Week 2

Quantum Certificates and the Hybrid Method

2.1 The hybrid method

Recall that last week, we saw that $\mathrm{Q}(f) = \Omega(\sqrt{\mathrm{bs}(f)})$ follows from $\mathrm{Q}(\textsc{PromiseOR}_n) = \Omega(\sqrt{n})$, but we did not prove the latter. Let's now attempt to do so.

The approach we'll use was first used by [BBB+97], and is called the *hybrid method*. For an algorithm Q and a fixed input $x \in \{0,1\}^n$, the hybrid method lets us roughly talk about "where the algorithm queried" when Q was run on x. It lets us do this even though a quantum algorithm queries the in superposition, so there is no true answer about where the algorithm queried; still, we will be able to say *something* about the relative query weights that were placed on each bit of the input x.

Before we tackle the hybrid method, let's look at a classical analogue of it, which can be used to lower bound randomized query complexity. This analogue takes the form of the following theorem.

Theorem 2.1. *Let R be a randomized query algorithm on n bits, and let $x \in \{0,1\}^n$. Let m_i^t be the probability that, when R is run on x, it queries bit i during query number t. Further, let $m_i = \sum_t m_i^t$ be the expected number of times R queries bit i of x before terminating. Suppose $B \subseteq [n]$ is a block such that R distinguishes between x and x^B to error ϵ (that is, $\Pr[R(x) = 1] \geq 1 - \epsilon$ and $\Pr[R(x^B) = 0] \geq 1 - \epsilon$ or vice versa). Then*

$$\sum_{i \in B} m_i \geq 1 - 2\epsilon.$$

Before we prove this theorem, let's see why it's useful. Note that $\sum_{i\in[n]} m_i$ is the expected number of queries R makes on x, so it is at most the worst-case number of queries of R. If R computes f to error $1/3$ using $\mathrm{R}(f)$ queries, then we have $\sum_{i\in[n]} m_i \leq \mathrm{R}(f)$. Also, for each sensitive block B of x, we have $\sum_{i\in B} m_i \geq 1/3$. If there are $\mathrm{bs}(f,x)$ different disjoint sensitive blocks for x, then the sum of the m_i within each block is at least $1/3$, so the sum of the m_i in all the blocks is at least $\mathrm{bs}(f,x)/3$. Picking x to be the input with largest block sensitivity, this shows that $\mathrm{R}(f) \geq \mathrm{bs}(f)/3$. In fact, if we use $\mathrm{R}_\epsilon(f)$ to denote the worst-case randomized query complexity of f to error ϵ (instead of to error $1/3$), then this argument (using Theorem 2.1) shows that $\mathrm{R}_\epsilon(f) \geq (1-2\epsilon)\,\mathrm{bs}(f)$. In other words, this argument analyzing "where the algorithm looked" for a single block in a single input is enough to prove the block sensitivity lower bound.

OK, let's prove Theorem 2.1.

Proof. Recall that a randomized query algorithm R is a probability distribution over deterministic decision trees. The sum $\sum_{iinB} m_i$ is the expected number of times a decision tree $D \sim R$ queries some $i \in B$ when it is run on the input x. This expectation is at least the probability p that a tree $D \sim R$ queries some $i \in B$ at least once when run on x. We now lower bound p.

Now, the key observation is that any decision tree which does not query any $i \in B$ must behave identically on x and x^B, and in particular, must output the same answer. That is to say, with probability $1-p$ the algorithm R does not query in B when run on x, so with probability $1-p$ it must also not query in B when run on x^B. Further, if q is the conditional probability that R outputs 1 when run on x given that it did not query in B, then $\Pr[R(x)=1]$ is at least $(1-p)q$ and $\Pr[R(x^B)=1]$ is also at least $(1-p)q$. Assume without loss of generality that $q \geq 1/2$ (otherwise, consider the probability $1-q$ that R outputs 0 on x conditioned on not querying in B instead of 1). Then $\Pr[R(x)=1] \geq (1-p)/2$ and $\Pr[R(x^B)=1] \geq (1-p)/2$. However, one of x and x^B is a 0 input, so one of these probabilities must be at most ϵ. Hence $(1-p)/2 \leq \epsilon$, or $p \geq 1-2\epsilon$, as desired. □

Hopefully Theorem 2.1 is reasonably intuitive: it just says that if an algorithm detects the difference between x and x^B, then it must query a bit inside the set B. Next, we will state the quantum version of this theorem.

Theorem 2.2. *Let Q be a T-query quantum algorithm on n bits, and let $x \in \{0,1\}^n$. Consider running Q on x until just before query number t, and then measuring the index register (the one specifying the bit $i \in [n]$ to query next). Let m_i^t be the probability that the outcome of this measurement is i. Let $m_i = \sum_t m_i^t$. Suppose $B \subseteq [n]$ is a block such that Q distinguishes between x and x^B to error ϵ (that is, $\Pr[Q(x)=1] \geq 1-\epsilon$ and $\Pr[Q(x^B)=0] \geq 1-\epsilon$ or vice versa). Then*

$$\sum_{i\in B} m_i \geq \frac{(1-2\epsilon)^2}{4T}.$$

Note that just like in the classical case, we have $\sum_i m_i = T$; that is, the sum of all m_i is the total number of queries the algorithm Q makes. If we interpret m_i to be the expected number of times Q queries bit i when run on x, then this theorem is saying that to detect a difference between x and x^B, a quantum algorithm needs to query at least $\Omega(1/T)$ total times instead of the block B, instead of the $\Omega(1)$ required by classical algorithms. That is, the longer the quantum algorithm runs for, the fewer *total* queries it needs to make inside of B to detect whether this block B has been flipped.

