Quantum Computing 105: Bernstein-Vazirani Algorithm

Kshipra Wadikar

March 26, 2025

Quantum computing has the potential to solve certain problems exponentially faster than classical computers. One such problem is finding a hidden binary string encoded within a function using the Bernstein-Vazirani algorithm. This algorithm demonstrates quantum speedup by determining an unknown n-bit string with just one query, whereas a classical approach would require n function evaluations.
In this article, we explore the Bernstein-Vazirani algorithm step by step, beginning with the mathematical representation of quantum states and their decimal mappings. We then describe the problem setup, explain the classical and quantum approaches, and finally, analyze how the quantum circuit efficiently determines the hidden bit string.

1 Mapping Quantum States to Decimal Numbers

In quantum computing, a system with n qubits can represent 2^n different states, where each state is a binary number of length n. Each binary state $|x_1x_2\ldots x_n\rangle$ can be mapped to a unique decimal number using the following formula:

$$\text{Decimal Equivalent} = \sum_{i=1}^{n} x_i \cdot 2^{(n-i)}$$

where:

- $x_1, x_2, \ldots, x_n$ are the individual bits in the quantum state.
- x_1 is the most significant bit (MSB).
- x_n is the least significant bit (LSB).

Example: 2-Qubit System
For a 2-qubit system, the mapping follows: Thus, for an n-qubit system, this formula

Quantum State $\|x_1x_2\rangle$	Decimal Equivalent
$\|00\rangle$	$0 \cdot 2^1 + 0 \cdot 2^0 = 0$
$\|01\rangle$	$0 \cdot 2^1 + 1 \cdot 2^0 = 1$
$\|10\rangle$	$1 \cdot 2^1 + 0 \cdot 2^0 = 2$
$\|11\rangle$	$1 \cdot 2^1 + 1 \cdot 2^0 = 3$

provides a systematic way to convert quantum states to their corresponding decimal values. This notation is widely used in quantum algorithms for efficient data processing.

2 Problem Statement: Bernstein-Vazirani Algorithm

Throughout this article, unless explicitly stated otherwise, the **oracle function** $f(x)$ is as defined below.

$$f(x) = (a_1 \cdot x_1) \oplus (a_2 \cdot x_2) \oplus \cdots \oplus (a_n \cdot x_n)$$

where:

- $x = (x_1, x_2, \ldots, x_n)$ is an n-bit input vector.
- $a = (a_1, a_2, \ldots, a_n)$ is the unknown n-bit key we need to determine.
- $\cdot$ represents bitwise multiplication (AND operation).
- $\oplus$ represents the bitwise XOR operation.
- $f(x)$ produces a single-bit output (0 or 1).

Example Scenario:
Consider $n = 2$, hidden key $a = 11_2$ which means $a_1 = 1, a_2 = 1$.
For 2-bits, the input vectors are $00, 11, 01, 10$.
When $x = 00$, $f(x) = (1 \cdot 0) \oplus (1 \cdot 0) = 0 \oplus 0 = 0$
When $x = 01$, $f(x) = (1 \cdot 0) \oplus (1 \cdot 1) = 0 \oplus 1 = 1$
When $x = 10$, $f(x) = (1 \cdot 1) \oplus (1 \cdot 0) = 1 \oplus 0 = 1$
When $x = 11$, $f(x) = (1 \cdot 1) \oplus (1 \cdot 1) = 1 \oplus 1 = 0$

2.1 Problem statement

Given such a function $f(x)$ our goal is to determine the unknown key a using a minimal number of function evaluations.

2.2 Finding a for an Unknown 2-Bit Key

We are given the function

$$f(x) = (a_1 \cdot x_1) \oplus (a_2 \cdot x_2)$$

where $a = (a_1, a_2)$ is **unknown** and our **goal** is to determine a using the least number of function evaluations.
To determine each bit of a, we select inputs where only one bit is set to 1 at a time:
When $x = 10$, $f(x) = a_1$ and thus $f(10)$ directly reveals a_1.
When $x = 01$, $f(x) = a_2$ and thus $f(01)$ directly reveals a_2.
By evaluating $f(10)$ and $f(01)$ we directly determine $a_1 = f(10), a_2 = f(01)$ in just two function evaluations.

2.3 Generalization for Any n-Bit Key

For an n-bit unknown key, we make n queries:

- $x = 10...0 \rightarrow$ Determines a_1.
- $x = 01...0 \rightarrow$ Determines a_2.
- ...
- $x = 00...1 \rightarrow$ Determines a_n.

Each function call isolates one bit of a, and after n queries, we fully reconstruct a.

2.4 Step-by-Step Execution of Bernstein-Vazirani Algorithm for a 2-Qubit System

The Bernstein-Vazirani algorithm is a quantum algorithm used to efficiently determine a hidden n-bit string a with just one function evaluation.
Let's go step by step for a 2-qubit system where $a = a_1a_2$ is unknown.

