CS 710: Complexity Theory 3/18/2010

Lecture 18: Error-Correcting Codes

Instructor: Dieter van Melkebeek Scribe: Dmitri Svetlov and Matt Elder

In the last lecture, we showed how to construct a pseudorandom generator (PRG) if a class of "average-hard" languages exist, i.e. if there is some $L \in \mathrm{E}$ with average-case hardness. Throughout this lecture and the next, we extend our earlier results by relaxing these average-case hardness hypotheses and replacing them with worst-case hypotheses, then developing the notion of *error-correcting codes* to prove that at length m, the worst-case hardness $C_L(m)$ can replace the average-case hardness $H_L(m)$ in such a way that we can construct a language $L\prime$ with average-case hardness very close to the worst-case hardness of any given language L.

1 Motivations

1.1 Circuit Lower Bounds Yield Pseudorandom Generators

Recall from last time the construction of one particular PRG. Specifically, say we are given a function $f : \{0,1\}^m \to \{0,1\}$ and an $(m, \log r)$ design $S_1, \ldots, S_r$ over $[\ell]$. Then we construct the function $G_r : \{0,1\}^\ell \to \{0,1\}^r$ with $G_r(\sigma) = (f(\sigma|S_1), \ldots, f(\sigma|S_r))$. Each of these designs is of size m and the intersection of any two distinct sets is at most r. From this follows the security condition that for a hardness $H_{L(f)}(m) \geq r^2$, G_r is a $(1/r)$-PRG for circuits of size at most r. We argued this by assuming the contrapositive, constructing an appropriate predictor circuit and finding the hardness. Another important parameter of these G functions is their complexity, which depends on f (specifically, the complexity of computing each ith element. We would like the complexity to be linear-exponential in ℓ, in which case (if $L(f) \in \mathrm{E}$) we say that G_r is "quick" for the purpose of derandomization (although for cryptographic purposes this may not be enough). Some instantiations that we have already seen, *e.g.* polynomial graphs such that $\ell = O(m^2)$, are able to achieve this quickness.

To illustrate some of the power of these PRGs, suppose we have a BPP algorithm A for some problem and a hard language that allows the construction of such a PRG. Then we can solve the same problem using the following deterministic algorithm $A\prime$.

1. Construct the PRG G_r taking a random seed of length ℓ and outputting a pseudorandom string of length r.
2. For every seed $\theta \in \{0,1\}^\ell$, run A, replacing its random bits by $G_r(\theta)$.
3. Output the majority vote of A on the set of all seeds.

Note that the utility of $A\prime$ as an algorithm for the problem that A solves depends on the complexity of A being small enough that G_r is known to fool it. Thus we can obtain the following set relations involving BPP by imposing hardness conditions of varying strengths.

1. If there exists a language $L \in \mathrm{E}$ such that $H_L(m) \geq m^{\omega(1)}$, then we can build a G_r with $\ell(r) = r^O(1)$, so $BPP \subseteq \mathrm{SUBEXP} = \bigcap_{c>0} \mathrm{DTIME}(2^{n^c})$. [This is one of the weakest hardness

conditions - recall the relationship between the second level of the polytime hierarchy and size-n^k circuits.]

2. If there exists a language $L \in \mathrm{E}$ such that $H_L(m) \geq 2^{m^{\Omega(1)}}$, then we can build a G_r with $\ell(r) = (\log r)^O(1)$, *i.e.* polylog input bits, so $BPP \subseteq \mathrm{QP} = \bigcap_{c>0} \mathrm{DTIME}(2^{\log cn})$.

3. If there exists a language $L \in \mathrm{E}$ such that $H_L(m) \geq 2^{\Omega(m)}$, then we can build a G_r with $\ell(r) = O(\log r)$, so $BPP \subseteq \mathrm{P}$. [This is the strongest hardness condition known - that of a linear-exponential dependence on m. It requires also a $(c \log r, \log r)$ design with $\ell = O(c^2 \log r)$ to show that a G_r taking only $O(\log r)$ input bits is constructible. We assert without proof that such a design does exist.]

As an aside, this strongest hardness condition implies the following corollary showing the inclusion of BPP in ZPP.

