Week 2

Some Quantum Query Algorithms

2.1 Revisiting the quantum query model

Recall the definition of a quantum query algorithm: it is a tuple $Q = (U_0, U_1, \ldots, U_T)$ of $T+1$ unitary matrices, each of which acts on four registers: a work register $|\cdot\rangle_W$, an output register $|\cdot\rangle_O$ (which is sometimes considered part of the work register), an index register $|\cdot\rangle_I$, and a query-output reigster $|\cdot\rangle_B$ (used to store the answers to queries). When run on input x, the quantum algorithm Q outputs $Q(x)$, defined as the random variable we get by computing

$$U_T U^x U_{T-1} U^x \ldots U^x U_1 U^x U_0 |\psi_{init}\rangle$$

and then measuring the output register $|\cdot\rangle_O$. Here U^x is the unitary matrix which performs a query: it maps $|o\rangle_O |w\rangle_W |i\rangle_I |b\rangle_B \to |o\rangle_O |w\rangle_W |i\rangle_I |b \oplus x_i\rangle_B$, where $\oplus$ denotes addition modulo 2 (or any other reversible operation). This map U^x is extended linearly to all vectors defined on these registers. The value of $|\psi_{init}\rangle$ does not really matter, since U_0 can change it to whatever the algorithm wants the initial state to be, so without loss of generality we can take $|\psi_{init}\rangle = |0\rangle_O |0\rangle_W |1\rangle_I |0\rangle_B$.

2.1.1 An alternative definition

The above is a natural way to define quantum query complexity, and is the way we defined it last week. However, there is an alternative way of defining quantum query complexity, which modifies the action of the query matrix U^x. Recall that U^x needs to somehow take an index $|i\rangle$ and return a bit $|x_i\rangle$, but it must remain unitary, so we cannot erase $|i\rangle$ and overwrite it with $|x_i\rangle$ (since this will "destroy information" and not be reversible, which means the action cannot be unitary). The solution we came up with was to have U^x map $|i\rangle |b\rangle \to |i\rangle |b \oplus x_i\rangle$, where b will usually be 0 and is just a placeholder for the query output x_i.

Instead, we can implement a quantum query in a different way: we can encode x_i in the *phase*. In other words, we can define U^x to map $|i\rangle$ to $(-1)^{x_i} |i\rangle$. This action is reversible: indeed, we can undo U^x by applying it again, so it is its own inverse. Actually, we will also want to give the algorithm the ability not to query any index; one simple way to do so is to add back the register B, and map $|i\rangle |b\rangle \to (-1)^{bx_i} |i\rangle |b\rangle$. This way, if $b=1$, the query is implemented in the phase, while if $b=0$, no query is made at all (so $|b\rangle_B$ becomes effectively an on/off switch). If we extend this map linearly (and define it to act as identity on the work registers), we get a unitary matrix U^x which can form the basis of another definition of quantum query complexity.

In this new definition, a T-query quantum algorithm will still be a tuple $Q = (U_0, U_1, \ldots, U_T)$, and $Q(x)$ will still denote the output of measuring the output register after applying $U_T U^x \ldots U^x U_0 |\psi_{init}\rangle$.

The only difference is that U^x now means something different (it implements a query to x in phase), so the algorithm Q needs to be designed differently to take this input account.

It turns out that these two definitions of quantum query complexity are equivalent. To see this, let's examine what happens when we apply a Hadamard matrix to the B register before and after we apply U^x. The Hadamard matrix is the unitary matrix $\frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$. Applying it to $|b\rangle$ gives $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^b|1\rangle)$. We extend H to act as identity on the other registers by taking the tensor (Kronecker) product with the identity matrix. This matrix will then send

$$|i\rangle|b\rangle \to \frac{1}{\sqrt{2}}(|i\rangle|0\rangle + (-1)^b|i\rangle|1\rangle).$$

Applying the matrix U^x which queries in phase then gives $\frac{1}{\sqrt{2}}(|i\rangle|0\rangle + (-1)^b(-1)^{x_i}|i\rangle|1\rangle)$, and applying the Hadamard matrix again gives $|i\rangle|b \oplus x_i\rangle$.

