the $\sqrt{X}(sx)$ error median 3.207×10^{-4} which measures the average error of the (sx) gate that has been used as a basis for the circuit. The machine implements only one multi-qubit gate which is the CNOT gate which is why it has its own median error 8.642×10^{-3} and the gate time 398.222 ns which defines how long the connection will last.

We can find similar values for other machines as well. While there may be a slight difference we will always see the influence of these uncertainties in the final result and for more gates or more complicated connections these errors will have a greater contribution.

4 Bernstein-Vazirani Algorithm With Two Secret Strings and a Probabilistic Oracle

Our systematic review of the original Bernstein-Vazirani algorithm in the previous sections has equipped us with enough knowledge to generalize it to the case of finding two secret strings. We follow literature on the probabilistic Bernstein-Vazirani algorithm [7] to choose a rather fundamental two string case because its understanding will really just simplify a further generalization to finding i secret strings. This generalization is an extension of the Bernstein-Vazirani algorithm in that it extends it so that the failure of a classical oracle is implied since it will not be able to find any bits of the secret strings with certainty. A classical algorithm may only obtain probability distributions of zero and one for each position of the secret strings. It cannot know which secret string it has gained information from and so cannot guess either of them. For this problem, a probabilistic approach is implemented; it one secret string using a single query to the probabilistic oracle. Running this oracle multiple times will generate the keys with equal probability eventually revealing them both. Which is a drastic improvement from the classical approach which would not be able to find a single one. It finds one of the first i strings with certainty and the rest of them with a high probability; we shall probe into its inner workings in the following sections. The probabilistic Bernstein-Vazirani (pBV) algorithm has a more complex oracle function due to multiple secret strings. Querying the oracle classically for each possible combination of secret strings would then require an exponential number of queries in the worst case. The quantum mechanical treatment of this problem then allows for a wider range of oracle functions to be used. In this section, we look at the running algorithm and the circuit behind the problem in order to understand how the probabilistic oracle finds multiple, particularly two, secret strings.

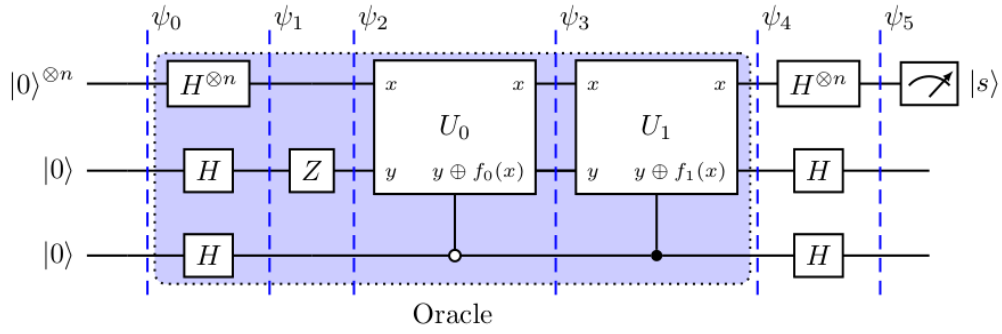
4.1 Problem

The goal of the pBV algorithm is to find multiple strings using a probabilistic oracle O_k. The functions are defined the same way as in Section (3.2). To recap, i distinct secret strings of n bits are defined as $f_i(x) = a_i \cdot x$, for $i = 0, 1, ..., k-1$ and here, $1 \leq k \leq 2^n$. For any input string, the oracle returns one of the function outputs with the equal probability of $\frac{1}{k}$. The pBV is, essentially, then first finding the first secret string and then the rest of the strings with a high probability with the least number of queries to the oracle. One of the reasons for why oracle is "probabilistic" is because for a large number of secret strings, the probability of finding all of them approaches one if number of calls to the oracle approach infinity.

