CS 710: Complexity Theory 11/17/2011

Lecture 21: Worst-case to Average-case reductions

Instructor: Dieter van Melkebeek Scribe: Gautam Prakriya

In the last lecture we began working towards the following goal: Given a language $L' \in$ E we would like to construct a language $L \in$ E s.t. the average case hardness of L is close to the worst-case hardness of L', $H_L \simeq C'_L$. (We will quantify "close to" later)

This transformation allows us to relax the hardness condition under which we know how to derandomize BPP. Our earlier hypothesis required a language in E with large average-case hardness ($H_L(m) \geq 2^{\Omega(m)} \forall m$). With the above transformation we can replace H_L by C_L in the hypothesis.

Now, let $\chi_{L'|m'}$ be the characteristic sequence of L' on inputs of length m'. (This is a binary string of length $2^{m'}$. The ith bit of the string specifies the membership of the ith string of length m').

Our transformation works as follows: We use an error correcting code to obtain a string of length more than $2^{m'}$ from $\chi_{L'|m'}$. We treat the resultant string as the characteristic sequence of the language L on inputs of length $m(> m')$. The intution being, if we have a small circuit that approximates L on inputs of length m fairly well (i.e. can compute L on noticably more than half the inputs), then we can view the characteristic sequence of this circuit as the received word and retrieve the information word($\chi_{L'|m'}$) from the characteristic sequence of the the circuit. Ofcourse, we need to perform the decoding with a small circuit as well.

In this Lecture we continue working towards the reduction from average-case hardness to worst-case hardness . We introduce another error correcting code, the Reed-Müller code. Later in the lecture we introduce the notion of List-Decoding.

1 Error correcting code (Contd.)

In the last lecture we looked at Hadamard and Reed-Solomon codes. We saw that the Hadamard code is a $[2^K, K]_2$ code. The minimum distance of this code is 1/2, which in the case of binary codes is the best distance one can hope for. Although it has a very good minimum distance, the Hadamard code has a very poor rate. This code can handle an error fraction of upto 1/4, which is not good enough for the result we are after. We will show later how List-Decoding will allow us to handle an error fraction of upto 1/2.

The Reed-Solomon code is obtained by treating the information word as the coefficients of some polynomial over $GF(q)$. It has a minimum distance that is close to 1.

$$\delta \geq 1 - \frac{K}{q}$$

This implies that the code can handle an error fraction that is close to 1/2. However, it is not locally decodable — To recover a single bit of the information word, we need to look at atleast K bits of the received word. (K is the size of the information word.). So the decoding cannot be done with a small circuit.

We will now describe the Reed-Müller code which is more suited for our purposes than the codes we have seen so far. As we will soon see it has a good rate and is locally decodable. The only issue is that it is not binary. We resolve this by concatenating it with the Hadamard code.

1.1 Reed-Müller codes

Instead of using univariate polynomials as we did with Reed-Solomon codes, we use m-variate polynomials over a finite field to construct Reed-Müller codes. As we will soon see this will allow local decodability.

Another point of departure from the Reed-Solomon codes is that we view the information word as the value of the m-variate polynomial at K distinct points in $GF(q)^m$.

1.1.1 Encoding

Consider the m-dimensional vector space $V(= GF(q)^m)$ over $GF(q)$. Let S be an $\underbrace{s \times s \times \cdots \times s}_{m}$ sub-cube of V. Let $K = s^m$, we view the information word as a sequence elements that specify the value of some function defined on S.

Our encoding will be a polynomial that interpolates this function. The length of the codeword will be q^m - the value of the polynomial at each point in V.

It is enough to obtain polynomials that interpolate the following functions.

$$\delta_{z^*}(z) = \begin{cases} 1 & \text{if} \quad z = z^* \\ 0 & \text{if} \quad z \neq z^*, z \in S \end{cases}$$

This is because any function can be written as a linear combination of the δ_{z^*}'s . It is easy to see that the following polynomial interpolates δ_{z^*}.

$$\prod_{i=1}^{m} \prod_{\substack{1 \leq j \leq s \\ j \neq z_i^*}} \left(\frac{z_i - j}{z_i^* - j} \right)$$

Any function over S can be interpolated by a linear combination of polynomials of this form. Therefore the degree of the interpolating polynomial is atmost $s \times (m-1)$. The codeword is a string of length q^m in which the i^{th} element corresponds to the value of the interpolating polynomial at the i^{th} point in V.

Clearly, $N = q^m$, by Schwartz-Zippel lemma $\delta \geq 1 - \frac{ms}{q}$. We note that the codeword contains an exact copy of the information word in it (The bits corresponding to the points in S.) Codes with this property are called systematic codes.

1.1.2 Decoding

As already mentioned multivariate polynomials afford us local decodability. We would like to be able to decode the received word when upto than $1/2 - \epsilon$ bits have been corrupted. However, in this lecture we only handle cases where a relatively small fraction - $1/3ms$ of bits have been corrupted. A decoding algorithm for Reed-Müller codes which corrects higher errors is presented in page 14 of [1].

