CS 710: Complexity Theory 3/23/2007

Lecture 19: Worst-Case to Average-Case Reductions

Instructor: Dieter van Melkebeek Scribe: Matt Elder & Brian Rice

Two lectures ago, we constructed a family of "quick" pseudo-random generators, based on the assumption that there exists a language $L \in \mathrm{E}$ with high average-case hardness. In this lecture, we extend these results; we show that the worst-case hardness at length m, $C_L(m)$, can be substituted for the average-case hardness at length m, $H_L(m)$. To do this we will further develop the tools we introduced last time: certain error-correcting codes.

1 Goal

Our goal for this and the next lecture will be the following theorem:

Theorem 1. *There are $c, d > 0$ so that*

$$(\forall L \in E)(\exists L' \in E) H_{L'}(cm) \geq \left(\frac{c_L(m)}{m}\right)^d.$$

This says that, given a language L in E with large circuit complexity, there is another language L' in E with large average-case hardness. Combining this with our earlier result, this will yield efficient time-bounded pseudorandom number generators under the assumption that there is a language in E with large circuit complexity.

The idea of the proof is as follows. From the characteristic function $\chi_L(m)$, we would like to construct the characteristic function $\chi_{L'}(cm)$ by applying an error-correcting code. Then, assuming that we can compute "most" of the values of $\chi_{L'}$, we can decode to find χ_L.

2 Return to Error-Correcting Codes

Not any error-correcting code (ECC) will do for our purposes. Ideally, the ECC we use would have the following properties:

- Encoding in polynomial time.
- Efficient and local decoding.
- $q = 2$ (where q is the number of symbols).
- Handle error rates very close to 1/2.

Recall the two error-correcting codes we considered last time: the Hadamard code and the Reed-Solomon code. Let us consider how their properties stack up against our requirements.

2.1 Hadamard Code

Recall that the Hadamard code, with $q = 2$, takes $a \in \{0,1\}^k$ and outputs the string $(\langle a, x \rangle)_{x \in \{0,1\}^k}$, where $\langle a, x \rangle$ indicates the dot product over $GF(2)$.

This code has the good property that the distance between codewords is relatively high (half the length of the codeword). Furthermore, the Hadamard code does have local decoding: to compute the bit a_i, we need only compare $r(x)$ and $r(x \oplus e_i)$, where $r(x)$ is the bit of the received word at position x, and e_i is the string of 0's except for a 1 at position i. If both of these are correct (transmitted without error) then they if and only if $a_i = 1$. By choosing several such x and taking majority vote, we correctly decode a_i with high probability.

On the other hand, the Hadamard code is far too inefficient for our purposes. The length of the codeword (and hence time to encode) is exponential in the length of the message, not polynomial as required.

2.2 Reed-Solomon Code

Recall that the Reed-Solomon Code transforms the information word $a \in GF(q)^K$ to the codeword $(P(x))_{x \in GF(q)}$, where P is the polynomial of degree $K - 1$ whose coefficients are the "digits" of a (i.e. $a = (a_0, a_1, ..., a_{K-1})$ and $P(x) = a_0 + a_1 x + ...a_{K-1} x^{K-1}$). This code relies on the fact that different polynomials of degree $K - 1$ can have at most $K - 1$ points of intersection, meaning the distance between distinct codewords is at least $N - K$.

With an appropriate choice of N and K, the Reed-Solomon code is very efficient. It is fairly easy to encode (certainly in polynomial time), and the distance between codewords is very large, so the code can handle error rates close to 1/2. However, this code is not satisfactory for our purposes. The important problem is that this code cannot be decoded locally: we must examine essentially the entire received word to retrieve one character of the message.

3 Reed-Müller Code

The Reed-Solomon code fails for our purposes because the decoding procedure is inherently non-local. The Reed-Müller code corrects this problem by using multi-variate rather than univariate polynomials. We will see in a moment how to take advantage of multi-variate polynomials to perform local decoding. First, consider the representation of the message. In the Reed-Solomon code, we represent the message as the coefficients for a polynomial. For the Reed-Müller code, we will think of the message as an m-variate polynomial with individual degrees less than s. We could use a similar encoding as was used for the Reed-Solomon code: interpret the message as the coefficients of the polynomial. However, this encoding inherently leads to non-local decoding - to recover the coefficients we need to recover the entire polynomial, meaning we would need to examine a large fraction of the received word bits.

