Lecture 11: 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 around $\sqrt{n}$ 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. That means we need at least $\Omega(\sqrt{n})$ queries.

Remember that the probability is computed over the inherent randomness of the algorithm, the algorithm should succeed with high probability on every input.

1.1 Classical lower bound

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.

Notice that even though unstructured search is defined over lot more inputs, the lower bound only needs that we answer correctly on no marked element and single marked element cases.

Note 1. It is interesting to note the same argument can be applied to any function where there are inputs x and $(y_1, y_2, \cdots, y_n)$ such that $f(y_i) = 1$ and they are at Hamming distance 1 from x.

A slightly more detailed argument can prove that any randomized algorithm, say A, will take $\Omega(n)$ queries. Since the name of the elements in the list are not important, we can denote marked element with 1 and unmarked element with 0. The problem is to find if there is a 1 in a list (string) of n bits. Like in deterministic case, our argument will rely on algorithm working correctly on Hamming weight 0 and Hamming weight 1 case. Denote by x to be all 0 input and y_i to be the input which has 1 at the i-th position only.

Exercise 1. What is Hamming weight?

Suppose R denotes the list of all possible $r's$ which can be taken as random strings for the algorithm. For example, if the algorithm utilizes k bits of randomness, R would be $\{0,1\}^k$. Notice that fixing $r \in R$,

we get a deterministic algorithm, and let p_r be the probability that we use r as the random string in the execution of the algorithm. The expected number of queries done by the algorithm on input z would be,

$$E[\textit{ number of queries on } z \textit{ by } A] = \sum_r p_r(\textit{ number of queries on input } (z,r) \textit{ by } A).$$

If the cost of A is T queries, the expected cost of the algorithm on all 0 input (x) is less than T.

The lower bound will follow from these two statements.

1. If the worst case cost (implying expected cost) is small, then there exists an index, $i \in [n]$, which is not queried with high probability.
2. The behaviour of the algorithm is similar for x and y_i. This contradicts that algorithm should output differently on these inputs.

Let us look at the first statement. Define P_i to be the probability that we query index i on x by A.

$$P_i = \sum_r p_r \mathbb{1}_{r,i}.$$

Here $\mathbb{1}_{r,i}$ outputs 1 if we query index i on input x with algorithm A (otherwise 0).

Exercise 2. Prove that $\sum_i P_i \leq T$.

Hint: $\sum_i P_i$ is your expected cost.

In other words, if $T \leq n/3$ then there exist an i which is queried with probability $P_i < 1/3$. Now the second statement will show that our algorithm will fail at either input x or y_i.

Notice that if for some $r \in R$, if we don't query i, then the algorithm behaves the same irrespective of input being x of y_i. Suppose $R_0 \subseteq R$ being the set of r's where we don't query i, then by definition $\sum_{r \in R_0} p_r = 1 - P_i$. Then A should output either 0 or 1 in at most half the time under $r \in R_0$, say it outputs 1 less than 0. Then the success probability of algorithm A on y_i is at most,

$$P_i + \frac{1 - P_i}{2} = \frac{1 + P_i}{2} < 2/3.$$

This is a contradiction, since A should succeed on any input with probability greater than $2/3$.

Note 2. It is interesting to note the same argument can be applied to any function where there are inputs x and $(y_1, y_2, \cdots, y_n)$ such that $f(y_i) = 1$ and they are at Hamming distance 1 from x (Exercise 16).

1.2 Quantum lower bound

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.

Note 3. 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).

Like the randomized algorithm lower bound, we will use the following steps. Again, denote by x to be all 0 input and y_i to be the input which has 1 at the i-th position only.

1. If the cost is small, then there exists an index, $i \in [n]$, which is not queried with high probability.
2. The *distance* between the final state of the algorithm on x and y_i is small.
3. The behaviour of the algorithm is similar for x and y_i. This contradicts that algorithm should output differently on these inputs.

We have an extra step (second step) as compared to classical case, which allows us to quantify the fact that the behaviour of the algorithm is similar for x and y_i in the quantum case. In a sense it is a natural consquence of the first step.

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

Similarly we can talk about the state of the algorithm at a certain point of time t on input z, we will denote it by $|\psi_t^z\rangle$. The algorithm will have two kind of registers. Since it needs to query, the first part will contain index i, and the rest can be thought of as workspace w. So, the state will be in linear combination of the basis states of the kind $|i, w\rangle$. At a point t on z the state can be written as,

$$|\psi_t^z\rangle = \sum_w \sum_i \alpha_{t,i,w}^z |i, w\rangle.$$

We will show that algorithm A which takes only $o(\sqrt{N}) = T$ queries CAN NOT answer correctly with high probability on all inputs $x, y_1, y_2, \cdots, y_n$.

