QuIC Seminar 5

Grover's Search Algorithm

Contents

And so the story goes...

Impressed by Simon and Shor,
Lov Grover wanted more.
"Searching takes too long—
We're doing it all wrong!"
Could quantum cause an uproar?

Though his quantum made a racket,
The speedup was quadratic.
Failure subsumed
Until Bennett penned a proof
That showed Lov was truly fantastic!

Lov Grover set out for an exponential speedup with a quantum algorithm for the classic *search problem*—given N items in no specific order, find me exactly the one I'm looking for. The best classical algorithm for such a problem is really no better than what a few Neanderthals could come up with on a rainy day: "Is this it? No. Is this it? No. Is this it? No. ... Hey I found it!"

It's clear from our "caveman algorithm" that in the worst case we'd have to ask $O(N)$ questions—exactly N questions in the worst case and $N/2$ questions on average. How could you possibly do better than this?

Lov Grover[1] had some key insight that led to a *quadratically* faster algorithm for the search problem—that is, one in which you have to ask $O(\sqrt{N})$ questions. Not only this, but his algorithm led to a general method in quantum computing known as *amplitude amplification* which could, and was, applied to many more problems.

However, Grover was initially *dis*satisfied with his technique! Here's Simon and Shor shattering ground with *exponential* speedups, but all Lov could manage was quadratic? What did he miss? How could this *mere quadratic speedup* be improved? Fortunately, it wasn't until the following year that Bennet, Bernstein, Brassard, and Vazirani published a landmark paper[2] which, among other results, demonstrated Grover's algorithm was *optimal*—you can't do better than $O(\sqrt{N})$ scaling!

We remark that these notes were based off four main resources: an article by Andy Matuschak and Michael Nielsen, a lecture by Scott Aaronson, lecture notes by Ronald de Wolf, and the classic text of Nielsen and Chuang. More details about each source, as well as links, are given in the end references.

5.1 Intuition for a Quadratic Speedup

What in the world could the quantum algorithm be doing to get $O\left(\sqrt{N}\right)$ scaling? Why the square root in particular, as opposed to a cube root or fourth root? How can we understand this intuitively?

Thankfully, per usual, Scott Aaronson has key intuition[3]. The insight can be understood in that we use an L2 norm in quantum computing rather than an L1 norm in classical probabilistic computing. That is—more specifically but still roughly speaking—suppose we assigned a number to each item in our "database"

$$[\alpha_1,\ \alpha_2,\ \cdots,\ \alpha_N] \tag{5.1}$$

In the classical case, these numbers are real numbers. We could assign a probability to each item $i = 1, ..., N$ via

$$p_i = \frac{\alpha_i}{\sum_{i=1}^{N} \alpha_i} \qquad \text{(classical)} \tag{5.2}$$

That is, p_i is the probability the ith item is the one we are searching for. Ignoring the normalization factor, we have a good chance of finding the right item, supposing it is index by m, when $p_m = O(N)$. This means that

$$\alpha_m = O(N) \qquad \text{(classical))}. \tag{5.3}$$

However, in the quantum case, we assign probabilities differently. The numbers α_i are amplitudes in the wavefunction, which are in general complex numbers. We get probabilities by taking the modulus squared of an amplitude

$$p_i = \frac{|\alpha_i|^2}{\sum_{i=1}^{N} |\alpha_i|^2} \qquad \text{(quantum)} \tag{5.4}$$

Again ignoring the normalization factor, we have a good chance of finding the right item when still $p_m = O(N)$. However, p_m now goes by $|\alpha_m|^2$. Thus, we have a good chance when

$$\alpha_m = O\left(\sqrt{N}\right) \qquad \text{(quantum)} \tag{5.5}$$

This is really just a heuristic argument (the rest of the seminar will explain Grover's algorithm in full detail) but is really at the heart of what's going on. If someone on the street asks you why Grover's algorithm is faster, you can say it's because quantum states are normalized by the L2 norm whereas classical states are normalized by the L1 norm.

5.2 The Problem Definition

We now formally define the problem we are trying to solve.

[1] L. K. Grover, A fast quantum mechanical algorithm for database search, May 1996.
[2] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, Strengths and Weaknesses of Quantum Computing, Jan. 1997.
[3] S. Aaronson, Quantum Algorithms III, Boulder School on Quantum Information, 2018

Definition 5.1 (The Search Problem). Suppose we are given integer indices $1, 2, ..., N$, which we will think of as labeling *objects* in no particular order. Without loss of generality, assume $N = 2^n$. Let integer y denote a particular object we are interested in.

Given access to an oracle $Q_s : \mathbb{Z} \to \{0, 1\}$ such that

$$Q_s(x) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

the **search problem** is to find object the y using the smallest number of queries possible[a].

[a]Note that we have defined the problem such that there is *exactly one* object y we are looking for. Variants of Grover's algorithm can solve the general case of looking for "multiple y objects." It's bad pedagogy to deal with these minor details at first, so we focus on the search problem with one and exactly one item as defined above.

We note that we are working in the *query model* of quantum computing which we have not discussed in full detail yet. Here, we give our quantum computer access to a black box which does some task for us. In the case of Grover's algorithm, the oracle (5.6) tells us if we have found the object we are looking for or not.