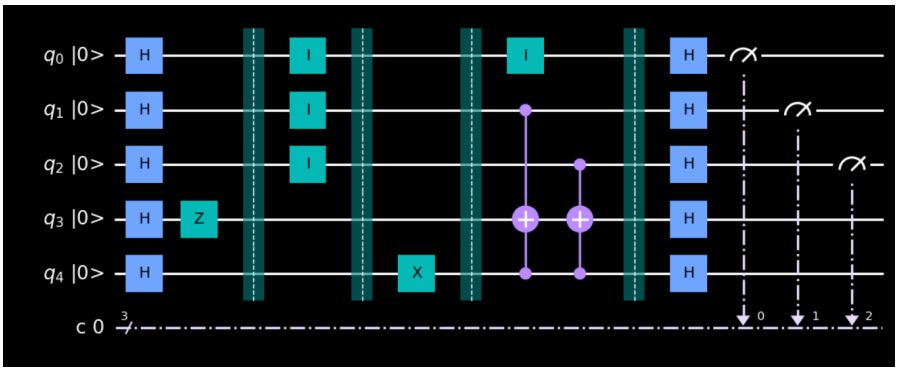
It is interesting to note here that if all of the secret strings are identical, then we just have our original BV algorithm problem back.

4.2 Building the Circuit

This section will expand on the circuit implementation of the pBV algorithm for the case of two secret keys. The schematic diagram of the oracle is shown in Figure (11a). The circuit's operation is similar in many ways to that of the BV algorithm's circuit that we have previously discussed. However, there are some key changes, one of them being the presence of a control qubit in addition to the already existing ancillary qubit. The number of control qubits required depends on the number of secret keys in the circuit. Here, the bottom-most ancillary qubit is used as the control qubit for the unitary gates U_0 and U_1. This is initialised to the state $|0\rangle$ and then transformed via the Hadamard gate as is customary for the bits in this algorithm. In addition to that the control qubit has an X gate that acts after the oracle for the first secret key. This X gate will flip the bit and since this bit is linked to both the oracles through the application of the Toffoli gate it activates one oracle at a time. This ensures that the unitary gates are selected with equal probability – which is $\frac{1}{2}$ in our case. [7].



(a) Circuit for implementation of the probabilistic BV algorithm for deciphering two secret keys.



(b) Detailed oracle implementation for the probabilistic BV problem, for the case of two secret keys (000, 110).

Figure 11: Implementation of the probabilistic Bernstein-Vazirani algorithm.

4.3 Mathematical Formulation

The mathematics in this section will simply expand on our quantum solution for the BV algorithm. The probabilistic oracle can be thought of as a mega-oracle, O_k that contains smaller oracles that can decode an individual secret string. The smaller oracles function the same way as those in the regular BV algorithm. Following the same formalism and logic as in Section (3.4), we define the action of the oracle O_k on n qubits initialized in the $|0\rangle$ state as,

$$O_k \left|0\right\rangle^{\otimes n} = \frac{1}{\sqrt{k}} \sum_{i=0}^{k-1} \left|\Psi_i\right\rangle, \tag{35}$$

where

$$\left|\Psi_i\right\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{a_i \cdot x} \left|x\right\rangle, \tag{36}$$

for $i = 0, 1, 2, ..., k-1$. For the two strings case, i will simply be 0 and 1. The output of the top n qubits would then be,

$$H^{\otimes n} O_k \left|0\right\rangle^{\otimes n} = H^{\otimes n} \left(\frac{1}{k} (\left|\Psi_0\right\rangle + \left|\Psi_1\right\rangle + ... + \left|\Psi_{k-1}\right\rangle \right) \tag{37}$$

A lengthy calculation for k secret strings can be continued from Equation (37) onward; however, since the logic behind the generalisation to Equation (37) has been shown step-wise, from here on, we limit our calculation to the two strings case (i = 0,1). The output is then,

$$= H^{\otimes n} \left(\frac{1}{k} \left(\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{a_0 \cdot x} \left|x\right\rangle + \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{a_1 \cdot x} \left|x\right\rangle \right) \right) \tag{38}$$

$$= \frac{1}{k} \left(\frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} (-1)^{a_0 \cdot x + y \cdot x} \left|y\right\rangle + \frac{1}{2^n} \sum_{y=0}^{2^n-1} \sum_{x=0}^{2^n-1} (-1)^{a_1 \cdot x + y \cdot x} \left|y\right\rangle \right). \tag{39}$$

