Grover's Quantum Search Algorithm for $O(\sqrt{N})$ Speedup in Unstructured Databases

Ankit Pradhan, Venu Madhav Yatam

IIT Bhubaneswar

April 26, 2019

Introduction

Grover's algorithm is a quantum algorithm that discovers the unique input to an oracle (black box) function that produces a known output value, using just $O(\sqrt{N})$ evaluations of the function, where N is the size of the function's domain.

This algorithm was devised by Lov Grover in 1996, an Indian-American computer scientist.

Lov Glover

Figure 1: Lov Grover

Introduction

The analogous search problem in classical computation cannot be solved in fewer than $O(N)$ evaluations. At roughly the same time that Grover published his algorithm, Bennett, Bernstein, Brassard, and Vazirani proved that any quantum solution to the problem needs to evaluate the function $\Omega(\sqrt{N})$ times, so Grover's algorithm is asymptotically optimal.

Implications for Cryptography

Unlike other quantum algorithms, which may provide exponential speedup over their classical counterparts (Shor's Factoring Algorithm), Grover's algorithm provides only a quadratic speedup. However, even quadratic speedup is considerable when N is large. Grover's algorithm could brute-force a 128-bit symmetric cryptographic key in roughly 2^{64} iterations, or a 256-bit key in roughly 2^{128} iterations. As a result, it is sometimes suggested that symmetric key lengths be doubled to protect against future quantum attacks.

Oracle

Given a search space of N elements, we use the index of an element as the primary search key. This is a number in the range 0 to $N - 1$.

For convenience we make the following assumptions:

- $N = 2^n$, enabling the index to be stored in n bits.
- The search problem has exactly M solutions, with $1 \leq M \leq N$.

We can represent a particular instance of the search problem by a function f, taking as input an integer $x \in \{0, 1, ..., N - 1\}$ such that

$$f(x) = \begin{cases} 1 & x \text{ is a solution to the search problem} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Oracle

Suppose a quantum oracle (black box) with the ability to "recognize" solutions to the search problem is available. An oracle qubit q is used to signal this recognition and the oracle is modelled as an unitary operator O which operates as:

$$|x\rangle |q\rangle \xrightarrow{O} |x\rangle |q \oplus f(x)\rangle$$

where

- $|x\rangle$: index register
- $\oplus$: addition modulo 2
- $|q\rangle$: oracle qubit (value is flipped if $f(x) = 1$, otherwise unchanged)

It can be verified that x is a solution to the search problem by preparing $|x\rangle |0\rangle$, applying the oracle, and checking to see if the oracle qubit has been flipped to $|1\rangle$.

Oracle

It is useful to prepare the oracle qubit in the state

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

If x is not a solution to the search problem, applying the oracle to the state $|x\rangle |-\rangle$ does not change the state. Otherwise if x is a solution, then the final state is $-|x\rangle |-\rangle$. Thus the oracle exihibits the following effect:

$$|x\rangle |-\rangle \xrightarrow{O} (-1)^{f(x)} |x\rangle |-\rangle$$

The oracle thus *marks the solutions* of the search problem by *inverting their phase*. It turns out that for an N item search problem with M solutions, we need only apply the search oracle $O(\sqrt{N/M})$ times to obtain a solution on a quantum computer.

The Procedure

The search algorithm operates as shown in the diagram below.

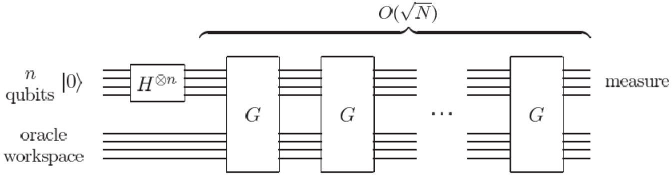


Figure 2: Schematic Circuit for Grover's Search Algorithm

The algorithm requires a single n qubit register. Since the internal workings of the oracle along with the extra work qubits needed by it are not necessary for describing the search algorithm, we omit the details here.