Initially, the query model feels like "cheating," since we are not fully working out the details of how the search black box would work. *We emphasize that this does not matter.* As long as the function Q_s is computable in the classical case, we can translate the classical circuit into a quantum circuit. We do not need to worry about whether Q_s is *efficiently* computable, since the speedup is relative.

So, in principle, all we need is the assumption that we can compute (5.6), then Grover's algorithm falls into place and we get the quadratic speedup. This is why Grover's algorithm is so powerful—it can be applied to a wide variety of problems. And it has. [References here.] Before describing how we could obtain a quantum circuit for Q_s from a classical circuit, we first give examples of problems that could be phrased as search problems. For the remainder, it should be understood that we want to solve the search problem with as few queries as possible. We can always solve the problem by querying everything, so this is not interesting. So, although we may not explicitly say *in as few queries as possible* in some places, keep in mind that this is implied.

Example 1: Search problems

The following are problems that can be framed as search problems, and thus problems for which Grover's algorithm could be applied.

1. The traveling salesman problem.

   Given an oracle to compute the distance of a path labeled by indices $1, 2, ..., N$, find the path of minimum distance[a].

2. An easier traveling salesman problem.

   Given an oracle to compute the distance of a path labeled by indices $1, 2, ..., N$, find a path with distance less than some threshold D. In this case, the search query becomes

   $$Q_s(x) = \begin{cases} 1 & d(x) < D \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

   where $d(x)$ is the distance of the path $x \in \{1, 2, ..., N\}$. Here, there could be one solution, more than one solution, or no solutions.

3. The MaxSat problem.

   Given an oracle for computing the *satisfiability* of a phrase $x_1 x_2 \cdots x_n$ where $x_i \in \{0, 1\}$, find the phrase of maximum satisfiability. Satisfiability is defined in terms of a cost function, for example

   $$C(x) = (x_1 \vee x_2 \wedge x_3) + ((x_1 \vee x_2) \wedge x_3) + (\neg x_1 \wedge x_2 \wedge \neg x_3). \tag{5.8}$$

   Note that each phrase can be labeled by an index $1, 2, 3, ..., 2^n$.

4. The phone book problem.

   Given a list of N names and a query Q_s for the one you are searching for, find the correct name with as few queries as possible.

   Interestingly, Grover mentions this in the abstract of his original paper, but it's not a good candidate to get a speedup with. The reason is that phone books are sorted! Namely, names are in alphabetical order. Using a binary search algorithm, for example, one could find the correct name in $\log_2 N$ time, which is exponentially faster than $\sqrt{N}$ time.

[a] This is an example of why we can assume $N = 2^n$ without loss of generality. Suppose we only had $M < N$ paths. Then, we could construct the search oracle Q_s to return 0 for all paths i such that $i > M$. Thus, it is like as though these paths labeled by $i > M$ do not exist, as desired. However, we'll still including because we will have a 2^n state register with n qubits.

These examples illustrate how we can think of problems in terms of a search problem. Using this thinking, many other algorithms can be expressed in this language, and it can be helpful to think of some.

Exercise 41: Think of more problems that can be phrased in terms of a search problem. What do the indices represent? What does the the search oracle do? Can the database be (efficiently) sorted, or would Grover's algorithm be a good candidate for a speedup?

5.3 Building the Black Box

Above we mentioned that the details of how we implement the search oracle (5.6) are unimportant for the relative speedup. While this is true, it can be insightful to consider how one may construct a quantum circuit for the black box. Any practical application of Grover's algorithm must take this into account, and since we are focused on both the theory and application of quantum algorithms, we should too.

However, this section can be considered optional and safely omitted on a first reading.

How could we ever hope to build these oracles we have mentioned? At the surface, they appear to be quite complex. Indeed they can be, but the key insight is to realize that all classical algorithms boil down to simple logical gates at the hardware level. If we are able to translate a universal classical gate set into quantum gates, then we are able to construct any oracle Q_s that we can implement on a classical computer.

There's numerous universal gate sets for classical computers. One of them is NOT and AND. Another is just NAND which is defined as

$$a \text{ NAND } b := \text{NOT } (a \text{ AND } b). \tag{5.9}$$

Note that, crucially, we are allowed to copy classical bits in addition to these gates. For example, if we input two bits a and b, we can assume we can create another copy of a or b at the start, or copy an intermediate bit during the computation. In the author's opinion, this should be explicitly stated when defining a universal gate set, but it is commonly left out.

Example 2: OR out of NAND.
Here we show how a circuit for OR can be constructed out of three NAND gates. Assume we input bits a and b. We want to output a OR b using only NAND gates.

The first trick is getting NOT a from just a by doing a NAND a. One can work this through logically, or it may help to note that

$$x \text{ NAND } y = xy \oplus 1 \tag{5.10}$$

where $\oplus$ denotes addition module two. Note that this involves copying the bit a. The result is a register with $a \oplus 1$.

The next step is to do the same thing with bit b, which results in a register with $b \oplus 1$.