If y $= a_i$, then y $\oplus a_i = 0$, and $\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{(a_i \oplus y) \cdot x} \left|y\right\rangle \otimes \left|1\right\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{(0) \cdot x} \left|a_i\right\rangle$ $= \left|a_i\right\rangle$. If y is not equal to a_i, the corresponding coefficient of $\left|y\right\rangle$ would be 0. Then for two secret strings,

$$H^{\otimes n} O_k \left|0\right\rangle^{\otimes n} = \frac{1}{k}(\left|a_0\right\rangle + \left|a_1\right\rangle). \tag{40}$$

Therefore, we see that performing a measurement returns one of the secret strings with equal probability $\frac{1}{k}$. Then, only one query is needed to find one of the two secret strings. Repeated measurements will return both the secret strings with equal probability on an ideal quantum computer.

4.3.1 Worked Example

In this section, we offer a calculated example for how the states evolve at each step following the oracle circuit for the pBV algorithm for two strings. We define the two 3-bit secret strings $a_0 = 011$ and $a_1 = 101$. We begin with the initialised state $\left|00000\right\rangle$. Following our mathematical formalism and definition of the circuit for two keys, we first apply the first Hadamard gates on the qubits to get,

$$\begin{aligned}
&H\left|00000\right\rangle = \frac{1}{4\sqrt{2}}(\left|0\right\rangle + \left|1\right\rangle)(\left|0\right\rangle + \left|1\right\rangle)(\left|0\right\rangle + \left|1\right\rangle)(\left|0\right\rangle + \left|1\right\rangle)(\left|0\right\rangle + \left|1\right\rangle) \\
&= \frac{1}{4\sqrt{2}}(\left|00000\right\rangle + \left|00010\right\rangle + \left|00100\right\rangle + \left|00110\right\rangle + \left|00001\right\rangle + \left|00011\right\rangle + \left|00101\right\rangle + \left|00111\right\rangle \\
&+ \left|01000\right\rangle + \left|01010\right\rangle + \left|01100\right\rangle + \left|01110\right\rangle + \left|01001\right\rangle + \left|01011\right\rangle + \left|01101\right\rangle + \left|01111\right\rangle \\
&+ \left|10000\right\rangle + \left|10010\right\rangle + \left|10100\right\rangle + \left|10110\right\rangle + \left|10001\right\rangle + \left|10011\right\rangle + \left|10101\right\rangle + \left|10111\right\rangle \\
&+ \left|11000\right\rangle + \left|11010\right\rangle + \left|11100\right\rangle + \left|11110\right\rangle + \left|11001\right\rangle + \left|11011\right\rangle + \left|11101\right\rangle + \left|11111\right\rangle).
\end{aligned}$$

This ensures the the two mini oracles for the two secret strings are selected with equal probability. The Z gate is then applied to the third qubit,

$$\begin{aligned}
&= \frac{1}{4\sqrt{2}}(\left|00000\right\rangle + \left|00010\right\rangle - \left|00100\right\rangle - \left|00110\right\rangle + \left|00001\right\rangle + \left|00011\right\rangle - \left|00101\right\rangle - \left|00111\right\rangle \\
&+ \left|01000\right\rangle + \left|01010\right\rangle - \left|01100\right\rangle - \left|01110\right\rangle + \left|01001\right\rangle + \left|01011\right\rangle - \left|01101\right\rangle - \left|01111\right\rangle \\
&+ \left|10000\right\rangle + \left|10010\right\rangle - \left|10100\right\rangle - \left|10110\right\rangle + \left|10001\right\rangle + \left|10011\right\rangle - \left|10101\right\rangle - \left|10111\right\rangle \\
&+ \left|11000\right\rangle + \left|11010\right\rangle - \left|11100\right\rangle - \left|11110\right\rangle + \left|11001\right\rangle + \left|11011\right\rangle - \left|11101\right\rangle - \left|11111\right\rangle).
\end{aligned}$$