Just like in the classical case, we can use Theorem 2.2 to get a lower bound on $\mathrm{Q}(f)$ in terms of $\mathrm{bs}(f)$. To see this, consider the input x maximizing $\mathrm{bs}(f,x)$, which has $k=\mathrm{bs}(f)$ disjoint sensitive blocks. Fix a T-query quantum algorithm Q computing f to error ϵ, and define m_i relative to that Q and x. Then for each sensitive block B of those k blocks, we have $\sum_{i\in B} m_i \geq (1-2\epsilon)^2/4T$. Since the blocks are disjoint, we get $T=\sum_i m_i \geq k(1-2\epsilon)^2/4T$, or $T^2 \geq \mathrm{bs}(f)(1-2\epsilon)^2/4$. Taking square roots, this is $T \geq (1-2\epsilon)/2 \cdot \sqrt{\mathrm{bs}(f)}$, which means that $\mathrm{Q}_\epsilon(f) \geq (1-2\epsilon)/2 \cdot \sqrt{\mathrm{bs}(f)}$, where $\mathrm{Q}_\epsilon(f)$ denotes the quantum query complexity of f to worst-case error ϵ.

Corollary 2.3. *Let f be a (possibly partial) Boolean function. Then*

$$\mathrm{Q}_\epsilon(f) \geq \frac{1-2\epsilon}{2}\sqrt{\mathrm{bs}(f)}.$$

Note that this corollary also gives us $\mathrm{Q}(\textsc{PromiseOR}_n) = \Omega(\sqrt{n})$, as we wanted last class. We now prove Theorem 2.2.

Proof. Recall that a T-query quantum algorithm Q is a sequence of unitary matrices $U_0, U_1, \ldots, U_T$. The behavior of Q on an input x is by applying

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

where $\left|\psi_{init}\right\rangle = |0\rangle_O |0\rangle_W |1\rangle_I |0\rangle_B$. Let $|\psi_1^x\rangle := U_0 \left|\psi_{init}\right\rangle$, and for each $t \in \{1, 2, \ldots, T\}$ define $\psi_{t+1}^x := U_t U^x |\psi_t^x\rangle$. Then $|\psi_t^x\rangle$ is the state of the algorithm Q when run on x right before query t is made, and $|\psi_{T+1}^x\rangle$ is the final state of the algorithm before being measured.

Let Π_i be the projector that maps $|\psi\rangle$ on registers O, W, I, B to its component where register I contains $|i\rangle$. That is, Π_i is the matrix $I_O \otimes I_W \otimes |i\rangle\langle i|_I \otimes I_B$, where I_O, I_W, and I_B are the identity matrices on registers O, W, and B respectively, and $\otimes$ denotes the Kronecker (tensor) product of the matrices. Then by definition, $m_i^t = \|\Pi_i |\psi_t^x\rangle\|^2$, where $\|\cdot\|$ denotes the 2-norm.

Note that $|\psi_t^x\rangle$ is a quantum state, and therefore a unit vector. We will examine the squared distance between $|\psi_t^x\rangle$ and $|\psi_t^{x^B}\rangle$, that is, the states of the algorithm just before query t is made when run on x vs. when run on x^B. We have

$$\begin{aligned}
\left\| |\psi_t^x\rangle - |\psi_t^{x^B}\rangle \right\|^2 &= \left(\langle\psi_t^x| - \langle\psi_t^{x^B}|\right)\left(|\psi_t^x\rangle - |\psi_t^{x^B}\rangle\right) \\
&= \langle\psi_t^x|\psi_t^x\rangle + \langle\psi_t^{x^B}|\psi_t^{x^B}\rangle - \langle\psi_t^x|\psi_t^{x^B}\rangle - \langle\psi_t^{x^B}|\psi_t^x\rangle \\
&= 2 - 2\Re\left(\langle\psi_t^x|\psi_t^{x^B}\rangle\right),
\end{aligned}$$

where $\Re(\cdot)$ denotes the real part of a complex number. This identity will come in handy.