Hence if U^x is the matrix which queries in the phase, then $(I \otimes H)U^x(I \otimes H)$ is the matrix which queries regularly. Since $I \otimes H$ is its own inverse, we also get that if U^x is the matrix which queries regularly, then $(I \otimes H)U^x(I \otimes H)$ queries in the phase. This means that if we had a quantum algorithm by the first definition, we can turn it into one that works by the second definition simply by multiplying the matrices U_i by matrices $(I \otimes H)$. We can also convert backwards using the same trick.

2.1.2 A simplification

We can simplify the second definition of the quantum query model by *almost* removing the register $|b\rangle$. Recall that when $b = 0$, no query gets made. It feels a little bit "wasteful" to pass $|i\rangle$ to the oracle if the oracle is not going to do anything with it. Instead, we can do the following: we will pass only $|i\rangle$ to the query matrix U^x, and have U^x map $|i\rangle \to (-1)^{x_i}|i\rangle$. However, since we want to preserve the ability to not make any query, we will add the symbol $i = 0$ as a possible argument to pass to U^x, and set $U^x|0\rangle = |0\rangle$. That is, if we pass $|0\rangle$ to U^x, we are saying we don't want to make any queries.

This final model removes the register B, and hence is notationally simpler. This is model is most often used in the design of quantum algorithms. It turns out to be equivalent to the other two query models we've defined.

Finally, let's ask the question: did we need the additional argument $|0\rangle_I$? What would happen if we defined quantum query complexity with the phase definition of querying, but without any option not to make a query?

The answer is that without the ability to not make a query, there is a lot we cannot do, so this ability is necessary. In particular, if the input is $x = 1^n$, then if we didn't have the ability to make no queries, the matrix U^x would add a minus sign to every basis state, and hence we would have $U^{1^n} = -I$. On the other hand, if $x = 0^n$, we would have $U^{0^n} = I$. Since a measurements are invariant under a global phase, the algorithm's output would be the same in both cases; hence no quantum algorithm would be able to distinguish the string 0^n from the string 1^n. This would be a bad definition of quantum query algorithms! Instead, any of our three equivalent definitions allows a quantum algorithm to distinguish between 0^n and 1^n using a single query (see if you can understand how).

2.1.3 Extension to non-Boolean strings

We can also define quantum query complexity in the case where the symbols x_i of x do not have to come from $\{0,1\}$, but instead come from a larger finite alphabet. For simplicity, let's assume the larger alphabet is $\{0, 2, \dots, m-1\}$, where m is a positive integer. Then in the first definition of quantum query complexity, we can have U^x map $|i\rangle |b\rangle \to |i\rangle |b + x_i\rangle$, where $b \in \{0, 1, \dots, m-1\}$ and where the addition is computed modulo m. This addition modulo m is chosen because it is reversible; any other reversible operation will also work, and will be equivalent.

We can then apply a Fourier transform to the B register before and after the query matrix U^x. Doing so lets us query in phase: we get a new matrix which maps

$$|i\rangle |b\rangle \to e^{2\pi j b x_i/m} |i\rangle |b\rangle$$

where $j = \sqrt{-1}$. Note that b need not be Boolean here; it is an element of $\{0, 1, \dots, m-1\}$ that the algorithm can choose freely.

2.2 Basic quantum algorithms

In this section, we'll review some famous quantum algorithms. We'll begin with a very simple one called the Deutsch–Jozsa algorithm [DJ92]. This algorithm computes the parity of 2 bits using one quantum query (and the computation is exact, i.e. the algorithm never makes any errors).

Theorem 2.1 (Deutsch–Jozsa [DJ92])**.** $\mathrm{Q}_E(\textsc{Parity}_2) = 1$.