Suppose we want to retrieve the k^{th} element of our information word from the received word r (say). Let z^* be the point in S that corresponds to this element, we want to find the value of the multivariate polynomial (defined by the codeword) at z^*.

We do so by picking a random line through z^* and restrict our multivariate polynomial (which we will henceforth refer to as P) to this line. P restricted to this line is a univariate polynomial of

degree atmost ms, so if the received word agrees with P on a large fraction of points on the line, we can recover P on the line.

Now, the fraction of corrupted elements is $\leq 1/3ms$. We recover $P(z^*)$ as follows. Pick a random direction u. Evaluate $r(z^* + tu)$ for $t = 1 \ldots ms$ (clearly, $ms < q$). If $(\forall 1 \leq t \leq ms) r(z^* + tu) = P(z^* + tu)$ then we recover the polynomial $P(z^* + tu)$ exactly, return value at $t = 0$.

$$Pr[(\exists t : 1 \leq t \leq ms)\ r(z^* + tu) \neq P(z^* + tu)] \leq \frac{1}{3ms} ms = \frac{1}{3} (\text{By a Union Bound})$$

Therefore, with probability $2/3$ we obtain the correct value of the k^{th} element in the information word. We can boost this probability by repeating and taking the majority vote.

1.2 Parameters

We already have that $N = q^m$, $K = s^m$, and $d = 1 - \frac{sm}{q}$, $\sharp$queries $= sm$. We want to choose s, m, and q, so that: i) d is close to 1, meaning we can correct even with error rates close to $\frac{1}{2}$, ii) ms is small, so the number of queries in the decoding procedure is small, and iii) $N = K^{O(1)}$, so the encoding is polynomially long (and thus also polynomial-time computable).

For δ to be positive, we need $sm < q$. Then iii) 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 $m \leq O(\frac{\log K}{\log \log K})$.

ii) combined with the constraint that $K = s^m$ means that making m as large 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 (since we are dealing with membership bits we need a binary code). We get around this issue by concatenating the Reed-Müller code with the Hadamard code.

1.3 Concatenation of Reed-Müller and Hadamard

If we start with an $[N, K, d]$ code over an alphabet of q elements, it is not difficult to see that concatenation with the Hadamard code yields a binary $[Nq, K \log q, \frac{d}{2}]$ code. For the information word of the Reed-Müller code, we have K symbols over $GF(q)$. These can be encoded by $K \log q$ binary bits. We apply the hadamard code on each of these blocks (corresponding to elements of $GF(q)$) of size $\log q$, this gives us a string of length q for each symbol. The concatenation gives us a encoding of size Nq. Finally, positions where the codeword and received word (In the Reed-Müller setting) agree correspond to blocks which agree (in the concatenation). If positions disagree, then the corresponding blocks disagree on half the bits. Therefore the relative distance is halved.

This concatenated code is binary, locally decodable, and the encoding procedure runs in polynomial time. (N and K are exponential in q so encoding results in only a polynomial stretch.) For our strategy to work we need to handle errors close to $1/2$. To be able to handle a error fraction of η, we need the minimum distance to be 2η. The issue is the best we can hope for with binary codes is a minimum distance of $1/2$. (One can show that when the minimum distance is greater than $1/2$, there can only be constantly many codewords.)

To get around this we introduce the notion of List decodability, where we demand that there are a (relatively small) number of codewords close to each received word. Given a received word, we produce a small list of all information words whose encodings are close to the received word and we want to be able to recover this list efficiently. With a relatively small number of extra bits, we can also specify which element of the list we are interested in. Since we are dealing with circuits we can use non-uniformity and hardwire this information into the circuit.

Naturally, as the fraction η gets closer to half, the number of codewords within distance η to a given received word grows, but we will be able to show that this number does not grow too large. We will describe a list decoding algorithm for the Hadamard code in this lecture.

2 List decodability of Hadamard codes

Given a received word $r \in \{0,1\}^{2^k}, \epsilon > 0$ we want to find a list that with high probability includes all information words $x \in \{0,1\}^K$ s.t. $\Delta(E(x), r) \leq \frac{1}{2} - \epsilon$. Where $E(x)$ represents the encoding of x and Δ the relative Hamming distance. We require $1/2 - \epsilon$ since, if we go up to $1/2$ and the received word is actually a valid codeword then our list would include all information words (Two distinct code words have relative distance $1/2$.) which we cannot allow since we would like to produce the list efficiently.

