6.896 Quantum Complexity Theory October 9th, 2008

Lecture 11

Lecturer: Scott Aaronson

In this class, we'll finish the proof for $D(f) = O(Q(f)^6)$ for all total Boolean functions f.

1 Block Sensitivity

We already saw the first step in the proof: $D(f) \leq C(f)^2$ for all f, where $C(f)$ is the certificate complexity. We will now do the second step, which involves *block sensitivity*.

Definition 1 *Given input $x \in \{0,1\}^n$ and a subset $S \subseteq \{1,...,n\}$ of bits, let $X^{(S)}$ denote X with the bits in S flipped. We call S a* sensitive block *on input X if $f(X) \neq f(X^{(S)})$.*

$bs_x(f)$, the *block sensitivity* of f on input X, is the maximum number of *disjoint* sensitive blocks that can be found on input X.

We define the *block sensitivity* of f to be $bs(f) = \max_x bs_x(f)$.

Claim 1 *$bs(f) \leq C(f)$ for all f.*

This is because to prove the value of $f(X)$, a certificate must intersect each sensitive block. Otherwise, flipping the bits in a sensitive block would change $f(X)$ without affecting the certificate's value.

Lemma 2 (Nisan) *$C(f) \leq bs(f)^2$ for all f.*

Proof: Given an input X, consider a maximal set of $k = bs_x(f)$ disjoint sensitive blocks $S_1, ..., S_k$ for X. Note that $k \leq bs(f)$. Assume without loss of generality that each S_i is *minimal* (i.e., no sub-block of S_i is also sensitive). If S_i was not minimal, we could always use a sensitive sub-block of S_i instead, without affecting the maximality of the set $S_1, ..., S_k$.

Claim 3 *The union of bits $S_i \bigcup ... \bigcup S_k$ is a certificate, proving the value of $f(X)$.*

To show this, suppose we wanted to flip some bits in X to change the value of $f(X)$. The bits we flipped would have to comprise a sensitive block S_{k+1}. But if S_{k+1} didn't intersect any of $S_1, ..., S_k$, it would contradict the assumption that $S_1, ..., S_k$ was a maximal set of disjoint sensitive blocks.

So the bits must intersect $S_i \bigcup ... \bigcup S_k$. If we monitor $S_i \bigcup ... \bigcup S_k$ as a certificate, then we are forcing the value of $f(X)$, since it cannot be changed without the certificate also changing.

Now we put an upper-bound on the size of $S_i \bigcup ... \bigcup S_k$. As stated before, $k \leq bs(f)$. Now we will show that each S_i has at most $bs(f)$ bits, making

$$|S_i \bigcup ... \bigcup S_k| \leq bs(f)^2 \tag{1}$$

Consider $X^{(S_i)}$. We know that $f(X^{(S_i)}) \neq f(X)$. By the assumption that S_i is minimal, flipping any bit in S_i changes the value of $f(X^{(S_i)})$ back to $f(X)$; otherwise, there'd be a smaller

sensitive block for X. So each bit in S_i is itself a sensitive block for $X^{(S_i)}$. Hence, $|S_i| \leq bs(f)$. So $|S_i \bigcup ... \bigcup S_k| \leq bs(f)^2$.

Since $S_i \bigcup ... \bigcup S_k$ is a certificate, we can conclude that $C(f) \leq bs(f)^2$ for all f.

□

2 Block Sensitivity and Approximate Degree

The third and final step is to relate $bs(f)$ to the approximate degree $\widetilde{deg}(f)$. We've already proven that $\widetilde{deg}(OR_n) = \Omega(\sqrt{n})$. We will now claim a generalization of that result.

Lemma 4 *$\widetilde{deg}(f) = \Omega(\sqrt{bs(f)})$ for all Boolean functions f.*

Proof: Consider an input X of f with $k = bs(f)$ disjoint sensitive blocks $S_1, ..., S_k$. Assume without loss of generality that $f(X) = 0$. Now define a new Boolean function with k inputs: $f'(y_1, ..., y_k)$ equals $f(X$ with S_i flipped iff $y_i = 1)$. Suppose there were a polynomial p that approximated f. Let $p'(y_1, ..., y_k)$ equal $p(X$ with S_i flipped iff $y_i = 1)$. Symmetrize p' to get a univariate polynomial

$$q(k) = EX_{|Y|=k}[p'(Y)]. \tag{2}$$

$0 \leq q(0) \leq \frac{1}{3}$ and $\frac{2}{3} \leq q(1) \leq 1$. We don't know what $q(2), ..., q(k)$ are, but we know that as averages of 0's and 1's, they must be bounded between 0 and 1. By Markov's inequality, $deg(q) = \Omega(\sqrt{k})$, which means that $\widetilde{deg}(f) = \Omega(\sqrt{bs(f)})$. □

3 Putting It All Together

Thus, we can conclude that

$$D(f) \leq C(f)^2 \leq bs(f)^4 = O(\widetilde{deg}(f)^8) = O(Q(f)^8) \tag{3}$$

Note that we had claimed that $D(f) = O(Q(f)^6)$. Earlier, we proved that $D(f) \leq C(f)^2$, but it turns out that this bound can be tightened to $D(f) \leq C(f)bs(f)$.

Proof: To show that this is true, recall the algorithm that repeatedly picks a 1-certificate consistent with everything queried so far, and then queries all the bits in it. We proved that this algorithm cannot run for more than $C(f)$ steps, but actually, it cannot run for more than $bs(f)+1$ steps.

