Computational Complexity: A Modern Approach

Draft of a book: Dated January 2007
Comments welcome!

Sanjeev Arora and Boaz Barak
Princeton University
complexitybook@gmail.com

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to
complexitybook@gmail.com — **Thank You!!**

Chapter 17

Hardness Amplification and Error Correcting Codes

We pointed out in earlier chapters (e.g., Chapter **??** the distinction between worst-case hardness and average-case hardness. For example, the problem of finding the smallest factor of every given integer seems difficult on worst-case instances, and yet is trivial for at least half the integers – namely, the even ones. We also saw that functions that are average-case hard have many uses, notably in cryptography and derandomization.

In this chapter we study techniques for *amplifying* hardness. First, we see Yao's XOR Lemma, which transforms a "mildly hard" function (i.e., one that is hard to compute on a small fraction of the instances) to a function that is *extremely hard*, for which the best algorithm is as bad as the algorithm that just randomly guesses the answer. We mentioned Yao's result in the chapter on cryptography as a means to transform weak one-way functions into strong one-way functions. The second result in this chapter is a technique to use *error-correcting codes* to transform *worst-case hard* functions into average-case hard functions. This transformation unfortunately makes the running time exponential, and is thus useful only in derandomization, and not in cryptography.

In addition to their applications in complexity theory, the ideas covered here have had other uses, including new constructions of error-correcting codes and new algorithms in machine learning.

17.1 Hardness and Hardness Amplification.

We now define a slightly more refined notion of hardness, that generalizes both the notions of worst-case and average-case hardness given in Definition 16.7:

Definition 17.1 (Hardness)
Let $f : \{0,1\}^* \to \{0,1\}$ and $\rho : \mathbb{N} \to [0,1]$. We define $\mathsf{H}^{\rho}_{\mathsf{avg}}(f)$ to be the function from $\mathbb{N}$ to $\mathbb{N}$ that maps every number n to the largest number S such that $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < \rho(n)$ for every Boolean circuit C on n inputs with size at most S.

Note that, in the notation of Definition 16.7, $\mathsf{H}_{\mathsf{wrs}}(f) = \mathsf{H}^1_{\mathsf{avg}}(f)$ and $\mathsf{H}_{\mathsf{avg}}(f)(n) = \max\{S : \mathsf{H}^{1/2+1/S}_{\mathsf{avg}}(f)(n) \geq S\}$. In this chapter we show the following results for every two functions $S, S' : \mathbb{N} \to \mathbb{N}$:

Worst-case to mild hardness. If there is a function $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) = \mathsf{H}^1_{\mathsf{avg}}(f)(n) \geq S(n)$ then there is a function $f' \in \mathbf{E}$ such that $\mathsf{H}^{0.99}_{\mathsf{avg}}(f)(n) \geq S(\epsilon n)^\epsilon$ for some constant $\epsilon > 0$ and every sufficiently large n.

Mild to strong hardness. If $f' \in \mathbf{E}$ satisfies $\mathsf{H}^{0.99}_{\mathsf{avg}}(f')(n) \geq S'(n)$ then there is $f'' \in \mathbf{E}$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{avg}}(f'')(n) \geq S'(n^\epsilon)^\epsilon$.

Combining these two results with Theorem 16.10, this implies that if there exists a function $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ then there exists an $S(\ell^\epsilon)^\epsilon$-pseudorandom generator for some $\epsilon > 0$, and hence:

Corollary 1 If there exists $f \in \mathbf{E}$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq 2^{n^\epsilon}$ then $\mathbf{BPP} \subseteq \mathbf{QuasiP} = \cup_c \mathbf{DTIME}(2^{\log^c n})$.

Corollary 2 If there exists $f \in \mathbf{E}$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq n^{\omega(1)}$ then $\mathbf{BPP} \subseteq \mathbf{SUBEXP} = \cap_\epsilon \mathbf{DTIME}(2^{n^\epsilon})$.

