6.896 Quantum Complexity Theory October 14, 2008

Lecture 12

Lecturer: Scott Aaronson

1 A second non-symmetric example

1.1 Limitations of speed-ups without structure

In the preceeding lectures, we examined the relationship between deterministic query complexity and quantum query complexity. In particular, we showed

Theorem 1 $\forall$Boolean $f D(f) = O(Q(f)^6)$

so we won't ever obtain a superpolynomial improvement in the query complexity of *total* functions, and to obtain a superpolynomial speed-up, we can't treat the problem as a black-box like Grover's algorithm does. We need to exploit some kind of structure, possibly in the form of a promise on the function, as Shor's algorithm does. Moreover, for symmetric Boolean functions – OR, MAJORITY, PARITY, and the like – the largest separation possible is only quadratic, which is achieved by the OR function, as demonstrated by Grover's algorithm and our various lower bounds. For functions such as MAJORITY, which switches from 0 to 1 around the middle Hamming weight, the advantage only diminishes, and any quantum algorithm can be shown to require $\Omega(n)$ queries, just as classical algorithms do.

We don't have such a clear picture of the state of affairs for non-symmetric Boolean functions, but a few concrete examples have been worked out. A natural next question is to consider what happens when our simple functions are composed. For example, we saw last time that for the two level OR-AND tree (with $\sqrt{n}$ branches at each level), the quantum query complexity is $\Theta(\sqrt{n})$, with the upper bound provided by a recursive application of Grover's algorithm, and the lower bound obtained via Ambainis' adversary method (stated without proof)—the quantum extension of the BBBV hybrid argument and variants, where we argue about what a quantum algorithm can do step-by-step. By contrast, we still don't know how to obtain the lower bound using the polynomial method, where we reduce questions about quantum algorithms to questions about low-degree polynomials, (the "pure math" approach) which is elegant *when* it works. Thus, these two methods seem to have complementary strengths and weaknesses.

1.2 The AND-OR tree, or: the power of randomization in black-box query complexity

The second example which we understand is also an AND-OR tree, but it is deeper, consisting of $\log n$ levels with two branches at each node (see Figure 1). We think of this as modeling a $\log n$-round game of pure strategy between two players in which the players have two options at each round (we could think of it more generally as a constant number of moves), and the winner is determined by a black-box evaluation function—the natural computational question in such a game is, "is there a move I can make such that for every move you can make, I can force a win?" This is precisely the problem of game tree evaluation. This black-box assumption allows us to begin

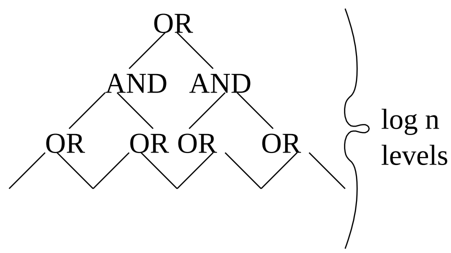

Figure 1: $\log n$-depth AND-OR tree

to address the question of what kind of a speed-up we can hope to obtain by using a quantum algorithm for playing games. The kind of question we know how to address is again about the number of queries we require—how many of the leaves of the tree (labeled with bits) we need to examine to determine its value.

The best classical algorithm turns out to be very simple:

Algorithm $EVAL-TREE$: For the tree rooted at vertex v,

1. Let u be the left or right child, chosen uniformly at random; run $EVAL-TREE(u)$.
2. If v is an AND and $EVAL-TREE(u) = 0$ or v is an OR and $EVAL-TREE(u) = 1$, return 0 or 1, respectively.
3. Otherwise, if w is the other child, return $EVAL-TREE(w)$.

The randomization is very important, since otherwise we might "get unlucky" at each level and need to evaluate every branch of the tree. It is actually an easy exercise to analyze the running time of this algorithm—what is more difficult to see is that this algorithm is actually *optimal*:

Theorem 2 (Saks-Wigderson '86) *$R(AND-OR) = \Theta(n^{\log \frac{1+\sqrt{33}}{4}})$. (where $\log \frac{1+\sqrt{33}}{4} \approx .754$)*

This function is conjectured to exhibit the largest separation possible between classical query complexity and randomized query complexity (without promises). It is certainly, at least, the largest known separation. Of course, the caveat about promises is essential here too—if we are promised that the input has Hamming weight either at least $2/3n$ or at most $1/3n$ and we need to decide which, then deterministic algorithms need more than $n/3$ queries, but randomized algorithms only need one query.

