CS225: Pseudorandomness Prof. Salil Vadhan

Lecture 20: Locally Decodable Codes

April 24, 2007

Based on scribe notes by Sasha Schwartz, Adi Akavia, and Kevin Matulef.

1 Recap

Last time we saw that locally decodable codes, if they exist, could be used to convert worst-case hard functions to average-case hard functions. Today, we will see how to obtain such codes.

Recall that we wanted a locally decodable code encoding messages $f : \{0,1\}^\ell \to \{0,1\}$ to codewords $\hat{f} : \{0,1\}^{\hat{\ell}} \to \{0,1\}$:

1. $\hat{\ell} = O(\ell)$, ie the block length ($2^{\hat{\ell}}$) is polynomial in the message length (2^ℓ), and moreover the encoding should be computable in time $2^{O(\ell)}$ (i.e. polynomial in the message length).
2. A local $(1/2 - \varepsilon)$-decoding algorithm that runs in time $\mathrm{poly}(\ell, 1/\varepsilon)$. That is, given oracle access to a received word $r : \{0,1\}^{\hat{\ell}} \to \{0,1\}$ at distance less than $(1/2 - \varepsilon)$ from a codeword $\hat{f}$ and any index $x \in \{0,1\}^\ell$, we should be able to compute $f(x)$ with high probability in time $\mathrm{poly}(\ell, 1/\varepsilon)$.

However, we will begin by working with the following modifications of the second requirement:

1. We will give a local δ-decoding algorithm for a small constant $\delta > 0$, rather than a local $(1/2 - \varepsilon)$-decoding algorithm. The reason is that achieving the latter will require a notion of local *list*-decoding, which we will introduce later.
2. We will give an algorithm for decoding *codeword* symbols rather than message symbols, as formalized by the following definition.

Definition 1 *A* local δ-decoding algorithm for codeword symbols *for a code $\mathcal{C} \subset \Sigma^{\hat{L}}$ is a probabilistic oracle algorithm* Dec *with the following property. Let $\hat{f} \in \mathcal{C}$ be any codeword, and let $g : \{0,1\}^{\hat{\ell}} \to \Sigma$ be such that $\Delta(g, \hat{f}) < \delta$. Then for all $x \in [\hat{L}]$ we have $\Pr[\mathrm{Dec}^g(x) = \hat{f}(x)] \geq 2/3$, where the probability is taken over the coins flips of* Dec.

This implies the standard definition of locally decodable codes under the mild constraint that the message symbols are explicitly included in the codeword.

Definition 2 *An encoding algorithm* $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$ *for the code $\mathcal{C} = \mathrm{Im}(\mathrm{Enc})$ is* systematic *if there is a polynomial-time computable function $I : [L] \to [\hat{L}]$ such that for all $f \in \{0,1\}^L$, $\hat{f} = \mathrm{Enc}(f)$, and all $x \in [L]$, we have $\hat{f}(I(x)) = f(x)$, where we interpret 0 and 1 as elements of Σ in some canonical way.*

Lemma 3 *If* $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$ *is systematic and* $\mathcal{C} = \mathrm{Im}(\mathrm{Enc})$ *has a local* δ*-decoding algorithm for codeword symbols running in time* t*, then* Enc *has a local* δ*-decoding algorithm (in the standard sense) running in time* $t + \mathrm{poly}(\log L)$*.*

2 Hadamard Code

Recall the m-variate Hadamard code, which consists of the truth tables of all $\mathbb{Z}_2$-linear functions $L : \{0,1\}^m \to \{0,1\}$.

