CS 810: Complexity Theory	4/18/2007

Lecture 29: Query Complexity

Instructor: Dieter van Melkebeek — Scribe: Jeff Kinne

The topic of today's lecture is query complexity. Suppose we have access to a database and wish to evaluate a predicate over the information contained in the database. The goal is to evaluate the predicate with as few queries to the database as possible. We formalize the problem and examine the query complexity of three models: deterministic, randomized, and quantum. We show that for fully specified problems the query complexities of all three models are within polynomial factors of each other; but for promise problems we give examples that demonstrate exponential gaps in the complexities for the three models.

1 Query Complexity Models

We first formalize the intuitive notion of a database described above. We let our database be specified by a bit string $x \in \{0,1\}^n$, and let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a predicate we wish to evaluate on x with as few accesses to x as possible. Most natural functions that depend on all the input bits (majority, parity, and, or, etc.) require full query complexity on deterministic machines. However, database queries are often very simple and only depend on few input bits. Also, some interesting functions have smaller query complexity in alternate models (see below).

We will look at the query complexity on three different models.

1. **Deterministic**. As we are only interested in the queries that are generated and not the complexity of the computation, we view the evaluation of f in this model as a *decision tree*. The root of the decision tree is the first bit of x queried, say x_i. The child on its left branch is the next bit queried if $x_i = 0$, and the child on the right branch for $x_i = 1$. The leaves of the tree correspond to our outputting a value for f. Notice that the *depth* of the decision tree measures the maximum number of queries that are asked for any string $x \in \{0,1\}^n$.

   We let $D(f)$ denote the deterministic query complexity of f - the minimum depth of a decision tree deciding f.

2. **Randomized**. We use the two-sided bounded-error setting. We would like to measure the worst-case number of queries that the machine asks over all inputs and random strings. The computation of the machine can be viewed as first picking a random string and then performing a deterministic evaluation. In other words, we pick a random string, and that random string determines a decision tree to use to evaluate the function. Thus we view the computation as a probability distribution over deterministic decision trees, with the worst-case query complexity corresponding to the maximum depth of any tree in the support of the distribution of trees.

   We let $R(f)$ denote the randomized query complexity of f - the minimum depth d such that there exists a probability distribution on decision trees of depth at most d that decides f correctly on every input with probability $> 2/3$.

3. **Quantum**. In the quantum setting, recall that we in general have access to unitary operations that act on superpositions of bits. Thus we provide a unitary operation U_x for accessing the input. We define U_x as $U_x : |i, b, z\rangle \rightarrow |i, b \oplus x_i, z\rangle$, where i is the index to the bit of x we want, b is an auxiliary bit for XORing the desired bit into the output, and z contains the remaining quantum bits we are working with. For quantum machines, we measure the query complexity as the number of times we need to apply U_x. As with the randomized case, we require that the quantum process computes the correct value on each input with probability $> 2/3$. Without loss of generality, we designate an output bit, which we measure at the end of the quantum computation to determine the output.

   We let $Q(f)$ denote the quantum query complexity of f - the minimum number of queries to U_x by a bounded-error quantum process solving f.

2 Examples

2.1 OR

$D(OR) = n$: given a decision tree of depth $n-1$, we can construct an x such that the decision tree gives the wrong value.

$R(OR) = \Theta(n)$. This can be argued by considering a randomized process deciding OR as a probability distribution over decision trees. Suppose $R(OR) = k$, that is that there is a probability distribution over depth at most k decision trees that decides f on each input with probability $> 2/3$. An averaging argument shows that there is at least one $i \in \{1, 2, ..., n\}$ such that $\Pr[i^{th}$ bit of input queried$] \leq \frac{k}{n}$ when using a decision tree from the distribution. Consider the strings 0^n and $0^{i-1}10^{n-i}$. Because $OR(0^n) = 0$ and $OR(0^{i-1}i0^{n-i}) = 1$, and we have assumed the distribution of trees solves OR with probability $> 2/3$, we conclude that

$$\Pr[\text{output different on } 0^n \text{ and } 0^{i-1}10^{n-i}] \geq 1/3.$$

But because the i^{th} bit is queried with probability at most $\frac{k}{n}$, $\Pr[\text{output different}] \leq \frac{k}{n}$. We conclude that $k \geq n/3$.

Alternatively, we can argue $R(OR) = \Theta(n)$ using the *Min-Max principle* described below.