The idea will be that this distance must be small at the beginning of the algorithm and large at the end, and can only change when the algorithm "queries inside" the block B where x and x^B differ. In this case where $t = 1$, we have $|\psi_1^x\rangle = |\psi_1^{x^B}\rangle$, since these both equal $U_0 \left|\psi_{init}\right\rangle$. Hence the distance between them is 0. Also, in the case where $t = T + 1$, the vectors $|\psi_{T+1}^x\rangle$ and $|\psi_{T+1}^{x^B}\rangle$ are the final states of the quantum algorithm when run on x and on x^B respectively. Now, write $|\psi_{T+1}^x\rangle = |\phi_0^x\rangle + |\phi_1^x\rangle$, where $|\phi_0^x\rangle$ and $|\phi_1^x\rangle$ are the orthogonal vectors corresponding to the output register being 0 and 1 respectively. Note that the norm of $|\phi_0^x\rangle$ and of $|\phi_1^x\rangle$ sum to 1, since they are orthogonal components of a unit vector. Then we have

$$\langle\psi_{T+1}^x|\psi_{T+1}^{x^B}\rangle = \big(\langle\phi_0^x| + \langle\phi_1^x|\big)\big(|\phi_0^{x^B}\rangle + |\phi_1^{x^B}\rangle\big) = \langle\phi_0^x|\phi_0^{x^B}\rangle + \langle\phi_1^x|\phi_1^{x^B}\rangle,$$

since all the vectors $|\phi_0^z\rangle$ are orthogonal to all the vectors $|\phi_1^z\rangle$. Using Cauchy-Schwartz, we get

$$|\langle\psi_{T+1}^x|\psi_{T+1}^{x^B}\rangle| \leq \|\phi_0^x\| \cdot \|\phi_0^{x^B}\| + \|\phi_1^x\| \cdot \|\phi_1^{x^B}\| = ab + (1-a)(1-b),$$

where $a = \|\phi_0^x\|$ and $b = \|\phi_0^{x^B}\|$. Note that a^2 is the probability that the quantum algorithm outputs 0 on input x, which is at most ϵ; on the other hand, b^2 is the probability that the quantum algorithm outputs 0 on x^B, which is at least $1-\epsilon$. It is not hard to see that the maximum possible value of $ab + (1-a)(1-b)$ under these constraints is $2\sqrt{\epsilon(1-\epsilon)}$. This gives

$$\left\| |\psi_{T+1}^x\rangle - |\psi_{T+1}^{x^B}\rangle \right\|^2 \geq 2 - 4\sqrt{\epsilon(1-\epsilon)}.$$

We now write

$$\begin{aligned}
\sqrt{2-4\sqrt{\epsilon(1-\epsilon)}} &\le \left\| |\psi_{T+1}^{x}\rangle - |\psi_{T+1}^{x^B}\rangle \right\| \\
&= \sum_{t=1}^{T} \left\| |\psi_{t+1}^{x}\rangle - |\psi_{t+1}^{x^B}\rangle \right\| - \left\| |\psi_{t}^{x}\rangle - |\psi_{t}^{x^B}\rangle \right\| \\
&= \sum_{t=1}^{T} \left\| U_t U^{x} |\psi_{t}^{x}\rangle - U_t U^{x^B} |\psi_{t}^{x^B}\rangle \right\| - \left\| |\psi_{t}^{x}\rangle - |\psi_{t}^{x^B}\rangle \right\| \\
&= \sum_{t=1}^{T} \left\| (U^{x^B})^\dagger U^{x} |\psi_{t}^{x}\rangle - |\psi_{t}^{x^B}\rangle \right\| - \left\| |\psi_{t}^{x}\rangle - |\psi_{t}^{x^B}\rangle \right\| \\
&\le \sum_{t=1}^{T} \left\| (U^{x^B})^\dagger U^{x} |\psi_{t}^{x}\rangle - |\psi_{t}^{x^B}\rangle - |\psi_{t}^{x}\rangle + |\psi_{t}^{x^B}\rangle \right\| \\
&= \sum_{t=1}^{T} \left\| \left(U^{x} - U^{B^x}\right) |\psi_{t}^{x}\rangle \right\|.
\end{aligned}$$

Here the fourth line followed by using the fact that the unitary matrices preserve the 2-norm, and applying $(U^{x^B})^\dagger U_t^\dagger$ inside the norm (recall that $U^\dagger = U^{-1}$ for a unitary matrix U). The fifth line followed using triangle inequality. The last line followed by multiplying the unitary U^{x^B} inside the norm.

Next, we note that $|\psi_t^x\rangle = \sum_{i=1}^n \Pi_i |\psi_t^x\rangle$. Furthermore, the matrix $U^x - U^{x^B}$ maps any vector that has i in the index register to 0 if $i \notin B$; this is because for $i \notin B$, we U^x and U^{x^B} behave the same on such a vector. We can therefore continue our inequality chain:

$$\begin{aligned}
&= \sum_{t=1}^{T} \left\| \left(U^{x} - U^{x^B}\right) \sum_{i=1}^{n} \Pi_i |\psi_{t}^{x}\rangle \right\| \\
&= \sum_{t=1}^{T} \left\| \sum_{i=1}^{n} \left(U^{x} - U^{x^B}\right) \Pi_i |\psi_{t}^{x}\rangle \right\| \\
&= \sum_{t=1}^{T} \left\| \sum_{i\in B} \left(U^{x} - U^{x^B}\right) \Pi_i |\psi_{t}^{x}\rangle \right\| \\
&= \sum_{t=1}^{T} \sqrt{\sum_{i\in B} \left\| \left(U^{x} - U^{x^B}\right) \Pi_i |\psi_{t}^{x}\rangle \right\|^2} \\
&\le \sum_{t=1}^{T} \sqrt{\sum_{i\in B} 4m_i^t} \\
&\le \sqrt{T \sum_{t=1}^{T} \sum_{i\in B} 4m_i^t} \\
&= \sqrt{4T \sum_{i\in B} m_i}.
\end{aligned}$$