Proof. It's clear that we need at least one query, since $\textsc{Parity}_2$ is not constant. To compute $\textsc{Parity}_2$ in one query, we query the superposition $\frac{1}{\sqrt{2}}(|1\rangle_I + |2\rangle_I)$ in phase, which gives us

$$\frac{1}{\sqrt{2}}((-1)^{x_1} |1\rangle_I + (-1)^{x_2} |2\rangle_I) = \frac{(-1)^{x_1}}{\sqrt{2}}(|1\rangle_I + (-1)^{x_1 \oplus x_2} |2\rangle_I).$$

We then apply a Hadamard matrix to the index register (note that this does not use any queries). This gives us the state

$$(-1)^{x_1}(|x_1 \oplus x_2\rangle_I).$$

We then simply copy that over to the output register; when it is measured, we get $x_1 \oplus x_2$, which is the parity of the 2-bit string x. $\square$

This algorithm doesn't appear to do much, since it only improves on the trivial deterministic algorithm by a factor of 2; however, it illustrates how quantum query algorithms are constructed.

Next, we'll introduce the Bernstein-Vazirani algorithm [BV97], which is effectively a higher-dimensional version of Deutsch–Jozsa. To do so, we will need a bit of notation.

Definition 2.2. *For a string $x \in \{0,1\}^n$ and a subset $S \subseteq [n]$, let $\phi(x)$ denote the string of length 2^n defined by $\phi(x)_y = \langle x, y\rangle$ for each $y \in \{0,1\}^n$. Here we've used y as an index into this string of length 2^n, which makes sense as there are 2^n distinct options for $y \in \{0,1\}^n$. We've also used the notation $\langle x, y\rangle$ the inner product modulo 2 of x and y.*

With this notation in hand, we can present the Bernstein-Vazirani algorithm.

Theorem 2.3 (Bernstein-Vazirani [BV97])**.** *Let $n \in \mathbb{N}$ be a positive integer, and let $N = 2^n$. Let $P = \{\phi(x) : x \in \{0,1\}^n\} \subseteq \{0,1\}^N$. Let $\textsc{Id}_P$ denote the identity function defined on P; that is, $\textsc{Id}_P(z) = z$ for all $z \in P$, and $\textsc{Id}_P(z)$ is undefined when $z \notin P$. Then $\mathrm{Q}_E(\textsc{Id}_P) = 1$.*

Before we prove this theorem, note that $\textsc{Id}_P$ is a function defined on strings of length $N = 2^n$, so one might expect its query complexity to be close to N. However, even a deterministic algorithm can compute $\textsc{Id}_P$ using only $n = \log N$ queries. Indeed, if the input is denoted by z, we can query the positions z_{e_i} for each $i \in [n]$, where e_i is the string that has 1 at position i and zeroes everywhere else. Since $z \in P$, we know $z = \phi(x)$ for some $x \in \{0,1\}^n$, and hence $z_{e_i} = \langle x, e_i \rangle = x_i$. Hence querying z_{e_i} for all i (a total of n queries) gives us all of x, after which we can simply compute $z = \phi(x)$ offline (with no additional queries).

The interesting property is that a quantum algorithm can compute this function using 1 query instead of using $\log N$ queries. Of course, this strongly relies on the promise we've placed on the input strings.

Proof. The quantum algorithm forms the uniform superposition over indices y to the input z:

$$\frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} |y\rangle_I .$$

It then queries the input in phase, giving the state

$$\frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{z_y} |y\rangle_I .$$

Finally, the algorithm applies $H_2^{\otimes n}$ to the index register, where H_2 is the 2×2 Hadamard matrix and where $\otimes n$ denotes the tensor (Kronecker) product of H_2 with itself n times. The result is a $2^n \times 2^n$ unitary matrix, with $H_2^{\otimes n}[y,w] = (-1)^{\langle y,w \rangle}/\sqrt{2^n}$. Applying this unitary, we get

$$\begin{aligned}
\frac{1}{2^n} \sum_{y \in \{0,1\}^n} \sum_{w \in \{0,1\}^n} (-1)^{z_y} (-1)^{\langle w,y \rangle} |w\rangle_I &= \frac{1}{2^n} \sum_{y \in \{0,1\}^n} \sum_{w \in \{0,1\}^n} (-1)^{\langle x,y \rangle + \langle w,y \rangle} |w\rangle_I \\
&= \frac{1}{2^n} \sum_{y \in \{0,1\}^n} \sum_{w \in \{0,1\}^n} (-1)^{\langle x+w,y \rangle} |w\rangle_I \\
&= \frac{1}{2^n} \sum_{w \in \{0,1\}^n} \left(\sum_{y \in \{0,1\}^n} (-1)^{\langle x+w,y \rangle} \right) |w\rangle_I .
\end{aligned}$$

