Quantum Computation **(CMU 15-859BB, Fall 2015)**

Lecture 4: Grover's Algorithm

September 21, 2015

Lecturer: John Wright *Scribe: Tom Tseng*

1 Introduction

In the previous lecture, we discussed how a quantum state may be transferred via quantum teleportation and how quantum mechanics is, in a sense, more powerful than classical mechanics in the CHSH game. Today, we are going to move on to discussing how we can apply quantum mechanics to computer science. We will see our first example of a quantum algorithm: Grover's algorithm, described in a paper published by Lov Grover in 1996 [Gro96]. Much of the excitement over quantum computation comes from how quantum algorithms can provide improvements over classical algorithms, and Grover's algorithm exhibits a quadratic speedup.

2 Grover's algorithm

2.1 What it solves

Grover's algorithm solves the problem of *unstructured search*.

Definition 2.1. In an *unstructured search* problem, given a set of N elements forming a set $X = \{x_1, x_2, \ldots, x_N\}$ and given a boolean function $f : X \to \{0, 1\}$, the goal is to find an element x^* in X such that $f(x^*) = 1$.

Unstructured search is often alternatively formulated as a database search problem in which we are given a database and we want to find an item that meets some specification. For example, given a database of N names, we might want to find where your name is located in the database.

The search is called "unstructured" because we are given no guarantees as to how the database is ordered. If we were given a sorted database, for instance, then we could perform binary search to find an element in logarithmic time. Instead, we have no prior knowledge about the contents of the database. With classical circuits, we cannot do better than performing a linear number of queries to find the target element.

Grover's algorithm leads to the following theorem:

Theorem 2.2. *The unstructured search problem can be solved in $\mathcal{O}(\sqrt{N})$ queries using quantum computation.*

In fact, this is within a constant factor of the best we can do for this problem under the quantum computation model, i.e. the query complexity is $\Theta(\sqrt{N})$ [BBBV97]. In particular, since we cannot solve the unstructured search problem in logarithmic time, this problem cannot be used as a way to solve NP problems in polynomial time; if we did have a logarithmic time algorithm, we could, given n variables arranged in clauses, solve the SAT problem by searching all 2^n possibilities in $\mathcal{O}(\log(2^n)) = \mathcal{O}(n)$ queries after a few manipulations. Nonetheless, solving the search problem in $\Theta(\sqrt{N})$ queries is still significantly better than what we can do in the classical case.

In addition, Grover's algorithm uses only $\mathcal{O}(\sqrt{N}\log N)$ gates. If we think of the number of gates as being roughly running time, then we have running time almost proportional to $\sqrt{N}$.

There is a caveat: the algorithm only gets the correct answer with high probability, say with probability greater than $\frac{2}{3}$. Of course, we can then make the probability of correctness an arbitrarily large constant. We do this by running the algorithm multiple times to get many different outputs. Then we can run each of our inputs through f to see if any match the criterion desired. This strategy only fails if all outputs fail, and since these are all independent outputs, our probability of failure is $\left(\frac{1}{3}\right)^m$.

Note that even with classical algorithms that only need to output the correct answer with two-thirds probability, we still need a linear number of queries. Therefore, this caveat does not provide an unfair advantage to the quantum algorithm.

2.2 The classical case

We need some sort of model for how we are allowed to interact with our database. In a classical algorithm, our medium of interaction is an *oracle* O_f that *implements* a function f. The function f encodes information about the element of interest, and we query the oracle to fetch results from the function. Think of the oracle as a quantum circuit or a big gate. This oracle, given an input of i, outputs $f(i)$ in constant time. We do not care about the details of how it works, and instead treat it as a black box.

As we do with any good algorithm, we want to limit resources used, which in this case are running time and number of queries to the oracle. Of course, in any classical algorithm, the best we can do is $\Theta(N)$ queries and $\Theta(N)$ running time – because the data is unstructured, we cannot help but look at every element in the worst case no matter how cleverly we choose our queries.