Several lines here require explanation. The fourth line follows because each of the vectors $\big(U^x - U^{x^B}\big)\Pi_i |\psi_t^x\rangle$ for different values of i are all orthogonal. The fifth line follows because, by triangle inequality,

$$\left\|(U^x - U^{x^B})\Pi_i |\psi_t^x\rangle\right\| \le \|U^x\Pi_i |\psi_t^x\rangle\| + \|U^{x^B}\Pi_i |\psi_t^x\rangle\| = 2\|\Pi_i |\psi_t^x\rangle\| = 2\sqrt{m_i^t}.$$

The sixth line follows by Cauchy-Schwartz on the sum over t, and the last line by rearranging the sums and by the definition of m_i.

We therefore conclude

$$\sum_{i\in B} m_i \ge \frac{1-2\sqrt{\epsilon(1-\epsilon)}}{2T}.$$

To finish the proof, we note that

$$2\sqrt{\epsilon(1-\epsilon)} = \sqrt{1-(1-2\epsilon)^2} \le 1-(1-2\epsilon)^2/2,$$

from which we get $1-2\sqrt{\epsilon(1-\epsilon)} \ge (1-2\epsilon)^2/2$. □

As mentioned, Theorem 2.1 is a classical randomized version of what is known as the quantum hybrid method. This term "hybrid method" is historical. The phrase usually refers to taking a sequence of progress measures $W_0, W_1, \dots, W_k$, arguing that W_0 is small and W_k is large, and then placing a bound on the distance between consecutive terms, $W_{i+1} - W_i \le \alpha$, in order to conclude that k must be large: $k \ge (W_k - W_0)/\alpha$. That is to say, if we've transformed from an object W_0 to another object W_k in k steps, we look at the "hybrid" object that we must have become in the middle, and argue that if the hybrid object did not change quickly, then the number of steps was large.

While such an argument did show up in the quantum version of the proof, this is not unique to the so-called quantum hybrid method; other methods, such as the "adversary methods" we will study later, also have this format. The name is therefore somewhat misleading, but it has stuck.

2.2 Linear programming duality

Before we proceed further, we will make a short detour into the theory of linear programming, which plays an important role in both quantum and classical lower bounds. Hopefully, many students will have already seen linear programming before; it has a rich theory, most of which we will not need to get into. The main concept in linear programming that will be crucial is the notion of *duality*, and in particular, the *strong duality theorem of linear programming*.

First things first: what is a linear program? The term "program" is historical, and has nothing to do with programming (it originally meant something closer to planning or scheduling, like how a television program uses the word "program"). In a linear program, we have some set of n real variables, which we arrange in a vector $x \in \mathbb{R}^n$. We wish to minimize some linear function of these variables, which we will denote by $\langle w, x\rangle$ where $w \in \mathbb{R}^n$ is a fixed, known vector of constants. This minimization is subject to constraints which are also linear; for example, one constraint might say that $\langle a, x\rangle \ge c$, where $a \in \mathbb{R}^n$ is a fixed vector and $c \in \mathbb{R}$ is a fixed constant.

More generally, since we will have many constraints, we will arrange them in a matrix A; all the constraints together will therefore be written $Ax \ge b$, where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ for some $m \in \mathbb{N}$. Note that constraints of the form $\langle a, x\rangle \le c$ (instead of greater than c) are also allowed, but we can still store them as one row of A (and one entry of b) by negating the equation to turn it into a greater than condition (this negates a and c). Similarly, constraints of the form $\langle a, x\rangle = c$ are allowed as well, but can be represented as one greater than inequality and one less than inequality. Strict constraints such as $<$ or $>$ are not allowed.

A general linear program will therefore have the form

$$\begin{array}{ll} \text{minimize} & \langle w, x\rangle \\ \text{subject to} & Ax \geq b, \end{array}$$

where $x \in \mathbb{R}^n$ is a vector of variables, and $w \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m\times n}$ are constants. The goal is to solve for x.

When first encountering a linear program, it might be tempting to try to use multivariable calculus to solve this task. However, most multivarable calculus techniques will have a step saying "now check the edge cases", and in linear programming, the optimal solution will always be an edge case. Visually, you can imagine each constraint (each row of $Ax \geq b$) to be saying that the vector x is on one side of a hyperplane; in other words, that x is in one half space. Since we have many constraints, what they effectively say is that x must lie in an intersection of half spaces. In other words, $Ax \geq b$ defines a polytope: a convex area of the space $\mathbb{R}^n$ that is cut off by linear hyperplanes. The maximization task $\langle w, x\rangle$ tells us to find the point inside of the polytope that is the furthest in some specific direction (the one defined by w). That is, if we rotated the polytope a certain way, then the optimization task would just be to find the lowest point in the polytope. It is not hard to see that this lowest point is always a vertex of the polytope (or tied with a vertex).