Note that if $x + w$ is not the all-zero string (modulo 2), then the inner sum will be 0, because half of the strings y will cause the term $(-1)^{\langle x+w,y \rangle}$ to be 1 and the other half will cause it to be -1. If $x + w$ is the all-zero string modulo 2, then we have $w = x$, and furthermore $(-1)^{\langle x+w,y \rangle} = 1$ for all y. Hence the above state is exactly equal to $|x\rangle_I$. We can now simply convert x to $\phi(x)$ (which is reversible, and hence can be done by a unitary matrix) without any further queries, and we can place $\phi(x)$ in the output register. □

2.3 Shor's algorithm

Next, we'll introduce Shor's algorithm [Sho97], which solves a problem called period-finding.

Definition 2.4. *Call a string $x \in [n]^n$* periodic with period $k \in \mathbb{N}$ *if*

1. *$x_i = x_{i+k}$ for all $i \in [n]$, and*
2. *for all positive integers $\ell < k$ and all $i \in [n]$, $x_i \neq x_{i+\ell}$.*

Here addition is modulo n (and x_0 refers to x_n).

We can now state Shor's algorithm, which amounts to the following theorem.

Theorem 2.5 (Shor [Sho97])**.** *Let P be the set of all periodic strings in $[n]^n$ with period at least $\sqrt{n}$. Let $f\colon P \to [n]$ be the function that takes an input $x \in P$ and returns its period (which is an integer in $[\sqrt{n}, n]$). Then $\mathrm{Q}(f) = O(1)$.*

Proof. The quantum algorithm will work with regular (non-phase) queries. It will start by setting up the uniform superposition

$$\frac{1}{\sqrt{n}} \sum_{i\in[n]} |i\rangle_I\, |0\rangle_B\,.$$

It will then make a query, yielding

$$\frac{1}{\sqrt{n}} \sum_{i\in[n]} |i\rangle_I\, |x_i\rangle_B\,.$$

Next, it will apply the quantum Fourier transform to the index register. This is the $n \times n$ unitary matrix F_n given by $F_n[a,b] = e^{2\pi jab/n}/\sqrt{n}$, where $j = \sqrt{-1}$. Applying this matrix to the index register does not cost any queries, and gives us

$$\frac{1}{n} \sum_{i\in[n]} \sum_{\ell\in[n]} e^{2\pi ji\ell/n}\, |\ell\rangle_I\, |x_i\rangle_B\,.$$

All we are going to do next is to measure the index register (and then repeat this algorithm a few times and do some classical post-processing). However, before we deal with the measurement, let's make some simplifications to this quantum state.

Note that if the period of x is k, then there are only k distinct symbols in the string x; if we denote the set of these symbols by $S \subseteq [n]$ with $|S| = k$, then we can write this state as

$$\frac{1}{n} \sum_{\alpha\in S} \sum_{\ell\in[n]} \left(\sum_{i\in[n]:x_i=\alpha} e^{2\pi ji\ell/n} \right) |\ell\rangle_I\, |\alpha\rangle\,.$$

The inner sum is over i such that $x_i = \alpha$. Since x is periodic, we know that the set of such i forms an arithmetic sequence; we can write $i = a + bk$ where b is an integer which ranges from 0 to $\lfloor (n-a)/k \rfloor = n/k - 1$ and where $a \in [1,k]$ is the position of the first occurrence of α. Let's use w to denote $e^{2\pi j/n}$. Then the inner sum is

$$w^{\ell a} \sum_{b=0}^{n/k-1} w^{b\ell k}.$$

There are two cases. If ℓk is a multiple of n, then $w^{b\ell k}$ is an integer power of $w^n = 1$; in this case, the above becomes $w^{\ell a} n/k$. If ℓk is not a multiple of n, then the sum is a geometric sequence, and the above becomes

$$w^{\ell a} \frac{1 - w^{\ell k(n/k)}}{1 - w^{\ell k}} = 0.$$

Hence the terms where ℓ is not a multiple of n/k are all 0. We can take $\ell = cn/k$ for $c = 1, 2, \ldots, k$; this way, the quantum state becomes

$$\frac{1}{k} \sum_{a=1}^{k} \sum_{c=1}^{k} e^{2\pi jca/k}\, |cn/k\rangle_I\, |x_a\rangle_B\,.$$