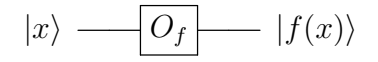
(Note that, in general, number of queries and running time are not the same, but in this case they are both linear. We care more about queries because they are simpler to reason about, plus they represent a lower bound to running time. Running time by itself involves many other factors that we do not know much about. In this course, we will be more concerned about number of queries as a measure of resources used.)

This is the end of the story in the classical situation; we have tight upper and lower bounds, and not much more can be done. Now we shall switch to the quantum setting.

2.3 Modifications in the quantum case

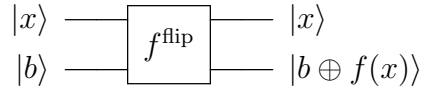
Right off the bat, we're going make some simplifications. First, we are going to assume that the element x^* that satisfies $f(x^*) = 1$ is unique. Second, we are going to assume the size of our database is is a power of two, letting $N = 2^n$ where N is the size of the database. Lastly, we are also going assume that the data is labeled as n-bit boolean strings in $\{0,1\}^n$ rather than being indexed from 1 to N. In turn, our boolean function f that we are studying will map $\{0,1\}^n$ to $\{0,1\}$. These conditions will be more convenient for us, and we do not lose much; the algorithm can be extended to relax the uniqueness assumption, we can always add garbage entries to the database to make the size of the database a power of two, and the naming of the elements is arbitrary.

Our interface with our database is not much different compared to the interface in the classical case. We have an oracle O_f that we give $|x\rangle$, and it will output $|f(x)\rangle$, as shown below:



We immediately see a problem: this oracle gate is not a valid quantum gate. It takes an n-bit input and gives a 1-bit output, and thus is not unitary or even reversible. There are a couple of ways we can fix this.

Our first possible fix is to give the gate an extra bit $|b\rangle$ to use for the output. Call this new gate the f^{flip} gate.

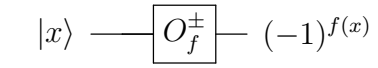


In particular, if we have $|b\rangle$ be an ancillary $|0\rangle$ qubit, it will end up as $|f(x)\rangle$ after passing through the gate. However, this gate is somewhat unwieldy because we have to carry around that extra qubit $|b\rangle$ with us. That brings us to our next fix.

Our second possible fix is to flip the input if and only if $f(x)$ is 1. We will call the gate that does this $O_f^{\pm}$. Given an input $|x\rangle$, $O_f^{\pm}$ will output

$$(-1)^{f(x)}|x\rangle = \begin{cases} |x\rangle & \text{if } f(x) = 0 \\ -|x\rangle & \text{if } f(x) = 1. \end{cases}$$

The diagram of the gate is given below.



This avoids the annoyance of the extra bit that our first fix had.

These two gates are both valid versions of the oracle, and they are equivalent in the sense that one can be constructed from the other. For example, we will prove that $O_f^{\pm}$ may be constructed using f^{flip}.

Proof. Suppose we wanted to construct the $O_f^{\pm}$ gate given access to the f^{flip} gate. Then we choose our ancillary bit that we send along with $|x\rangle$ into f^{flip} to be $|-\rangle$.

Before we hit the f^{flip} gate, we have the state

$$|x\rangle \oplus |-\rangle = |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}(|x0\rangle - |x1\rangle).$$

After we pass the gate, there are two cases. If $f(x) = 0$, then the state is unchanged. Otherwise, if $f(x) = 1$, then we end up with the state

$$|x\rangle \otimes \frac{1}{\sqrt{2}}(|0 \oplus 1\rangle - |1 \oplus 1\rangle) = |x\rangle \otimes \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = \frac{1}{\sqrt{2}}(|x1\rangle - |x0\rangle)$$

