Lecture 12: Query complexity

Rajat Mittal

IIT Kanpur

After going through Grover's algorithm and some of its variations, let us prove that Grover's algorithm is optimal for the search problem. This, in an intuitive sense, also shows that quantum computing is not simply *computing everything in parallel*. If that was the case, we should be able to do search in just one step.

The argument in the next section shows the limitation of a quantum computer and is one of the fundamental result in *quantum complexity theory*. This argument will bound the time required for search by bounding the number of queries to the search oracle (query complexity). We will briefly discuss the query complexity framework in the end and see general techniques to give bounds in this framework. The bounds in this framework allow us to lower bound quantum complexity of many more functions (e.g., Majority and Parity).

1 Optimality of Grover search

We have shown that the search problem can be solved with $O(\sqrt{n})$ oracle queries. Can we do better?

In this section, we will show that Grover search is optimal. That means, we will prove that any quantum algorithm with less queries will not be able to give the correct answer with high probability. Remember that the probability is computed over the inherent randomness of the algorithm, the algorithm should succeed with high probability on every input.

Let us first see if we can formally argue that a classical deterministic algorithm will take n queries for unstructured search. Suppose the best algorithm does only $n-1$ queries (remember that the algorithm can decide which queries to do depending upon the output of previous queries). We look at the algorithm when we answer all queries with unmarked element. After $n-1$ queries, algorithm still hasn't looked at one index, say i. Then the answer to the two inputs, all unmarked elements and marked element at the i-th place is same for the algorithm. This means the algorithm doesn't work for a particular input, a contradiction.

A slightly more detailed argument can prove that a randomized algorithm will take $\Omega(n)$ queries. What about a quantum algorithm? We know Grover search can do better, $O(\sqrt{n})$.

Why can't we make Grover's algorithm faster than $O(\sqrt{n})$?

Remember that we defined two states in Grover's algorithm, state $|M\rangle = |x^0\rangle$ (marked state) and $|U\rangle = \frac{1}{\sqrt{2^n-1}} \sum_{x \neq x^0} |x\rangle$ (equal superposition over unmarked states). The first approach could be to say that if we start with a state close to $|U\rangle$, a query can only move the vector a small distance (or make a small rotation). Ultimately we should end up in state $|M\rangle$, which is very far away from $|U\rangle$. So lot of queries are required.

The problem with this argument is, the algorithm doesn't need to start with something close to $|U\rangle$. On the other hand, a priori, algorithm doesn't know which element is marked. We want to use the fact (like the argument against a deterministic algorithm) that the algorithm works on all inputs. Since the algorithm should work for all oracles O_x, we *should be able to* pick an O_x where x is far from the starting state of the algorithm. In other words, whatever be the starting state of the algorithm, there should exist some marked x such that the marked state is far from the starting state.

We will restrict our attention to input x's with Hamming weight 1. In other words, x can be identified with the index where it is 1. The following argument shows that any algorithm which finds x (or the index where it is 1) using O_x, with high probability, will use at least $O(\sqrt{n})$ queries. This argument requires that we find the marked index, and not just output if the marked element exists. Later techniques will show the same lower bound even if the algorithm only shows the existence of marked element. We have seen that the complexity of these two problems is related by logarithmic factors anyway.

Note 1. The following argument shows that any algorithm which returns the correct answer on every input of Hamming weight at most 1 will take at least $O(\sqrt{n})$ queries. Grover accomplishes much more; it works for all inputs (even when Hamming weight is bigger than 1).

A generic algorithm in the query framework:

Let us see how a generic algorithm progresses in this framework of queries from an oracle. Any generic algorithm will start with a state $|\psi\rangle$ and apply unitaries (independent of input) and oracle (dependent on input) one after another. If there are l oracle queries, the final state can be written as,

$$|\psi_l^x\rangle = U_l O_x U_{l-1} O_x \cdots U_1 O_x |\psi\rangle.$$

Exercise 1. Why is this the most general form of a quantum algorithm?