Finally, we take the NAND of the two registers to get a OR b. To see verify this works, note we can write the entire circuit as

$$(a \text{ NAND } a) \text{ NAND } (b \text{ NAND } b) = ab \oplus a \oplus b. \tag{5.11}$$

The quantity on the right hand side can be easily verified via a truth table to be a OR b. Perhaps even simpler is to observe that the quantity on the right hand side is only zero when both a and b are zero, which is the definition of the OR gate.

Exercise 42: Construct XOR out of NAND.

The proof that AND and NOT gates (plus copy) form a universal gate set will be beyond our scope, but it is a known result. What we will focus on is how to translate these into quantum gates. Once we know this, then will have shown a method for constructing any quantum oracle (quantum circuit) from a classical oracle (classical circuit).

Note that this is another way of seeing that classical computing is a special case of quantum computing. Namely, classical computing is quantum computing restricted to the diagonal basis. Any quantum algorithm that's entirely on the diagonal is merely classical. We emphasize that Grover's algorithm, as with any good quantum algorithm, uses coherence (off-diagonal elements). We don't see that here because we are focusing only on constructing the oracle.

5.3.1 Translating Classical Gates to Quantum Gates

The translation for classical NOT to quantum NOT is as easy as changing symbols—NOT becomes Pauli-X. These two gates are exactly identical when acting on computational basis states.

The translation for classical AND to quantum AND requires a bit more thought. The CNOT gate gives us the OR of two bits as

$$\text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle. \tag{5.12}$$

Here, we have used $|x\rangle$ as the control qubit and $|y\rangle$ as the target qubit.

The key to getting a quantum analogue of AND is to control the CNOT operation on another qubit. This means that we have two control qubits and one target qubit. The target qubit gets flipped if and only if *both* control qubits are in the $|1\rangle$ state. Note that the "both" part will give us the desired "and." This gate is commonly called the *Toffoli* gate, formally defined below.

Definition 5.2 (Toffoli gate). The **Tofolli gate** is a quantum gate that acts on three qubits with the following action:

$$\text{Tof}|x\rangle|y\rangle|z\rangle := |x\rangle|y\rangle|z \oplus xy\rangle. \tag{5.13}$$

A matrix representation of the Toffoli gate in the computational basis is

$$\text{Tof} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{5.14}$$

Exercise 43: Prove that the Tofolli gate is unitary. No matrix multiplications allowed.

Exercise 44: Prove that the Tofolli gate is self-inverse.

If we input a a and b in the first two registers of the Toffoli gate, and an ancilla qubit in the $|0\rangle$ state into the third register, then the action of the Toffoli gate produces

$$\text{Tof}|a\rangle|b\rangle|0\rangle = |a\rangle|b\rangle|ab\rangle. \tag{5.15}$$

This is precisely the AND operation we were seeking. Note that AND itself is irreversible (i.e., not unitary). Thus, it cannot be a quantum gate.

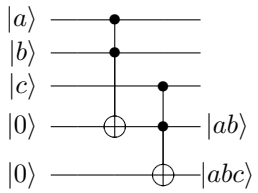

Figure 5.1: Chain of Toffoli gates for the quantum circuit analogue of a AND b AND c. The circuit inputs $|a\rangle, |b\rangle$, and $|c\rangle$ and uses two ancilla qubits to output $|ab\rangle$ and $|abc\rangle$

Example 3: AND is irreversible
Suppose I had two bits a and b, and I told you a AND $b = 1$. Could you tell me what a and b were?
In this case, yes, because there is only one unique input which gives 1 as output.
However, if I told you a AND $b = 0$, could you tell me what a and b were?
In this case, the answer is definitively no. There are three different inputs which give 0 as output, so one could never know with certainty.

There's a known trick for making classical computation reversible. It's as simple as you might expect—keeping around the input to the function. In the example above, if we knew the output was 0 and the input was $a = 0$ and $b = 1$, of course we could reproduce what the input was—we kept track of it. This is what the Toffoli gate is doing. The first two registers remain the same to make the gate reversible (unitary), and we write the AND of the two bits into the third register (again in a reversible way).

Exercise 45: The NAND operation is also irreversible. How can you modify the gate operation to make it reversible?

Exercise 46: The XOR operation is also irreversible. How can you modify the gate operation to make it reversible?

Now that we have quantum analogues for a universal classical gate set, we can implement any classical circuit on a quantum computer! This means that we can implement any oracle for which their exists a classical circuit. We state this result as a formal theorem below.

Theorem 5.1 (Quantum circuits for classical circuits.). Any classical circuit can be implemented on a quantum computer using Toffoli and NOT gates.

Proof. The proof is in the text leading up to this theorem. The main steps are:

1. Prove that any classical circuit can be constructed with only AND and NOT gates.
2. Prove that the quantum analogue for NOT is Pauli-X.
3. Prove that the quantum analogue for AND is the Toffoli gate.

We did not prove (1) explicitly, but it is a standard result that can be found in, for example, elementary textbooks on electrical engineering. Steps (2) and (3) were proven by construction. □

Exercise 47: We saw that Toffoli is the quantum analogue of AND. What's the quantum analague of NAND? (*Hint:* Try doing it with Toffoli and NOT, and try doing it with just a single Toffoli and a cleverly chosen qubit for the output register of the Toffoli.)

Example 4: Translating classical AND gates into quantum circuits

Here we construct a quantum circuit for the simple classical circuit a AND b AND c. We know the quantum analogue for AND is Toffoli. The circuit, shown in Figure 5.1, thus uses two Toffoli gates.