Measuring the index register then gives a uniform distribution over integers cn/k for $c \in \{1, 2, \ldots, k\}$. If we run this twice, we get two integers $c_1 n/k$ and $c_2 n/k$ where c_1 and c_2 are uniformly random in $\{1, 2, \ldots, k\}$. When k is large enough, the probability that two uniformly random integers less than k are relatively prime is around $6/\pi^2 \geq 0.6$. This means that if we simply take the greatest common factor of the two outputs $c_1 n/k$ and $c_2 n/k$, we will get n/k with probability at least 0.6, provided k is large enough. We can then classically compute k from n/k, since we know n. Finally, if we want the success probability to be larger than 0.6, we can repeat this process a constant number of times and output the largest k we've seen; this can boost the success probability to any constant of our choice.

We note that although we used classical post-processing, we could have done all of this computation using the work tape of the quantum algorithm; this is because quantum computation can simulate classical computation. □

Shor's algorithm can find the period of a periodic string using a constant number of queries. How long does this take for a classical algorithm? Some lower bounds are shown in [Cle04; CFMW10]. Note that if a string x is periodic, then the period k has to divide evenly into the length n of the string; this means that even without making any queries, we know something about the period k.

Luckily, it turns out that Shor's algorithm can also find the period of a nearly-periodic string, in which n is not a multiple of k. Proving this formally requires an annoying analysis of the errors this introduces, and we won't do so here; however, at least when k is in the vicinity of $\sqrt{n}$ (or a constant factor away), Shor's algorithm can still work using $O(1)$ queries.

To get a clean separation between quantum and classical algorithms, we can define our period-finding function to have the promise that x is nearly periodic (so its length does not need to be an integer multiple of the period) and that its period is in the range $[\sqrt{n}, 4\sqrt{n}]$. We can then define the function f to have Boolean outputs: we'll ask for the output to be 0 if the period is smaller than $\sqrt{2n}$, and for the output to be 1 if the period is in $[\sqrt{2n}, \sqrt{4n}]$.

This function f defined on strings of length n has $\mathrm{Q}(f) = O(1)$ and $\mathrm{R}(f) = \Omega(n^{1/4})$; see [CFMW10], for example. This gives quite a large separation between quantum and randomized query complexity. However, note that the strings of the input had a non-Boolean alphabet. If we want the function to be defined on Boolean strings, we could use $O(\log n)$ bits to represent one alphabet symbol; of course, the quantum algorithm will then require $O(\log n)$ to compute the function, since it needs to read constantly many alphabet symbols. This gives us a Boolean function with $\mathrm{Q}(f) = O(\log n)$ and $\mathrm{R}(f) = \Omega(n^{1/4})$.

2.4 Grover's algorithm

Next, we examine Grover's algorithm [Gro96].

Theorem 2.6. *Let $P \subseteq \{0,1\}^n$ be the set of all strings with Hamming weight 1 (that is, strings that have exactly one non-zero entry). Define a function $f\colon P \to [n]$ by setting $f(x)$ to be the unique index $i \in [n]$ such that $x_i = 1$. Then $\mathrm{Q}(f) = O(\sqrt{n})$*

Proof. The algorithm starts with the uniform superposition over indices,

$$|\phi\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} |i\rangle_I .$$

This state is independent of the input x, and can be constructed with no queries. From this starting state, the algorithm then repeatedly applies two unitaries: first, the unitary U^x which queries in phase; and second, the unitary $U = I - 2\left|\phi\right\rangle\!\left\langle\phi\right|$. This is repeated k times, which gives us the state

$$(UU^x)^k \left|\phi\right\rangle.$$

To analyze the resulting state, we observe that the algorithm is acting in a two-dimensional space; that is, even though the quantum states used (such as $\left|\phi\right\rangle$) are n-dimensional, they all lie in the the same 2-dimensional plane, and the algorithm never causes the state to leave this plane. Let i be the index where $x_i = 1$; then the 2-dimensional plane is the one spanned by the orthogonal vectors $\left|i\right\rangle$ and $\left|\psi\right\rangle = \frac{1}{\sqrt{n-1}}\sum_{j\neq i}\left|j\right\rangle$.

