1 Find the Hidden String Using Simon's Algorithm

Simon's algorithm is a quantum algorithm that demonstrates an exponential speedup over the best-known classical algorithms for a specific problem known as Simon's problem. It finds a hidden bit string $s \in \{0,1\}^n$ associated with a function $f : \{0,1\}^n \to \{0,1\}^n$. The algorithm is a cornerstone of quantum computing as it inspired Shor's algorithm for factoring. Unlike deterministic algorithms like Bernstein-Vazirani, Simon's algorithm is probabilistic and requires a classical post-processing step to determine the final answer.

1.1 Problem Statement

The problem involves a "black-box" function (oracle) f that is guaranteed to have the following property for a secret, non-zero string $s \in \{0,1\}^n$:

$$f(x) = f(y) \iff x \oplus y = s \text{ or } x \oplus y = 0^n$$

This means the function is 2-to-1, where every output corresponds to exactly two inputs, x and $x \oplus s$. The goal is to find the hidden string s efficiently. A classical approach would require, on average, $O(2^{n/2})$ queries, whereas Simon's algorithm can solve it in $O(n)$ queries.

1.2 Algorithm Implementation

The implementation uses a hybrid quantum-classical approach. The quantum part is used to generate vectors that satisfy a specific linear relationship with s, and the classical part solves a system of linear equations to find s.

1.2.1 Circuit Preparation

The quantum circuit is initialized with $2n$ qubits, organized into two registers of n qubits each.

- The first register (input) is initialized to the $|0\rangle^{\otimes n}$ state. Hadamard gates (H) are applied to each qubit in this register to create a uniform superposition of all possible input strings:
$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0\rangle^{\otimes n}$$
- The second register (output) remains in the $|0\rangle^{\otimes n}$ state.

1.2.2 Oracle U_f

The oracle U_f applies the function f to the superposition state. It is a unitary transformation defined as:

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$$

Applying the oracle to our prepared state yields:

$$|\psi_2\rangle = U_f |\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

The implementation of the oracle in the provided code first copies the input register to the output register using CNOT gates, and then applies additional CNOTs based on the secret string s to enforce the 2-to-1 property.

1.2.3 Re-apply Hadamard Gate and Measurement

After the oracle call, a second set of Hadamard gates is applied to the first (input) register. This interference step is the core of the quantum part of the algorithm. Measuring the first register yields a random n-bit string z that is guaranteed to satisfy the condition:

$$z \cdot s = 0 \pmod 2$$

This provides one linear equation about the bits of s. However, a single z is not enough to determine s.

1.2.4 Classical Post-Processing

The quantum circuit is run multiple times to obtain a set of vectors $\{z_1, z_2, \ldots, z_k\}$. The goal is to collect $n-1$ linearly independent vectors. These vectors form a system of linear equations over the field F_2:

$$\begin{cases} z_1 \cdot s = 0 \\ z_2 \cdot s = 0 \\ \vdots \\ z_{n-1} \cdot s = 0 \end{cases}$$

This system can be solved efficiently using classical methods, such as Gaussian elimination, to find the unique non-zero solution, which is the secret string s. The provided code implements helper functions ‘rank$_F$2‘$to check for linear independence and$‘$solve_linear_system$‘$to finds$.

1.3 Code Implementation

The Python code uses Qiskit to implement Simon's algorithm. The key components are:

- `main(n)`: Orchestrates the entire process. It generates a random secret string s, builds the circuit, and repeatedly runs it using the ‘Sampler‘ primitive until $n-1$ linearly independent ‘z‘ vectors are found.
- `create_simon_oracle(s)`: Constructs the quantum oracle U_f for a given secret string s.
- `rank_F2(matrix)`: A classical helper function to compute the rank of a binary matrix over F_2, used to check for linear independence of the measured z vectors.
- `solve_linear_system(matrix, n)`: A classical function that solves the system of linear equations $Z \cdot s = 0$ using Gaussian elimination to find s.

1.4 Test Case and Results

The code was executed for the case where $n = 4$. The results are summarized below.

- **Secret String** s: A random non-zero string was generated: `0111`.

- **Quantum Execution**: The circuit was run multiple times to find 3 linearly independent ‘z‘ vectors.

Table 1: Measurement Results for $n = 4, s = 0111$

Run	Measured z	Dot Product $z \cdot s$	Status	Linearly Independent?
1	`0110`	0	Correct	Yes (1/3)
2	`0011`	0	Correct	Yes (2/3)
3	`0110`	0	Correct	No (Dependent)
4	`1000`	0	Correct	Yes (3/3)

The linearly independent vectors found were:

$$Z = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

- **Classical Solution**: Solving the system $Z \cdot s = 0$ yielded the calculated string `0111`.

- **Outcome**: The algorithm successfully identified the secret string.
  Original ‘s‘: 0111
  Calculated ‘s‘: 0111.

1.5 Circuit Diagram

The quantum circuit diagram for the $n = 4$ case with the secret string $s = 0111$ is shown below. It consists of 8 qubits (4 for the input register, 4 for the output register) and 4 classical bits for the measurement outcome.

1.6 Conclusion

Simon’s algorithm was successfully implemented using Qiskit. The hybrid quantum-classical approach correctly identified the hidden bit string s by generating a system of linear equations through repeated quantum measurements and solving it classically. The experiment for $n = 4$ confirms that the algorithm performs as expected, finding $n - 1$ linearly independent vectors and recovering s. This successfully demonstrates the exponential quantum advantage of Simon’s algorithm for this specific problem.

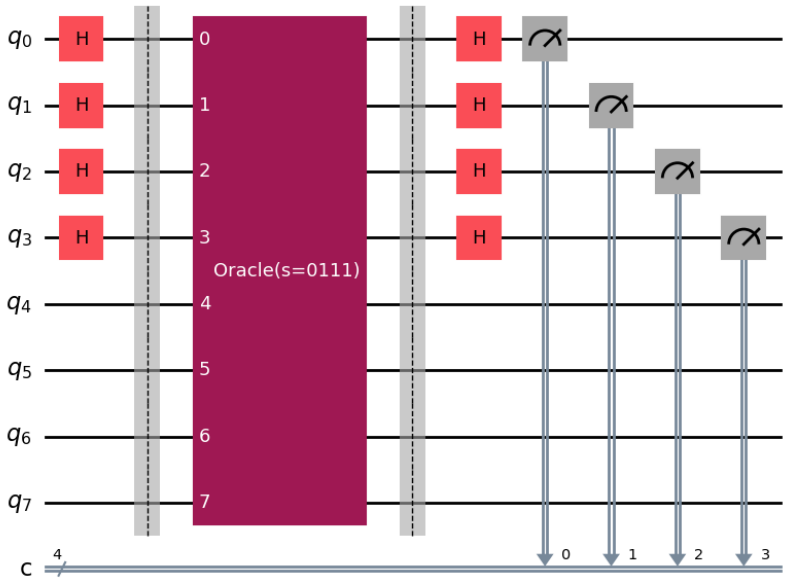

Figure 1: Simon's Algorithm Circuit for $n = 4$ and $s = 0111$