Proposition 4 *The* m*-variate Hadamard code has a local* $(1/4-\varepsilon)$*-decoding algorithm for codeword symbols running in time* $\mathrm{poly}(m, 1/\varepsilon)$*.*

Proof: We are given oracle access to $g : \{0,1\}^m \to \{0,1\}$ that is at distance less than $1/4 - \varepsilon$ from some (unknown) linear function L, and we want to compute $L(x)$ at an arbitrary point $x \in \{0,1\}^m$. The idea is *random self-reducibility*: we can reduce computing L at an arbitrary point to computing L at uniformly random points, where g is likely to give the correct answer. Specifically, $L(x) = L(x \oplus r) \oplus L(r)$ for every r, and both $x \oplus r$ and r are uniformly distributed if we choose $r \xleftarrow{\mathrm{R}} \{0,1\}^m$. The probability that g differs from L at either of these points is at most $2 \cdot (1/4 - \varepsilon) = 1/2 - 2\varepsilon$. Thus $g(x \oplus r) \oplus g(r)$ gives the correct answer with probability noticeably larger than 1/2. We can amplify this success probability by repetition. Specifically, if we define $\mathrm{Dec}^g(x) = \mathrm{maj}_{1 \le j \le t}\{g(r_j) \oplus g(r_j \oplus x)\}$, where $r_1, \ldots, r_t$ are chosen independently at random and where $t = O(1/\varepsilon^2)$, then we get error probability at most 1/3 in computing $L(x)$. ■

This local decoding algorithm is optimal in terms of its decoding distance and running time, but the problem is that the Hadamard code has exponentially small rate.

3 Reed–Muller Code

Recall that the m-variate Reed–Muller code of degree d over $\mathbb{F}_q$ consists of all multivariate polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ of total degree at most d. This code has minimum distance $\delta = 1 - d/q$. Reed–Muller Codes are a common generalization of both Hadamard and Reed–Solomon codes, and thus we can hope that for an appropriate setting of parameters, we will be able to get the best of both kinds of codes. That is, we want to combine the efficient local decoding of the Hadamard code with the good rate of Reed-Solomon codes.

Theorem 5 *The Reed-Muller Code of degree* d *in* m *variables has a local* $1/12$*-decoding algorithm for codeword symbols running in time* $\mathrm{poly}(m, |\mathbb{F}|)$ *provided* $d \le |\mathbb{F}|/9$ *and* $|\mathbb{F}| \ge 36$*.*

Note the running time of the decoder is roughly the m'th root of the block length $\hat{L} = |\mathbb{F}|^m$. When $m = 1$, our decoder can query the entire string and we simply obtain a global decoding algorithm for Reed-Solomon Codes (which we already know how to achieve). But for large enough m, the decoder can only access a small fraction of the received word. In fact, one can improve the running time to $\mathrm{poly}(m, d, \log|\mathbb{F}|)$, but the weaker result is sufficient for our purposes.

The key idea behind the decoder is to do restrictions to random *lines* in $\mathbb{F}^m$. The restriction of a Reed–Muller codeword to such a line is a Reed–Solomon codeword, and we can afford to run our global Reed–Solomon decoding algorithm on the line.

Formally, for $a, b \in \mathbb{F}^m$, we define the *line through a in direction b* as the function $\ell_{a,b} : \mathbb{F} \to \mathbb{F}^m$ given by $\ell_{a,b}(x) = a + xb$. If $g : \mathbb{F}^m \to \mathbb{F}$ is any function and $\ell : \mathbb{F} \to \mathbb{F}^m$ is a line, then we use $g|_\ell$ to denote the *restriction of g to ℓ*, which is simply the composition $g \circ \ell : \mathbb{F} \to \mathbb{F}$. Note that if p is any polynomial of total degree at most d, then $p|_\ell$ is a (univariate) polynomial of degree at most d.

So we are given an oracle g of distance less than δ from some degree d polynomial $p : \mathbb{F}^m \to \mathbb{F}$, and we want to compute $p(x)$ for some $x \in \mathbb{F}^m$. We begin be choosing a random line ℓ through x. Every point of $\mathbb{F}^m \setminus \{x\}$ lies on exactly one line through x, so the points on ℓ (except x) are distributed uniformly at random over the whole domain, and thus g and p are likely to agree on these points. Thus we can hope to use the points on this line to reconstruct the value of $p(x)$. If δ is sufficiently small compared to the degree (e.g. $\delta = 1/3(d+1)$), we can simply interpolate the value of $p(x)$ from $d+1$ random points on the line. This gives rise to the following algorithm.