1. **Quantum Circuit**
   We start with $3 = (2 + 1)$ qubits. Qubits q_1, q_2 both are initialized to $|0\rangle$.
   The ancilla qubit is initialized to $|1\rangle$.
   The **initial state of the system** is $|0\rangle\,|0\rangle\,|1\rangle$

2. **Applying Hadamard Gates to Initialize the State**
   Applying a Hadamard transform on each qubit:

   $$H^{\otimes 3}\,|0\rangle\,|0\rangle\,|1\rangle$$

   This gives:

   $$\frac{1}{2}\sum_{x=0}^{3}|x\rangle \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

   which results in an equal superposition of all inputs x.

3. **Applying the Oracle (Encoding a)**
   The oracle applies a phase shift to the state based on $f(x)$:

   $$|x_1, x_2, y\rangle \to |x_1, x_2, y \oplus f(x)\rangle$$

   The ancilla qubit y is initialized to the state $\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}$. Therefore, the operation becomes:

   $$|x_1, x_2\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \to (-1)^{f(x)}\,|x_1, x_2\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

   This introduces a phase shift based on $f(x)$.

4. **Applying Hadamard Transform Again on the First Two Qubits**
   Applying another Hadamard transform on the first two qubits extracts a.

   $$H\,|x\rangle = \frac{1}{2}\sum_{z=0}^{3}(-1)^{a\cdot z}\,|z\rangle$$

   After measurement, the first two qubits directly reveal a in a single query!

5. **Measuring the First Two Qubits**

   - If the measurement result is a_1a_2, we have found a in one function call.
   - Unlike the classical approach (which requires 2 queries), the quantum approach does this in only one query.

3 Algorithm: Bernstein-Vazirani for n Qubits

1. **Initialize the Quantum Register:**

   - Start with $n + 1$ qubits in the state:

   $$|0\rangle \otimes n \otimes |1\rangle$$

- The first n qubits represent the input register, and the last qubit (ancillary) assists in function evaluation.

2. **Apply Hadamard Gates:**

   - Apply Hadamard gates (H) to all $n+1$ qubits:

   $$H^{\otimes(n+1)} |0\rangle^{\otimes n} |1\rangle$$

   - This creates a superposition of all 2^n input states in the computational qubits, while the ancillary qubit is transformed to:

   $$\frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

   - The resulting quantum state is:

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

3. **Apply the Oracle (Function Encoding $f(x)$):**

   - The oracle applies the function transformation:

   $$|x\rangle |y\rangle \to |x\rangle |y \oplus f(x)\rangle$$

   - Since the ancillary qubit was initialized to $|1\rangle$, and we intialize it to $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ to cause phase kickback, this transformation introduces a phase factor:

   $$(-1)^{f(x)} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

   - This results in the state:

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

   - The ancillary qubit is now separable and no longer needed.

4. **Apply Hadamard Transform Again to the First n Qubits:**

   - Another Hadamard transform extracts the hidden string a:

   $$H^{\otimes n} \sum_{x=0}^{2^n-1} (-1)^{a \cdot x} |x\rangle$$

   - This transformation ensures that the computational qubits collapse into the state $|a\rangle$, meaning the measurement will always yield a.

5. **Measure the First n Qubits:**
   The measurement of the computational qubits directly gives the hidden binary string a with 100% probability.

4 Custom Qiskit Implementation

The following libraries are required.

Listing 1: required libraries

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

Listing 2: Qiskit Implementation for the Bernstein-Vazirani Algorithm

```
def bernstein_vazirani(n, a):
    qc = QuantumCircuit(n+1, n)
    qc.x(n)  # Initialize ancilla to |1>
    qc.h(range(n+1))  # Apply Hadamard gates

    # Apply CNOT gates based on the hidden string 'a'
    for i in range(n):
        if a[i] == '1':
            qc.cx(i, n)

    qc.h(range(n))  # Apply Hadamard gates again
    qc.measure(range(n), range(n))  # Measure first n qubits

    return qc
```

Listing 3: Run the Algorithm

```
n = 6
a = "101110"  # Hidden binary string
bv_circuit = bernstein_vazirani(n, a)

#  Use Qiskit Aer simulator
simulator = Aer.get_backend('aer_simulator')  #  Correct in
   Qiskit 1.24
compiled_circuit = transpile(bv_circuit, simulator)
job = simulator.run(compiled_circuit, shots=1024)
result = job.result()

# Get measurement counts
counts = result.get_counts()
print(counts)
plot_histogram(counts)
```

The output of the above code is

```
{'011101': 1024}
```

5 Next Steps: Simon's Algorithm

While the Bernstein-Vazirani algorithm demonstrates an exponential advantage in determining a hidden bit string with a single query, another groundbreaking quantum algorithm—Simon's Algorithm—paves the way for more advanced quantum speedups.

Simon's Algorithm introduced the first provable exponential separation between quantum and classical computing, laying the foundation for Shor's factoring algorithm.

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2010. See Section 3.4 for a discussion of quantum oracles and their role in the Bernstein-Vazirani algorithm, including its application in query complexity.

[2] IBM Quantum, "Qiskit Documentation," Available at: https://qiskit.org/documentation/. Reference for implementing the Bernstein-Vazirani algorithm using Qiskit's latest version.