We will present a randomized procedure that outputs the list in time polynomial in $/\epsilon$. This also implies that the list contains atmost polynomial in k/ϵ words. Recall that to find the i$^t h$ bit of the indormation word x, the local decoding procedure for Hadamard codes randomly picked a point a and queried two points in the received word $r(e_i + a)$ and $r(a)$ to obtain x_i . Since the fraction of errors is less than $1/4 - \epsilon$, this procedure give us the correct value with probability $1/2 - 2\epsilon$. However in the current situation, the fraction of errors is up to $1/2 - \epsilon$. In this case the standard decoding procedure gives the correct value with probability $\geq 2\epsilon$ which is of no use to us.

However, we still do something similar - The idea is to query the first point while assuming that we have the correct value for the second point.

We focus on obtaining particular bit of the information word x - the i^{th} bit x_i (say). Pick t points uniformly at random $a_1, a_2 \ldots, a_t \in \{0,1\}^K$ (t will be determined later). Let x be the information word. Recall that the encoding is obtained by taking inner product with all possible vectors $a \in \{0,1\}^K$.

Consider $y_c = \sum_{i=1}^{t} c_i a_i$ for every $c \in \{0,1\}^t, c \neq 0$. For every c, y_c is uniformly distributed since the a_i's are picked uniformly at random. Moreover different values of c give pairwise independent random variables.

Now, suppose $y_c + e_i$ (once a_i's are fixed) is a position where r is not corrupted. Then $(x, y_c) + r(y_c + e_i) = x_i$. Since atmost a $1/2 - \epsilon$ fraction is corrupted,

$$\Pr_{a_1, a_2 \ldots, a_t \sim \{0,1\}^K}[(x, y_c) + r(y_c + e_i) = x_i] \geq \frac{1}{2} + \epsilon$$

Now, we would like to boost our confidence from $1/2 + \epsilon$ by making not 1 but $2^t - 1$ queries. (one for each value of c) and taking the majority vote. We will choose a t s.t this can be done. We will see that pairwise independence is enough to boost probability using the Majority vote.

Let I_c be the indicator random variable for $(x, y_c) + r(y_c + e_i) = x_i$.

$$\Pr[\text{MAJ is incorrect}] \leq \Pr[\sum I_c \leq \frac{1}{2}(2^t - 1)]$$

Since, atmost a $(\frac{1}{2} - \epsilon)$ fraction is corrupted, the expected value of $I = \sum I_c$ is

$$E(I) = (\frac{1}{2} - \epsilon)(2^t - 1)$$

. Therefore,

$$\Pr[\text{MAJ is incorrect}] = \Pr[I \leq E(I) - \epsilon(2^t - 1)]$$

by Chebyshev's inequality,

$$\Pr[\text{MAJ is incorrect}] \leq \frac{\sigma^2(I)}{(\epsilon(2^t - 1))^2} \quad = \frac{(2^t - 1)}{4(\epsilon(2^t - 1))^2} = \frac{1}{4\epsilon^2(2^t - 1)}$$

We used the fact that the variance of an indicator random variable is atmost $1/4$. Since the indicator random variables are pair-wise independent, their variances are additive.

We obtain the following by a union bound

$$\Pr[\exists j \text{ We output } x_j \text{ incorrectly}] \leq \frac{K}{4\epsilon^2(2^t - 1)}$$

We need $\Pr[\exists j$ We output x_j incorrectly$] \leq \frac{1}{3}$. $t = \mathrm{O}\ (\log \frac{K}{\epsilon})$ works.

Thus far, we have shown how one can recover each bit of the information word provided we know the value of (x, y_c) for all c. This seems an unreasonable assumption since x is what we want to find. This is where list decoding comes in. The algorithm needs a sequence of values $((x, y_c))_{c \neq 0^t}$. We run the algorithm on all possible values of this sequence and output the list of information words obtained. Then with probability $\geq 2/3$, x is present in this list. We obtain x on atleast a $2/3$ fraction of the runs on the correct sequence.

However, one a potential problem is that there are $2^{2^t - 1}$ possible sequences. But since the inner product is linear,

$$(x, y_c) = (x, \sum_{j=1}^{t} c_j a_j) = \sum_{j=1}^{t} c_j (x, a_j).$$

Which means we only need to run through all possible sequences of values of $((x, a_j))_{j \in \{1 \ldots t\}}$. Which is just $2^t = \mathrm{poly}(\frac{K}{\epsilon})$.

Next Time

In the next lecture we will wrap up our discussion of worst-case to average-case reductions, Introduce Extractors and look at some of their applications.

Acknowledgements

In writing the notes for this lecture, I perused the notes by Tom Watson and Matt Elder for lecture 16,17 from the Spring 2007 offering of CS 810, and the notes by Brian Rice and Theodora Hinkle for lectures 19,20 from the Spring 2010 offering of CS 710.

References

[1] Madhu Sudan, Luca Trevisian, and Salil Vadhan. *Pseudorandom Generators without the XOR Lemma*. J. of Computer and System Sciences, 62(2):236-266, 2001. Preliminary version: Proc. of 31st ACM STOC, 1999. Accessed at http://www.cs.berkeley.edu/