If there was no oracle then the state would have been,

$$|\psi_l\rangle = U_l U_{l-1} \cdots U_1 |\psi\rangle.$$

The idea for the lower bound of $\Omega(\sqrt{n})$:

Assuming we have done only a small number of queries,

- If the algorithm succeeds with high probability, then for all x, the state $|\psi_l^x\rangle$ should be close to $|x\rangle$. Since the state $|\psi_l\rangle$ cannot be close to many $|x\rangle$'s, the state $|\psi_l\rangle$ and $|\psi_l^x\rangle$, for almost all x's, should be far.
- The state $|\psi_l\rangle$ and $|\psi_l^x\rangle$, for almost all x's, should be close as we have done only small number of queries. Notice that they are same for $l = 0$.

Hence, we arrive at the contradiction. It is important that we consider all x's, otherwise the argument will not work.

To prove it formally, consider the potential function,

$$\Phi_l = \sum_{x:|x|=1} \|\psi_l^x - \psi_l\|^2.$$

Notice that the summation is over x's whose Hamming weight is only 1, there are only n such x's.

Suppose a successful algorithm takes L queries, the lower bound will follow from two parts,

- $\Phi_L \geq O(n)$, since algorithm succeeds with probability $\geq 1/2$.
- After l queries, $\Phi_l \leq 4l^2$ (since one query can't change Φ_l by a large amount).

Exercise 2. Show that the lower bound follows from the two parts.

Exercise 3. Read about Cauchy-Schwarz inequality.

1.1 $\Phi_L \geq O(n)$

We know that,

$$\Phi_L = \sum_x \|\psi_L^x - \psi_L\|^2.$$

The idea is, ψ_L^x is close to x, so many ψ_L^x's will be far from ψ_L, making Φ_L big.

First, we will deal with a simpler situation, assume $\psi_L^x = x$. The following lemma just relies on the fact that ψ_L is a constant vector (does not depend on x).

Lemma 1. *For any unit vector $|\psi\rangle$,*

$$\sum_x \|x - \psi\|^2 \geq 2n - 2\sqrt{n} = O(n).$$

Proof. First we will write $|\psi\rangle$ in the basis of x,

$$|\psi\rangle = \sum_x a_x |x\rangle.$$

Here $\sum_x |a_x|^2 = 1$. Then looking at the concerned quantity,

$$\sum_x \|x - \psi\|^2 = \sum_x \left(\sum_{y \neq x} |a_y|^2 + |1 - a_x|^2 \right).$$

Simplifying,

$$\sum_x \|x - \psi\|^2 = (n-1) + \sum_x |1 - a_x|^2 \geq 2n - 2\sum_x |a_x|.$$

Exercise 4. Prove that, $\sum_x |a_x| \leq \sqrt{n}$.

Applying Cauchy-Schwarz on the last term, we get the required result.

□

Now we generalize: even if ψ_L^x is close to x (and not exactly x as in Lem. 1), we will show that Φ_L is $O(n)$. Similar to the style of previous proof, let ψ_L be equal to $\sum_x \alpha_x |x\rangle$.

We also know that ψ_L^x gives x with probability at least $1/2$. This means, the projection of ψ_L^x on $|x\rangle$ should be at least $1/\sqrt{2}$. Let $(\psi_L^x)_y$ denote the projection of ψ_L^x on y.

Then, looking at the expression for Φ_L,

$$\Phi_L = \sum_x \|\psi_L^x - \psi_L\|^2 = \sum_x \left(\sum_{y \neq x} |(\psi_L^x)_y - \alpha_y|^2 + |(\psi_L^x)_x - \alpha_x|^2 \right).$$

We use the assumption $(\psi_L^x)_x \geq 1/\sqrt{2}$.

$$\sum_x \|\psi_L^x - \psi_L\|^2 \geq \sum_x \left(\sum_{y \neq x} |(\psi_L^x)_y - \alpha_y|^2 + |(\psi_L^x)_x - \alpha_x|^2 \right).$$