The first Toffoli is to write the state $|ab\rangle$ into the first ancilla qubit. Once we have this state, it's an easy step to get the final desired $|abc\rangle$ by performing another Toffoli gate and writing this into a third ancilla qubit. This type of "chain" of Toffoli gates, plus appropriate NOT gates, is a general way to construct the quantum analogue of any classical circuit.

Exercise 48: Translate the classical OR gate into a quantum circuit using Toffoli and NOT gates. Compare this with the quantum circuit using CNOT.

5.3.2 Uncomputation

Reversible computing is all fun and games in the classical world[4], but for quantum computing it serves a specific purpose. Notice that in Example 5.3.1 and Figure 5.1 we did not *just* compute $|abc\rangle$. We had some extra information lying around in the ancilla qubit, specifically the $|ab\rangle$ ancilla. For certain cases, this is not an issue, but for others it could pose a "problem." The problem is that we *just* wanted to compute $|abc\rangle$ and have the other registers remain in their original states. The reason we would want this is so that we could reuse the ancilla qubits later in the computation. If we used them in their current state, the problem would be that it would interfere with the proper coherence pattern we are trying to choreograph.

The easy way to overcome this problem is to just introduce more ancilla qubits. This works in principle, but qubits are precious resources, and we would like to limit the number of qubits needed for our algorithm. There's a way to overcome this problem, known as *uncomputation*, which introduces more quantum gates (also precious resources) to reduce the number of ancilla qubits needed. Uncomputation, formally defined below, consists of three steps, which are quite natural upon reflection.

Definition 5.3 (Uncomputation). **Uncomputation** (also known as *uncomputing*) is the act of reversibly computing a result in a quantum circuit such that input states remain unchanged. The three steps of uncomputation are:

(1) Use a quantum circuit to compute the desired output.
(2) Use a CNOT gate to copy the output into an ancilla qubit.
(3) Perform the quantum gates from step (1) in reverse to restore the initial input state.

Thus, uncomputation consists of one additional ancilla qubit. If the circuit for computing the desired output contains k gates, the total uncomputation circuit contains $2k + 1$ gates. Below, we show an example of this natural process.

Example 5: Translating classical AND gates into quantum circuits with uncomputation.

Here we construct a quantum circuit for the same classical circuit a AND b AND c considered a previous example, but here we perform the calculation using uncomputing.

The steps we perform are exactly those of Definition 5.3. After performing the Toffoli chain of Figure 5.1, we copy the result into an ancillary qubit, then perform the Toffoli chain in reverse. (Recall that the Toffoli gate is self-inverse.) The complete circuit is shown in Figure 5.2.

Exercise 49: After doing Exercise 5.3.1, construct a quantum circuit for the OR gate using Tofolli in a clean manner (i.e., using uncomputation).

[4] Actually it can have a purpose—it can be used to reduce power consumption in computations, which is very important in certain cases.

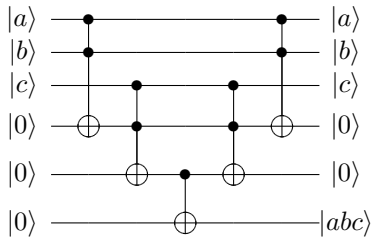

Figure 5.2: Chain of Toffoli gates for the quantum circuit analogue of a AND b AND c using uncomputation. The circuit inputs $|a\rangle, |b\rangle$, and $|c\rangle$ and uses three ancilla qubits to output *only* $|abc\rangle$, leaving the other input registers unchanged.

5.4 The Classical Complexity

This section is currently just a note.

As we have argued above in the introduction, the average time classical query complexity is of order N. Does this need to be proved explicitly? Is there an interesting discussion of the classical complexity beyond "average case is $N/2$, worst case is N?"

We shall consider these questions in the future.

5.5 Grover's Algorithm

At this point, we know the following:

1. The classical query complexity for search is $O(N)$.
2. We can implement a quantum circuit for the search oracle Q_s, defined in (5.6).

The final thing to show is that:

Theorem 5.2 (Quantum query complexity for search). The quantum query complexity for the search problem is $O\left(\sqrt{N}\right)$.

The proof of this theorem will come from Grover's algorithm, which we now discuss.

Grover's algorithm depends on a key concept known as the Grover iteration, which we'll denote G. Each application of G will ask the oracle how close the current state is to a solution, and use that information to move the current state closer to a solution. Before we define G, we first need to define a place to start.

Recall from our definition of the search problem that each object is encoded in integers $1, ..., N$, which we will write equivalently as $0, 1, ..., N-1$. We do this to encode the integers in binary $x \in \{0,1\}^n$, where $n = \log_2 N$ is integral by assumption that N is a power of two. So, our answer will be some bit string in z which we don't a priori know.

What should the starting point for our algorithm be? We have no idea what the solution z is other than it is one of the bit strings $x \in \{0,1\}^n$. This is to say, as far as we are concerned, *every bit string is equally probable* to be the solution. We should thus start in the equal superposition state

$$|\Phi\rangle := \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle. \tag{5.16}$$

Recall that this state can be prepared from the ground state $|0\rangle^{\otimes n}$ by performing a Hadamard gate on each qubit.