The Procedure

Firstly, the algorithm initializes the circuit to a state $|0\rangle^{\otimes n}$ and a Hadamard transform converts this state to an equal superposition state

$$|\psi\rangle = \frac{1}{N^{1/2}} \sum_{x=0}^{N-1} |x\rangle$$

The algorithm then repeatedly applies a quantum subroutine, know as the **Grover iteration** or **Grover operator**, denoted by **G**. The quantum circuit for Grover iteration is as follows:

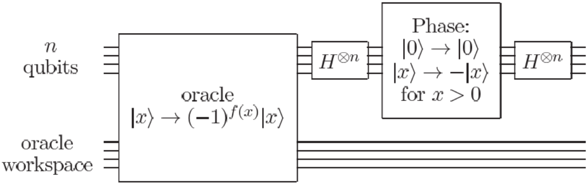


Figure 3: Circuit for Grover Iteration G

The Procedure

The four steps below illustrate the operation of the operation of the Grover Iteration circuit:

- Apply the oracle O
- Apply the Hadamard transform $H^{\otimes n}$
- Perform a conditional phase shift, with every computational basis state except $|0\rangle$ receiving a phase shift of -1, $|x\rangle|-\rangle \rightarrow -(-1)^{\delta x0}|x\rangle|-\rangle$.
- Apply the Hadamard transform $H^{\otimes n}$.

Each of the operation in the Grover iteration can be efficiently implemented on a quantum computer. Steps 2 and 4, require $n = log_2(N)$ operations each while Step 3, may be implemented using $O(n)$ gates. The cost of the oracle call depends upon the specific application, but for now, we note that the Grover iteration requires only a single oracle call.

The Procedure

The combined effect of steps 2, 3, and 4 reduces the initial state to:

$$H^{\otimes n}(2\,|0\rangle\,\langle 0| - I)H^{\otimes n} = 2\,|\psi\rangle\,\langle\psi| - I$$

where $|\psi\rangle$ is the equal superposition state.
Thus the Grover iteration, G, may be written as

$$G = (2\,|\psi\rangle\,\langle\psi| - I)O$$

G can be regarded as a rotation in the two-dimensional space spanned by the starting vector $|\psi\rangle$ and the state consisting of a uniform superposition of solutions to the search problem. This improves the probability amplitude of the state associated with the solution of the search algorithm.

The Procedure

The upper bound on the number of iterations required for this purpose:

$$R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil$$

Thus $R = O(\sqrt{N/M})$ Grover iterations (equivalently oracle calls) must be performed to obtain a solution to the search problem with high probability, which is a quadratic improvement over the $O(N/M)$ oracle calls required classically.

The algorithm

Inputs: (1) a black box oracle O which performs the transformation $O|x\rangle|q\rangle = |x\rangle|q \oplus f(x)\rangle$, where $f(x) = 0$ for all $0 \leq x < 2^n$ except x_0, for which $f(x_0) = 1$; (2) $n+1$ qubits in the state $|0\rangle$.

Outputs: x_0.

Runtime: $O(\sqrt{2^n})$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle^{\otimes n}|0\rangle$ — initial state
2. $\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$ — apply $H^{\otimes n}$ to the first n qubits, and HX to the last qubit
3. $\rightarrow \left[(2|\psi\rangle\langle\psi| - I)O\right]^R \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \approx |x_0\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$ — apply the Grover iteration $R \approx \lceil \pi\sqrt{2^n}/4 \rceil$ times
4. $\rightarrow x_0$ — measure the first n qubits

Figure 4: Grover's Algorithm

The algorithm

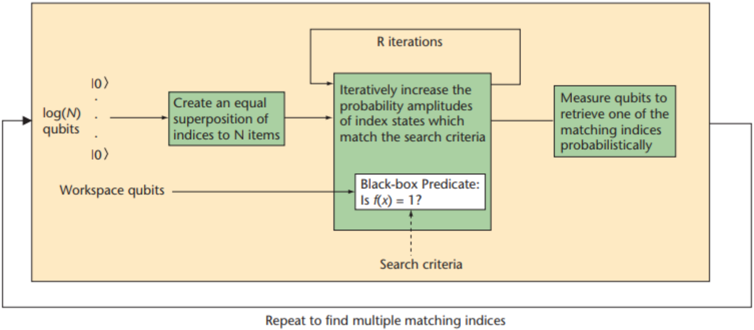


Figure 5: Circuit for Grover's Algorithm

Quantum Search on an Unstructured Database

Consider a database with $N \equiv 2^n$ items, each of length l bits labelled as $d_1, ..., d_N$. The organization to determine where a particular l bit string, s, is in the database follows.

The quantum computer consists of two units, a CPU and a memory. Assuming that the CPU contains four registers:

- An n qubit **index** register initialized to $|0\rangle$
- An l qubit register initialized to $|s\rangle$ and the register remains in the same state for the entire computation
- An l qubit **data** register initialized to $|0\rangle$
- A 1 qubit register initialized to $|-\rangle$

Organization

The memory unit can be implemented in the following ways.