Simplifying,

$$\sum_x \|\psi_L^x - \psi_L\|^2 \geq \sum_x |(\psi_L^x)_x - \alpha_x|^2 \geq \sum_x \left(|(\psi_L^x)_x|^2 - 2|\alpha_x| \right) = O(n) - 2\sum_x |\alpha_x|.$$

Apply Cauchy-Schwarz on the last term ($\sum_x |\alpha_x|^2 \leq 1$),

$$\Phi_L \geq O(n) - O(\sqrt{n}) \geq O(n).$$

1.2 $\Phi_l \leq 4l^2$

Now, we need to prove that Φ_l does not increase much after the application of a query. We will prove this by induction on l, note that $\Phi_0 = 0$.

We will express Φ_{l+1} in terms of Φ_l (notice that a unitary does not change the distance between two vectors),

$$\Phi_{l+1} = \sum_x \|O_x\psi_l^x - \psi_l\|^2 = \sum_x \|(O_x\psi_l^x - O_x\psi_l) + (O_x\psi_l - \psi_l)\|^2. \tag{1}$$

Notice that both the terms in the summation should be small (why)?

$$\begin{aligned}\sum_x \|O_x\psi_l^x - O_x\psi_l + O_x\psi_l - \psi_l\|^2 \\ &\leq \sum_x \left(\|O_x(\psi_l^x - \psi_l)\|^2 + 2\|O_x(\psi_l^x - \psi_l)\|\|(O_x - I)\psi_l\| + \|(O_x - I)\psi_l\|^2\right) \\ &\leq 4l^2 + 4l\sqrt{\left(\sum_x \|(O_x - I)\psi_l\|^2\right)} + \sum_x \|(O_x - I)\psi_l\|^2 \quad \text{(using Cauchy-Schwarz).}\end{aligned} \tag{2}$$

The quantity $\sum_x \|(O_x - I)\psi_l\|^2$ can be upper bounded because ψ_l is a fixed vector. Thinking of x as an index, O_x puts a negative sign in front of the amplitude of x. We get, $\sum_x \|(O_x - I)\psi_l\|^2 = 4\left(\sum_x |\langle x|\psi_l\rangle|^2\right)$. Substituting in the previous equation,

$$\Phi_{l+1} \leq 4l^2 + 8l\sqrt{\left(\sum_x |\langle x|\psi_l\rangle|^2\right)} + 4\sum_x |\langle x|\psi_l\rangle|^2.$$

The sum $\left(\sum_x |\langle x|\psi_l\rangle|^2\right)$ is 1, so

$$\Phi_{l+1} \leq 4l^2 + 8l + 4 \leq 4(l+1)^2.$$

Exercise 5. Make sure that you can prove every part of the argument above.

2 Quantum query complexity

The decision problem of search, whether there is a marked element or not, can be thought of as computing the boolean OR function.

Exercise 6. Convince yourself that this is the case. Notice that name of elements is not important, only if they are marked or not.

The search problem can be posed as, given an input $x \in \{0,1\}^n$, find if there is a 1 or not. We are given an oracle which gives x_i on input i. The number of queries needed to compute OR of x, is called the *query complexity* of OR.

This question can be posed for different functions too, like parity or majority of bits. In general, given a function $f : \{0,1\}^n \to \{0,1\}$ and an oracle to query the input, query complexity of f is the minimum number of queries required to compute f in the bounded error setting. In other words, it is the number of queries used in the worst case by the best algorithm for f. Query complexity is important because it is a good substitute of time complexity in many cases.

The decision problem for search (OR) might seem simple, but we will show below that even this requires $O(\sqrt{n})$ queries. In the process, polynomial method will be introduced, one of the main techniques to lower

bound the query complexity of a general function f. The other technique, adversary method, is given as extra reading.

From Grover search and the lower bound through polynomial method, the query complexity of OR is $\Theta(\sqrt{n})$.

Query complexity framework: Let us look at the query complexity framework more formally. We are interested in computing a function $f : \{0,1\}^n \to \{0,1\}$ on a quantum computer. Unlike the classical case, input is hidden using an oracle: O_x on input i gives the i-th bit x_i.