Visualize $\sum_{j\neq i}\left|j\right\rangle$ as the x-axis and $\left|i\right\rangle$ as the y-axis. Then $\left|\phi\right\rangle$ is a unit vector in the first quadrant which is close to the x-axis. Indeed, the angle θ between $\left|\phi\right\rangle$ and the x-axis satisfies

$$\cos(\theta) = \langle\phi|\psi\rangle = \frac{1}{\sqrt{n(n-1)}}\sum_{j\neq i}\langle j|j\rangle = \sqrt{\frac{n-1}{n}},$$

which means that $\sin(\theta) = 1/\sqrt{n}$, and hence $\theta \approx 1/\sqrt{n}$.

The unitary operator U^x acts on this 2-dimensional space by flipping the vector across the x-axis (since it negates $\left|i\right\rangle$ and leaves $\left|j\right\rangle$ unchanged for $j \neq i$). On the other hand, by construction U acts on this 2-dimensional space by flipping across the line spanned by $\left|\phi\right\rangle$. If a vector has angle α with the positive x axis, then U^x maps it to the vector with angle $-\alpha$, and then U maps it to the vector with angle $\alpha + 2\theta$. Hence after k steps of the algorithm, the state $(UU^x)^k\left|\phi\right\rangle$ is precisely

$$\sin((2k+1)\theta)\left|i\right\rangle + \frac{1}{\sqrt{n}}\sum_{j\neq i}\cos((2k+1)\theta)\left|j\right\rangle,$$

with $\theta = \arcsin(1/\sqrt{n})$.

If we measure the index register I, the probability that we will find the correct index i is $\sin^2((2k+1)\theta)$. Since $\theta \approx 1/\sqrt{n}$, if we choose k to be around $(\pi/4)\sqrt{n}$, the success probability will be at least a constant. We can then do some classical post-processing: we can query x_i where i is the result of the measurement to check if $x_i = 1$. If so, we know that i is correct, and we can return i. If not, we can rerun the quantum algorithm to try again. After a constant number of repetitions, the probability that we failed in all of them can be made to be below any small constant of our choice. □

The above application of Grover search is very specific: there is a special promise (that the Hamming weight is exactly 1) and the output of the function is not Boolean. However, it turns out that we can use the same basic algorithm to get a quadratic speedup on many related problems.

First, consider the partial function PROMISEOR_n, where the Hamming weight is promised to be either 0 or 1 and where our goal is to determine which is the case. By running Grover's algorithm as above, we get some index i that the algorithm claims contains a 1; by querying x_i, we can then check whether the algorithm succeeded. If so, we output 1, and if not, we output 0. This algorithm makes constant error.

Next, consider the situation where there is more than one marked item; that is, where the Hamming weight of x is $h > 1$. We can run essentially the same algorithm as in Grover search, except that our two axes will be (1) the uniform superposition over unmarked items (as the x-axis) and (2) the uniform superposition over marked items (as the y-axis). The angle θ will now be $\arcsin(\sqrt{h/n})$, and so the number of times we wish to rotate by 2θ before measuring will be $\sqrt{n/h}$. This means we can find the position i of a marked item using $O(\sqrt{n/h})$ quantum queries; however, this requires knowing h (at least to within a factor of 2 or so), so that we know when to stop rotating.

This means we can distinguish strings of Hamming weight h from the all-0 string using $O(\sqrt{n/h})$ quantum queries, so long as we know h. In fact, we can even distinguish strings of Hamming weight between $[h, 2h]$ from the all-0 string using $O(\sqrt{n/h})$ while knowing only h (and not the exact Hamming weight); this is because Grover's algorithm still works with some constant success probability if we run for a number of iterations that is slightly sub-optimal. However, if we run for a number of iterations that is much smaller or much larger than the "correct" number, the angle that the final vector makes with the x-axis might end up being close to 0 or to π, in which case the success probability might be very small (like $1/n$ or even 0).