2.1.1 Min-Max principle

The *Min-Max principle* can be used to argue lower bounds on R for many functions. To determine the query complexity of f, we think of playing a game between two players:

I) Pick a decision tree A of depth $\leq k$,
II) Pick an input x.

f has query complexity $\leq k$ if there is a randomized strategy for player I that results in $A(x) = f(x)$ with probability $\geq 2/3$. To formalize this, we introduce some notation. Let $M_{A,x} = \chi[A(x) \neq f(x)]$ be a matrix indicating for each decision tree of depth $\leq k$ which inputs it is incorrect on. Player I's strategy can be modeled as a probability distribution p over decision trees of depth $\leq k$. Then player I has a winning strategy (and thus $R(f) \leq k$) iff

$$\min_p \max_x (p^T M e_x) < 1/3.$$

Some manipulations give alternate formulations of this equation. We can also allow player II to have a randomized strategy; let q be the distribution over inputs x that player II chooses. Then

$$\min_p \max_x (p^T M e_x) = \min_p \max_q (p^T M q)$$

because e_x is a particular q and $p^T M q$ is a convex combination of the $p^T M e_x$. For any fixed q',

$$\min_p \max_q (p^T M q) \geq \min_p (p^T M q') = \min_A (e_A^T M q')$$

for similar reasons.

We can thus show that $R(f) > k$ by demonstrating a distribution q with $\min_A(e_A^T M q) \geq 1/3$. That is, we need to demonstrate a distribution of inputs so that all decision trees of depth at most k fail on this distribution with high probability.

We note that in fact an application of linear programming duality shows that $\min_p \max_q(p^T M q) = \max_q \min_p(p^T M q)$, which can be used to give tight bounds on $R(f)$ by picking q appropriately.

2.1.2 OR (continued)

We use the Min-Max principle on the function $f = OR$. Consider the distribution

$$q = \begin{cases} 1/3 \text{ on } 0^n, \\ 2/(3n) \text{ on } 0^{i-1}10^{n-i} \text{ for } 1 \leq i \leq n. \end{cases}$$

If $R(OR) = k$, then by the Min-Max principle, no matter which decision tree A of depth $\leq k$ we pick, A fails on this distribution with probability less than 1/3. Consider the all 0's path through A. Because 1/3 of the weight is on 0^n and A is supposed to fail with probability less than 1/3, the answer on the all 0's path cannot be 1. As the depth of A is at most k, at least $\frac{2}{3}(1 - \frac{k}{n})$ of the weight in the distribution is on strings whose true output is 1 but will go down the all 0's path of A. Then $\frac{2}{3}(1 - \frac{k}{n}) < \frac{1}{3}$, and we conclude that $k > \frac{n}{2}$.

We note that $Q(OR) = \Theta(\sqrt{n})$. This follows by Grover's search algorithm, a celebrated quantum algorithm which we do not cover in this course.

2.2 Hadamard Code Property Testing

Let f be the function that tests whether a string is either a Hadamard codeword or is far away from one. We point out that this is a promise problem. Recall that the linearity test used with PCPs for the Hadamard Code only required three queries (namely $x, y, x+y$). We conclude that $R(f) = \Theta(1)$. As a quantum process can simulate a random process, we also conclude that $Q(f) = \Theta(1)$. It can be shown that $D(f) = \Theta(n)$ by using an adversarial argument. Thus this function demonstrates a promise problem with an exponential gap between deterministic and randomized query complexity.

2.3 Simon's Problem

We demonstrate a promise problem with an exponential gap between randomized and quantum query complexity. Recall Simon's Problem: x is the characteristic sequence of $g : \{0,1\}^\ell \to \{0,1\}^\ell$ with the promise that either g is 1-1 or $\exists s \neq 0$ such that $g(y) = g(z) \Leftrightarrow y = z + s$. In the lecture on quantum effects, we gave a quantum algorithm to find s using ℓ queries. We cast the problem as a promise problem by setting $f(x) = 1$ if g_x is 1-1 and $f(x) = 0$ if there is an s as above. The quantum algorithm gives us $Q(f) = \Theta(\log n)$. It can be shown that $R(f) = \Omega(\sqrt{n})$ using the Min-Max principle and the birthday paradox rule.

3 Main Theorem

We have seen examples of promise problems where there is an exponential gap between the deterministic, randomized, and quantum query complexity. The main theorem of this lectures shows that this can only be true for promise problems. We show that for fully specified functions the query complexities D, R, Q are polynomially related.

Theorem 1. *For any fully specified f, $Q(f) \leq R(f) \leq D(f) \leq O((Q(f))^6)$.*

The first two inequalities follow from the efficient simulations of deterministic processes by randomized processes and of randomized processes by quantum processes. So we must show that $D(f) \leq O((Q(f))^6)$. We point out that the largest known gap in query complexity is the quadratic gap known for OR, so this result may not be tight.

3.1 First Step

We introduce a few related complexity measures. The first step in the proof is to relate $D(f)$ to these new complexity measures with a series of lemmas.

Definition 1. *The* certificate complexity *of f on input x is defined as*

$$W(f;x) = \min_{I \subseteq [n]} \{|I| \mid (\forall y) y|_I = x|_I \Rightarrow f(y) = f(x)\}.$$

In words, $W(f;x)$ is the minimum number of bits of x that need to be checked to uniquely determine the value of $f(x)$.

