Bernstein-Vazirani Algorithm

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

PHY 318: Project Draft 1

Friday, April, 7, 2023

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 Mathematical Formulation

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 $a \cdot x \pmod 2$. 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 $f(x) = a \cdot x \pmod 2$, 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 Building the circuit

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].

Consider Figure (1). Assuming that the initial input is 0 for both qubits, after the

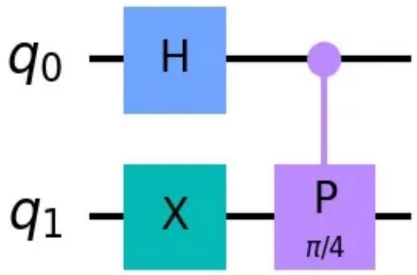


Figure 1: A schematic to explain phase kickback.

action of the Hadamard on the first qubit we get,

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

Then applying the Pauli-X gate leads us to,

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

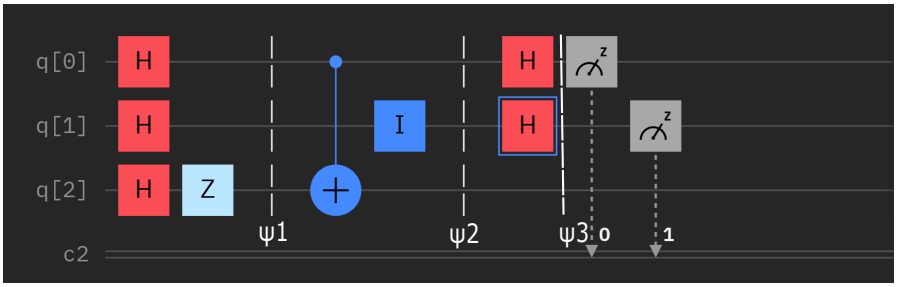
Finally the application of the controlled phase gate leads to,

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

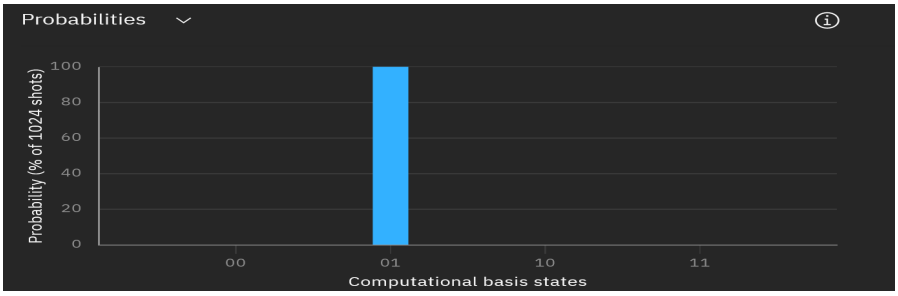
$$=(\frac{|0\rangle + e^{\iota \frac{\pi}{4}} |1\rangle}{\sqrt{2}}) |1\rangle \,. \tag{11}$$

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.



(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.

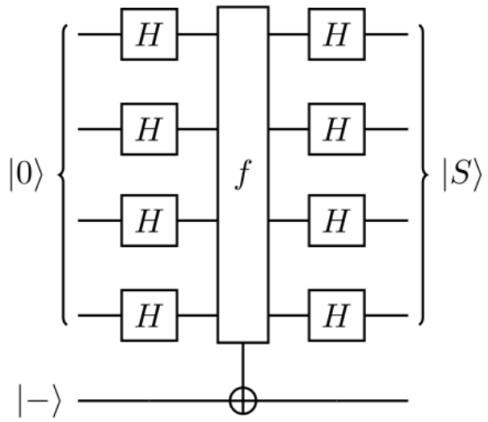


Figure 3: A schematic to ease the visualisation of how the quantum solution is approached [2].

3.4 Quantum Solution

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).

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.