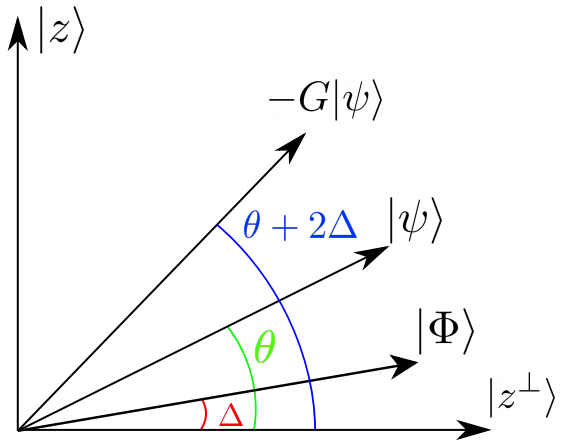

Figure 5.3: A graphical representation of Grover's algorithm. Starting in any state $|\psi\rangle$, the Grover iteration G first reflects about the solution $|z\rangle$, then reflects about the maximally coherent state $|\Phi\rangle$. The resulting state $-G|\psi\rangle$ is is 2Δ closer (in terms of angles) to the solution $|z\rangle$.

5.5.1 The Grover Iteration

Now that we have our initial starting point, we are ready to discuss how we evolve this state via the Grover iteration G. The idea is best presented by the diagram shown in Figure [2D subspace figure here].

Since $|\Phi\rangle$ is an equal superposition over *all* bit strings, it includes the correct bit string, which we are labeling z:

$$|\Phi\rangle = \frac{1}{\sqrt{2^n}}\left[|\text{wrong}\rangle + |\text{wrong}\rangle + \cdots + |\text{wrong}\rangle + |z\rangle + |\text{wrong}\rangle + \cdots + |\text{wrong}\rangle\right] \tag{5.17}$$

If we measured in the computational basis now, we would have merely $1/2^n$ probability of getting the correct bit string, which is very small for large n. In other words, we'd just be guessing! We want some way to increase, or *amplify*, this probability. This is the goal of the Grover iteration, which achieves this goal in a clever way.

We'll state the idea of the previous paragraph in a slightly different (but equivalent) way to motivate the graphical representation of Grover's algorithm shown in Figure [2D subspace figure here]. As we have seen, the overlap of the correct string $|z\rangle$ with our initial guess is exponentially small in the number of qubits n:

$$\langle z|\Phi\rangle = \frac{1}{\sqrt{2^n}}. \tag{5.18}$$

This means, geometrically, that the state $|\Phi\rangle$ is nearly orthogonal to $|z\rangle$. It is exactly orthogonal in the limiting case $n \to \infty$, and for all practical purposes, for large enough n we can consider them to be essentially orthogonal. Indeed, for $n = 50$, the overlap is on the order of 10^{-8}. This motivates the graphical representation in Figure [2D subspace figure here].

Exercise 50: However, if the two states $|\Phi\rangle$ and $|z\rangle$ were *exactly* orthogonal, Grover's algorithm would fail. After reading through the explanation of Grover's algorithm below, come back to this exercise and write a proof of this.

Recall that the angle between two unit vectors is the inverse cosine of their overlap. The angle between $|\Phi\rangle$ and the vector exactly orthogonal to $|z\rangle$ is thus the inverse sine of (5.18). Calling this angle Δ, we thus have

$$\Delta := \arcsin\frac{1}{\sqrt{2^n}} \approx \frac{1}{\sqrt{2^n}} \tag{5.19}$$

where in the second step we used the small angle approximation which is valid for large n.

The Grover iteration, geometrically, consists of the following steps:

1. Reflect about the correct solution $|z\rangle$.
2. Reflect about the equal superposition state (initial guess) $|\Phi\rangle$.

We will now show that these two reflections rotate any state vector $|\psi\rangle$ closer to the correct state $|z\rangle$.

The first reflection can be written

$$R_z := 2|z\rangle\langle z| - I \tag{5.20}$$

and the second reflection can be written

$$R_\Phi := 2|\Phi\rangle\langle\Phi| - I \tag{5.21}$$

where I is the identity operator.

Exercise 51: If the above two equations are unfamiliar to you, think about what a reflection about some vector $\mathbf{v}$ does. If you are parallel to $\boldsymbol{v}$, reflecting does nothing. If you are perpendicular to $\mathbf{v}$, reflecting gives you the opposite sign. Generally, for an arbitrary vector $\boldsymbol{w}$, reflecting about $\boldsymbol{v}$ keeps the parallel component and gives an opposite sign to the perpendicular component. Argue that $R_{\boldsymbol{v}} := 2\boldsymbol{v}\boldsymbol{v}^T - I$ is the correct mathematical expression for such an operation.