which is the negative of the state we began with. Thus we have

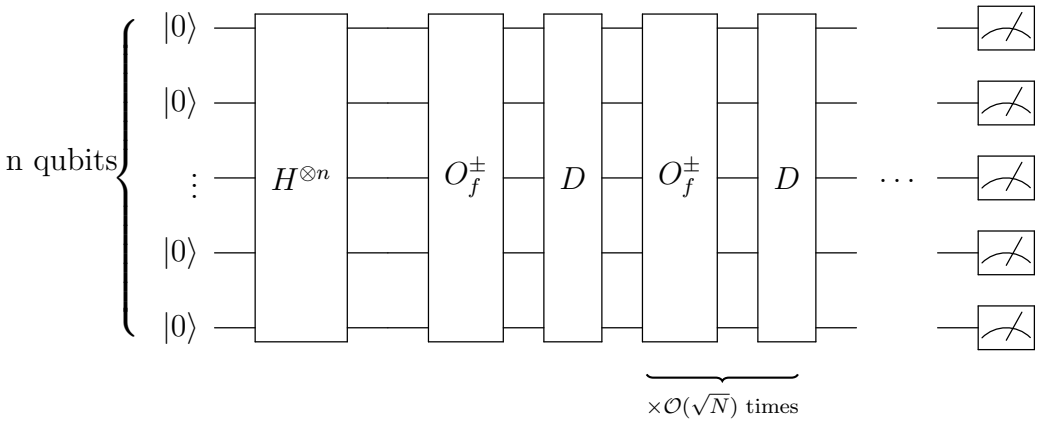
$$|x\rangle \otimes |-\rangle \mapsto (-1)^{f(x)} (|x\rangle \otimes |-\rangle)$$

as desired to simulate the $O_f^{\pm}$ gate. □

Because the gates are equivalent, we may use either. For the circuits in this lecture, we will use the $O_f^{\pm}$ gate.

2.4 The algorithm

Recall that our goal is to, given the ability to query f using our oracle gate, find x^* such that $f(x^*) = 1$. We shall start by diagramming the entire circuit below and explain the details following that.



We start with n qubits all initialized to $|0\rangle$. Think of these as forming an n-bit string, or an input to f. Let us make our goal concrete by saying that we would like the amplitude of $|x^*\rangle$ to be at least .1 in magnitude at the end of the circuit so that when we measure at the end, we have a $.1^2 = .01$ probability of measuring $|x^*\rangle$.

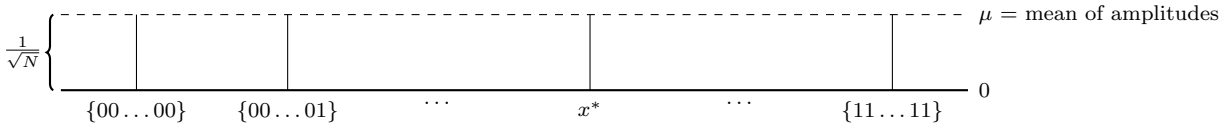
What can we do with our quantum circuit that is not possible classically? We can perhaps exploit superposition. We shall begin by transforming our input into the uniform superposition

$$\sum_{x \in \{0,1\}^n} \frac{1}{\sqrt{N}} |x\rangle .$$

The uniform amplitudes of all the n-bit strings, in a sense, represents our current complete uncertainty as to what x^* is.

We achieve this uniform superposition of all the basis states by sticking a Hadamard gate on every wire, or equivalently a $H^{\otimes n}$ gate on all wires. This superposition trick is common in quantum algorithms, so get used to it.

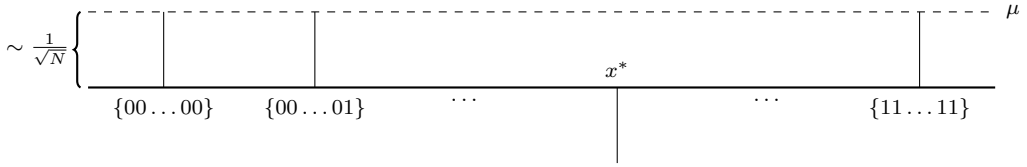
The current state can be visualized with a diagram representing amplitudes in a fashion similar to a bar graph.