What we will be primarily interested in is the notion of *duality*. To introduce duality, it will be easiest to give a concrete example. Consider the following linear program.

$$\begin{array}{lrl} \text{minimize} & 3x_1 - x_2 & \\ \text{subject to} & x_1 + x_2 & \geq 0 \\ & 2x_1 - 2x_2 & \geq -1 \\ & 5x_1 - x_2 & \geq 2 \end{array}$$

How would I convince you that the optimal solution has objective value at most 5? This is easy to do: I would just need to give you some feasible solution, such as $x_1 = 2$ and $x_2 = 1$, and you could check that this solution satisfies the constraints and has objective value 5. Therefore, proving that the optimal value of a minimization program is *at most* something is easy.

What if I wanted to convince you that the optimal value is *at least* something? Well, note that if we add the first two constraints together, we get $3x_1 - x_2 \geq -1$. Since the objective function is $3x_1 - x_2$, this proves the that optimal value must be at least -1. There is actually another lower bound we can get in this way: we can add $1/4$ times the second constraint and $1/2$ times the third constraint to get $3x_1 - x_2 \geq 3/4$, a better bound. This bound turns out to be tight. To prove this, I just need to give you a feasible solution with objective value $3/4$. Such a solution is $(x_1, x_2) = (5/8, 9/8)$.

As we saw, to prove lower bounds on the minimization problem, we can take linear combinations of the constraints in order to construct a new constraint that effectively says "the objective function is at least c" for some constant c. What is the best lower bound we can construct in this way? Well, to be more precise, constructing a lower bound in this way means picking some weight y_i for each constraint, so we have $y_1, y_2, y_3 \in \mathbb{R}$. We want to multiply each constraint by its weight and add them together; however, this only gives us a new valid constraint if the y_i weights are non-negative, since our constraints are inequalities. So we have $y_1, y_2, y_3 \geq 0$. Next, we want the resulting constraint to look exactly the same as the objective function $3x_1 - x_2$. This means we want the new coefficient of x_1 to be 3, which means $y_1 + 2y_2 + 5y_3 = 3$. It also means we want the new coefficient of x_2 to be -1, which means $y_1 - 2y_2 - y_3 = -1$. Finally, we want to maximize the right hand side of the new constraint, so we want to maximize $-y_2 + 2y_3$. In other words, finding the best lower bound of this form amounts to solving a new linear program!

$$\begin{array}{llllll} \text{maximize} & & -\ y_2 & +2y_3 & \\ \text{subject to} & y_1 & +2y_2 & +5y_3 & =3 \\ & y_1 & -2y_2 & -\ y_3 & =-1 \\ & y_1 & ,\ \ y_2 & ,\ \ y_3 & \geq 0 \end{array}$$

This linear program is called the *dual* of the previous one. More generally, if the minimization program had the form

$$\begin{array}{ll} \text{minimize} & \langle w, x\rangle \\ \text{subject to} & Ax \geq b, \end{array}$$

then its dual will be a maximization program of the form

$$\begin{array}{lll} \text{maximize} & \langle b, y\rangle & \\ \text{subject to} & A^T y & = w \\ & y & \geq 0. \end{array}$$

Note that any feasible solution to the minimization program must have objective value larger than that of any feasible solution to the maximization program, because the maximization program was designed to give lower bounds for the minimization program. Moreover, we can take the dual of the dual, which turns out to equal the original (primal) linear program.

The fundamental property you need to know about linear program is the following magical theorem.

Theorem 2.4 (Strong duality of linear programs). *The optimal objective value of a linear program equals the optimal objective value its dual.*

This theorem is highly nontrivial and should be surprising; after all, why should it be that lower bounds we can get by naively combining constraints turn out to be the best possible lower bounds? However, it turns out to be true (though we will not prove it).

One thing to note: if the original (primal) program has contradictory constraints, then it of course has no solution. If it is a minimization program, we say its optimal objective value is ∞, to denote that it has no solutions at all (any solution would be better than ∞ for a minimization program). Its dual can still be defined, and it will be a maximization program with optimal objective value ∞; this means it will be *unbounded*, with no upper limit on how good the feasible solutions can get. Conversely, if the minimzation program is unbounded, we denote its optimal objective value by $-\infty$, and its dual will be a maximization program which is infeasible. In other words, strong duality holds even in the degenerate cases where there is no optimal solution to the linear programs.

2.3 Fractional certificates

We will now define a new query complexity measure called fractional certificate complexity, and show that it is a better lower bound on quantum query complexity than the block sensitivity.

Recall that the certificate complexity of an input x relative to a function f was the minimum size of a certificate consistent with x, or in other words, the minimum number of bits of x I need to show you to convince you of the value of $f(x)$. Viewed another way, this is the minimum number of bits of x that must be revealed such that each y with $f(y) \neq f(x)$ must disagree with x on one of those revealed bits.

We can relax this definition to allow for *fractional* certificates. In a fractional certificate, each bit is included with some non-negative weight w_i. The total cost or size of a fractional certificate is $\sum_{i\in[n]} w_i$. The condition a fractional certificate for x must satisfy is that for all inputs y with $f(y)\neq f(x)$, we have $\sum_{i:y_i\neq x_i} w_i \geq 1$. That is, each string y that disagrees with x on f-value must conflict with x on a set of bits whose *total weight* is at least 1 (under the weight scheme defined by the fractional certificate). Note that if we restricted all the weights to be in B, then fractional certificates would become the same as regular certificates.