Corollary 1. $\mathrm{BPP} \subseteq \mathrm{ZPP}^{\mathrm{NP}} \subseteq \Sigma_2^P \cap \Pi_2^P$.

Proof. Recall that the definition of BPP as the class of languages with poly-time, bounded-error randomized algorithms. From the third implication above, such algorithms can be derandomized by functions $f : \{0,1\}^m \to \{0,1\}$ in E with hardness $H(f) \geq 2^{\Omega(m)}$.

So for a given function f, consider its characteristic string χ_f, the length of which is 2^m. By a counting argument on all possible characteristic strings, we know that most functions on m bits will have the required hardness. Therefore, given a BPP algorithm A, consider the following $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm.

1. Pick a random function $f : \{0,1\}^{O(\log r)} \to \{0,1\}$. [Essentially, guess the characteristic sequence of this presumably hard function. Since it is $\log r$ in size, this can be done in $O(r^2)$ time.]

2. Query the NP oracle whether a small (size less than r) circuit computes f. [Again, this is an NP predicate since f is $\log r$ in size. Think of this alternatively as changing the hardness question to an NP/coNP question: a witness to nonhardness is a short circuit.] If there is a short circuit for f, halt and return "?."

3. Otherwise, construct a PRG G built from the hard function f. Then run the derandomized version of A using G and output its result.

Since, as we've noted, most candidates for f are hard, this algorithm halts and fails with low probability but gives the correct answer in all other cases, so it is a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm. Therefore, every language solvable by an algorithm in BPP can also be solved by one in $\mathrm{ZPP}^{\mathrm{NP}}$. Thus, $\mathrm{BPP} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. (Note that this is yet another proof of the membership of BPP in the second level of the poly-time hierarchy, for those still unconvinced.) □

These results apply to a wide variety of computational models, from typical models –like the time- and space-bounded settings –to extreme models such as constant-depth circuits. [Relate the first two hardness conditions explored above to these situations.]

We now turn to the converse of these entailments, and show that not only do hard languages imply the existence of PRGs, but also that PRGs imply the existence of hard languages.

1.2 Pseudorandom Generators Yield Circuit Lower Bounds

Theorem 1. *If there exists an ϵ-PRG G computable in* E *that fools circuits of size at most r, then there exists a language $A \in$* E *with circuit complexity greater than r.*

Proof. The idea behind this proof is that we can use a given PRG G to construct a *discerning* language for G, *i.e.*, one that differentiates between the uniform distribution and the distribution generated by G. Such a language must have a high circuit complexity since G is a PRG.

We note first that if the existence of G is to be non-trivial, then the output of G must be longer than its input, and ϵ must be less than 1. Thus by looking at only the first $\ell + 1$ output bits of G, we can assume that G takes ℓ to $\ell + 1$ bits. Let A be the language $\{G(\alpha) \mid \alpha \in \{0,1\}^\ell\}$.

Now suppose the circuit C decides A. Then we find that

$$\Pr_{\sigma \in \{0,1\}^\ell}[C(G(\sigma)) = 1] = 1, \text{and}$$

$$\Pr_{\sigma \in \{0,1\}^{\ell+1}}[C(\sigma) = 1] \leq \frac{1}{2}.$$

The first result holds by the definition of C; the second holds since G maps its 2^ℓ inputs to no more than 2^ℓ distinct outputs. These results taken together show that C differentiates between the uniform distribution and the distribution generated by G. But since G fools circuits up to size r, any circuit discerning G must have size greater than r. Since we have chosen an arbitrary C deciding A, this must be true for all circuits deciding A, so A has circuit complexity greater than r.

It now remains to show that $A \in$ E. So consider the naive algorithm for testing membership of $x \in A$: compute $G(\theta)$ for all seeds θ and compare the outputs to x to see if it is one of them. Since we assume that $G \in$ E and it cycles over 2^ℓ seeds, this computation can be done in time $2^O(\ell)$; thus, $A \in$ E. $\square$

We have now shown a stronger, bidirectional relationship: the existence of quick PRGs fooling circuits of size r is equivalent to the existence of functions in E with circuit complexity greater than r.

2 Error-Correcting Codes

An *error-correcting code* (ECC) can be thought of as a function from a set of *information words* in some alphabet to a set of *codewords* in that alphabet such that the codeword, even if some not-too-large fraction of its bits are flipped, can still be decoded and the information word retrieved. In other words, ECCs encode information in such a way that a certain amount of error (in data transmission, say) can be corrected.