To get to $\mathbf{BPP} = \mathbf{P}$, we need a stronger transformation. We do this by showing how to transform in one fell swoop, a function $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f) \geq S(n)$ into a function $f' \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq S(\epsilon n)^\epsilon$ for some $\epsilon > 0$. Combined with Theorem 16.10, this implies that $\mathbf{BPP} = \mathbf{P}$ if there exists $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f) \geq 2^{\Omega(n)}$.

17.2 Mild to strong hardness: Yao's XOR Lemma.

We start with the second result described above: transforming a function that has "mild" average-case hardness to a function that has strong average-case hardness. The transformation is actually quite simple and natural, but its analysis is somewhat involved (yet, in our opinion, beautiful).

THEOREM 17.2 (YAO'S XOR LEMMA)
For every $f : \{0,1\}^n \to \{0,1\}$ and $k \in \mathbb{N}$, define $f^{\oplus k} : \{0,1\}^{nk} \to \{0,1\}$ as follows: $f^{\oplus k}(x_1, \ldots, x_k) = \sum_{i=1}^k f(x_i) \pmod 2$.
For every $\delta > 0, S$ and $\epsilon > 2(1-\delta/2)^k$, if $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$ then

$$\mathsf{H}^{1/2+\epsilon}_{\mathsf{avg}}(f^{\oplus k}) \geq \frac{\epsilon^2}{100 \log(1/\delta\epsilon)} S$$

The intuition behind Theorem 17.2 derives from the following fact. Suppose we have a biased coin that, whenever it is tossed, comes up heads with probability $1-\delta$ and tails with probability δ. If δ is small, each coin toss is fairly predictable. But suppose we now toss it k times and define a composite coin toss that is "heads" iff the coin came up heads an odd number of times. Then the probability of "heads" in this composite coin toss is at most $1/2 + (1-2\delta)^k$ (see Exercise 1), which tends to $1/2$ as k increases. Thus the parity of coin tosses becomes quite unpredictable. The

analogy to our case is that intuitively, for each i, a circuit of size S has chance at most $1-\delta$ of "knowing" $f(x_i)$ if x_i is random. Thus from its perspective, whether or not it will be able to know $f(x_i)$ is like a biased coin toss. Hence its chance of guessing the parity of the k bits should be roughly like $1/2 + (1-2\delta)^k$.

We transform this intuition into a proof via an elegant result of Impagliazzo, that provides some fascinating insight on mildly hard functions.

DEFINITION 17.3 (δ-DENSITY DISTRIBUTION)
For $\delta < 1$ a *δ-density* distribution H over $\{0,1\}^n$ is one such that for every $x \in \{0,1\}^n$, $\Pr[H = x] \leq \frac{2^{-n}}{\delta}$.

REMARK 17.4
Note that in Chapter 16 we would have called it a distribution with min entropy $n - \log 1/\delta$.

The motivating example for this definition is the distribution that is uniform over some subset of size $\delta 2^n$ and has 0 probability outside this set.

A priori, one can think that a function f that is hard to compute by small circuits with probability $1-\delta$ could have two possible forms: **(a)** the hardness is sort of "spread" all over the inputs, and it is roughly $1-\delta$-hard on every significant set of inputs or **(b)** there is a subset H of roughly a δ fraction of the inputs such that on H the function is *extremely hard* (cannot be computed better than $\frac{1}{2}+\epsilon$ for some tiny ϵ) and on the rest of the inputs the function may be even very easy. Such a set may be thought of as lying at the core of the hardness of f and is sometimes called the *hardcore set*. Impagliazzo's Lemma shows that actually *every* hard function has the form **(b)**. (While the Lemma talks about distributions and not sets, one can easily transform it into a result on sets.)

LEMMA 17.5 (IMPAGLIAZZO'S HARDCORE LEMMA)
For every $\delta > 0$, $f : \{0,1\}^n \to \{0,1\}^n$, and $\epsilon > 0$, if $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$ then there exists a distribution H over $\{0,1\}^n$ of density at least $\delta/2$ such that for every circuit C of size at most $\frac{\epsilon^2 S}{100\log(1/\delta\epsilon)}$,

$$\Pr_{x \in_R H}[C(x) = f(x)] \leq 1/2 + \epsilon\,,$$

Proof of Yao's XOR Lemma using Impagliazzo's Hardcore Lemma.