Following the action of the first Toffoli gate, which we know is built in a way that it contains information about the first secret string and likewise for the second Toffoli gate and the second secret string, we obtain,

$$= \frac{1}{4\sqrt{2}}(|00000\rangle + |00010\rangle - |00100\rangle - |00110\rangle + |00001\rangle + |00011\rangle - |00101\rangle - |00111\rangle$$
$$+ |01000\rangle + |01010\rangle - |01100\rangle - |01110\rangle + |01001\rangle + |01011\rangle - |01101\rangle - |01111\rangle$$
$$+ |10000\rangle + |10010\rangle - |10100\rangle - |10110\rangle + |10011\rangle + |10001\rangle - |10111\rangle - |10101\rangle$$
$$+ |10001\rangle + |10011\rangle - |10101\rangle - |11110\rangle + |11011\rangle + |11001\rangle - |11111\rangle - |11101\rangle).$$

The action of the second Toffoli gate gives us,

$$= \frac{1}{4\sqrt{2}}(|00000\rangle + |00010\rangle - |00100\rangle - |00110\rangle + |00001\rangle + |00011\rangle - |00101\rangle - |00111\rangle$$
$$+ |01000\rangle + |01010\rangle - |01100\rangle - |01110\rangle + |01011\rangle + |01001\rangle - |01111\rangle - |01101\rangle$$
$$+ |10000\rangle + |10010\rangle - |10100\rangle - |10110\rangle + |10011\rangle + |10001\rangle - |10111\rangle - |10101\rangle$$
$$+ |10001\rangle + |10011\rangle - |10111\rangle - |11110\rangle + |11001\rangle + |11011\rangle - |11101\rangle - |11111\rangle)$$

Now we apply the X gate on the controlled qubit,

$$= \frac{1}{4\sqrt{2}}(|00001\rangle + |00011\rangle - |00101\rangle - |00111\rangle + |00000\rangle + |00010\rangle - |00100\rangle - |00110\rangle$$
$$+ |01001\rangle + |01011\rangle - |01101\rangle - |01111\rangle + |01010\rangle + |01000\rangle - |01110\rangle - |01100\rangle$$
$$+ |10001\rangle + |10011\rangle - |10101\rangle - |10111\rangle + |10010\rangle + |10000\rangle - |10110\rangle - |10100\rangle$$
$$+ |10000\rangle + |10010\rangle - |10110\rangle - |11111\rangle + |11000\rangle + |11010\rangle - |11100\rangle - |11110\rangle).$$

The action of the X gate on the controlled qubit flips the state of the controlled qubit. The controlled qubit is linked to the qubit that corresponds to a 1 in the secret string. The controlled not Toffoli gate is on the ancillary qubit with one conditional being on the qubit corresponding to a 1 in the secret string and the second one on the control qubit. These conditionals then dictate whether the state of the ancillary qubit will be flipped or not. The state of the control qubit being flipped ensures that the action of the previous gate on the n-qubit state will not be repeated. In other words, the X gate ensures that either one oracle will work or the other. This also ensures that the second oracle works with a half probability and that both of the oracles are unique, i.e. one oracle works on one uniquely-coded string. We then apply the third Toffoli gate on the overall state,

$$= \frac{1}{4\sqrt{2}}(|00001\rangle + |00011\rangle - |00101\rangle - |00111\rangle + |00000\rangle + |00010\rangle - |00100\rangle - |00110\rangle$$
$$+ |01001\rangle + |01011\rangle - |01101\rangle - |01111\rangle + |01010\rangle + |01000\rangle - |01110\rangle - |01100\rangle$$
$$+ |10011\rangle + |10001\rangle - |10111\rangle - |10101\rangle + (|10010\rangle) + |10010\rangle - |10110\rangle - |10100\rangle$$
$$+ |10000\rangle + |10010\rangle - |10110\rangle - |11101\rangle + |11000\rangle + |11000\rangle - |11100\rangle - |11110\rangle),$$

followed by the fourth Toffoli gate,

$$= \frac{1}{4\sqrt{2}}(|00001\rangle + |00011\rangle - |00111\rangle - |00101\rangle + |00000\rangle + |00010\rangle - |00100\rangle - |00110\rangle$$
$$+ |01001\rangle + |01011\rangle - |01111\rangle - |01101\rangle + |01010\rangle + |01000\rangle - |01110\rangle - |01100\rangle$$
$$+ |10011\rangle + |10001\rangle - |10101\rangle - |10111\rangle + (|10010\rangle) + |10010\rangle - |10110\rangle - |10100\rangle$$
$$+ |10000\rangle + |10010\rangle - |10110\rangle - |11111\rangle + |11000\rangle + |11000\rangle - |11100\rangle - |11110\rangle).$$