In any case, the situation with randomized algorithms is similar to the one we faced with quantum algorithms. In the following, let R_ϵ denote the query complexity of a randomized algorithm that is allowed to err with probability ϵ. It is easy to see that for "Las Vegas" algorithms (i.e., $\epsilon = 0$, like our pruning algorithm), $R_0(f) \geq C(f)$, since the algorithm must see a certificate before it halts—otherwise, there's both a 0-input and a 1-input that are consistent with the bits queried so far, so we would err on some input. Since we also saw $D(f) \leq C(f)^2$, we find that $D(f) \leq R_0(f)^2$ for all Boolean total functions f, but we don't know whether or not this is tight—the $\log n$-level AND-OR tree obtains the largest *known* speed-up. Likewise, for "Monte-Carlo" algorithms ($\epsilon > 0$), observe that if f has block sensitivity k, then we have to examine each of our disjoint blocks with

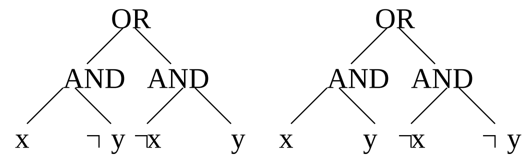

Figure 2: Two-level AND-OR trees computing $x \oplus y$ (left) and $\neg(x \oplus y)$ (right).

probability at least $1 - 2\epsilon$, since again otherwise there would be a 1-input and a 0-input that only differ in some unexamined block, and then even if we randomly output 0 or 1, we are still incorrect with proabability greater than $(1/2)2\epsilon = \epsilon$. Since we showed $D(f) \leq \mathsf{bs}(f)^3$, $D(f) = O(R_\epsilon(f)^3)$. Once again, the AND-OR tree obtains the largest known gap, and we don't know if this cubic gap is exhibited by any function. These facts, along with the application to game playing, are why this AND-OR tree is considered to be an extremely interesting example (the "fruit fly" of query complexity).

1.3 The quantum query complexity of the AND-OR tree

We now turn examining the quantum query complexity of this problem. We can get an easy quantum lower bound by a reduction from the parity problem. Observe that the parity of two bits x and y and its negation are computed by two-level binary AND-OR trees (see Figure 2). By recursively substituting these trees for the variables x and y, it is easy to see that we obtain the parity of n bits in $2 \log n$ levels—that is, in a tree with $n' = O(n^2)$ leaves. Given an instance of the parity problem, it is not hard to see how, given the path to a leaf, we could recursively decide which bit x_i of the input we should place at that leaf and whether or not that bit should be negated. Since we saw using the polynomial method that the parity of n bits required $n/2$ queries to compute, this AND-OR tree requires $\Omega(\sqrt{n'})$ queries to evaluate. (Since a more efficient query algorithm for evaluating the AND-OR tree would yield an impossibly query efficient algorithm for computing the parity of n bits.) Alternatively, it is possible to use Ambainis' adversary method to obtain the $\Omega(\sqrt{n})$-lower bound as well.

We don't know how to obtain a better lower bound. It is also not easy to see how we can obtain an algorithm that is more efficient than our $O(n^{.753})$-query classical algorithm—it isn't clear how to obtain an upper bound by applying, e.g., Grover's algorithm recursively to this problem since we only have subtrees of size two and moreover there is a recursive build-up of error at the $\omega(1)$ internal nodes. Despite this, in 2006, Farhi, Goldstone, and Gutmann found a $O(\sqrt{n})$-query "Quantum walk" algorithm – sort of a sophisticated variant of Grover's algorithm – for evaluation of these AND-OR trees using intuitions from scattering theory and particle physics. (Contrary to our experience earlier in the course, this is an example where knowledge of physics turned out to be useful in the design of an algorithm.) Thus in fact, our easy $\Omega(\sqrt{n})$ lower bound turns out to be tight. Interestingly, this is an example of a function where the quantum case is simpler than the classical (randomized) case—$\Theta(\sqrt{n})$ versus $\Theta(n^{.754})$, where the classical lower bound was a highly nontrivial result.