We will use such ECCs to express the desired relation discussed above –that between a language L' with high average-case circuit complexity to a language L with high worst-case circuit complexity. Let $\chi_{L(m)}$ be the characteristic sequence of L on inputs of length m, *i.e.*, the ith bit of $\chi_{L(m)}$ is 1 exactly if the ith m-bit word is in L. Then, for some m', let L' be the language whose characteristic sequence $\chi_{L'}(m')$ on inputs of length m' is the codeword corresponding to the information word $\chi_L(m)$ according to some ECC.

Our desire is to show that if L is worst-case hard for circuits of size s, then L' is average-case hard for circuits of slightly smaller size s' –alternatively, that $H_{L'}(m') \simeq C_L(m)$, where $m' = \theta(m)$ (in which case $2^{m'} = O(2^{m^c})$. To prove this, think contrapositively: assume to the contrary that there is a small circuit that can approximate $\chi_{L'}$ in many cases (specifically, more than half of them). Then, given a decoding algorithm for the ECC, a small circuit for $\chi_L(m)$.

Naturally, not all possible functions from information words to codewords will be able to accomplish our precise objective; in particular, any suitable ECC will need to have three specific properties. We will list them first, to guide our exploration of possible candidates, and show in a later lecture why they are in fact necessary. Any such ECC must exhibit

1. Robustness: It must be able to handle very high error rates (*i.e.*, a flipping of up to almost half of the bits). Alternatively, it must correct error rates as high as $1/2 - \epsilon$ for small ϵ.

2. Local Decoding: To obtain any single bit of the information word, only a small portion of the codeword must be examined. Furthermore, this examination must not take too long.

3. Polytime Encoding: This ensures that if $L \in \mathrm{E}$, then $L' \in \mathrm{E}$ and $m' = O(m)$.

We now formally define error-correcting codes and explore some candidates in light of these prerequisites.

Definition 1. *An (N, K, D)-*error-correcting code *(ECC) is a mapping $E : \Sigma^K \to \Sigma^N$ such that for all distinct $x_1, x_2 \in \Sigma^K$, $E(()x_1)$ and $E(()x_2)$ differ in at least D positions (in other words, the Hamming distance between them is greater than D.)*

Let us now express the use of ECCs in these terms. Given an information word $x \in \Sigma^K$, we will pass it as input to the mapping function and obtain the codeword $y = E(x)$. Then suppose that uncontrollable events perturb y and cause some bits to flip at random, erasing our memory of y and producing a received word z that is distinct from y. If fewer than D bit flips occurred between y and z, then by definition z cannot be a legal codeword, and we can detect that errors were introduced. Then, if fewer than $\lfloor (D-1)/2 \rfloor$ flips have occurred, z is closer in Hamming distance to y than to any other legal codeword, so z can be corrected back to y. [Note that throughout this discussion, the introducible errors are all bit flips. Naturally, this restricts Σ to $\{0, 1\}$. We will discuss later whether other alphabets are also possible and what implications the choice of alphabet has on the introduction and correction of errors.]

The parameters of these ECCs that will determine their usefulness for our purpose –and therefore the ones we will seek to optimize –are

1. the *ratio* N/K, which quantifies the redundancy introduced by the encoding;

2. the *relative distance* D/N between valid codewords; and

3. the complexity of encoding and decoding.

Naturally, we wish to keep the ratio low, so as not to introduce too much redundancy, and to minimize the complexity of encoding and decoding while achieving a large relative distance, allowing us to correct many errors. However, the first two parameters are in a sense opposed to one another: decreasing the redundancy in turns shrinks the relative distance and lessens the number of errors that can be corrected; on the other hand, increasing the relative distance will eventually require greater redundancy and may also increase the complexity of the encoding and decoding algorithms beyond reasonable limits.

In our treatment of error-correcting codes, we will focus on the particular class of *linear ECCs*, which are mappings from $[q]^K$ to $[q]^N$, where $[q]$ denotes the set $\{0, 1, ..., q\}$ for some prime power q. We denote such codes with square brackets, using the notation $[N, K, D]_q$.

