CmpE 598 Lecture Notes

Kayhan Ata

April 10, 2018

1 Grover's Algorithm

Up to now, we have dealt with algorithms that use fixed finite memory, Grover's algorithm uses more memory as the problem gets bigger. And there are arbitrarily many qubits to get more power by using superposition.
Assume that we are given a program which can compute a function

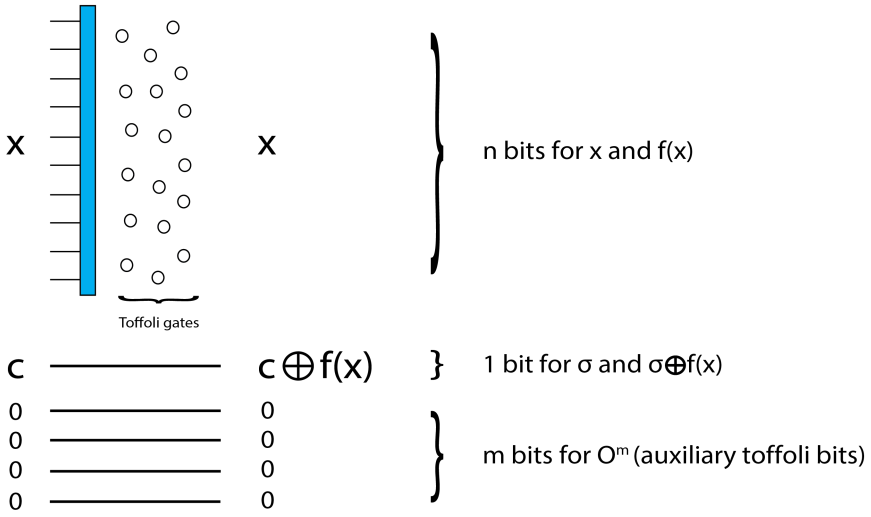
$$f : \{0,1\}^n \longmapsto \{0,1\}$$

We can't see the solution by just looking at the code due to complicated boolean formulas, although it is easy to write such a function evaluater

f has only one n-bit input which makes it true, all other 2^{n-1} inputs make it false and we want to find that particular input. The classical algorithms run $poly(n)2^n$ time while Grover's algorithm solves it in $poly(n)2^{n/2}$, providing a quadratic improvement.
We will use $n + 1 + m$ qubits where m is large enough so we can compute the transformation

$$|x\sigma O^m\rangle \longmapsto |x(\sigma \oplus f(x))O^m\rangle$$

We know from earlier lectures that classical AND and NOT gates can be represented by a Toffoli gate which is also a quantum gate.



1.1 Grover's Search Algorithm

Initialize everything to zero;
Apply Hadamard operations to first n qubits;
for $i = 1...2^{n/2}$ **do**
  Step 1. Reflect around $|e\rangle$;
    1.1 Compute $|x\sigma O^m\rangle \longmapsto |x(\sigma \oplus f(x))O^m\rangle$;
    1.2 If n+$1^{st} = 1$ then multiply vector by -1, otherwise do nothing ;
    1.3 Compute $|x\sigma O^m\rangle \longmapsto |x(\sigma \oplus f(x))O^m\rangle$;
  Step 2. Reflect around $|u\rangle$;
    2.1 Apply Hadamard to first n qubits;
    2.2 Reflect around $|0\rangle$;
    2.2.1 If first n qubits are not all zero then flip the n+1^{st} qubit;
    2.2.2 If n+1^{st} qubit is 1 then multiply by -1;
    2.2.3 If first n qubits are not all zero then flip the n+1^{st} qubit ;
    2.3 Apply Hadamard to first n qubits;
end
Measure the first n qubits, check if the value "a" you read makes $f(a) = 1$.