We denote the minimum size of a fractional certificate for x with respect to f by $\mathrm{FC}(f,x)$. We then define $\mathrm{FC}(f)$ to be the maximum value of $\mathrm{FC}(f,x)$ over all $x\in\mathrm{Dom}(f)$. Observe the following theorem.

Theorem 2.5. *Let f be a (possibly partial) Boolean function. Then*

$$\mathrm{R}_\epsilon(f) \geq (1-2\epsilon)\,\mathrm{FC}(f),$$

$$\mathrm{Q}_\epsilon(f) \geq \frac{1-2\epsilon}{2}\sqrt{\mathrm{FC}(f)}.$$

Proof. Let R be a randomized algorithm computing f to error ϵ using $T=\mathrm{R}_\epsilon(f)$ queries, and let $x\in\mathrm{Dom}(f)$ be such that $\mathrm{FC}(f,x)=\mathrm{FC}(f)$. Let m_i be defined with respect to R and x as in the hybrid argument, Theorem 2.1. Then fix any $y\in\mathrm{Dom}(f)$ such that $f(y)\neq f(x)$, and let B be the set of bits $i\in[n]$ such that $y_i\neq x_i$. Then $y=x^B$, and since R distinguishes x from x^B, we must have $\sum_{i\in B} m_i \geq 1-2\epsilon$. Now define $w_i = m/(1-2\epsilon)$. Then $\sum_{i:y_i\neq x_i} w_i \geq 1$ for all y such that $f(y)\neq f(x)$, so the vector w is a fractional certificate for x with respect to f. Its objective value is $\sum_i w_i = (1-2\epsilon)^{-1}\sum_i m_i = \mathrm{R}_\epsilon(f)/(1-2\epsilon)$, which is an upper bound on $\mathrm{FC}(f,x)$, and therefore on $\mathrm{FC}(f)$. The desired result for randomized algorithms follows.

The proof of the quantum claim is similar. Let Q be a quantum algorithm computing f to error ϵ using $T=\mathrm{Q}_\epsilon(f)$ queries, and let $x\in\mathrm{Dom}(f)$ be such that $\mathrm{FC}(f,x)=\mathrm{FC}(f)$. Let m_i be defined as in the quantum hybrid argument Theorem 2.2 with respect to Q and x, and let $w_i = 4Tm_i/(1-2\epsilon)^2$. Then for any $y\in\mathrm{Dom}(f)$ with $f(y)\neq f(x)$, the algorithm Q must distinguish x and y to error ϵ, so $\sum_{i:y_i\neq x_i} w_i = (4T/(1-2\epsilon)^2)\sum_{i:y_i\neq x_i} m_i \geq 1$, meaning the vector w is a fractional certificate for x with respect to f. The total cost of w is $\sum_i w_i = (4T/(1-2\epsilon)^2)\sum_i m_i = 4T^2/(1-2\epsilon)^2$, so $\mathrm{Q}_\epsilon(f) = T \geq \sqrt{\mathrm{FC}(f)}\cdot(1-2\epsilon)/2$. □

Note that since $\mathrm{FC}(f,x)$ is a minimization over weights, and since it has strictly less constraints than $\mathrm{C}(f,x)$ (it does not have the constraints that the weights are in $\{0,1\}$), we must have $\mathrm{FC}(f,x)\leq \mathrm{C}(f,x)$ for all f and x. This also means that $\mathrm{FC}(f)\leq\mathrm{C}(f)$. Note that while we know that $\mathrm{FC}(f)$ gives a lower bound for randomized and quantum algorithms, the same is not true for $\mathrm{C}(f)$, at least for partial functions.

To better understand $\mathrm{FC}(f)$, we note that the problem of finding a fractional certificate for a specific input x is actually a linear program. That is, we have a vector of variables $w\in\mathbb{R}^n$, which must be non-negative, meaning $w\geq 0$. We also have other constraints: for each y such that $f(x)\neq f(y)$, we have a constraint that says $\sum_{i:x_i\neq y_i} w_i \geq 1$, which is linear in the variables w_i. Finally, the objective value is $\sum_i w_i$, which is still linear. In other words,

$$\begin{array}{rlll}\mathrm{FC}(f,x) = & \text{minimize} & \sum_{i\in[n]} w_i & \\ & \text{subject to} & \sum_{i:y_i\neq x_i} w_i \geq 1 & \forall y\in\mathrm{Dom}(f): f(y)\neq f(x) \\ & & w_i \geq 0 & \forall i\in[n].\end{array}$$

The dual of this linear program is called the fractional block sensitivity, $\mathrm{fbs}(f,x)$.

$$\begin{array}{rlllll} \mathrm{fbs}(f,x) = & \text{maximize} & \sum_{y:f(y)\neq f(x)} u_y & & & \\ & \text{subject to} & \sum_{y:y_i\neq x_i} & u_y & \leq 1 & \forall i \in [n] \\ & & & u_y & \geq 0 & \forall y \in \mathrm{Dom}(f) : f(y) \neq f(x). \end{array}$$

