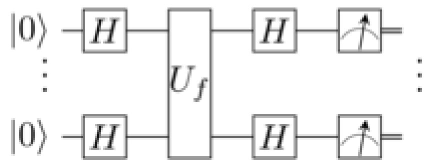
Bernstein–Vazirani algorithm



The **Bernstein–Vazirani algorithm**, which solves the **Bernstein–Vazirani problem** is a quantum algorithm invented by Ethan Bernstein and Umesh Vazirani in 1992[1] . It's a restricted version of the Deutsch–Jozsa algorithm where instead of distinguishing between two different classes of functions, it tries to learn a string encoded in a function[2]. The Bernstein–Vazirani algorithm was designed to prove an oracle separation between complexity classes BQP and BPP.[1]

Contents

Problem statement

Given an oracle that implements some function $f\colon\{0,1\}^n \rightarrow \{0,1\}$, It is promised that the function $f(x)$ is a dot product between x and a secret string $s \subset \{0,1\}^n$ modulo 2. $f(x) = x \cdot s = x_1 s_1 + x_2 s_2 + \cdots + x_n s_n$, find s.

Algorithm

Classically, the most efficient method to find the secret string is by evaluating the function n times where $x = 2^i$, $i \subset \{0, 1, \ldots, n-1\}$[2]

$$\begin{aligned}
f(1000\cdots0_n) &= s_1\\
f(0100\cdots0_n) &= s_2\\
f(0010\cdots0_n) &= s_3\\
&\vdots\\
f(0000\cdots1) &= s_n
\end{aligned}$$

In contrast to the classical solution which needs at least n queries of the function to find s, only one query is needed quantumly. The quantum algorithm is as follows: [2]

Apply a Hadamard transform to the n qubit state $|0\rangle^{\otimes n}$ to get

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

Perform a controlled negation of every state in the superposition generated by the previous Hadamard transformation for which the oracle when applied to this state returns 1. This transforms the superposition into

$$\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n}(-1)^{f(x)}|x\rangle.$$

Another Hadamard transform is applied to each qubit which makes it so that for qubits where $s_i = 1$, its state is converted from $|-\rangle$ to $|1\rangle$ and for qubits where $s_i = 0$, its state is converted from $|+\rangle$ to $|0\rangle$.

To obtain s, a measurement on the Standard basis ($\{|0\rangle, |1\rangle\}$) is performed on the qubits.

Implementation

An implementation of the Bernstein–Vazirani algorithm in Cirq.[3]

```
"""Demonstrates the Bernstein-Vazirani algorithm.

The (non-recursive) Bernstein-Vazirani algorithm takes a black-box oracle
implementing a function f(a) = a·factors + bias (mod 2), where 'bias' is 0 or 1,
'a' and 'factors' are vectors with all elements equal to 0 or 1, and the
algorithm solves for 'factors' in a single query to the oracle.
=== EXAMPLE OUTPUT ===
Secret function:
f(a) = a·<0, 0, 1, 0, 0, 1, 1, 1> + 0 (mod 2)
Sampled results:
Counter({'00100111': 3})
Most common matches secret factors:
True
"""
import random
import cirq

def main(qubit_count = 8):
    circuit_sample_count = 3

    # Choose qubits to use.
    input_qubits = [cirq.GridQubit(i, 0) for i in range(qubit_count)]
    output_qubit = cirq.GridQubit(qubit_count, 0)

    # Pick coefficients for the oracle and create a circuit to query it.
    secret_bias_bit = random.randint(0, 1)
    secret_factor_bits = [random.randint(0, 1) for _ in range(qubit_count)]
    oracle = make_oracle(input_qubits,
                         output_qubit,
                         secret_factor_bits,
                         secret_bias_bit)
    print('Secret function:\nf(a) = a·<{}> + {} (mod 2)'.format(
        ', '.join(str(e) for e in secret_factor_bits),
        secret_bias_bit))

    # Embed the oracle into a special quantum circuit querying it exactly once.
    circuit = make_bernstein_vazirani_circuit(
        input_qubits, output_qubit, oracle)

    # Sample from the circuit a couple times.
    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=circuit_sample_count)
    frequencies = result.histogram(key='result', fold_func=bitstring)
    print('Sampled results:\n{}'.format(frequencies))

    # Check if we actually found the secret value.
    most_common_bitstring = frequencies.most_common(1)[0][0]
    print('Most common matches secret factors:\n{}'.format(
        most_common_bitstring == bitstring(secret_factor_bits)))

def make_oracle(input_qubits,
                output_qubit,
                secret_factor_bits,
                secret_bias_bit):
    """Gates implementing the function f(a) = a·factors + bias (mod 2)."""

    if secret_bias_bit:
```

```
        yield cirq.X(output_qubit)

    for qubit, bit in zip(input_qubits, secret_factor_bits):
        if bit:
            yield cirq.CNOT(qubit, output_qubit)

def make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle):
    """Solves for factors in f(a) = a·factors + bias (mod 2) with one query."""

    c = cirq.Circuit()

    # Initialize qubits.
    c.append([
        cirq.X(output_qubit),
        cirq.H(output_qubit),
        cirq.H.on_each(*input_qubits),
    ])

    # Query oracle.
    c.append(oracle)

    # Measure in X basis.
    c.append([
        cirq.H.on_each(*input_qubits),
        cirq.measure(*input_qubits, key='result')
    ])

    return c

def bitstring(bits):
    return ''.join(str(int(b)) for b in bits)

if __name__ == '__main__':
    main()
```

See also

- Hidden linear function problem

References

1. Ethan Bernstein and Umesh Vazirani (1997). "Quantum Complexity Theory". *SIAM Journal on Computing*. **26** (5): 1411–1473. doi:10.1137/S0097539796300921 (https://doi.org/10.1137%2FS0097539796300921).
2. S D Fallek, C D Herold, B J McMahon, K M Maller, K R Brown, and J M Amini (2016). "Transport implementation of the Bernstein–Vazirani algorithm with ion qubits". *New Journal of Physics*. **18**. doi:10.1088/1367-2630/aab341 (https://doi.org/10.1088%2F1367-2630%2Faab341).
3. The Cirq Developers. "Implementation of the Bernstein-Vazirani algorithm" (https://github.com/quantumlib/Cirq/blob/master/examples/bernstein_vazirani.py). Retrieved 2019-06-30.