- A quantum memory containing $N = 2^n$ cells of l qubits each, containing the database entries $|d_x\rangle$
- A classical memory containing $N = 2^n$ cells of l bits each, containing the database entries d_x

But unlike the traditional classical memory, this implementation can be used to address a quantum index x which can be in a superposition of multiple values allowing a superposition of cell values to be loaded from memory.

Memory Access

Memory access works in the following way: if the CPU's index register is in the state $|x\rangle$ and the data register is in the state $|d\rangle$, then the contents d_x of the x^{th} memory cell are added to the data register: $|d\rangle \rightarrow |d \oplus d_x\rangle$, where the addition is done bitwise, modulo 2.

Quantum Addressing

In order for the oracle to function correctly on superpositions it may seem as though the memory needs to be quantum mechanical. But with some caveats, the memory can actually be implemented classically, making it much more resistant to the effects of noise. Here, a quantum addressing scheme is needed. The picture in the next slide depicts a conceptual diagram of a 32 cell classical memory with a 5 qubit quantum addressing scheme.

Quantum Addressing

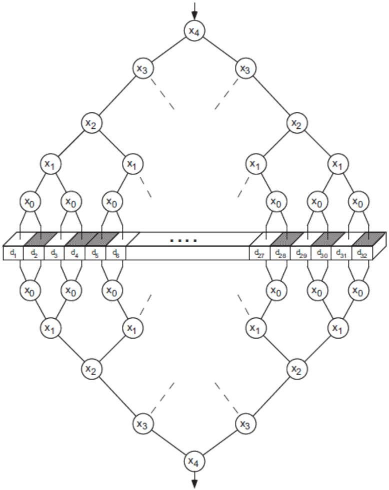


Figure 6: 32 cell classical memory with 5 qubit quantum addressing scheme

Quantum Addressing

Each circle represents a switch which addresses the qubit inscribed within it. For example, when $|x_4\rangle = |0\rangle$, the corresponding switch routes the input qubit towards the left and when $|x_4\rangle = |1\rangle$ the switch routes the input qubit to the right. If $|x_4\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, then an equal superposition of both routes is taken. The data register qubits enter at the top of the tree, and are routed down to the database, which changes their state according to the contents of the memory. The qubits are then routed back into a definite position, leaving them with the retrieved information.

Physically, this could be realized using, for example, single photons for the data register qubits, which are steered using nonlinear interferometers.

Effect of Oracle

The key part of implementing the quantum search algorithm lies in the implementation of the oracle, which must flip the phase of the index which locates s in the memory. Suppose the CPU is in the state $|x\rangle |s\rangle |0\rangle |-\rangle$, applying the *LOAD* operation transforms this state to $|x\rangle |s\rangle |d_x\rangle |-\rangle$. Then, the second and third registers are compared, and if they are the same, then a bit flip is applied to register 4, otherwise nothing is changed. The effect of this operation is

$$|x\rangle |s\rangle |d_x\rangle |-\rangle \rightarrow \begin{cases} -|x\rangle |s\rangle |d_x\rangle |-\rangle & d_x = s \\ |x\rangle |s\rangle |d_x\rangle |-\rangle & d_x \neq s \end{cases} \quad (2)$$

Effect of Oracle

The data register is then restored to the state $|0\rangle$ by performing the *LOAD* operation again. The total action of the oracle thus leaves registers 2, 3 and 4 unaffected, and unentangled with register 1. Using this oracle's

implementation we can apply the quantum search algorithm to determine the location of s in the database, using $O(\sqrt{N})$ *LOAD* operations, compared to the N *LOAD* operations that were required classically.

Example: Search on an index of size 4

Consider four locations represented by two qubits and the data at each location as follows.

$$
\begin{aligned}
|00\rangle &\rightarrow b \\
|01\rangle &\rightarrow d \\
|10\rangle &\rightarrow c \\
|11\rangle &\rightarrow a
\end{aligned}
$$

Search key : c

As the number of elements is four, at max 1 iteration is sufficient for the Grover's search algorithm.