Let us now consider the effect of the Grover iteration $G := R_\Phi R_z$ on the initial state $|\Phi\rangle$. Using the definitions (5.20) and (5.21), it is easy to see that

$$R_z|\Phi\rangle = \frac{2}{\sqrt{2^n}}|z\rangle - |\Phi\rangle \tag{5.22}$$

where we have used that $\langle z|\Phi\rangle = 1/\sqrt{2^n}$. Next, we have that

$$R_\Phi R_z|\Phi\rangle = R_\Phi\left(\frac{2}{\sqrt{2^n}}|z\rangle - |\Phi\rangle\right) = \left(\frac{4}{2^n} - 1\right)|\Phi\rangle - \frac{2}{\sqrt{2^n}}|z\rangle. \tag{5.23}$$

Note that we can express this trigonometrically using $\sin\Delta = 1/\sqrt{2^n}$. We state this result below.

Theorem 5.3 (Effect of the Grover iteration on initial state). The Grover iteration $G := R_\Phi R_z$ has the effect

$$G|\Phi\rangle = \left(4\sin^2\Delta - 1\right)|\Phi\rangle - 2\sin\Delta|z\rangle \tag{5.24}$$

where $N := 2^n$ and n is the number of qubits.

As we can see, we "got a little more $|z\rangle$ amplitude" after applying the Grover iteration.

Exercise 52: What is the probability of measuring $|z\rangle$ before and after the Grover iteration? (*Warning! The vectors $|\Phi\rangle$ and $|z\rangle$ are not orthogonal.*)

Your answer to the above exercise may not be too insightful—depending on how much you worked it out and/or how many trigonometric identities you tried. I'll spare you the grunt work (though you should really try it yourself) and walk you through the steps to see the key result.

Exercise 53: Recall that the angle between two normalized vectors $|u\rangle$ and $|v\rangle$ is

$$\cos\theta = \langle u|v\rangle. \tag{5.25}$$

(a) Argue that $G|\Phi\rangle$ and $-G|\Phi\rangle$ produce the same probability distribution and are thus the same state up to global phase.

(b) Show that the angle between $|\Phi\rangle$ and $-G|\Phi\rangle$ is

$$\cos\theta = \langle\Phi| - G|\Phi\rangle = 1 - 2\sin^2\Delta. \tag{5.26}$$

(b) Use the half-angle identity

$$\sin^2 x = \frac{1 - \cos 2x}{2} \tag{5.27}$$

to show that $\cos\theta = \cos 2\Delta$.

(d) Conclude that the angle between $|\Phi\rangle$ and $-G|\Phi\rangle$ is 2Δ.

Here is the eureka! moment. We started with a vector, namely $|\Phi\rangle$, that had angle Δ relative to $|z^\perp\rangle$. After applying the Grover iteration, we ended with a vector $-G|\Phi\rangle$ which as angle 2Δ relative to $|\Phi\rangle$. (Equivalently, $-G|\Phi\rangle$ has angle 3Δ relative to $|z^\perp\rangle$.) Thus, the effect of the Grover iteration is to move the initial vector an angle 2Δ closer to the correct vector!

We would state this in a theorem since the result is so important, but we can actually state a stronger theorem. The Grover iteration G has this effect on *any* vector, not just $|\Phi\rangle$, so long as that vector is not $|z^\perp\rangle$ up to global phase.

Theorem 5.4 (Effect of Grover iteration). Let $|\psi\rangle$ be any state such that $\langle z^\perp|\psi\rangle =: \cos\theta \neq 0$. Then, the angle between $|z\rangle$ and $G|\psi\rangle$ (modulo a global phase) is

$$\cos^{-1}\langle z|G|\psi\rangle = \theta + 2\Delta \tag{5.28}$$

where $\Delta := \sin^- 1/\sqrt{N} \approx 1/\sqrt{N}$.

Proof. The proof of this theorem is a generalization of the above discussion and is left as an exercise. □

Exercise 54: Prove Theorem 5.4.

5.5.2 The Full Algorithm

Now that we know the effect of the Grover iteration G, one can guess how simple the algorithm is: keep applying G!

No, really, that's it. Grover's algorithm is simply $G^k|\Phi\rangle$ for some integer k. The number of times we need to implement k is precisely the (oracle) complexity of the algorithm, which we now address.

Exercise 55: Does it matter what state we start in for Grover's algorithm? For example, if $G^k|\Phi\rangle$ gives us high overlap with $|z\rangle$, for which $|\phi\rangle$ will $G^m|\phi\rangle$ also give high overlap with $|z\rangle$ where m is some integer.

5.5.3 The Quantum Complexity

We want our final state to be as close to the vector $|z\rangle$ as possible. The angle between $G^k|\Phi\rangle$ and $|z\rangle$ (i.e., our final state after k iterations of G) is

$$\theta_k := \Delta + 2k\Delta = (2k+1)\Delta. \tag{5.29}$$

We would like this angle to be $\pi/2 - \Delta$.

Exercise 56: Why do we want the final state $G^k|\Phi\rangle$ of Grover's algorithm to have angle $\pi/2 - \Delta$ from $|z\rangle$?

By setting $\theta_k = \pi/2 - \Delta$, we see that $2(k+1)\Delta = \pi/2$, or equivalently

$$\boxed{k = \frac{\pi}{4\Delta} - 1.} \tag{5.30}$$

Recall that $\Delta := \sin^{-1} 1/\sqrt{N} \approx 1/\sqrt{N}$. Thus, we have established the complexity of Grover's algorithm and the quadratic speedup:

$$\boxed{k \approx \frac{\pi}{4}\sqrt{N} - 1.} \tag{5.31}$$

Note that we want k to be an integer because we're going to apply G an integer number of times. (We can of course raise a unitary to any power we like, but given that we have a circuit prescription for G, it makes sense to think of doing G an integer number of times.)