An algorithm A computes f with bounded error iff it gives the correct answer with probability more than $2/3$. The algorithm applies oracle queries and unitaries (independent of the input) in succession. The query complexity of A is the number of queries required by A on the worst input.

The query complexity of f, $Q_\epsilon(f)$, is the query complexity of the best algorithm.

$$Q_\epsilon(f) = \min_A \max_{x \in \{0,1\}^n} \text{(number of queries used by } A \text{ on input } x).$$

Here, minimum is taken over all A's which can compute f in the bounded error setting.

2.1 Polynomial method

Polynomial method is the first lower bounding technique for query complexity, i.e., it allows us to prove statements like:

" Any algorithm which computes f (bounded error setting) should make $\Omega(\cdot)$ number of queries."

In polynomial method, we would like to represent our computation/quantity of interest as a polynomial and then use properties of polynomials to bound the cost of computation/quantity of interest. This will become clearer with an example.

How many classical queries will you need to compute a Boolean function $f : \{0,1\}^n \to \{0,1\}$. It is known that there is a unique multilinear (degree of each variable is 1 in each term) polynomial $p : \mathbb{R}^n \to \mathbb{R}$ which agrees with f on all inputs $\{0,1\}^n$. We will skip the proof of this fact, please see any book on Analysis of Boolean functions for this well known fact (e.g. [1]).

The classical query complexity of f can be lower bounded by the degree of polynomial representing f. To give the proof idea, notice that a classical query algorithm looks like a tree (called *decision tree*), where each path is identified with a set of queries and their output. Some of these paths will lead to output 1 and some of them to 0. The indicator for a path captures all inputs which go through that path in the algorithm. For example, if a path P is specified by $x_1 = 1, x_3 = 0, x_4 = 1$, the indicator polynomial is,

$$1_P = x_1(1 - x_3)x_4.$$

The function f can be written as $f = \sum_{all\ p\ that\ output\ 1} 1_P$.

Exercise 7. How does this prove the degree is a lower bound on classical query complexity?

The main idea behind polynomial method for quantum query complexity is: *the amplitudes of the final state in a t-query algorithm are polynomials of input variables $(x_1, x_2, \cdots, x_n)$ with degree at most t.*

Before we prove the main idea, let us see how it implies a lower bound on the query complexity. Since we are considering decision problems, the final stage of a quantum algorithm will make a measurement on the final state. Some of the basis states (of measurement) will be accepting states. If the measurement falls in the accepting states, algorithm outputs 1, otherwise 0. The probability of being in the accepting states is a polynomial of degree at most $2t$ using the main idea.

Exercise 8. Prove the above statement.

Let $p_A(x)$ denote the probability of acceptance of x under an algorithm A. Remember that it is a polynomial with degree less than $2t$. If A computes f then,

- $p_A(x) \geq 2/3$ if $f(x) = 1$,

– $p_A(x) \leq 1/3$ if $f(x) = 0$.

Intuitively, $p_A(x)$ is close to $f(x)$ for all inputs x. In other words, p_A *approximates* the function f very nicely. Mathematicians have looked at many different notions of approximating a function f. It is a well studied field in mathematics. Let us take a look at the definition of our interest.

A polynomial p approximates f iff $|f(x) - p(x)| \leq 1/3$ for all x. The smallest degree of a polynomial p which approximates f is called *the approximate degree* of f. It is denoted by $\widetilde{deg}(f)$.

Exercise 9. Why are we taking smallest degree?

Again, this field has been studied in detail and we already have many bounds and techniques on approximate degrees of different f. For instance, it is known that OR on n variables has approximate degree $\sqrt{n}$.

Note 2. The proof for the approximate degree of OR first converts the approximating polynomial into a single variable (Hamming weight of the input) polynomial which approximates OR on that Hamming weight. This single degree polynomial has the same degree as the approximating polynomial. Though no such small degree polynomial can exist from standard results in mathematics. For a complete proof, please look at these notes: `https://www.cse.iitk.ac.in/users/rmittal/prev_course/f21/reports/6_approx.pdf`.