First step:

Let us consider the state on input x (all 0 input). We will drop superscripts to ease notation,

$$|\psi_t\rangle = \sum_w \sum_i \alpha_{t,i,w} |i, w\rangle.$$

The amount (probability) of querying on input i can be thought of as $p_i := \sum_{t=1}^{T} \sum_w |\alpha_{t,i,w}|^2$. Notice that since we query in superposition, there is *no actual probability* of querying i. Though p_i is a good intuitive replacement of the p_i in the classical case.

Summing it over all i,

$$\sum_i p_i = \sum_i \sum_{t=1}^{T} \sum_w |\alpha_{t,i,w}|^2 = \sum_{t=1}^{T} \sum_i \sum_w |\alpha_{t,i,w}|^2 = \sum_{t=1}^{T} 1 = T.$$

By averaging, there exist an i s.t. $p_i \leq T/n$. We will show that algorithm cannot answer correctly on x and y_i both. This finishes the first part. For the remaining parts, this index i is fixed (the one we did not query a lot).

Second step:

The idea for bounding the distance is, only the $\alpha_{t,i,w}$ parts can be affected differently by x and $y := y_i$. This will allow us to bound the separation at each time step, and the final distance can be bounded by triangle inequality. Notice that $|\psi_0^x\rangle$ and $|\psi_0^y\rangle$ are the same states.

$$\begin{aligned}
\||\psi_T^x\rangle - |\psi_T^y\rangle\| &= \sum_{t=0}^{T-1} \big\||\psi_{t+1}^x\rangle - |\psi_{t+1}^y\rangle\big\| - \||\psi_t^x\rangle - |\psi_t^y\rangle\| \\
&= \sum_{t=0}^{T-1} \|O^x|\psi_t^x\rangle - O^y|\psi_t^y\rangle\| - \||\psi_t^x\rangle - |\psi_t^y\rangle\| \qquad \textit{(Unitaries do not affect the distance)} \\
&= \sum_{t=0}^{T-1} \|O^xO^y|\psi_t^x\rangle - |\psi_t^y\rangle\| - \||\psi_t^x\rangle - |\psi_t^y\rangle\| \qquad \textit{(Unitaries do not affect the distance)} \\
&\leq \sum_{t=0}^{T-1} \|O^xO^y|\psi_t^x\rangle - |\psi_t^x\rangle\| \qquad \textit{(Triangle inequality)} \\
&= \sum_{t=0}^{T-1} \|(O^x - O^y)|\psi_t^x\rangle\| \qquad \textit{(Why?)}
\end{aligned}$$

Now, notice that this change will only happen because of $\alpha_{t,i,w}$'s. On all these coefficients, we will get different signs because of O^x and O^y. All other coefficients will cancel because O^x and O^y will act the same on them. In other words,

$$(O^x - O^y)|\psi_t^x\rangle = 2\sum_w \alpha_{t,i,w}|i,w\rangle.$$

Hence,

$$\|(O^x - O^y)|\psi_t^x\rangle\| = \sqrt{4\sum_w |\alpha_{t,i,w}|^2}.$$

We can bound the total distance,

$$\||\psi_T^x\rangle - |\psi_T^y\rangle\| \leq \sum_{t=1}^{T} \sqrt{4\sum_w |\alpha_{t,i,w}|^2}.$$

We have an upper bound on $p_i = \sum_{t=1}^{T}\sum_w |\alpha_{t,i,w}|^2$, which can be put in to the previous equation by applying Cauchy-Schwarz inequality.

$$\||\psi_T^x\rangle - |\psi_T^y\rangle\| \leq \sqrt{4T\sum_{t=1}^{T}\sum_w |\alpha_{t,i,w}|^2} = \sqrt{4Tp_i}.$$

By the bound on $p_i \leq T/n$, this distance is less than constant if $T = o(\sqrt{n})$.

The final step:

We will show that if the algorithm answers correctly on x, y, then the distance $\||\psi_T^x\rangle - |\psi_T^y\rangle\|$ should be constant. This gives a contradiction with the previous step. The following lemma is in general useful, it says that if the algorithm distinguishes between x and y, then their final states should be far apart (a very natural statement).

Lemma 1. *Suppose there are two inputs x and y such that $f(x) \neq f(y)$, and an algorithm works correctly on both inputs (with error ϵ). Let the final states be $|\psi^x\rangle$ and $|\psi^y\rangle$ respectively, then*

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

Exercise 4. Finish the lower bound argument for unstructured search using the lemma above with $\epsilon = 1/3$. Hint: Show that $\| |\psi^x\rangle - |\psi^y\rangle \|^2 \geq 2 - 2\sqrt{\epsilon(1-\epsilon)}$.