We also define $W_b(f) = \max_{x|f(x)=b} W(f;x)$ and $W(f) = \max(W_0(f), W_1(f))$.

For the OR function, we have that $W_0(OR) = n$ and $W_1(OR) = 1$. We point out that $D(f) \geq W_b(f)$ for all functions f because each path on a decision tree contributes a certificate for the inputs that end up there. This may not be the best certificate, so $W_b(f)$ can be smaller than $D(f)$.

Definition 2. *The* sensitivity *of f on input x is defined as*

$$S(f;x) = |\{i \in [n] \mid f(x + e_i) \neq f(x)\}|.$$

In words, $S(f;x)$ is the number of bits of x that if changed would change the value of f.

We also define $S_b(f) = \max_{x|f(x)=b} S(f;x)$ and $S(f) = \max(S_0(f), S_1(f))$.

For the OR function, we have that $S_0(OR) = n$ and $S_1(OR) = 1$.

Definition 3. *The* block sensitivity *of f on input x is defined as*

$$BS(f;x) = \textit{Max } \# \textit{ disjoint blocks } B_i \subseteq [n] \textit{ s.t. } (\forall i) f(x + \chi_{B_i}) \neq f(x).$$

In words, $BS(f;x)$ is the maximum number of disjoint subsets of bits of x that if flipped change the value of f.

We also define $BS_b(f) = \max_{x|f(x)=b} BS(f;x)$ and $BS(f) = \max(BS_0(f), BS_1(f))$.

For the OR function, BS and S are the same. There are other functions on which they differ. We point out that because S is a restriction of BS, $S_b(f) \leq BS_b(f)$ for all f and b. Further, $BS_b(f) \leq W_b(f)$ because flipping any bit in a minimal certificate flips the output of the function.

For the measures we have defined, we already have the following inequalities:

$$S_b(f) \leq BS_b(f) \leq W_b(f) \leq D(f).$$

We want to upper-bound $D(f)$ by the smaller measures, which we will do with two lemmas.

Lemma 1. *For all functions f and bits b, $W_b(f) \leq BS_b(f) \cdot S_{\overline{b}}(f)$.*

Proof. Consider any x with $f(x) = b$ and $k = BS_b(f;x)$ with corresponding blocks $B_1, .., B_k$. Without loss of generality, we assume the B_i are minimal sets - meaning that flipping values of x in a subset of any of them does not change the value of f.

We first show that for each i, $|B_i| \leq S_{\overline{b}}(f)$. Consider the input $y = x + \chi_{B_i}$. By the properties of B_i, $f(y) = \overline{b}$. Because B_i is minimal, flipping the bits in a subset of B_i still produces output b. Then $S(f;y) \geq |B_i|$ as flipping any of the bits indexed by B_i in y changes the output back to b again.

Second, we claim that $\cup_{i=1}^{k} B_i$ induces a certificate for $f(x)$. Suppose not. Then there are a set of positions outside of $\cup_{i=1}^{k} B_i$ that can be flipped to flip the output of f, and we would have that $BS_b(f) \geq k+1$.

Then for x, we have a certificate of size $\sum_{i=1}^{k} |B_i| \leq BS(f;x) \cdot S_{\overline{b}}(f) \leq BS_b(f) \cdot S_{\overline{b}}(f)$. □

The second lemma upper bounds $D(f)$ as a function of the other measures.

Lemma 2. *For any f and b, $D(f) \leq W_b(f) \cdot BS_{\overline{b}}(f)$.*

Proof. We wish to construct a decision tree for f of depth at most $W_b(f) \cdot BS_{\overline{b}}(f)$. We use the following algorithm to determine $f(x)$:

(1) $I \leftarrow \emptyset$, I is set of indices we have queried in x.
(2) **while** $x|_I$ does not force f
(3) Pick a b-certificate consistent with $x|_I$.
(4) Add these indices to the set I of queried bit positions.
(5) Output $f(x)$.

To show that $|I| \leq W_b(f) \cdot BS_{\overline{b}}(f)$, we note that each iteration of the loop queries at most $W_b(f)$ bit positions, and then show that the number of iterations is at most $BS_{\overline{b}}(f)$. We omit this proof. □

By combining the two above lemmas with the fact that $S_b(f) \leq BS_b(f)$, we conclude that

$$D(f) \leq (BS(f))^3.$$

3.2 Second Step - Multi-variate Polynomials

The second main ingredient in the proof of the main theorem is to relate the quantum query complexity of a function f to the degree of a polynomial approximating f.