From this definition of approximation, it is clear that $p_A(x)$ approximates f and has degree at most $2t$. So, $2t$ should be bigger than $\widetilde{deg}(f)$. That means, $\widetilde{deg}(f)/2$ is a lower bound on the query complexity of f.

Theorem 1. *The quantum query complexity of a function f is lower bounded by its approximate degree.*

$$Q_\epsilon(f) = \Omega(\widetilde{deg}(f))$$

Applying this theorem to OR, we can show that the query complexity of OR is $\Omega(\sqrt{n})$.

Proof of the main idea. We want to show that the amplitude of any state, after t queries in an algorithm, is a polynomial of degree at most t in variables $x_1, x_2, \cdots, x_n$. Initially, with 0 queries, the state $|\psi\rangle$ is independent of x and hence every amplitude is a 0 degree polynomial.

Remember that the state after t queries can be written as,

$$|\psi_t^x\rangle = U_t O_x U_{l-1} O_x \cdots U_1 O_x |\psi\rangle.$$

Since a unitary is a fixed linear operator, it does not increase the degree of polynomials representing amplitudes (Why?).

Proof follows by induction, by showing that after each query the degree of the polynomial representing amplitude can increase by at most 1 (and amplitudes still remains a polynomial). We know the action of the oracle,

$$O_x|i, b, z\rangle = |i, b \oplus x_i, z\rangle,$$

where z is a basis state of the workspace.

A simple manipulation gives,

$$O_x|i, b, z\rangle = x_i|i, \bar{b}, z\rangle + (1 - x_i)|i, b, z\rangle.$$

So, a query can only increase the degree of the polynomial representing amplitudes by 1. This finishes the proof of the main idea.

Exercise 10. Is this increase necessary?

□

2.2 Extra reading: Adversary method

The proof for optimality of Grover search can be thought of as an *argument* between the algorithm and an *adversary*. The algorithm claims that it can solve OR with less than $\sqrt{n}$ queries.

The adversary picks *pairs* of input which have different output and says that the algorithm should produce lot of difference (the potential function) between these pairs of input. But one query can't produce much difference. So summing up, $o(\sqrt{n})$ queries won't be able to produce required difference.

For the case of OR function the input pairs were $0, x$, where 0 is all 0 string and x is a string with exactly one 1.

We will see a generic form of this Adversary method. It is very important to come up with these pairs of inputs where function value differs.

Suppose we want to give a lower bound on the query complexity of a function $f : \{0,1\}^n \to \{0,1\}$. To encode the input pairs, we will keep a matrix W with rows and columns indexed by inputs x. $W(x,y)$ gives a non-negative weight to the input pair we are considering, with $W(x,y) = 0$ if $f(x) = f(y)$.

Suppose the state of the algorithm after l queries to input x is $|\psi_l^x\rangle$. We will define the potential function in a very similar way as before,

$$\Phi_l = \sum_{x,y} W_{x,y}\beta_x\beta_y\langle\psi_l^x|\psi_l^y\rangle,$$

where β is the eigenvector corresponding to the maximum eigenvalue (absolute value).

Total change in Φ_l Again we know that $\Phi_0 = \sum_{x,y} W_{x,y}\beta_x\beta_y$.

Exercise 11. Prove that $\Phi_0 = \|W\|$.

Say the algorithm takes t queries. If the algorithm succeeds with probability ϵ, then $|\psi_l^x\rangle$ will be close to $|f(x)\rangle$, maximum angle θ such that $\sin^2\theta = \epsilon$. Similarly $|\psi_l^y\rangle$ will be close to $|f(y)\rangle$. If $f(x) \neq f(y)$ then $|\psi_l^y\rangle$ will be at least at an angle of $\pi/2 - 2\theta$ with $|\psi_l^x\rangle$. Hence,

$$\langle\psi_l^x|\psi_l^y\rangle \leq 2\sqrt{\epsilon(1-\epsilon)}.$$

Exercise 12. Formally prove the argument in the paragraph above.