For the purposes of contradition, suppose it did; suppose we've picked and queried more than $bs(f)+1$ certificates. Because the algorithm hasn't stopped yet, there must still be both a 0-input and a 1-input compatible with our queries. Let X be such a 0-input. For this X, we can find more than $bs(f)$ disjoint sensitive blocks, a contradiction. This is because in each iteration, we queried a potential 1-certificate, meaning that had a subset of those bits been flipped, f would have been *forced* to be 1. That subset is a sensitive block. Since there are $bs(f)+1$ certificates and they're disjoint, we have more than $bs(f)$ disjoint sensitive blocks. □

Thus,

$$D(f) \leq C(f)bs(f) \leq bs(f)^3 = O(\widetilde{deg}(f)^6) = O(Q(f)^6) \tag{4}$$

This is still the best relation known. Another open problem is whether or not there is a similar polynomial relation between $D(f)$ and $Q(f)$ for *almost*-total Boolean functions, where the function is defined for only $(1-\epsilon)$ fraction of the inputs.

4 Interesting Examples of Boolean Functions

This is a discussion of particular Boolean functions that people are interested in and have done research on.

Recall the previously discussed OR/AND tree problem, a.k.a. "Is there an all-ones row?" The classical query complexity for this problem is n.

An upper-bound for the quantum query complexity is $O(n^{\frac{3}{4}})$, which comes from applying Grover's algorithm separately to each AND subtree.

Applying Grover's algorithm recursively, we can get $O(\sqrt{n}\log n)$. (The $\log n$ comes from the need to use amplification to reduce the error.) A more careful recursive application of Grover's algorithm can even reduce that to $O(\sqrt{n})$.

The OR/AND tree problem can also be thought of as a game, where Player 1 picks a subtree and Player 2 picks a leaf within that subtree. Player 1 wins if a 1 is reached, Player 2 wins if a 0 is reached, and the question is who wins?

To find a lower-bound on the quantum query complexity, we can apply the BBBV lower-bound to a single subtree and get a bound of $\Omega(n^{1/4})$. Raising that lower-bound to $\Omega(\sqrt{n})$ had been an open problem for several years. The difficulty comes from the OP/AND tree not being a symmetric Boolean function, so symmetrization doesn't yield anything. Proving that the approximate degree is $\Omega(\sqrt{n})$ remains an open problem, but we *can* prove it is $\Omega(n^{\frac{1}{3}})$.

4.1 Ambainis's Quantum Adversary Method

Andris Ambainis introduced a completely new quantum lower bound method that has since had an enormous influence on the field. We're not going to go into the details of the proof, but we'll show what the theorem is and how it's applied.

Theorem 5 (Ambainis) *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, X be some set of 0-inputs, and Y be some set of 1-inputs. Let $R \subset X \times Y$ be a relation such that*

1. *For every $x \in X$, there are at least m $y \in Y$ such that $(x,y) \in R$.*
2. *For every $y \in Y$, there are at least m' $x \in X$ such that $(x,y) \in R$.*
3. *For every $x \in X$ and $i \in [n]$, there are at most s $y \in Y$ such that $(x,y) \in R$ and $x_i \neq y_i$.*
4. *For every $y \in Y$ and $i \in [n]$, there are at most s' $x \in X$ such that $(x,y) \in R$ and $x_i \neq y_i$.*

Then any quantum algorithm to compute f needs $\Omega(\sqrt{\frac{mm'}{ss'}})$ queries.

The basic proof idea is to consider a superposition of inputs. If the algorithm succeeds at distinguishing 0-inputs from 1-inputs, then by the end, this pure state must evolve into a mixed state; certain off-diagonal entries in the density matrix must be 0. The theorem bounds how long this takes to happen.

The conditions for the theorem basically state that every 0-input has many 1-inputs as neighbors, but not too many 1-inputs as neighbors that differ from it at any specific bit. The same goes for 1-inputs with 0-inputs as neighbors.

4.2 Application of the Quantum Adversary Method

Let's apply the theorem to the OR problem. The set of 0-inputs X contains only the all-0 input. The set of 1-inputs Y will consist of all inputs with Hamming weight 1. In general, we want the 1-inputs to be close to the 0-inputs to maximize the lower bound generated by the theorem.

Every 0-input has $m = n$ neighbors. Every 1-input has $m' = 1$ neighbor. Every 0-input has $s = 1$ neighbor that differs from it at a specific bit. Every 1-input has $s' = 1$ neighbor that differs from it at a specific bit. Plugging these values in, we get $\Omega(\sqrt{n})$, confirming Grover's algorithm's tightness.

Now let's apply the theorem to the OR/AND tree problem, modeled as a $\sqrt{n} \times \sqrt{n}$ matrix where we need to determine if there's a row of all 1's. A useful trick is often to use inputs that are right at the threshold between being 0-inputs or 1-inputs.

Let's say that each row in a matrix has either zero or one 0 in it; that is to say, each row is either all 1's or almost all 1's. A 1-input would have a single row of all 1's and all other rows have 1's for all but one bit. A 0-input would have each row have one zero. If a 0-input matrix x and a 1-input matrix y differ at just a single bit, we'll make them neighbors, i.e. $(x, y) \in R$.

Every 0-input has $m = \sqrt{n}$ neighbors, by changing one of the $\sqrt{n}$ 0 bits into a 1 bit. Every 1-input has $m' = \sqrt{n}$ neighbors, by changing one of the $\sqrt{n}$ 1 bits in the all 1's row into a 0 bit. Every 0-input has at most $s = 1$ neighbor that differs from it at a specific bit. Every 1-input has at most $s' = 1$ neighbor that differs from it at a specific bit. Plugging those values in, we get that an algorithm that solves the OR/AND problem must have quantum query complexity $\Omega(\sqrt{n})$.

Next time, we'll talk about the collision problem.

6.845 Quantum Complexity Theory
Fall 2010