Definition 4. $\widetilde{Deg}(f)$ = *the minimum degree of a multi-linear n-variate polynomial A over $\mathbb{R}$ s.t.*

$$(\forall x \in \{0,1\}^n) A(x) \in [0,1], \ \ \textit{and} \ |A(x) - f(x)| \leq \frac{1}{3}.$$

Lemma 3. *For any function f, $\widetilde{Deg}(f) \leq 2Q(f)$.*

Proof. Consider a quantum algorithm computing a function f with bounded error which applies U_x $Q(f)$ times. We first show that the amplitudes after k applications of U_x can be written as multi-variate polynomials in $x_1, ..., x_n$ of total degree $\leq k$. The proof is by induction. Let $\alpha'_{|i,b,z\rangle}$ be the amplitude after applying U_x and $\alpha_{|i,b,z\rangle}$ be the amplitude before. By the definition of U_x, we have that

$$\alpha'_{|i,b,z\rangle} = x_i \alpha_{|i,\bar{b},z\rangle} + (1 - x_i)\alpha_{|i,b,z\rangle}.$$

We claim that if we are looking at the k^{th} application of U_x, then each of $\alpha_{|i,\bar{b},z\rangle}$ and $\alpha_{|i,b,z,\rangle}$ are of degree $\leq k-1$. By induction, the amplitudes immediately after applying U_x for the $(k-1)^{th}$ time were of total degree $\leq k-1$. In between applications of U_x there may be some unitary operations not involving the input. As these do not do not involve the input, the amplitudes immediately before the k^{th} application of U_x are still of degree at most k. Then $\alpha_{|i,\bar{b},z\rangle}$ and $\alpha_{|i,b,z\rangle}$ each have total degree at most $k-1$ and $\alpha'_{|i,b,z\rangle}$ has total degree at most k.

The output distribution of the quantum algorithm is the sum of squares of certain $\alpha_{|i,b,z\rangle}$'s. The polynomial approximation we give is the sum of squares of these amplitudes. As these amplitudes are expressible as polynomials of degree at most $Q(f)$ and squaring multiplies the degree by 2, we have a polynomial of degree at most $2Q(f)$. It is linear in each variable x_i because we can replace x_i^2 with x_i since we are looking at boolean variables. The polynomial approximates f because we are looking at a bounded-error quantum algorithm. And finally, the polynomial takes values in $[0,1]$ because it computes a probability. ☐

3.3 Wrap-up

The final lemma we need to prove the main theorem is the following.

Lemma 4. *For any function f, $BS(f) \leq 4(\widetilde{Deg}(f))^2$.*

Proof sketch. Let $k = BS(f) = BS(f;x)$ with blocks $B_1, B_2, ..., B_k$, $b = f(x)$, and let $\tilde{f}$ be an approximating polynomial for f. The result is proved by making a number of transformations to $\tilde{f}$ based off of the blocks such that: i) the degree with each transformation does not go up, and ii) the final result is a univariate polynomial whose degree can be lower bounded based off of the number of blocks.

For the first step, consider $g : \{0,1\}^k \to \mathbb{R}$ defined by $g(y) = \tilde{f}(x + \sum_{i=1}^k y_i \cdot \chi_{B_i})$. Notice that $deg(g) \leq deg(\tilde{f})$, and that

$$\begin{aligned} &\text{on weight } 0 \ y\text{'s}, \quad |g(y) - b| < 1/3, \\ &\text{on weight } 1 \ y\text{'s}, \quad |g(y) - \bar{b}| < 1/3. \end{aligned}$$

Intuitively, g should have high degree to be able to accomplish this, but it is difficult to analyze as it is multi-variate. Notice that the above property only relates to the weight of the input - an input has weight 0 or 1 under any permutation of its bits.

The second step is to convert g into a univariate polynomial h such that $h(w)$ captures the behavior of g on inputs of weight w. Formally, we define h to be the *symmetrization* of g, that is

$$h(w) = \text{Average of } g \text{ over } y\text{'s of weight } w.$$

h is defined by the formula $h(w) = \frac{\sum_{\pi \in S_k} g(\pi(w))}{k!}$. From this, it is clear that $deg(h) \leq deg(g)$. Also notice that h has the following properties: 1) $h(0)$ and $h(1)$ differ by at least 1/3, and 2) $h(w) \in [0,1]$ for all $w \in \{0, 1, ..., k\}$. Informally, h must have a large derivative somewhere in the interval $[0,1]$ but at the same time h is small on a number of points in the interval $[0,k]$. Tools from analysis can be used to formalize the intuition that h should have high degree. In particular, it can be shown that $deg(h) \geq \sqrt{\frac{k}{4}}$. Combined with the fact that $deg(h) \leq deg(\tilde{f})$, we get the stated result. $\square$

We get the result of the main theorem by combining the results we have proved:

$$D(f) \leq (BS(f))^3 \leq (4(\widetilde{Deg}(f))^2)^3 \leq 64(Q(f))^6.$$