So the total change in Φ_l is $\Omega(\|W\|)$. We will upper bound the change because of every query and hence prove a lower bound on the number of queries.

Change through one query After $l+1$ queries the state on input x is $U_{l+1}O_x|\psi_l^x\rangle$. Where U_{l+1} is independent of input x. So,

$$\Phi_{l+1} - \Phi_l \leq \sum_{x,y} W_{x,y}\beta_x\beta_y|\langle\psi_l^x|\psi_l^y\rangle - \langle O_x\psi_l^x|O_y\psi_l^y\rangle|. \tag{3}$$

The term in the summation can be simplified to,

$$|\langle\psi_l^y|O_yO_x - I|\psi_l^x\rangle| \leq \sum_{i:x_i\neq y_i} 2|\alpha_{x,i}||\alpha_{y,i}|.$$

Here $\alpha_{x,i}$ is the amplitude of $|i\rangle$ in $|\psi_l^x\rangle$, so $\sum_i |\alpha_{x,i}|^2 = 1$.

From Eq. 3,

$$\Phi_{l+1} - \Phi_l \leq \sum_{x,y} 2W_{x,y}\beta_x\beta_y \sum_{i:x_i\neq y_i} |\alpha_{x,i}\alpha_{y,i}|.$$

For convenience, define a matrix Δ_i of the same dimensions as W, s.t., $\Delta_i(x,y)$ is 1 if $x_i \neq y_i$ and 0 otherwise. So,

$$\Phi_{l+1} - \Phi_l \leq 2\sum_{x,y}\sum_i (W \circ \Delta_i)_{x,y}\beta_x\beta_y|\alpha_{x,i}\alpha_{y,i}|.$$

Here, $W \circ \Delta_i$ denotes the matrix obtained by taking entry-wise multiplication of W and Δ_i.
Again, say v_i is a vector indexed by x, s.t., $v_i(x) = \beta_x |\alpha_{x,i}|$.
The change in potential can be bounded as,

$$\begin{aligned}
\Phi_{l+1} - \Phi_l &\leq 2 \sum_{x,y} \sum_i (W \circ \Delta_i)_{x,y} v_i(x) v_i(y) \\
&= 2 \sum_i \sum_{x,y} (W \circ \Delta_i)_{x,y} v_i(x) v_i(y) \\
&\leq 2 \sum_i \|W \circ \Delta_i\| \|v_i\|^2 \\
&\leq 2 \max_i \|W \circ \Delta_i\|
\end{aligned} \tag{4}$$

The last equation is true because $\sum_i \|v_i\|^2 = 1$.

Exercise 13. Prove that $\sum_i \|v_i\|^2 = 1$.

From Sec. 2.2 and the bound on change through every query, we get the following theorem.

Theorem 2. *Given a function $f : \{0,1\}^n \to \{0,1\}$ and a $2^n \times 2^n$ matrix W (positive entry-wise), such that $W_{x,y} = 0$ if $f(x) = f(y)$. The query complexity of f is,*

$$\Omega\left(\frac{\|W\|}{\max_i \|W \circ \Delta_i\|}\right).$$

To give a lower bound, we need to come up with a good W. Generally the idea is to put more weight on pairs which differ on small number of bits but there function value is different. You can prove a lower bound on OR using this theorem, the weight on input pairs can be figured out by the proof of optimality of Grover search. This is given as an exercise in the assignment.

3 Assignment

Exercise 14. Prove that the query complexity of OR is $\Omega(\sqrt{n})$ using Thm. 2.

Exercise 15. What is the approximate degree of $f : [n] \to 0, 1$, where f is 0 if and only if input is odd.

Exercise 16. Construct a polynomial which exactly represents a function $f : \{0,1\}^n \to \mathbb{R}$.

References

1. Ryan O'Donnell. *Analysis of Boolean Functions.* Cambridge University Press, 2014.