Local δ-Decoder for $\delta = 1/3(d+1)$. **Input** : $g \colon \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial p of total degree at most d, and $x \in \mathbb{F}^m$
Output : $p(x)$
Steps :

1. Choose $y \xleftarrow{\text{R}} \mathbb{F}^m$. Let $\ell = \ell_{x,y} = \{x + ty\}_{t \in \mathbb{F}}$
2. Query $\beta_0 = g|_\ell(\alpha_0), ..., \beta_d = g|_\ell(\alpha_d)$ where $\alpha_0, ..., \alpha_d \in \mathbb{F} \setminus \{0\}$ are any fixed points
3. Interpolate to find a unique univariate polynomial q of degree at most d s.t. $\forall i, q(\alpha_i) = \beta_i$
4. Output $q(0)$

Analysis : Observe that for all $\alpha_i \in \mathbb{F} \setminus \{0\}$, $\ell_{x,y}(\alpha_i)$ is uniform over the $y \in \mathbb{F}^m$. This implies that

$$\Pr_\ell[g_|\ell(\alpha_i) \neq p|_\ell(\alpha_i)] \leq \delta = \frac{1}{3(d+1)}$$

By union bound

$$\Pr_\ell[\exists i, g_|\ell(\alpha_i) \neq p|_\ell(\alpha_i)] \leq (d+1)\delta = \frac{1}{3}$$

Namely, $\forall i, q(\alpha_i) = p|_\ell(\alpha_i)$, w.p. at least $2/3$. Therefore, $q(0) = p(x)$, w.p. at least $2/3$. The running time of the algorithm is $\text{poly}(m, |\mathbb{F}|)$.

Local δ-Decoder for $\delta = 1/12$. We alter steps 2 and 3 in the above algorithm. In step 2, instead of querying only $d+1$ points, we query over *all* points in ℓ. In step 3, instead of interpolation, we use the *global* $1/3$-decoding algorithm for Reed–Solomon codes to decode the univariate polynomial $p|_\ell$. (A $1/3$-decoder for Reed–Solomon codes follows from the $(1 - 2\sqrt{d/|\mathbb{F}|})$ list-decoding algorithm we saw in Lecture 15. Since $1/3 \leq 1 - 2\sqrt{d/|\mathbb{F}|}$, the list-decoder will produce a list containing all univariate polynomials at distance less than $1/3$, and since $1/3$ is smaller than half the minimum distance $(1 - d/|\mathbb{F}|)$, there will be only one good decoding.) Formally, the algorithm proceeds as follows.

Local δ-Decoder for $\delta = 1/12$. **Input** : $g\colon \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial p of total degree at most d, and $x \in \mathbb{F}^m$
Output : $p(x)$
Steps :

1. Choose $y \xleftarrow{\text{R}} \mathbb{F}^m$. Let $\ell = \ell_{x,y} = \{x + ty\}_{t \in \mathbb{F}}$
2. Query g at all points on ℓ.
3. Run the 1/3-decoder for Reed–Solomon codes on $g|_\ell$ to obtain the (unique) polynomial q at distance less than 1/3 from $g|_\ell$ (if one exists).
4. Output $q(0)$.

Analysis : The expected distance (between $g|_\ell$ and $p|_\ell$) is small:

$$\mathop{\mathrm{E}}_{\ell}[\Delta(g|_\ell, p|_\ell)] \leq \frac{1}{|\mathbb{F}|} + \delta < \frac{1}{36} + \frac{1}{12} = \frac{1}{9}$$

(where the term $\frac{1}{|\mathbb{F}|}$ is due to the fact that the point x is not random). Therefore, by Markov's Inequality,

$$\Pr[\Delta(g|_\ell, p|_\ell) \geq 1/3] \leq 1/3$$

Thus, with probability at least 2/3, we have that $p|_\ell$ is the unique polynomial of degree at most d at distance less than 1/3 from $g|_\ell$ and thus q must equal $p|_\ell$.