What if we don't know h? Can we still find a marked item if we don't know how many there are to look for? Well, we can do the following trick: we can run the Grover rotation for a *random* number of iterations. We can choose this random number k uniformly in $[0, 100\sqrt{n}]$. The upper bound is picked to be large enough to ensure that even if there is only one marked item, we will rotate full-circle many times. This way, regardless of θ (and hence regardless of the number of marked elements), the angle we end up with will effectively be uniformly random in the full range $[0, 2\pi]$. Then with probability around 1/2, the angle at the end will be in $[\pi/4, 3\pi/4] \cup [5\pi/4, 7\pi/4]$; if this happens, the success probability of our measurement will be at least 1/2. Hence the probability that this process succeeds in finding a marked item is at least 1/4. We can then check the resulting index i by querying x_i to see if $x_i = 1$; if not, we can try again. Repeating this procedure a constant number of times lets us find a marked item with any constant probability we wish (or else lets us conclude there is no marked item at all).

The upshot of this is that we get the following conclusion.

Theorem 2.7 (Grover Search [Gro96]). *Let* OR_n *be the OR function on* n *bits. Then* $\mathrm{Q}(\mathrm{OR}_n) = O(\sqrt{n})$.

It is not hard to argue that $\mathrm{R}(\mathrm{OR}_n) = \Omega(n)$, so OR_n is a total function that gives a quadratic quantum speedup. As we've discussed last week, it is known that $\mathrm{R}(f) = O(\mathrm{Q}(f)^4)$ and even $\mathrm{D}(f) = O(\mathrm{Q}(f)^4)$ for total functions. There's also a power 3 separation known between $\mathrm{R}(f)$ and $\mathrm{Q}(f)$, and a tight power 4 separation between $\mathrm{D}(f)$ and $\mathrm{Q}(f)$.

It is not currently known whether the largest possible separation between $\mathrm{Q}(f)$ and $\mathrm{R}(f)$ for total functions is a power 3 separation or a power 4 separation (or something in between).

2.5 Amplitude amplification

As we've noted before, a quantum algorithm can be amplified: if the algorithm computes a Boolean function to error ϵ, repeating it k times and taking a majority vote of the outputs gives a new algorithm which uses k times as many queries and which has smaller error.

How much does it cost to perform this amplification? In other words, how large does k need to be? It turns out that if we start with a constant-error algorithm and wish to get a very-small-error algorithm with error ϵ, we only need to pick k to be $\Theta(\log 1/\epsilon)$. This means that, for example, if we wanted the error to be $1/n^{100}$, we would only need to pay a multiplicative factor of $O(\log n)$ in the number of queries used.

What if instead of starting with constant error, we start with very large error? Recall that an error of 1/2 or larger makes query complexity trivial (because the algorithm can guess the answer without making any queries). Suppose, however, that we have an algorithm that computes a Boolean function to error $(1-\gamma)/2$, where $\gamma > 0$ is very small (say, $1/\sqrt{n}$ or something). We wish to turn this into an algorithm that succeeds with constant error. How much does this cost?

It turns out that this is a much more expensive operation. Using the strategy of repeating k times and taking the majority vote, it turns out that we would need to pick $k = \Theta(1/\gamma^2)$ before the error level becomes constant. If $\gamma = 1/\sqrt{n}$, this would mean that we have to use n times as many queries as before – but since every Boolean function can be computed with n queries, this doesn't even beat the trivial deterministic algorithm of querying all the bits, and is therefore useless.

A better amplification strategy in this type of setting exists – for quantum algorithms only (the classical ones are stuck paying $1/\gamma^2$). This strategy relies on *amplitude estimation*. Roughly speaking, the task of quantum amplitude estimation works as follows: we have a quantum algorithm Q, which we can view as a unitary $U = U_T U^x \dots U^x U_0$ that depends on x, a start state $|\psi_{init}\rangle$, and a measurement; we will represent the measurement using a projection matrix Π, which projects onto the space where the algorithm outputs 1. In this notation, the probability that $Q(x)$ outputs 1 is $p = \|\Pi U |\psi_{init}\rangle\|^2$.