Applying Hadamard operation to n qubits originally at 0's causes an equiprobable superposition of all states corresponding to all n bit strings, as an example for $n = 2$

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$$

Tensor product of the two vector gives

$$\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \otimes \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix}$$

Therefore the probabilities to see $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ are all 1/2.
In Step 1.2, the n+1^{st} qubit is multiplied by -1 if it is 1, therefore the composite state becomes

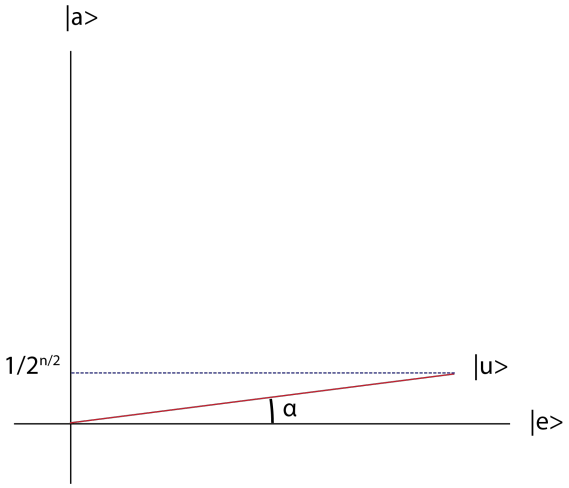
$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \otimes - \begin{pmatrix} 0 \\ 1 \end{pmatrix} = - \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

In Step 2.2.1, for the n+1^{st} qubit 0 becomes 1 and 1 becomes 0.

1.2 Graphical Description of the Algorithm

Let $u = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}} |x\rangle$ and let "a" be the special input that we are looking for.
Define "e" as the equal superposition of all 2^{n-1} basis other than "a", that is the probability to see all 2^{n-1} vectors on "e" are equal but the probability to see "a" is zero.

$$|e\rangle = \sum_{x \neq a} |x\rangle$$

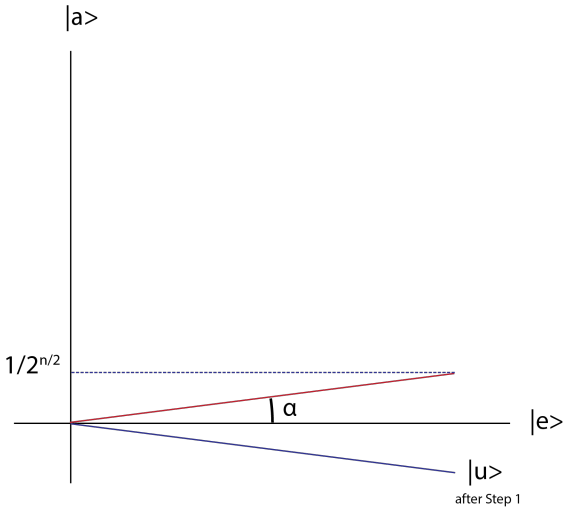


If we write u in an explicit form;

$$u = \frac{1}{2^{n/2}}|00...0\rangle + \frac{1}{2^{n/2}}|00...1\rangle + ... + \frac{1}{2^{n/2}}|a\rangle + ... + \frac{1}{2^{n/2}}|11...1\rangle$$