We now use Lemma 17.5 to transform the biased-coins intuition discussed above into a proof of the XOR Lemma. Let $f : \{0,1\}^n \to \{0,1\}$ be a function such that $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$, let $k \in \mathbb{N}$ and suppose, for the sake of contradiction, that there is a circuit C of size $\frac{\epsilon^2}{100\log(1/\delta\epsilon)}S$ such that

$$\Pr_{(x_1,\ldots,x_k)\in_R U_n^k}\left[C(x_1,\ldots,x_k) = \sum_{i=1}^{k} f(x_i) \pmod 2\right] \geq 1/2 + \epsilon\,, \tag{1}$$

where $\epsilon > 2(1-\delta/2)^k$.

Let H be the hardcore distribution of dens ity at least $\delta' = \delta/2$ that is obtained from Lemma 17.5, on which every circuit C' fails to compute f with probability better than $1/2 + \epsilon/2$. Define a distribution G over $\{0,1\}^n$ as follows: for every $x \in \{0,1\}^n$, $\Pr[G = x] = (1 - \delta' \Pr[H = x])/(1-\delta')$.

Note that G is indeed a well-defined distribution, as H has density at least δ'. Also note that if H was the uniform distribution over some subset of $\{0,1\}^n$ of size $\delta' 2^n$, then G will be the uniform distribution over the complement of this subset.

We can think of the process of picking a uniform element in $\{0,1\}^n$ as follows: first toss a δ'-biased coin that comes up "heads" with probability δ. Then, if it came up "heads" choose a random element out of H, and with probability $1-\delta'$, and otherwise choose a random element out of G. We shorthand this and write

$$U_n = (1-\delta')G + \delta' H \,. \tag{2}$$

If we consider the distribution $(U_n)^2$ of picking *two* random strings, then by (2) it can be written as $(1-\delta')^2 G^2 + (1-\delta')\delta' GH + \delta'(1-\delta')HG + \delta'^2 H^2$. Similarly, for every k

$$(U_n)^k = (1-\delta')^k G^k + (1-\delta')^{k-1}\delta' G^{k-1}H + \cdots + \delta'^k H^k \,. \tag{3}$$

For every distribution $\mathcal{D}$ over $\{0,1\}^{nk}$ let $P_{\mathcal{D}}$ be the probability of the event of the left-hand side of (1) that $C(x_1,\ldots,x_k) = \sum_{i=1}^k f(x_i) \pmod 2$ where $x_1,\ldots,x_k$ are chosen from $\mathcal{D}$. Then, combining (1) and (3),

$$1/2 + \epsilon \leq P_{(U_n)^k} = (1-\delta')^k P_{G^k} + (1-\delta')^{k-1}\delta' P_{G^{k-1}H} + \cdots + \delta'^k P_{H^k} \,.$$

But since $\delta' = \delta/2$ and $\epsilon > 2(1-\delta/2)^k$ and $P_{G^k} \leq 1$ we get

$$1/2 + \epsilon/2 \leq 1/2 + \epsilon - (1-\delta')^k \leq (1-\delta')^{k-1}\delta' P_{G^{k-1}H} + \cdots + \delta'^k P_{H^k} \,.$$

Notice, the coefficients of all distributions on the right hand side sum up to less than one, so there must exist a distribution $\mathcal{D}$ that has at least one H component such that $P_{\mathcal{D}} \geq 1/2 + \epsilon/2$. Suppose that $\mathcal{D} = G^{k-1}H$ (all other cases are handled in a similar way). Then, we get that

$$\Pr_{X_1,\ldots,X_{k-1}\in_R G, X_k \in_R H}[C(X_1,\ldots,X_{k-1},X_k) = \sum_{i=1}^k f(X_i) \pmod 2] \geq 1/2 + \epsilon/2 \,. \tag{4}$$