2 The Collision Problem

We will conclude our unit on quantum query complexity with an interesting non-boolean problem, "the collision problem." It is a problem that exhibits more structure than Grover's "needle-in-a-haystack" problem, but less structure than the period finding problem, "interpolating" between the two (in some sense). Informally, rather than looking for a "needle in a haystack," we are merely looking for two identical pieces of hay:

The Collision Problem. Given oracle access to a function $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ with n even (i.e., the oracle maps $|x\rangle|b\rangle$ to $|x\rangle|b \oplus f(x)\rangle$) promised that either f is one-to-one or two-to-one, decide which.

Variant: Promised that f is two-to-one, find a collision.

Clearly, the decision problem is easier than this search variant, since an algorithm for the search problem would provide a witness that a function is not one-to-one, i.e., that the function must be two-to-one given the promise (and of course, it would fail to find a collision in a one-to-one function).

Key difference: One way that this problem has "more structure" than Grover's problem is that any one-to-one function must have distance at least $n/2$ from any two-to-one function. Thus, any fixed one-to-one function can be trivially distinguished from any fixed two-to-one function by querying a random location. (They agree with probability at most 1/2.)

Obviously, the deterministic query complexity of this problem is (exactly) $n/2 + 1$, since if we are unlucky, we might see up to $n/2$ distinct elements when we query a two-to-one function. It is also not hard to see that the randomized query complexity is $\Theta(\sqrt{n})$ by the so-called "Birthday Paradox." Similar to our analysis of Simon's algorithm, suppose we choose a two-to-one function f uniformly at random. Then, for any fixed pair of distinct locations, x_i and x_j, the probability that $f(x_i) = f(x_j)$ is $\frac{1}{n-1}$, so we know that by a union bound, the total probability of seeing a collision in any k queries is $\binom{k}{2}\frac{1}{n-1}$. Thus, in particular, for $k(n) = o(\sqrt{n})$, the probability of seeing a collision is $o(1)$ and we can't obtain a sufficiently small error probability. By contrast, for $k(n) = \Omega(\sqrt{n})$ queries to random locations in any two-to-one function, it is easy to see (using, e.g., Chebyshev's inequality) that the expected number of collisions is $\Omega(1)$, so we can find a collision with constant probability, solving the search variant of the problem.

2.1 Motivation

This problem is interesting for a few reasons. First of all, graph isomorphism reduces to this problem. Fix a graph G, and consider the map $\sigma \mapsto \sigma(G)$ (for $\sigma \in S_n$). Notice that, for the graph $G \cup H$ (assuming G and H are rigid, i.e., only have a trivial automorphism), this map is two-to-one if G and H are isomorphic and it is one-to-one otherwise. So, one might wonder if there could be a $O(\log n)$-query algorithm for the collision problem, since that would lead to a polynomial-time algorithm for the graph isomorphism problem—in particular, if we could find a random collision, we could remove the restriction on rigidity (we could use approximate counting). In any case, note that since the map here is on a domain of size $(2n)!$, we would need a $\operatorname{poly}\log(2n!) \sim \operatorname{poly}(n)$-query algorithm for this application. That is, we wonder whether or not there is an efficient algorithm for graph isomorphism when we ignore the group structure.

There's also an application to breaking cryptographic (collision-resistant) hash functions, used in digital signatures on the internet, for example—the hash function is applied to some secret message (credit-card number, etc.) to create a "signature" of that message. Often, the security of a protocol depends on the assumption that it is computationally intractible to find a collision in the hash function – to find two messages m and m' that map to the same hash value – e.g., this might happen if the purpose of the hash value was to commit to a value without revealing it. (It may also be that finding a collision helps us find a stronger break in a hash function.)

There is a classical attack based on the "Birthday Paradox," – the "birthday attack" – which proceeds by hashing random messages and storing their results until two messages are found that hash to the same value. For $N = 2^n$, it follows from the Birthday Paradox that this attack finds a collision in $O(\sqrt{N})$ evaluations with constant probability—a quadratic improvement over what one might naively expect. Again, if there were a $\operatorname{poly}\log N$-query quantum algorithm for this problem or, more generally, for finding collisions in any k-to-one function – we can assume that the collisions are evenly distributed, since this minimizes the total number of collision pairs, and any unbalanced distribution generally only makes an algorithm's task easier (this is not a proof, but in what follows, our upper bounds will work in the non-uniform case, and our lower bounds will apply in the uniform case, so we will have covered this, in any case) – that would yield a poly(n) time quantum algorithm for breaking any cryptographic hash function.