4 The Encoding

Given $f\colon \{0,1\}^\ell \to \{0,1\}$, we want to encode it as a Reed–Muller codeword $\hat{f}\colon \{0,1\}^{\hat{\ell}} \to \Sigma$ s.t.:

- The encoding time is $2^{O(\ell)}$
- $\hat{\ell} = O(\ell)$
- The code is systematic in the sense of Definition 2. Informally, this means that f should be a "restriction" of $\hat{f}$.

Note: the usual encoding for Reed–Muller codes, where the message gives the coefficients of the polynomial does not suffice, since this encoding is not systematic.

4.1 Multilinear Extension

Claim 6 *For any $f\colon \{0,1\}^\ell \to \{0,1\}$ there exists a (unique) polynomial $\hat{f}\colon \mathbb{F}^\ell \to \mathbb{F}$ s.t. $\hat{f}_{|\{0,1\}^\ell} \equiv f$ of degree at most 1 in each variable.*

Proof: We prove the existence of the polynomial $\hat{f}$. Define

$$\hat{f}(x_1, ..., x_\ell) = \sum_{\alpha \in \{0,1\}^\ell} f(\alpha)\delta_\alpha(x)$$

for

$$\delta_\alpha(x) = \left(\prod_{i:\ \alpha_i = 1} x_i\right)\left(\prod_{i:\ \alpha_i = 0} (1 - x_i)\right)$$

Note that for $x \in \{0,1\}^\ell$, $\delta_\alpha(x) = 1$ only when $\alpha = x$, therefore $\hat{f}_{|\{0,1\}^\ell} \equiv f$. Uniqueness of $\hat{f}$ is derived from counting degrees of freedom. The bound on the total degree is by inspection. ■

Thinking of $\hat{f}$ as an encoding of f, let's inspect the properties of this encoding.

- For the local 1/12-decoding algorithm above, we need $|\mathbb{F}| \geq 9\ell$, therefore, we can take $|\mathbb{F}| = \Theta(\ell)$.
- The encoding time is $2^{O(\ell)}$, as computing a single point of $\hat{f}$ requires summing over $2^{O(\ell)}$ elements.
- The code is systematic, since $\hat{f}$ is an extension of f.
- However, we run into troubles in terms of the input length $\hat{\ell} = \ell \log |\mathbb{F}| = \Theta(\ell \log \ell)$. This is slightly too large; we aim for having $\hat{\ell} = O(\ell)$.

4.2 Low-Degree Extension

To solve the problem of the input length $\hat{\ell}$ in the multi-linear encoding, we reduce the dimension of the polynomial $\hat{f}$ by changing the embedding of the domain of f: Instead of interpreting $\{0,1\}^\ell \subseteq \mathbb{F}^{\hat{\ell}}$ as an embedding of the domain of f in $\mathbb{F}^{\hat{\ell}}$, we map $\{0,1\}^\ell$ to $\mathbb{H}^m$ for some subfield $\mathbb{H} \subseteq \mathbb{F}$, and as such embed it in $\mathbb{F}^m$.

More precisely, we fix a subfield $\mathbb{H} \subseteq \mathbb{F}$ of size (roughly) $|\mathbb{H}| = \sqrt{|\mathbb{F}|}$. Choose $m = \frac{\ell}{\log|\mathbb{H}|}$, and fix some efficient one-to-one mapping from $\{0,1\}^\ell$ into $\mathbb{H}^m$. With this mapping, view f as a polynomial $f\colon \mathbb{H}^m \to \mathbb{F}$.

Analogously to before, we have the following theorem.