We use a different encoding that will allow local decoding. The information word a is composed of s^m elements from $GF(q)$. We view each element as the evaluation of some m-variate polynomial P on a particular element of $GF(q)^m$ taken from a sub-cube of size s^m. This is illustrated in Figure 1. Encoding is performed by determining the polynomial P from the points given in the message, and then evaluating P on all possible m-tuples over $GF(q)^m$.

It is immediate that for the Reed-Müller code, $K = s^m$ and $N = q^m$. By the Schwartz-Zippel Lemma, the code achieves relative distance $d = 1 - \frac{ms}{q}$.

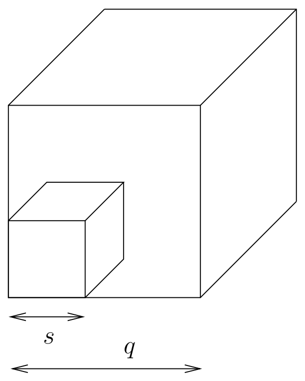

Figure 1: An illustration of the scheme used for the Reed-Müller code for the case of $m = 3$. The message gives the evaluation of a polynomial on a sub-cube of size s^m, and the encoding is the evaluation of an interpolating polynomial on the entire space.

3.1 Encoding

We have specified the correct encoding of the Reed-Müller code above, but it is not immediate that this can be done efficiently: we need to determine the unique polynomial P with each individual degree less than s that interpolates a given set of values on a sub-cube of size s^m. Let $a_i \in GF(q)$ denote the i^{th} value of a. Let ϕ be some invertible map between the integers 1 through s^m and a subcube of $GF(q)^m$ of size s^m. We define want to determine P so that $P(\phi(i)) = a_i$.

We show that we can construct P by building polynomials that interpolate each element of the sub-cube. The polynomial δ_c is defined for an m-variable constant vector c, which is inside the s^m-subcube. We want $\delta_c(c) = 1$ and $\delta_c(x) = 0$ for all other values x inside the s^m-subcube. We can construct δ_c as follows:

$$\delta_c(x) = \alpha \prod_{i=1}^{m} \prod_{\substack{j \neq c_i \\ 0 \leq j < s}} (x_i - j).$$

Here, α is just a normalization constant; we use this so that $\delta_c(c) = 1$. For all values of $x \neq c$ in the s^m-subcube, there is some index i such that $x_i \neq c_i$. By the construction, there must then be a multiplicand in $\delta_c(x)$ of the form $x_i - x_i$, so $\delta_c(x) = 0$. Note that the degree of δ_c in each variable is less than s.

So, we can compose the desired polynomial P as a linear combination of the polynomials δ_c:

$$P(x) = \sum_{i=1}^{s^m} a_i \delta_{\phi(i)}(x).$$

As the degree of each δ_c in each variable is less than s, the same is true of P. We conclude that this formula determines the unique such polynomial interpolating the given points.

Once we have constructed P from the information word a, we produce the codeword b by concatenating the value of $P(x)$ for all values of x in $GF(q)^m$. As an aside, we point out that the the codeword contains an exact copy of the message a (the portion of the codeword where we evaluate P on the elements of the sub-cube that were used to construct P). Codes with this property are called *systematic codes.*

3.2 Decoding

To decode, we could query the received word at every position and construct a polynomial that differs from the received word in the fewest number of positions. We do not do this as our goal is to perform as few queries of the received word as possible. Recall that each element of the message corresponds to the evaluation of P on some point in the s^m-subcube. The basic idea is to look at P restricted to a random line through the point corresponding the position in the message we want to decode. P restricted to this line is a univariate polynomial, so if the received word agrees with P on a large fraction of the points on the line, we can recover the values of P on the line (including the point we are interested in). The construction is illustrated in Figure 2.

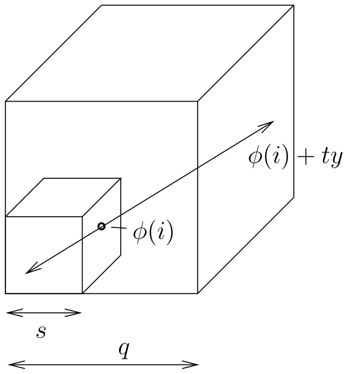

Figure 2: Reed-Müller decoding. To determine $a_i = P(\phi(i))$, construct a line $\phi(i) + ty$ through $\phi(i)$ for a randomly chosen y, then determine the univariate polynomial $P(\phi(i) + ty)$. For $t = 0$, this gives $P(\phi(i))$.