2.2 Algorithms for the collision problem

We now consider quantum algorithms for this problem. We have talked at length about how this problem has "more structure" than Grover's problem; how can we use the additional structure exhibited by this problem to find a better algorithm? We start by showing a different (easy) $\sqrt{N}$-query algorithm for finding collisions: we query $f(1)$, and we use Grover search to find the other index i such that $f(i) = f(1)$. We could also have used Grover's algorithm on all possible collision pairs (x_i, x_j).

It is possible to combine this Grover search algorithm with our "Birthday" algorithm to obtain a $O(n^{1/3})$-query algorithm as follows:

Algorithm. (Brassard-Høyer-Tapp 1997)

1. Make $n^{1/3}$ classical queries at random: $f(x_1), \ldots, f(x_{n^{1/3}})$
2. Enter superposition over the remaining $n^{2/3}$ positions.
3. Apply Grover search to find an element in the initial list.

Grover's algorithm takes $O(n^{1/3})$ queries to find such an element in the final step, so this algorithm uses $O(n^{1/3})$ queries overall. (Clearly, the $n^{1/3}$ was chosen to optimize the trade-off we obtain.)

We know, by the correctness of Grover's algorithm, that this algorithm will work when there's a collision between the elements sampled in phase 1 and the elements in the superposition in phase 2. Thus, to see that this algorithm works, we only need to examine the probability of a collision. We observe that there are $n^{1/3}n^{2/3} = n$ pairs of phase-1 and phase-2 elements. A uniformly chosen pair of distinct elements would have probability $\frac{1}{n-1}$ of being a collision pair, but this doesn't quite apply here since our n pairs are not uniformly chosen—there are correlations. Nevertheless, the probability of a collision between a fixed pair in the two lists is still $\Omega(1/n)$ so we can apply an analysis similar to the standard analysis of the Birthday Paradox to find that a collision exists with constant probability.

Is this optimal? It is challenging to find a super-constant lower bound for this problem: for example, it is hard to find a hybrid argument, since it is hard to interpolate between any one-to-one function and any two-to-one function (since they differ in so many places). Likewise, if we tried to apply our block sensitivity lower bound, we find that the block sensitivity of this problem is 2 (there are at most two disjoint ways of changing a one-to-one function to a two-to-one function), so our block sensitivity lower bound yields a mere $\Omega(\sqrt{2})$-query lower bound.

There is another, more illuminating way of framing the difficulty: a quantum algorithm can "almost" find a collision pair in just a constant number of queries! Suppose we prepare a superposition $\frac{1}{\sqrt{n}} \sum_{x=1}^{n} |x\rangle|f(x)\rangle$ and measure the second register; this yields a superposition $\frac{|x\rangle+|y\rangle}{2}$ for a random pair x and y such that $f(x) = f(y)$ in the first register. Thus, if we could only measure this state twice, we could find a collision pair. (By constrast, it isn't clear how measuring twice would allow one to solve the Grover problem in a constant number of queries.)

Next time, we'll see a $\Omega(n^{1/5})$-query lower bound (A. 2002) which was improved in subsequent weeks to $\Omega(n^{1/4})$ and then $\Omega(n^{1/3})$ by Yao and Shi, with some restrictions—only when the range of the function was much larger than n. These restrictions were later removed by Kutin and Ambainis, so the $O(n^{1/3})$ algorithm again turns out to be optimal.

2.3 The collision problem and "hidden variable" interpretations of quantum mechanics

An additional motivation for this problem (described in A. 2002) was studying the computational power of "hidden variable theory" interpretations of quantum mechanics. This school of thought says that a quantum superposition is at any time "really" in only one basis state. This basis state is called a "hidden variable" (although ironically, it is the one state that is not "hidden," but rather is directly experienced). So, like in many-worlds interpretations, there is a quantum state with amplitudes for basis states for the many possibilities of states that the world could be in, but in contrast to the many-worlds interpretation, most of these states are just some guiding field in the background, and the world is actually in one distinguished state.

What significance does this have to quantum computing? One might think, "absolutely nothing," since all of these interpretations make the same experimental predictions, and thus lead to the same computational model. But, if we could see a complete history of these true states, we could solve the collisions problem – and hence the graph isomorphism problem – in a constant number of queries. We could prepare a superposition over a collision pair as described earlier, and apply transformations (e.g., Hadamards) to "juggle" the true state between the basis states of the superposition. Thus, given a lower bound for the collision problem, we see that the additional information in this history provably gives additional computational power.

6.845 Quantum Complexity Theory
Fall 2010