By the averaging principle, (4) implies that there *exist* $k-1$ strings $x_1,\ldots,x_{k-1}$ such that if $b = \sum_{i=1}^{k-1} f(x_i) \pmod 2$ then,

$$\Pr_{X_k \in_R H}[C(x_1,\ldots,x_{k-1},X_k) = b + f(X_k) \pmod 2] \geq 1/2 + \epsilon/2 \,. \tag{5}$$

But by "hardwiring" the values $x_1,\ldots,x_k$ and b into the circuit C, (5) shows a direct contradiction to the fact that H is a hardcore distribution for the function f. ■

17.3 Proof of Impagliazzo's Lemma

Let f be a function with $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$. To Prove Lemma 17.5 we need to show a distribution H over $\{0,1\}^n$ (with no element of weight more than $2\cdot 2^{-n}/\delta$) on which *every* circuit C of size S' cannot compute f with probability better than $1/2+\epsilon$ (where S',ϵ are as in the Lemma's statement).

Let's think of this task as a game between two players named *Russell* and *Noam*. Russell first sends to Noam some distribution H over $\{0,1\}^n$ with density at least δ. Then Noam sends to Russell some circuit C of size at most S'. Russell then pays to Noam $\mathsf{E}_{x\in_R H}[\mathsf{Right}_C(x)]$ dollars, where $\mathsf{Right}_C(x)$ is equal to 1 if $C(x) = f(x)$ and equal to 0 otherwise. What we need to prove is that there is distribution that Russell can choose, such that no matter what circuit Noam sends, Russell will not have to pay him more than $1/2 + \epsilon$ dollars.

An initial observation is that Russell could have easily ensured this if he was allowed to play second instead of first. Indeed, under our assumptions, for *every* circuit C of size S (and so, in particular also for circuits of size S' which is smaller than S), there exists a set S_C of at least $\delta 2^n \geq (\delta/2)2^n$ inputs such that $C(x) \neq f(x)$ for every $x \in S_C$. Thus, if Noam had to send his circuit C, then Russell could have chosen H to be the uniform distribution over S_C. Thus H would have density at least $\delta/2$ and $\mathsf{E}_{x\in_R H}[\mathsf{Right}_C(x)] = 0$, meaning that Russell wouldn't have to pay Noam a single cent.

Now this game is a *zero sum game*, since whatever Noam gains Russell loses and vice versa, tempting us to invoke von-Neumann's famous *Min-Max Theorem* (see Note 17.7) that says that in a zero-sum game it does not matter who plays first *as long as we allow randomized strategies*.[1] What does it mean to allow randomized strategies in our context? It means that Noam can send a *distribution* $\mathcal{C}$ over circuits instead of a single circuit, and the amount Russell will pay is $\mathsf{E}_{C\in_R\mathcal{C}}\mathsf{E}_{x\in_R H}[\mathsf{Right}_C(x)]$. (It also means that Russell is allowed to send a distribution over $\delta/2$-density distributions, but this is equivalent to sending a single $\delta/2$-density distribution.)

Thus, we only need to show that, when playing second, Russell can still ensure a payment of at most $1/2 + \epsilon$ dollars even when Noam sends a distribution $\mathcal{C}$ of S'-sized circuits. For every distribution $\mathcal{C}$, we say that an input $x \in \{0,1\}^n$ is *good for Noam* (*good* for short) with respect to $\mathcal{C}$ if $\mathsf{E}_{C\in_R\mathcal{C}}[\mathsf{Right}_C(x)] \geq 1/2 + \epsilon$. It suffices to show that for every distribution $\mathcal{C}$ over circuits of size at most S', the number of good x's with respect to $\mathcal{C}$ is at most $1 - \delta/2$. (Indeed, this means that for every $\mathcal{C}$, Russell could choose as its distribution H the uniform distribution over the bad inputs with respect to $\mathcal{C}$.)