Proof of Lemma 1. If the algorithm fails with probability ϵ, then $|\psi^x\rangle$ will be close to $|f(x)\rangle$. This means it can make at most an angle θ with $|f(x)\rangle$ such that $\sin^2\theta = \epsilon$. Similarly $|\psi^y\rangle$ will be close to $|f(y)\rangle$ and will make at most angle θ.

If $f(x) \neq f(y)$ then $|\psi^y\rangle$ will be at least at an angle of $\pi/2 - 2\theta$ with $|\psi^x\rangle$. This is true even if all the concerned vectors are not in plane (plane is the worst case). Hence,

$$\langle \psi^x | \psi^y \rangle \leq \cos(\frac{\pi}{2} - 2\theta) = \sin(2\theta) = 2\sqrt{\epsilon(1-\epsilon)}.$$

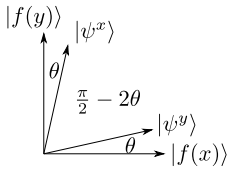

Fig. 1. Worst case for the final states

Please look at Figure 1 for a proof by picture.

□

Note 4. Like the deterministic and randomized case, the same argument can be applied to any function where there are inputs x and $(y_1, y_2, \cdots, y_n)$ such that $f(y_i) = 1$ and they are at Hamming distance 1 from x (Exercise 17).

Interestingly, the lower bound of $\Omega(\sqrt{n})$ on unstructured search was proven before Grover's algorithm itself. At that point, researchers were interested in whether quantum computing can show exponential improvement on NP-hard problems. This was a negative result in the sense that a generic exponential speedup using unstructured search for all NP-hard problems can't happen.

It seems they thought that no speedup was possible (even a square root one), and the $\Omega(\sqrt{n})$ lower bound could be pushed to $\Omega(n)$ [1, Lecture 23]. Though, Grover thought otherwise.

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 5. 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. [2]).

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 6. 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 7. 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 8. 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 $\sqrt{n}$ [3].

Note 5. 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 9. 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. Though, such an argument cannot give better than $\Omega(\sqrt{n})$ bound. Though, using polynomial method, we know that there are functions (like Parity) whose quantum query complexity is $\Omega(n)$.

We will see a generic form of this Adversary method. The idea would be to not fix the input on one side of the function value. 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 10. 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 11. 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,

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

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. 1,

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

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,

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

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

Exercise 12. 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.

As the simplest weight scheme, give weight 1 to any input pair x,y at Hamming distance 1 such that $f(x)\neq f(y)$. Such a weight matrix is known as the sensitivity graph of f, called G_f. The largest eigenvalue of the sensitivity graph G_f is known as $\lambda(f)$, you will show that it is a lower bound on quantum query complexity of f (Exercise 18).

3 Assignment

Exercise 13. Prove that the query complexity of OR is $\Omega(\sqrt{n})$ using Thm. 2.

Exercise 14. What is the approximate degree of $f : [n] \to 0, 1$, where f is 0 if and only if input is odd.

Exercise 15. Construct a polynomial which exactly represents a function $f : \{0,1\}^n \to \mathbb{R}$.

Exercise 16. Let $f : \{0,1\}^n \to \{0,1\}$ be a function and x be an input. Let $y_1, y_2, \cdots, y_m$ be inputs such that each of them are Hamming distance 1 from x and $f(y_i) \neq f(x)$. Show that the number of randomized queries needed are at least $\Omega(m)$.

Exercise 17. Let $f : \{0,1\}^n \to \{0,1\}$ be a function and x be an input. Let $y_1, y_2, \cdots, y_m$ be inputs such that each of them are Hamming distance 1 from x and $f(y_i) \neq f(x)$. Show that the number of quantum queries needed are at least $\Omega(\sqrt{m})$.

Exercise 18. The simplest way to assign weights in adversary method would be to give weight 1 to any input pair x, y at Hamming distance 1 such that $f(x) \neq f(y)$. Such a weight matrix is known as the sensitivity graph of f. Show that the largest eigenvalue of the sensitivity graph is a lower bound on quantum query complexity of f.

References

1. S. Aaronson. Introduction to quantum information science lecture notes, 2018. `https://www.scottaaronson.com/qclec.pdf`.
2. Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.
3. R. Paturi. On the degree of polynomials that approximate symmetric boolean functions. *STOC 1992*, 1992.