This dual assigns a non-negative weight u_y to each input $y \in \mathrm{Dom}(f)$ with $f(y) \neq f(x)$, such that on every bit i, the total weight assigned to inputs that disagree with x is at most 1. Another way to look at this is to view each y as defining a sensitive block B for x, which is the set of all bits i on which x and y disagree; that is, the set B such that $x^B = y$. Then this dual program assigns a weight to each sensitive block B of x such that for each bit $i \in [n]$, the total weight of the blocks that contain bit i is at most 1. Note that if the weights were in $\{0,1\}$, this would force all the blocks of non-zero weight to be disjoint, that is, the weight scheme would simply define a set of disjoint sensitive blocks of x. The objective value is the sum of the weights, which in the $\{0,1\}$ case would be the number of disjoint sensitive blocks.

Since we are maximizing, and since the fractional block sensitivity removes a constraint (the constraint that the weights are in $\{0,1\}$) relative to the definition of $\mathrm{bs}(f,x)$, we have $\mathrm{fbs}(f,x) \geq \mathrm{bs}(f,x)$ for all f and x. This means we also have $\mathrm{fbs}(f) \geq \mathrm{bs}(f)$. By strong duality, we know that $\mathrm{fbs}(f,x) = \mathrm{FC}(f,x)$, which also means that $\mathrm{fbs}(f) = \mathrm{FC}(f)$. This measure is usually denoted by $\mathrm{fbs}(f)$ rather than being denoted by $\mathrm{FC}(f)$.

Now, since $\mathrm{fbs}(f) \geq \mathrm{bs}(f)$ and since we have shown that $\mathrm{fbs}(f)$ lower bounds $\mathrm{R}(f)$ (and $\sqrt{\mathrm{fbs}(f)}$ lower bounds $\mathrm{Q}(f)$), we remark that this lower bound is actually an improvement over the block sensitivity lower bound we saw earlier. That is, not only did the hybrid argument give us the block sensitivity lower bound, it actually also gave us the fractional block sensitivity lower bound—which is stronger.

2.4 Some examples

Example 1. Consider the following function defined on $\binom{n}{2}$ bits. The bits of the input will represent the edges of a graph on n vertices; if an input bit is 1, the corresponding edge is present in the graph, and if it is 0, the edge is absent. Inputs x to the function f will therefore be undirected graphs on n vertices. The function f will be a promise problem; the input is promised to be either the all-0 string (the graph with no edges), or else a star. A *star* is a graph that has one vertex v that is connected to all other vertices, that is, it has degree $n-1$, and no other edges (that is, $n-1$ total edges). We define $f(x) = 0$ if x is the all-zero string, and $f(x) = 1$ otherwise. What is $\mathrm{R}(f)$ and $\mathrm{Q}(f)$?

Note that a strategy of picking random bits and querying them has some chance of finding an edge if the input is a star; this probability is $(n-1)/\binom{n}{2} = 2/n$ per query. After $O(n)$ queries, this strategy finds a 1 in the string with high (constant) probability if the input is a star. Hence $\mathrm{R}(f) = O(n)$. Using Grover search, a quantum algorithm can speed this up quadratically, to $\mathrm{Q}(f) = O(\sqrt{n})$.

What about the lower bounds? It might be tempting to try to use block sensitivity, but it turns out that the block sensitivity of this function is 1. That's because if the input is a star, the only block to flip to get a 0 input is the entire star. On the other hand, if the input is all-0, then to get a 1-input we must flip an entire star, but any two stars overlap on some bit, and so are not disjoint (this is because the star centered at vertex u and the star centered at vertex v overlap on the edge between u and v).

Instead, we use fractional block sensitivity. We pick the all-zero string, and assign a weight to each of its sensitive blocks—that is, to each star. There are n stars (one centered on each vertex), and we will give them each weight $1/2$. Then for every bit i, this bit corresponds to an edge $\{u, v\}$, and there are only two sensitive blocks that use this edge: the star centered on u and the star centered on v. Hence for each bit, the weight of all blocks containing it is 1. On the other hand, the total weight of all blocks is $n/2$. We therefore get $\mathrm{R}(f) = \Omega(n)$ and $\mathrm{Q}(f) = \Omega(\sqrt{n})$, meaning that our upper bounds were tight.

Example 2. Consider the function f on $2n$ bits whose promise is that the Hamming weight of the input is 1. If the 1 occurs in the first n bits of x, we have $f(x) = 0$, whereas if the 1 is in the second n bits of x, we have $f(x) = 1$. What are $\mathrm{R}(f)$ and $\mathrm{Q}(f)$?

This function f wants you to find the position of the 1; this can be solved using $O(\sqrt{n})$ queries quantumly, but seems to require roughly n queries classically. But how do we prove these lower bounds?

We can look at the block sensitivity of this function. At a given input x, say with $x_1 = 1$, to flip the value of $f(x)$ we must move the 1 from the left to the right; this requires flipping two bits, x_1 and another bit x_m with $m \in (n, 2n]$. Note that all of these sensitive blocks for x overlap on the bit x_1. This means the block sensitivity of x is 1.