The only remaining (important) question is: After k such iterations, how close are we to the desired state $|z\rangle$? Certainly if we neglected rounding and did exactly $\pi/(4\sin^{-1} N^{-1/2}) - 1$ iterations of G, we would have the state $|z\rangle$ exactly. But we round k to an integer, so we don't get to $|z\rangle$ exactly. However, the farthest k can be from an integer is one-half.

Let θ_k^* be the optimal angle, and let θ_k be the angle we rotate by after rounding k^* to the closest integer k. Since $|k - k^*| \leq 1/2$, we have

$$|\theta_k - \theta_{k^*}| = 2|k - k^*|\Delta \leq \Delta. \tag{5.32}$$

Thus, the effect of rounding error is $\theta_k = \theta_{k^*} \pm \Delta$. In other words, we are at worst an angle Δ away from $|z\rangle$ at the end of Grover's algorithm. For the worst case when the final angle is Δ away from optimal, the probability of success is

$$p(\text{measure } |z\rangle) = |\langle z|G^k|\Phi\rangle|^2 = \cos^2\Delta = 1 - \sin^2\Delta = 1 - 1/N. \tag{5.33}$$

We can now state our findings below in a formal theorem.

Theorem 5.5 (The complexity of Grover's algorithm). For large N, after

$$k := \text{round}\left(\frac{\pi}{4}\sqrt{N} - 1\right) \tag{5.34}$$

applications of the Grover iteration $G := R_\Phi R_z$ to the initial state $|\Phi\rangle$, the probability of measuring the correct solution is

$$p(\text{measure } |z\rangle) = 1 - 1/N. \tag{5.35}$$

Note that $k = O(\sqrt{N})$ proves Theorem 5.2.

Exercise 57: Perform the above analysis of Grover's algorithm starting in an initial state $|\phi\rangle$ such that $\langle\phi|z\rangle = \cos\theta$. (Keep the same Grover iteration $G := R_\Phi R_z$.)

Exercise 58: Modify the Grover iteration $G := R_\Phi R_z$ to be $R_\phi R_z$, where R_z is still $R_z := 2|z\rangle\langle z| - I$, but now $R_\phi := 2|\phi\rangle\langle\phi| - I$ where $\langle\phi|z\rangle = \cos\theta$. Starting in the initial state $|\Phi\rangle$ how does the runtime and success probability of Grover's algorithm change?

5.6 Implementing the Grover Iteration

This section is a continuation of Section 5.3, and as such has the same caveat:

This section can be considered optional and safely omitted on a first reading.

Nonetheless, the content of this section is important for the practicality of Grover's algorithm—proceed at will.

We noted above that the key step in Grover's algorithm is the Grover iteration

$$G := R_\Phi R_z = (2|\Phi\rangle\langle\Phi| - I)(2|z\rangle\langle z| - I). \tag{5.36}$$

While it is straightforward to consider the effect of this unitary, it is important to consider how to implement G in a quantum circuit.

Exercise 59: Verify that G is unitary. More generally, verify that the product of two reflections is unitary.

They key insight to implementing G is actually the intuitive interpretation of reflections: Do nothing to the parallel component, and add a minus sign to the perpendicular component. For instructive purposes, we

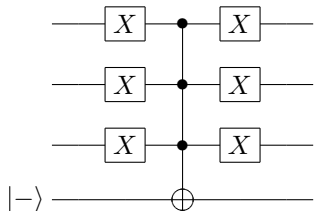

Figure 5.4: A quantum circuit that implements a reflection about $|0\rangle^{\otimes n}$, i.e. the unitary $R_0 := 2|0\rangle\langle 0| - I$. Here $n = 3$ for concreteness but the pattern easily extends to any n. A single ancilla in the $|-\rangle$ state is utilized for any n. The multi-controlled Toffoli gate can be constructed "standard" Toffoli gates via the circuit of Fig. 5.1 or Fig. 5.2.

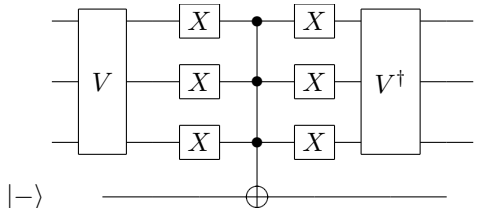

Figure 5.5: A quantum circuit for performing the reflection $R_v := 2|v\rangle\langle v| - I$ where $|v\rangle = V|0\rangle$.

can consider the reflection about $|0\rangle^{\otimes n}$. The interpretation above says that if *every* qubit is in the 0 state, we apply a minus sign. Otherwise, we do nothing. Hence, we need to control on the value of each of the n qubits being 0. If they are, we apply a minus sign, else we do nothing.

Controlling on all zeros is easy, but what do we do if the bitstring is all zeros? The answer is the common *phase kickback* trick and works by noting that

$$X|-\rangle = -|-\rangle \tag{5.37}$$

where $|-\rangle := (|0\rangle - |1\rangle)/\sqrt{2}$.