For these codes, the range of possible codewords is a linear subspace of $[q]^N$ and the codeword bits can be thought of as linear functions of the information bits. The minimum distance between two codewords is the minimum Hamming weights of the code, analogously to the general case. As an aside, linear ECCs also have some additional useful properties, such as generator matrices for them, although we will not discuss them further here.

Let us now consider two simple examples of linear ECCs and investigate whether their parameters suffice for our goal.

2.1 The Hadamard Code

In the Hadamard code, a linear ECC with $q = 2$, a binary information word a of length K is mapped to the codeword $(< a, x >)_{x\in\{0,1\}^K}$ –in other words, the concatenation of the inner (or, dot) products of a and x across all binary words x of length K. It is easy to calculate that for this code, $N = 2^K$, and (by a counting argument) $D = 2^{K-1}$. Therefore, the relative distance equals $D/N = 1/2$ (which is rather good), but the redundancy ratio equals $\frac{2^K}{K}$, which is incredibly large.

Now consider a possible decoding process for received words after encoding and perturbation. Suppose that the total error rate of transmission is no more than 1/4. Let $r(x)$ be the portion of the received word r at position x. Then we have $\Pr[r(x) = a \cdot x] \geq 3/4 + \epsilon$. To find the information bit a_i, we can look at two values of x that differ at bit i. Pick x at random, and consider both this bit and the bit $x \oplus e_i$. Since both x and $x \oplus e_i$ are chosen uniformly at random, we find that

$$\begin{aligned}\Pr[r(x) \oplus r(x \oplus e_i) = a_i e_i] &\geq \Pr[r(x) \text{ is correct}] \Pr[r(x \oplus e_i) \text{ is correct}] \\ &\geq (3/4 + \epsilon)^2 \\ &\geq 1/2 + 2e.\end{aligned}$$

Therefore, with a probability greater than 1/2, we can decode each information bit by examining just two others. Further, if the error rate of transmission is slightly below 1/4, then all errors can be corrected, and since no ECC has a relative distance greater than 1/2, this error-correction rate is arbitrarily close to the best rate theoretically possible.

But nevertheless, the Hadamard code fails to achieve our goal: we can handle error rates only up to 1/4, and while local decoding is clearly possible, the encoding (and therefore the time required to encode) is exponentially long in the length of the input.

2.2 The Reed-Solomon Code

To try to address these deficiencies, we consider another code that also happens to have a geometric inspiration. The Reed-Solomon code maps an information word $a \in [q]^K$ to the codeword $P(x) = (\Sigma_{i=1}^K a_i x^{i-1})_{x\in[q]^K}$. In other words, P is the polynomial of degree $K - 1$ whose coefficients are the "digits" of a; *i.e.*, for $a = a_0 a_1 ... a_K$, $P(x) = a_0 + a_1 x + ... + a_K x^{K-1}$.

The key fact upon which this code relies is that distinct polynomials of degree $K - 1$ can intersect at no more than $K - 1$ distinct points, so the minimum distance between any two distinct codewords is at least $N - K$. We can thus determine that the parameters of the Reed-Solomon code are $N = q$, $K \leq q$, and the relative distance is at least $1 - K/N$.

Note that there is some apparent flexibility in these parameters. Essentially, the Reed-Solomon makes the inherent trade-off between ration and relative distance "adjustable," so by choosing $K = 2\epsilon N$, we can handle error rates up to $1/2 - \epsilon$. Furthermore, since we are dealing with low-degree polynomials (compared to the length of the information word), both encoding and decoding can be done efficiently, *i.e.*, in polynomial time. For these reasons, Reed-Solomon codes are used extensively for many practical applications, such as communication and information retrieval.

However, note that the decoding algorithm is not local –we must still examine essentially the entire codeword to decode any particular information bit. So while we have made progress toward finding a single ECC that serves our purposes, more work remains to pin such a code down.

3 Next Lecture

In the next lecture, we will solve the problem of nonlocal decoding with the Reed-Müller code, which uses multivariant polynomials rather than the univariant polynomials that create Reed-Solomon codes. We will then tie these various results together to obtain a single error-correcting code that meets all our specifications and will allow us to complete our relation of hardness in the worst case to hardness in the average case.