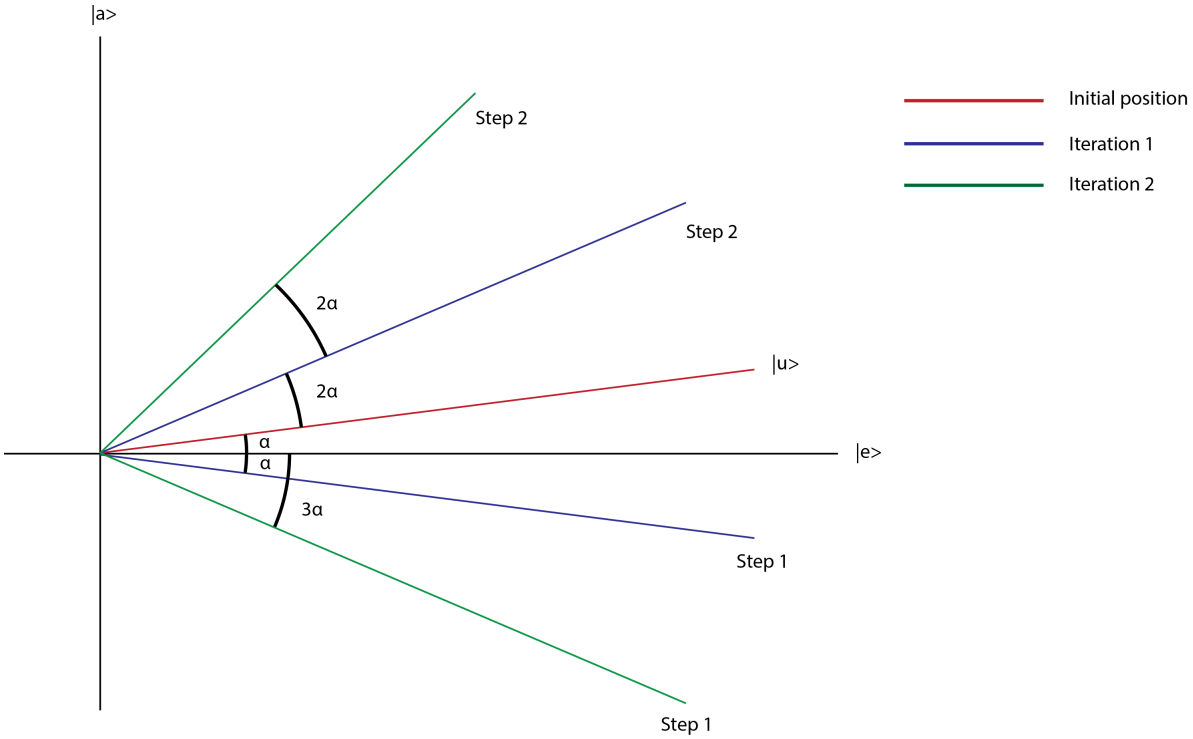
After Step 1, the sign of "a" changes but the rest of the terms remain same, therefore vector "u" is reflected around "e"

$$u = \frac{1}{2^{n/2}}|00...0\rangle + \frac{1}{2^{n/2}}|00...1\rangle + ... - \frac{1}{2^{n/2}}|a\rangle + ... + \frac{1}{2^{n/2}}|11...1\rangle$$



In Step 2.1, by applying Hadamard operation the coordinate transformation is done from $|u\rangle$ to $|0^n\rangle$. Then the vector is reflected around $|0^n\rangle$ in Step 2.2. Finally, in Step 2.3 the coordinate transformation is done from $|0^n\rangle$ to $|u\rangle$ by Hadamard operation. After Step 2, the vector is reflected around "u".

After several iterations "u" gets closer to "a" as seen in the figure



We know α initially, therefore we know how many times to do iterations. If n is big, then $1/2^{n/2}$ is small and for small angles $sin\alpha \approx \alpha$. Initially "u" needs to cover

$$\frac{\pi}{2} - arcsin\left(\frac{1}{2^{n/2}}\right)$$

in order to get close to "a" and in each iteration it covers

$$2arcsin\left(\frac{1}{2^{n/2}}\right)$$

Therefore $O(2^{n/2})$ iterations are enough.

2 Shor's Algorithm for Factorization

Given a positive integer find its factors.
There exists a fast classical algorithm for detecting whether the number is prime. If so, problem solved.
There exists another classical algorithm for detecting whether the number is a power, a^b for a,b is integer and $b > 1$:

$$x = a^b$$

$$logx = bloga \Longrightarrow \text{b can not be big}$$

If you can find a factor, you can find all other factors using the same method again and again. Shor's algorithm enables an exponential improvement over classical algorithms. Next lecture we will assume we are given number pq where p and q are primes.