Theorem 7 *There exists a (unique) $\hat{f}\colon \mathbb{F}^m \to \mathbb{F}$ of degree at most $|\mathbb{H}| - 1$ in each variable s.t. $\hat{f}|_{H^m} \equiv f$.*

The total degree of $\hat{f}$ is $m(|\mathbb{H}| - 1) \leq m\,|\mathbb{H}| \leq \ell\sqrt{|\mathbb{F}|}$. So we can $\frac{1}{12}$-decode, as long as $|\mathbb{F}| \geq 81\ell^2$ (recall that our decoding algorithm requires that the degree is at most $|\mathbb{F}|/9$). Let's inspecting the properties of $\hat{f}$ as an encoding of f:

- The input length is $m \log|\mathbb{F}| = \frac{\ell}{\log|\mathbb{H}|}\log|\mathbb{F}| = 2\ell$. Namely, $\hat{\ell} = O(\ell)$ as desired.
- The code is systematic as long as our mapping from $\{0,1\}^\ell$ to $\mathbb{H}^m$ is efficient.

5 Implications for Worst-Case/Average-Case Connections

Plugging the local decoding algorithm for Reed–Muller codes given above into the general result from last time, we obtain the following

Theorem 8 *If there exists $f\colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against (non-uniform) time $t(\ell)$, then there exists $\hat{f}\colon \{0,1\}^{O(\ell)} \to \{0,1\}^{O(\log \ell)}$ in $\mathbf{E}$ that is $1/12$ average-case hard against (non-uniform) time $t(\ell)/poly(\ell)$.*

This differs from our original goal in two ways: $\hat{f}$ is not Boolean, and we only get hardness $1/12$ (instead of $1/2-\varepsilon$). These can be remedied by concatenating the Reed–Muller code with a Hadamard code. Note that the Hadamard code is for message space $\mathbb{F}$, so it can be 1/4-decoded by brute-force in time $\mathrm{poly}(|\mathbb{F}|)$ (which is the amount of time already taken by our decoder).[1] Using this, we obtain:

Theorem 9 *If there exists $f\colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against (non-uniform) time $t(\ell)$, then there exists $\hat{f}\colon \{0,1\}^{O(\ell)} \to \{0,1\}$ in $\mathbf{E}$ that is $1/48$ average-case hard against (non-uniform) time $t(\ell)/poly(\ell)$.*

Hardness Amplification. Now our goal is to boost this constant hardness $1/48$ to $1/2-\varepsilon$. There are some generic techniques for doing this, known as Direct Product Theorems or Yao's XOR lemma (for Boolean functions). In those methods we use independent copies of the function at hand. For example, it Yao's XOR lemma, we let f' consist of k independent copies of $\hat{f}$,

$$f'(x_1,...,x_k) = (\hat{f}(x_1),...,\hat{f}(x_k)).$$

Intuitively, if $\hat{f}$ is 1/12 average-case hard, then f' should $(1-(\frac{11}{12})^k)$-average case hard. Similarly, if we take the XOR of k independent copies of a Boolean function, the hardness should approach 1/2 exponentially fast. These statements are (basically) true, though proving them turns out to be harder than one might expect.

The main disadvantage of this approach (for our purposes) is that the input length is $k\ell$ while we aim for input length of $O(\ell)$. To overcome this problem, it is possible to use derandomization, namely, evaluate $\hat{f}$ on *dependent* inputs instead of independent ones.

In the next lecture, we will take a different approach, namely to generalize our notion and algorithms for locally decodable codes to locally *list*-decodable codes. (In fact, the aforementioned results on hardness amplification can be interpreted in a coding-theoretic language as well.)

6 Other Connections

As you will see on PS6, locally decodable codes are closely related to protocols for *private information retrieval*. Another connection, and actually the setting in which these local decoding

[1] This concatenation step is equivalent to applying the Goldreich–Levin hardcore predicate to $\hat{f}$, for those familiar with it. However, for the parameters we are using (where the message space is small and we are doing unique decoding), we do not need the sophisticated Goldreich–Levin algorithm (which can be interpreted as a "local list-decoding algorithm," a notion we will define next time).

algorithms were first discovered, is to *program self-correctors*. Suppose you have a program for computing a function, such as the determinant, which happens to be a codeword in a locally decodable code (e.g. the determinant is a low-degree multivariate polynomial). Then, even if this program has some bugs and gives the wrong answer on some small fraction of inputs, you can use the local decoding algorithm to obtain the correct answer on *all* inputs with high probability.