Example: Search on an index of size 4

- **Hadamard Operation**

$$|0\rangle \xrightarrow{H} (|0\rangle + |1\rangle)/\sqrt{2}$$
$$|1\rangle \xrightarrow{H} (|0\rangle - |1\rangle)/\sqrt{2}$$

Output of First Hadamard gate :
$(|00\rangle + |01\rangle + |10\rangle + |11\rangle)/2$

First Iteration :

- **Oracle's Output** - $(|00\rangle + |01\rangle - |10\rangle + |11\rangle)/2$
- **Hadamard Operation's Output** - $(|00\rangle - |01\rangle + |10\rangle + |11\rangle)/2$
- **Phase shift** - $(|00\rangle + |01\rangle - |10\rangle - |11\rangle)/2$
- **Hadamard Operation's Output** - $|10\rangle$

Classical Implementation

```
def oracle(amp):
    for k, v in list(amp.items()):
        if hash(k[1]) == target:
            amp[k] = v * -1
    return amp
def grover(key, data):
    n = len(data)
    rounds = int((pi / 4) * sqrt(n))
    def _grover(target, objects, n, rounds):
        y_pos = np.arange(n)
        tuples = [(i, objects[i]) for i in range(n)]
        amp = OrderedDict.fromkeys(tuples, 1/sqrt(n))
        for i in range(0, rounds):
            amp = oracle(amp)
            avg = mean(amp.values())
            for k, v in list(amp.items()):
                if oracle(k[1]) == target:
                    amp[k] = (2 * avg) + abs(v)
                    continue
                amp[k] = v-(2*(v-avg))
        return amp
    amp = _grover(key, data, n, rounds)
    return amp, max(amp, key=amp.get)
```

Implementation on HR Database

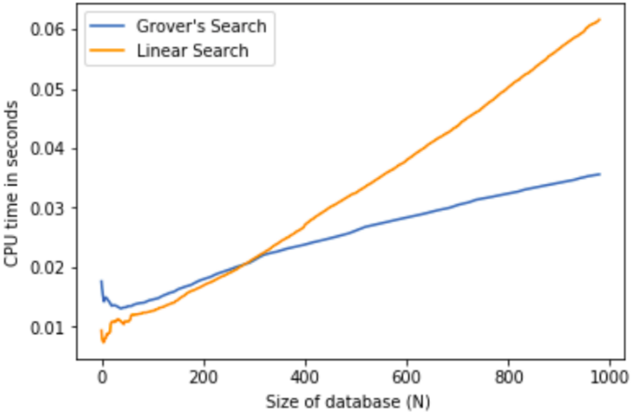
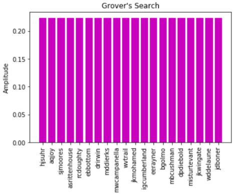
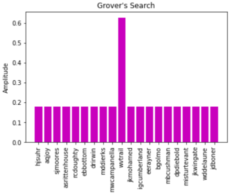


Figure 7: Average running time for Linear and Grover's Search Algorithms on HR Database

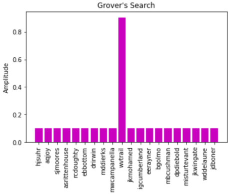
Amplitude Change wrt Rounds



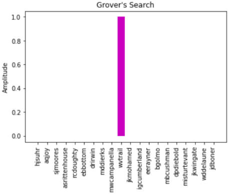
(a) Initialization



(b) After round 1



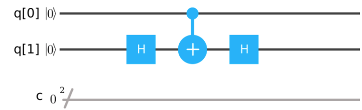
(c) After round 2



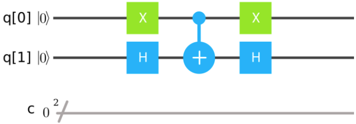
(d) After round 3

Figure 8: Amplitude changes wrt number of rounds for key username 'wvtrail' among 20 usernames

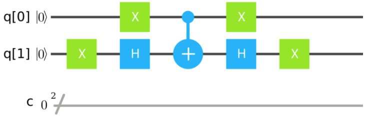
IBM Quantum Experience



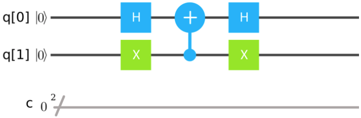
(a) 11



(b) 10



(c) 00



(d) 01

Figure 9: Exemplary Oracle for 2 qubits

IBM Quantum Experience

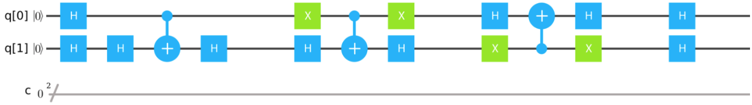


Figure 10: Grover's Diffusion Function

Bibliography

L. Grover, *A fast quantum mechanical algorithm for database search*, in Proc. 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 212–219.

M. Nielsen and I. Chuang, *Quantum computation and Quantum information*, Cambridge: Cambridge Univ. Press, 2000.

Wikipedia, *Grover's Algorithm*, `https://en.wikipedia.org/wiki/Grover%27s_algorithm`

Thank You!