Now we shall query this state by adding an oracle $O_f^{\pm}$ gate. That flips the amplitude of the x^* component and leaves everything else unchanged, giving us a state of

$$-\frac{1}{\sqrt{N}}\left|x^*\right\rangle + \sum_{\substack{x\in\{0,1\}^n \\ x\neq x^*}} \frac{1}{\sqrt{N}}\left|x\right\rangle .$$

Graphically, we now have



What we want is to increase the amplitude of x^* absolutely. This motivates the introduction of a new gate that performs what is called the *Grover diffusion operator*. What does this gate do? First, we have to define a new number

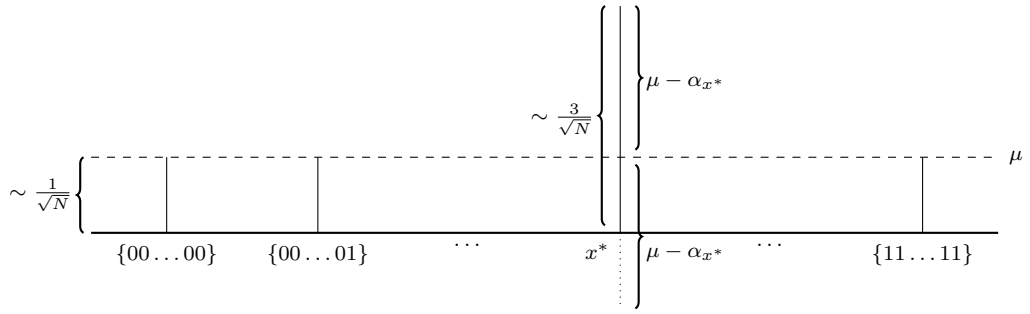
$$\mu = \frac{1}{N}\sum_x \alpha_x = \text{average of } \alpha_x \text{ for } x \in \{0,1\}^n.$$

where for α_x is the amplitude of x for a given x in $\{0,1\}^n$. Our Grover diffusion gate is defined to have the mapping

$$\sum_{x\in\{0,1\}} \alpha_x \left|x\right\rangle \mapsto \sum_{x\in\{0,1\}^n} (2\mu - \alpha_x)\left|x\right\rangle .$$

Note that this is a valid quantum gate; it is linear since all the operations in the mapping are linear, and it is also unitary. We can show it is unitary by verifying that the operation preserves the norm of the input. In addition, we will later construct the gate out of other unitary gates.

O.K., what does this gate *really* do? It flips amplitudes around the average, μ. To illustrate this, we will apply the gate, labeled D on the original circuit diagram, after the oracle gate. Here is what our amplitudes look like after application:

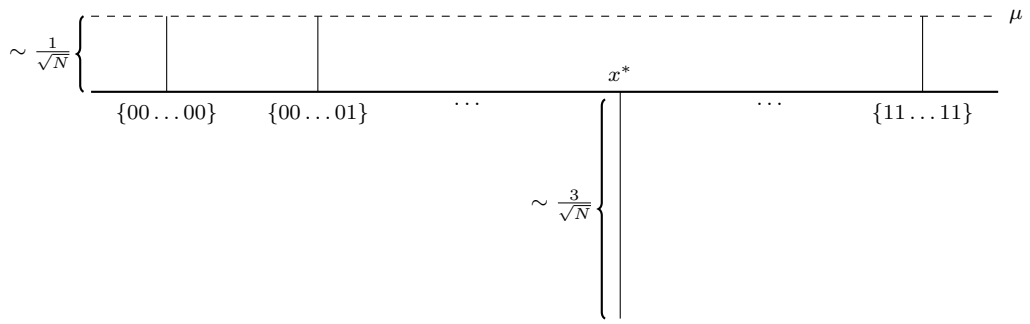


The mean amplitude prior to applying the gate was

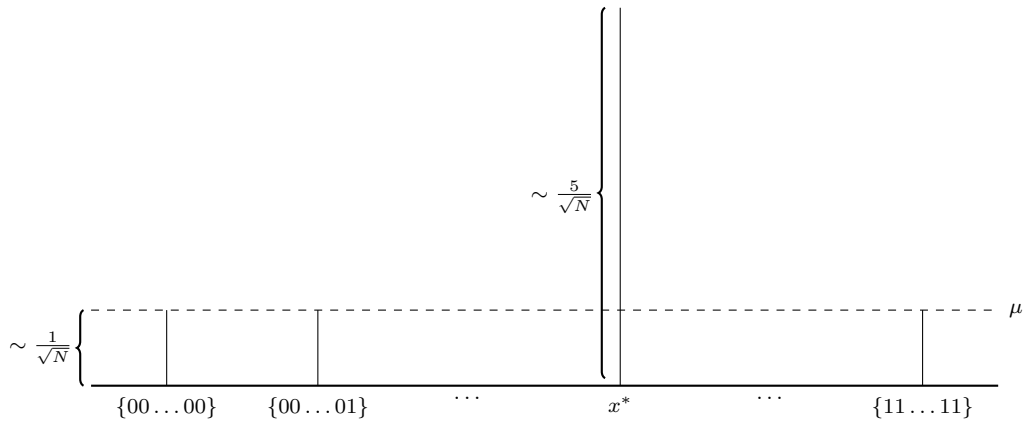
$$\sum_{x\in\{0,1\}^n} \alpha_x = \frac{2^n - 1}{\sqrt{N}} - \frac{1}{\sqrt{N}} = \frac{2^n - 2}{\sqrt{N}} \approx \frac{N}{\sqrt{N}} = \frac{1}{\sqrt{N}}$$

i.e. $\frac{1}{\sqrt{N}}$ minus an insignificant amount. That means when the "reflection around the mean" occurs with the application of the Grover diffusion operator, almost all the entries $x \in \{0,1\}^n$ stay roughly the same. But – and here is where the magic happens – the amplitude of x^* gets magnified to about $\frac{3}{\sqrt{N}}$ in magnitude!

Still, this is quite far from our goal of making the amplitude of x^* greater than a constant. To make further progress, we can try the most brain-dead thing possible: apply the oracle and the Grover diffusion gate again! After applying $O_f^{\pm}$ for the second time, we have



and after applying the Grover diffusion for the second time, we obtain



The computation to see that this is true is more or less the same. We flip x^* to its negative, and then flip it around the mean while everything else stays roughly unchanged.

The intuitive, high-level picture of what is happening is that α_{x^*} is going up by more than $1/\sqrt{N}$ each time we repeat the application of the pair of gates $O_f^{\pm}$ and D. With that logic, after $\mathcal{O}(\sqrt{N})$ steps, α_{x^*} should exceed the constant .1 as desired.

Of course, α_{x^*} cannot actually increase by $1/\sqrt{N}$ at every step since the amplitude cannot go above 1. The increase of α_{x^*} has to slow down or reverse at some point. For instance, if α_{x^*} is sufficiently large, then when we flip α_{x^*} to negative by using the $O_f^{\pm}$ gate, this large negative value might actually drag the mean down to be negative. With a negative mean, the magnitude of x^* will decrease after using the Grover diffusion gate. In light of that, we must be careful when we are analyzing this algorithm to make sure we never get that kind of negative progress. However, so long as α_{x^*} is not too large, it seems we will always step closer to our goal.

2.5 Analysis

We have constructed the circuit, and now it remains to formally show that the math behind the circuit actually works out.

First, we will introduce some notation. Let $\alpha^{(t)}$ be the amplitude of x^* after the t-th step, where one step consists of applying an $O_f^{\pm}$ gate and then a Grover diffusion gate. Let $\beta^{(t)}$ be the amplitude of any other $x \in \{0,1\}^n$ after the t-th step. Finally, let $\mu^{(t)}$ be the mean of all the amplitudes, which is $-\alpha^{(t)} + \frac{N-1}{N}\beta^{(t)}$, after the oracle gate on the t-th step.

For example, $\alpha^{(0)} = \beta^{(0)} = \frac{1}{\sqrt{N}}$. Also, if we know the values for a given step t, we can calculate $\alpha^{(t+1)} = 2\mu^{(t)} + \alpha^{(t)}$ and so forth.