We now formalize the decoding procedure. Suppose r is our received word. Let $r(x)$ denote the value of r in the position corresponding to $x \in GF(q)^m$; and let $P(x)$ denote the analogous position in the correct codeword. To retrieve the i^{th} digit of the message a, we would like to determine the value of $P(\phi(i)) = a_i$. To determine $P(\phi(i))$, we pick a random point $y \in GF(q)^m$, select ms distinct values for $t \neq 0$, and examine $r(\phi(i) + ty)$ at each one. We know that $P(\phi(i) + ty)$ is a polynomial in t of degree at most ms. Suppose $P = r$ for the values we have chosen. In this case, we can determine the polynomial $P(\phi(i) + ty)$, and evaluate it at $t = 0$ to get $P(\phi(i)) = a_i$.

So if $P = r$ on the chosen points, we recover the correct value. We now bound the probability that $P \neq r$ for at least one of the chosen points. Suppose $r(x)$ differs from $P(x)$ in at most γ fraction of locations. As each point we query r on is uniformly distributed, the probability that $r(x) \neq P(x)$ for each query x is at most γ, by our above assumption. So, the probability that $r(\phi(i) + ty) \neq P(\phi(i) + ty)$ for some selected value of t among the $m(s-1)$ values that we selected is at most $(ms)\gamma$. If we assume that $\gamma \leq \frac{1}{3ms}$, the probability we output an incorrect value for a_i is at most $1/3$. Furthermore, we can repeat the above process for different random values of y, and then take the majority vote for the value of a_i. In this way, we can make this decoding algorithm work with probability $1 - \epsilon$ for ϵ as small as we like.

The above analysis only works provided the error rate in transmission is no greater than $1/(3ms)$. Essentially the same techniques can be used with higher error rates, but we stuck to lower error rates in the above to keep the analysis simple.

We also point out that the decoding procedure is randomized, whereas we often want to have a deterministic decoding procedure. Since we will have circuits perform the decoding procedure in our worst-case to average-case reduction, requiring randomness will not be a problem - we will be able to hardwire "good" random bits into the circuits.

3.3 Parameters

We would like to set the parameters of this code in such a way to meet the three conditions we originally set out to meet. We already have that $N = q^m$, $K = s^m$, and $d = 1 - \frac{sm}{q}$. We want to choose s, m, and q, so that: (a) d is close to 1, meaning we can correct even with error rates close to $\frac{1}{2}$, (b) ms is small, so the number of queries in the decoding procedure is small, and (c) $N = K^{O(1)}$, so the encoding is polynomially long (and thus also polynomial-time computable).

We point out that for d to be positive, we need $sm < q$. Then (c) implies that

$$K^{O(1)} \geq N = q^m \geq (ms)^m = K \cdot m^m,$$

so $m^m \leq K^{O(1)}$. Taking logarithms, we have that $m \log m \leq O(\log K)$, and therefore $m \leq O(\frac{\log K}{\log \log K})$.

(b) combined with the fact that $K = s^m$ means that making m as small as possible will minimize sm, so we set $m = \Theta(\frac{\log K}{\log \log K})$. As $K = s^m$, this means that $s = \Theta(\log K)$.

We have set the parameters so that we get a code with polynomial stretch and requiring only a poly-logarithmic number of queries to locally decode. This is almost good enough for what we want to do. There are two issues that still need to be dealt with: the code is not binary, and (we will see) the distance is not good enough.

4 Concatenation of Reed-Müller and Hadamard

To deal with the problem of the Reed-Müller code being a non-binary code, we concatenate it with the Hadamard code. If we start with an $[N, K, d]$ code over an alphabet of q elements, concatenation with the Hadamard code yields a binary $[Nq, K \log q, \frac{d}{2}]$ code. This concatenated code has all of the properties that we want except that the distance is not good enough. Recall that we wanted to be able to handle error rates close to $\frac{1}{2}$. To do this with unique decoding, we need the distance $\frac{d}{2}$ to be close to 1. This is not possible for any code, since $d \leq 1$.

In the next lecture, we will see how to get around this problem by relaxing the requirements of uniquely decoding the correct message. The relaxed notion is called *list-decoding*, where we require the decoding procedure to output all messages whose encoding is close to the received word. In the next lecture we will see a list-decoding procedure for the Hadamard code, and give an idea of a list-decoding procedure for the concatenated Reed-Müller/Hadamard code that satisfies all of our original goals.