Our goal will be to *estimate* this probability p; that is, we want a new algorithm which uses the unitary U as a subroutine and which outputs an estimate $\hat{p}$, which is guaranteed to be close to p. If we had such an algorithm, and if it used the unitary U few times, and if the final distance between p and $\hat{p}$ was smaller than $\gamma/2$, then we could use this estimate on our original algorithm to see if its acceptance probability is less than or greater than $1/2$. This allows us to know whether or not the input x was a 1-input.

Formally, we have the following theorem.

Theorem 2.8 (Amplitude estimation)**.** *Suppose we have access to a unitary U (representing a quantum algorithm) which maps $|0\rangle$ to $|\psi\rangle$, as well as access to a projective measurement Π, and we wish to estimate $p := \|\Pi|\psi\rangle\|_2^2$ (representing the probability the quantum algorithm accepts). Fix $\epsilon, \delta \in (0, 1/2)$. Then using at most $O((1/\epsilon)\ln(1/\delta))$ controlled applications of U or $U^\dagger$ and at most that many applications of $I - 2\Pi$, we can output $\tilde{p} \in [0, 1]$ such that $|\tilde{p} - p| \le \epsilon$ with probability at least $1 - \delta$.*

Further, this can be tightened to a bound that depends on p, as follows. For any positive real number T, there is an algorithm which depends on ϵ, δ, and T (but not on p) which uses at most T applications of the unitaries (as above) and outputs $\tilde{p} \in [0, 1]$ with the following guarantee: if T is at least $100(1/\sqrt{\epsilon} + \sqrt{p}/\epsilon)\ln(1/\delta)$, then $|\tilde{p} - p| \le \epsilon$ with probability at least $1 - \delta$.

This theorem is proved in this week's readings; we do not prove it here. We note that using this theorem, we can get amplification from error $(1 - \gamma)/2$ to constant error while paying only $O(1/\gamma)$ overhead, as mentioned earlier. This fast amplification has many applications in quantum algorithms. For example, we get the following theorem.

Theorem 2.9. *Let $\textsc{GapMaj}_n$ be the promise problem in which the input is promised to have Hamming weight either at most $n/2 - \sqrt{n}$ or else at least $n/2 + \sqrt{n}$, and where we must determine which of these two cases we are given as input. Then $\mathrm{Q}(\textsc{GapMaj}_n) = O(\sqrt{n})$.*

Proof. Consider the randomized algorithm which picks an index $i \in [n]$ uniformly at random, and outputs x_i. This algorithm uses one query and computes $\textsc{GapMaj}_n$ to error $1/2 - 1/\sqrt{n}$. We can turn this into a single-query quantum algorithm which also achieves this error level. We then amplify this quantum algorithm to constant error using amplitude estimation. The new quantum algorithm computes $\textsc{GapMaj}_n$ to constant error and uses $O(\sqrt{n})$ queries, as desired. □

References

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum Complexity Theory. *SIAM Journal on Computing* (1997). Previous version in STOC 1993. DOI: 10.1137/s0097539796300921 (p. 3).

[CFMW10] Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Ronald de Wolf. New Results on Quantum Property Testing. *Proceedings of the 30th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. 2010. DOI: 10.4230/LIPICS.FSTTCS.2010.145. arXiv: 1005.0523 (p. 6).

[Cle04] Richard Cleve. The query complexity of order-finding. *Information and Computation* (2004). Previous version in CCC 2000. DOI: 10.1016/j.ic.2004.04.001. arXiv: quant-ph/9911124 (p. 6).

[DJ92] David Deutsch and Richard Jozsa. Rapid Solution of Problems by Quantum Computation. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* (1992). DOI: 10.1098/rspa.1992.0167 (p. 3).

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 1996. DOI: 10.1145/237814.237866. arXiv: quant-ph/9605043 (pp. 6, 8).

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing* (1997). Previous version in FOCS 1994. DOI: 10.1137/S0097539795293172. arXiv: quant-ph/9508027 (pp. 4, 5).