What about the fractional block sensitivity? It turns out that this is also 1! That's because all the blocks of x overlap on the same bit; since we're only allowed to have total weight 1 on that bit, we can only assign total weight 1 to all the sensitive blocks of x, so $\mathrm{fbs}(f, x) = 1$ for all x.

Now what? Well, it turns out that we can add another input to the domain of f to solve our problems. Consider the string 0^{2n}, which is not in the domain of f. Intuitively, 0^{2n} is hard to distinguish from both the 0-inputs of f and also from the 1-inputs of f; this can imply that the 0-inputs are hard to distinguish from the 1-inputs.

The way this trick will work is to consider a quantum algorithm Q for f, and then run Q on 0^{2n}. Of course, 0^n is not in the domain of f. However, it is still possible to run Q on it and see what happens. Q will output 1 with some probability p. We know that $p \in [0, 1]$, but since 0^{2n} is not in the domain of f, there are no further guarantees on p: it might even be equal to $1/2$. Still, let's look at whether p is closer to 0 or to 1. We must have either $p \leq 1/2$, or else $p \geq 1/2$. In the former case, Q distinguishes 0^{2n} from all 1-inputs, because it accepts 1-inputs with probability at least $2/3$ but accepts 0^{2n} with probability at most $1/2$; with some rebalancing and amplification, we can get an algorithm Q' (which costs only a constant factor more than Q) which distinguishes 0^{2n} from all 1-inputs. Similarly, if $p \geq 1/2$, we can get an algorithm Q' which distinguishes 0^{2n} from all 0-inputs of f.

Either way, the algorithm Q' solves a task that requires $\Omega(\sqrt{n})$ quantum queries to solve. This is because this task requires distinguishing an all-zero string from n different strings of Hamming weight 1. The block sensitivity of this task is now n, which means that $\Omega(\sqrt{n})$ quantum queries are required. Since Q' solves this task using only a constant factor more queries than Q, it must be the case that Q used $\Omega(\sqrt{n})$ queries as well, so $\mathrm{Q}(f) = \Omega(\sqrt{n})$. A similar argument can be used to establish the randomized lower bound.

2.5 Randomized and quantum certificates

Finally, we note that there is yet another interpretation for fractional certificates. For any input x to a function f, define a new function f_x which is a restriction of f to a promise. This function

f_x will have domain $\{x\} \cup \{y \in \mathrm{Dom}(f) : f(y) \neq f(x)\}$, and on this domain it will behave like f. Then clearly $\mathrm{D}(f_x) \leq \mathrm{D}(f)$, $\mathrm{R}(f_x) \leq \mathrm{R}(f)$, and $\mathrm{Q}(f_x) \leq \mathrm{Q}(f)$.

Now, it is not hard to see that $\mathrm{D}(f_x) = \mathrm{C}(f, x)$. That's because solving f_x only requires checking if the input equals x or has opposite f-value, which can be done by querying all the bits of a certificate of x with respect to f. This gives an interesting relationship between the measures $\mathrm{D}(f)$ and $\mathrm{C}(f)$.

As it turns out, we also have $\mathrm{R}(f_x) = \Theta(\mathrm{FC}(f, x))$. In one direction, if we had a fractional certificate for x, we could query each of the bits of the input with probability equal to their assigned weight in the fractional certificate; if we make the choice of querying or not independently for each bit, this will cost an expected $\mathrm{FC}(f, x)$ queries, and will discover a difference from x (if it is there) with probability at least $1/e$. With amplification, this can be turned into a bounded-error algorithm for f_x of expected cost $\mathrm{FC}(f, x)$, and using the Markov cutoff trick we can turn the expected cost into a worst-case cost. In the other direction, note that $\mathrm{fbs}(f_x, x) = \mathrm{fbs}(f, x)$, which we have already seen is a lower bound on $\mathrm{R}(f_x)$.

What is $\mathrm{Q}(f_x)$? Well, we know that $\mathrm{Q}(f_x) = \Omega(\sqrt{\mathrm{fbs}(f_x, x)}) = \Omega(\sqrt{\mathrm{fbs}(f, x)})$. It turns out the upper bound also holds: using a modification of Grover search combined with the randomized algorithm approach, we can get $\mathrm{Q}(f_x) = O(\sqrt{\mathrm{fbs}(f, x)})$, so we have $\mathrm{Q}(f_x) = \Theta(\sqrt{\mathrm{fbs}(f, x)})$.

For this reason, the measure $\mathrm{fbs}(f, x)$ (or equivalently $\mathrm{FC}(f, x)$) is sometimes called the randomized certificate complexity of f at x, because it is a randomized analogue of $\mathrm{C}(f, x)$. Similarly, $\sqrt{\mathrm{fbs}(f, x)}$ is sometimes called the quantum certificate complexity of f. These definitions were first defined in [Aar08].

References

[Aar08] Scott Aaronson. "Quantum certificate complexity". In: *Journal of Computer and System Sciences* 74.3 (2008). Computational Complexity 2003. DOI: 10.1016/j.jcss.2007.06.020. eprint: 1506.04719 (p. 11).

[BBB+97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. "Strengths and Weaknesses of Quantum Computing". In: *SIAM Journal on Computing (special issue on quantum computing)* 26 (5 1997) (p. 1).