Finally, when Hadamard gates are acted on all of the qubits and measurements are performed on the first, second, and third qubits, one secret string is revealed bit-by-bit. Repeating the measurements for a large number of times would result in the measurement of the secret strings with a close to equal probability as evidenced by the histogram that we obtained after running the algorithm on the Aer-simulator. We have provided the formulation for two strings; if one might wish to perform generalize this to a secret string number of their choice, only i would need to change following Equation (36). The logic behind the rest of the steps would remain the same. For x keys, running the algorithm on the Aer-simulator would give a histogram comprising x bars, of near-equal probabilities.

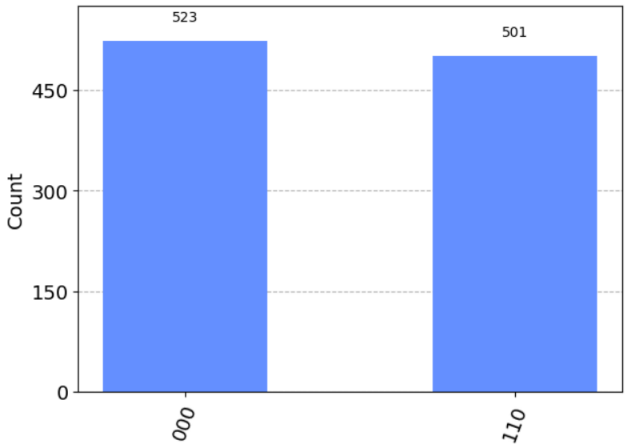


Figure 12: Histogram of Aer simulator-run results for two distinct secret strings.

5 Conclusion

Through this report, we have shown how algorithms like the Bernstein-Vazirani algorithm have demonstrated the clear advantage that quantum computers have over their classical analogs. In section 3, we have discussed the motivation behind the algorithm in detail; additionally, we also addressed its working principle, both in context of its circuit and the mathematics behind it. We also worked through an example for a more detailed understanding of the algorithm. Finally, we talk about complexity theory in context of the algorithm and discuss the pitfalls of real

quantum machines. Then, in section 4 we introduced a variation of the Bernstein-Vazirani algorithm that demonstrates a greater advantage by achieving something that would be impossible on a classical computer. Sections that followed expanded on the mathematical and circuital underpinnings of the circuit and concluded with a detailed example for the two secret strings case. In doing so we have presented a comprehensive overview of the Bernstein-Vazirani algorithm, which can have fascinating applications in quantum computing, such as those in quantum image representation and processing[8].

References

[1] Emilio Peláez. A clever quantum trick. https://medium.com/quantum-untangled/a-clever-quantum-trick-54f27e2518a4. Accessed: 07-04-2023.

[2] Qiskit. Bernstein-vazirani algorithm. https://qiskit.org/textbook/ch-algorithms/bernstein-vazirani.html. Accessed: 07-04-2023.

[3] Louis Chen. Quantum algorithm (6):bernstein-vazirani algorithm. https://quantumpedia.uk/quantum-algorithm-6-bernstein-vazirani-algorithm-f371ea83256e. Accessed: 26-04-2023.

[4] Ronald de Wold. *Quantum Computing: Lecture Notes*. QuSoft, CWI and University of Amsterdam, 2011.

[5] Alessandro Luongo. Quantum algorithms for data analysis. https://quantumalgorithms.org/chapter-intro.html. Accessed: 26-04-2023.

[6] John Watrous. Space-bounded quantum complexity. *Journal of Computer and System Sciences*, 2(59):281–326, 1999.

[7] Alok Shukla and Prakash Vedula. A generalization of bernstein–vazirani algorithm with multiple secret keys and a probabilistic oracle. https://arxiv.org/pdf/2301.10014.pdf.

[8] Alexandru-Gabriel Tudorache, Vasile-Ion Manta, and Simona Caraiman. Implementation of the bernstein-vazirani quantum algorithm using the qiskit framework. *Bulletin of the Polytechnic Institute of Iași. Electrical Engineering, Power Engineering, Electronics Section*, 67(2):31–40, 2021.