Now that notation is defined, we can move on to the body of the proof. We first show that α increases by a significant amount if α is not already too large.

Proposition 2.3. *Suppose $\alpha^{(t)} \leq 1/2$ and $N \geq 4$. Then $\alpha^{(t+1)} \geq \alpha^{(t)} + \frac{1}{\sqrt{N}}$.*

Proof. The sum of all the squares of the amplitudes is one, so

$$\begin{aligned} 1 &= (\alpha^{(t)})^2 + (N-1)(\beta^{(t)})^2 \\ &\leq \frac{1}{4} + (N-1)(\beta^{(t)})^2. \end{aligned}$$

With some rearranging, we get a bound on $\beta^{(t)}$ of

$$\beta^{(t)} \geq \sqrt{\frac{3}{4(N-1)}}.$$

Therefore,

$$\begin{aligned}
\mu^{(t)} &= \frac{-\alpha^{(t)} + (N-1)\beta^t}{N} \\
&\geq \frac{-\frac{1}{2} + (N-1)\sqrt{\frac{3}{4(N-1)}}}{N} \\
&= \frac{1}{2}\frac{(N-1)\sqrt{\frac{3}{N-1}} - 1}{N} \\
&= \frac{1}{2}\frac{\sqrt{3N-3} - 1}{N} \\
&\geq \frac{1}{2}\frac{1}{\sqrt{N}} \text{ given } N \geq 4.
\end{aligned}$$

Knowing this bound on $\mu^{(t)}$, we may calculate $\alpha^{(t+1)}$ to be

$$\alpha^{(t+1)} = 2\mu^{(t)} + \alpha^{(t)} \geq \frac{1}{\sqrt{N}} + \alpha^{(t)},$$

and the result is proven. □

The next proposition tells us that the increases of $\alpha^{(t+1)}$ are relatively controlled. This is useful because it will be used to show that $\alpha^{(t)}$ stays below one-half for a number of steps so that the previous proposition may be applied.

Proposition 2.4. *For any t, $\alpha^{(t+1)} \leq \alpha^{(t)} + \frac{2}{\sqrt{N}}$.*

Proof. For any given step t, $\alpha^{(t)}$ is positive, and therefore

$$\mu^{(t)} = \frac{-\alpha^{(t)} + (N-1)\beta^{(t)}}{N} \leq \frac{N-1}{N}\beta^{(t)}.$$

We also know that $(N-1)\left(\beta^{(t)}\right)^2 \leq 1$, and so $\beta^{(t)} \leq \frac{1}{\sqrt{N-1}}$. This gives us

$$\begin{aligned}
\alpha^{(t+1)} &= 2\mu^{(t)} + \alpha^{(t)} \\
&\leq 2\frac{N-1}{N}\beta^{(t)} + \alpha^{(t)} \\
&\leq 2\frac{N-1}{N}\frac{1}{\sqrt{N-1}} + \alpha^{(t)} \\
&= 2\frac{\sqrt{N-1}}{N} + \alpha^{(t)} \\
&\leq \frac{2}{\sqrt{N}} + \alpha^{(t)},
\end{aligned}$$

which is the desired result. □

Now we can show that we get $\alpha > .1$ with $\mathcal{O}(\sqrt{N})$ steps. If $N < 16$, then $\alpha^{(t)} = \frac{1}{\sqrt{N}} \geq .25$, so we are already done. Otherwise, if $N \geq 16$, as long as $t \leq \sqrt{N}/8$ we get

$$\begin{aligned}
\alpha^{(t)} &\leq \alpha^{(0)} + \frac{2}{\sqrt{N}} t \\
&\leq \alpha^{(0)} + \frac{2}{\sqrt{N}} \frac{\sqrt{N}}{8} \\
&= \frac{1}{\sqrt{N}} + \frac{1}{4} \\
&\leq \frac{1}{2}
\end{aligned}$$

by Proposition 2.4. This means that we may apply Proposition 2.3 for $\sqrt{N}/8$ steps to get

$$\alpha^{(\sqrt{N}/8)} \geq \frac{\sqrt{N}}{8} \frac{1}{\sqrt{N}} = \frac{1}{8} > .1,$$

and so we are indeed done with $\mathcal{O}(\sqrt{N})$ steps.