Exercise 60: Verify (5.37).

So, to achieve the minus one phase for the reflection, we can use an ancilla in the $|-\rangle$ state and a multi-controlled NOT operation. The circuit diagram is shown in Figure 5.4. One may wonder how to construct the multi-controlled NOT gate out of standard gates—but one shouldn't! We have already discussed this in Section 5.3, namely in Fig. 5.1 and Fig. 5.2.

Although this is a reflection about the $|0\rangle$ state, we can easily modify this to reflect about any state $|v\rangle$. Namely, we can write

$$R_v := 2|v\rangle\langle v| - I = 2V|0\rangle\langle 0|V^\dagger - I \tag{5.38}$$

where V is a unitary which prepares $|v\rangle$ from the ground state, i.e. $V|0\rangle = |v\rangle$. Thus, a circuit to reflect about any vector $|v\rangle$ has the same structure as that of Fig. 5.4 except we conjugate the circuit by the gate sequence which implements V. For concreteness, another circuit diagram is shown in Fig. 5.5.

Exercise 61: Using Fig. 5.5, what is the circuit that performs the reflection about $|\Phi\rangle$?

Finally, we can ask how to implement the reflection about the solution state $|z\rangle$, i.e. the reflection $R_z := 2|z\rangle\langle z| - I$. Should we proceed in the same manner as the circuit in Fig. 5.5? The answer is no: If we had a unitary $\mathcal{Z}$ such that $\mathcal{Z}|0\rangle = |z\rangle$, we wouldn't have a problem to solve in the first place. (Recall: We're trying to find an integer $z \in \{1, 2, ..., N\}$. If we had a circuit which prepared $|z\rangle$, we wouldn't need Grover's algorithm!)

This is where you need to be a little imaginative—or at least I do. All we are allowed is a query, the quantum version of the query in Def. 5.1. Specifically, the quantum query Q_s acts on a state of $n = \lg N$ qubits and sets an ancilla qubit to be $|1\rangle$ if the register is a solution to the search problem.

Definition 5.4 (The Grover Query). The **Grover query** is a unitary Q_s which acts on $n+1$ qubits as

$$Q_s|x\rangle|a\rangle = |x\rangle|a \oplus s(x)\rangle \tag{5.39}$$

where $s(x) = 1$ if x is the bitstring we are searching for, else $s(x) = 0$. (For complete clarity, $|x\rangle$ is a state on n qubits and $|a\rangle$ is an ancilla qubit with $a \in \{0, 1\}$.)

This is precisely the query we want to implement the reflection about $|z\rangle$. Namely, we simply act with the query Q_s on the input register and ancilla to perform the reflection $R_z := 2|z\rangle\langle z| - I$.

You may wonder how we can implement Q_s without knowing what the solution $|z\rangle$ is. The particular details always depend on the search problem being considered, but it is usually possible to understand why this is. For example, suppose we have N paths and we're trying to find one shorter than a distance d. Then, the query can simply compute the distance of the particular bitstring labeling a path and ask if it is less than d. If so, the query returns 1, else 0. In this manner, the query can still be constructed and implemented without circular reasoning. In practice, details of the circuit implementation of Q_s are of course important. But, hey, quantum algorithms are mostly theoretical, especially in the query model.

5.7 Closing Remarks

The technique used in Grover's algorithm can be applied in many places, as you may guess. As such, the technique goes by the name of **amplitude amplification**. This is a good name—it reflects the goal and effect of Grover's algorithm. Namely, increase, or *amplify*, the *amplitude* (and so the probability) of measuring the correct bitstring.

As briefly mentioned, there are variants of Grover's algorithm with more than one solution. The presentation is generally the same, but we may be more explicit than this and cover it in the future.

Lastly, if you read the optional sections, you may be wondering why I went through the pain of defining uncomputation, other than it's interesting. The reason is that we want to implement the black box in a "clean manner." Namely, we want to implement the Grover iteration G exactly, meaning that we implement each of the two reflections R_z and R_Φ exactly. The reflection R_z doesn't affect the input register of n qubits by construction, but for R_Φ we have to be more careful. If we *didn't* use uncomputation for the chain of Toffolis implementing the multi-controlled NOT gate, then our input register $|x\rangle$ would have been affected. We don't want this to happen, otherwise we would have to modify the rest of the algorithm. (Or, as is sometimes equivalently said, this would mess up the interference we're using in the algorithm.)

So, the point of introducing uncomputation was for the chain of Toffolis in the R_Φ reflection. But uncomputation is a more general trick, so it's good to cover it explicitly.

Exercise 62: Show that Grover's algorithm on a single qubit is no better than guessing! (in the worst case.) Specifically, let $|z\rangle = |1\rangle$ be the solution, and start in the state $|\Phi\rangle = |+\rangle$. What is the effect of the Grover iteration on $|\Phi\rangle$? Extrapolating, show that the probability of success is $1/2$, i.e., that Grover's algorithm is no better than a random guess.

References

These lectures notes were based on an article by Andy Matuschak and Michael Nielsen[5], a video lecture by Scott Aaronson, lecture notes by Ronald de Wolf, and the classic text of Nielsen and Chuang.

[5] https://quantum.country/search