Of course, with just this one pass through the circuit, our probability of measuring x^* is only $\left(\alpha^{(\sqrt{N}/8)}\right)^2 > .01$. However, if we repeat the process a constant number of times, say 110 times, our probability of finding x^* is

$$\begin{aligned}
\mathbf{Pr}[\text{one output out of 110 being } x^*] &= 1 - \mathbf{Pr}[\text{output is not } x^*]^{110} \\
&\geq 1 - .99^{110} \\
&\geq \frac{2}{3}
\end{aligned}$$

which is certainly a respectable probability.

2.6 Gate complexity

We also promised an $\mathcal{O}(\sqrt{N} \log N)$ bound on the number of gates used in the algorithm. To prove this bound, it suffices to construct the Grover diffusion gate. To do so, we will invent yet another gate Z_0 defined by

$$Z_0 |x\rangle = \begin{cases} |x\rangle & \text{if } |x\rangle = |0^n\rangle \\ -|x\rangle & \text{otherwise} \end{cases}$$

In unitary matrix form, $Z_0 = 2 |0^n\rangle \langle 0^n| - I$ where I is the $n \times n$ identity matrix. We can confirm that this matrix indeed implements Z_0 by trying different inputs. If we give the input $|0^n\rangle$, we get

$$Z_0 |0^n\rangle = 2 |0^n\rangle \langle 0^n|0^n\rangle - |0^n\rangle = 2 |0^n\rangle 1 - |0^n\rangle = |0^n\rangle$$

as desired, and if we give some input $|x\rangle$ that has no 0^n component, then we know that $\langle 0^n|x\rangle = 0$, so we get

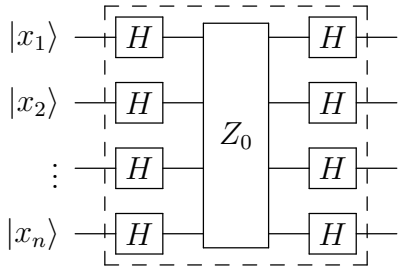
$$Z_0 |x\rangle = 2 |0^n\rangle \langle 0^n|x\rangle - |x\rangle = - |x\rangle$$

also as desired.

Now we can construct the Grover diffusion gate, D, with a Z_0 gate and $\mathcal{O}(n) = \mathcal{O}(\log N)$ Hadamard gates. In matrix form, knowing that the Hadamard gate is its own inverse, we can write the diffusion gate as

$$\begin{aligned} D &= H^{\otimes n} Z_0 H^{\otimes n} \\ &= H^{\otimes n}(2 |0^n\rangle \langle 0^n| - I) H^{\otimes n} \\ &= 2 \left((H |0\rangle)^{\otimes n} \left((H |0\rangle)^{\otimes n} \right)^\dagger \right) - H^{\otimes n} H^{\otimes n} \\ &= 2 |+^n\rangle \langle +^n| - I. \end{aligned}$$

It is the same as Z_0 but with $|+\rangle$ instead of $|0\rangle$. In diagram form, it is



Let us try giving the matrix an arbitrary input $|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$ to make sure it works. We get

$$\begin{aligned} D |\psi\rangle &= (2 |+^n\rangle \langle +^n| - I) |\psi\rangle \\ &= 2 |+^n\rangle \langle +^n|\psi\rangle - |\psi\rangle . \end{aligned}$$

Notice that we can rewrite $\langle +^n|$ to get

$$\begin{aligned} \langle +^n| &= \left(\frac{\langle 0| + \langle 1|}{\sqrt{2}} \right)^{\otimes n} \\ &= \sum_{x \in \{0,1\}^n} \frac{1}{\sqrt{N}} \langle x| , \end{aligned}$$

and therefore

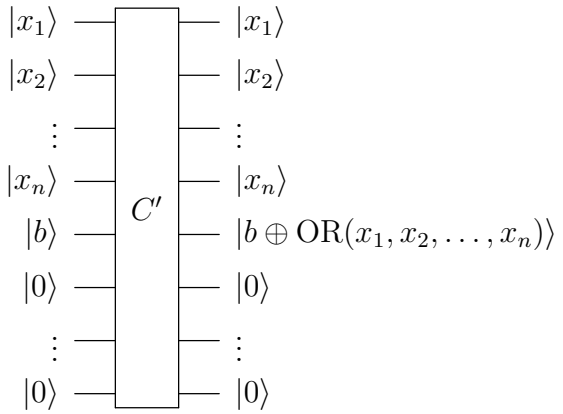
$$\begin{aligned} \langle +^n|\psi\rangle &= \sum_{x \in \{0,1\}^n} \frac{\alpha_x}{\sqrt{N}} \langle x|x\rangle \\ &= \sum_{x \in \{0,1\}^n} \frac{\alpha_x}{\sqrt{N}} \\ &= \mu \sqrt{N}. \end{aligned}$$

That means that we can continue simplifying $D|\psi\rangle$ to get

$$\begin{aligned}
D|\psi\rangle &= 2|+^n\rangle\langle+^n|\psi\rangle - |\psi\rangle \\
&= 2|+^n\rangle\mu\sqrt{N} - |\psi\rangle \\
&= 2\left(\frac{|0\rangle+|1\rangle}{\sqrt{N}}\right)\mu\sqrt{N} - |\psi\rangle \\
&= 2\left(\sum_{x\in\{0,1\}^n}\frac{1}{\sqrt{N}}|x\rangle\right)\mu\sqrt{N} - |\psi\rangle \\
&= 2\left(\sum_{x\in\{0,1\}^n}\mu|x\rangle\right) - |\psi\rangle \\
&= \sum_{x\in\{0,1\}^n}(2\mu-\alpha_x)|x\rangle
\end{aligned}$$

which is precisely what we expect from the diffusion operator.

Lastly, we need to show that we actually can construct the Z_0 gate. What does Z_0 do? It negates $|x\rangle$ if and only if the logical OR of $\{x_1, x_2, \ldots, x_n\}$ is 1, where x_i is the i-th qubit that makes up $|x\rangle$. These kinds of logical operations are easy to compute. In particular, there is a classical circuit that computes the logical OR of n bits with $\mathcal{O}(n) = \mathcal{O}(\log N)$ Toffoli or CNOT gates. This circuit is not reversible, but of course we may use the results of problem 4 on homework 1 to fix that. Given a circuit C that computes the logical OR of n bits, we can construct a circuit C' with $m \in \mathbb{N}$ ancillary bits that does the following:



Now if we send $|x\rangle \otimes |-\rangle \otimes |0^m\rangle$ into C', we get $(-1)^{\mathrm{OR}(x)}(|x\rangle \otimes |-\rangle \otimes |0^m\rangle)$ out, which is exactly what Z_0 does. There are some extra gabrage bits, but those can be ignored.

Ultimately, what this all means is that the Grover diffusion gate can be constructed with $\mathcal{O}(\log N)$ gates. That means that Grover's algorithm takes $\mathcal{O}(\sqrt{N}\log N)$ gates to implement.

3 Conclusion

We now are closely familiar with Grover's algorithm, one of the most famous quantum algorithms at present. It demonstrates a provable polynomial improvement over its classical counterpart by combining the quantum features of superposition and negative amplitudes to solve the unstructured search problem. In this lecture, we showed how to find a unique entry x^* from a database using $\mathcal{O}(\sqrt{N})$ iterations, but, in general, if there are k entries that match the desired criterion, then Grover's algorithm can be extended to find an item in $\mathcal{O}(\sqrt{N/k})$ iterations. Next lecture, we'll discuss the quantum query model and learn about some other algorithms that are based on querying the black-box implementation of a function.

References

[BBBV97] Charlies H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

[Gro96] Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996.