Suppose otherwise, that there is at least a $1 - \delta/2$ fraction of inputs that are good for $\mathcal{C}$. We will use this to come up with an S-sized circuit C that computes f on at least a $1 - \delta$ fraction of the inputs in $\{0,1\}^n$, contradicting the assumption that $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$. Let $t = 10\log(1/\delta\epsilon)/\epsilon^2$, choose $C_1, \ldots, C_t$ at random from $\mathcal{C}$ and let $C = \mathrm{maj}\{C_1, \ldots, C_t\}$ be the circuit of size $tS' < S$ circuit that on input x outputs the majority value of $\{C_1(x), \ldots, C_t(x)\}$. If x is good for $\mathcal{C}$, then by the Chernoff bound we have that $C(x) = f(x)$ with probability at least $1 - \delta/2$ over the choice of $C_1, \ldots, C_t$. Since we assume at least $1 - \delta/2$ of the inputs are good for $\mathcal{C}$, we get that

$$\mathsf{E}_{x\in_R\{0,1\}^n}\mathsf{E}_{C_1\in_R\mathcal{C},\ldots,C_t\in_R\mathcal{C}}[\mathsf{Right}_{\mathrm{maj}\{C_1,\ldots,C_t\}}(x)] \geq (1-\tfrac{\delta}{2})(1-\tfrac{\delta}{2}) \geq 1-\delta\,. \tag{6}$$

But by linearity of expectation, we can switch the order of expectations in (6) obtaining that

$$\mathsf{E}_{C_1\in_R\mathcal{C},\ldots,C_t\in_R\mathcal{C}}\mathsf{E}_{x\in_R\{0,1\}^n}[\mathsf{Right}_{\mathrm{maj}\{C_1,\ldots,C_t\}}(x)] \geq 1-\delta\,,$$

[1] The careful reader might note that another requirement is that the set of possible moves by each player is *finite*, which does not seem to hold in our case as Russell can send any one of the infinitely many $\delta/2$-density distributions. However, by either requiring that the probabilities of the distribution are multiples of $\frac{\epsilon}{100\cdot 2^n}$ (which won't make any significant difference in the game's outcome), or using the fact that each such distribution is a convex sum of uniform distributions over sets of size at least $(\delta/2)2^n$ (see Exercise 9 of Chapter 16), we can make this game finite.

which in particular implies that *there exists* a circuit C of size at most S such that $\mathsf{E}_{x\in_R U_n}[\mathsf{Right}_C(x)] \geq 1-\delta$, or in other words, C computes f on at least a $1-\delta$ fraction of the inputs. ■

REMARK 17.6

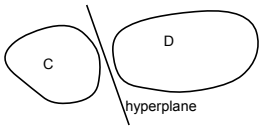
Taken in the contrapositive, Lemma 17.5 implies that if for every significant chunk of the inputs there is some circuit that computes f with on this chunk with some advantage over $1/2$, then there is a single circuit that computes f with good probability over all inputs. In machine learning such a result (transforming a way to weakly predict some function into a way to strongly predict it) is called *Boosting* of learning methods. Although the proof we presented here is non-constructive, Impagliazzo's original proof was constructive, and was used to obtain a boosting algorithm yielding some new results in machine learning, see [**?**].

NOTE 17.7 (THE MIN-MAX THEOREM)
A *zero sum game* is, as the name implies, a game between two parties in which whatever one party loses is won by the other party. It is modeled by an $m \times n$ matrix $A = (a_{i,j})$ of real numbers. The game consists of only a single move. One party, called the *minimizer* or *column player*, chooses an index $j \in [n]$ while the other party, called the *maximizer* or *row player*, chooses an index $i \in [m]$. The *outcome* is that the column player has to pay $a_{i,j}$ units of money to the row player (if $a_{i,j}$ is negative then actually the row player has to pay). Clearly, the *order* in which players make their moves is important. Surprisingly, if we allow the players randomized strategies, then the order of play becomes unimportant.

The game with *randomized* (also known as *mixed*) strategies is as follows. The column player chooses a *distribution* over the columns; that is, a vector $\mathbf{p} \in [0,1]^n$ with $\sum_{i=1}^n p_i = 1$. Similarly, the row player chooses a distribution $\mathbf{q}$ over the rows. The amount paid is the expectation of $a_{i,j}$ for j chosen from $\mathbf{p}$ and i chosen from $\mathbf{q}$. If we think of $\mathbf{p}$ as a column vector and $\mathbf{q}$ as a row vector then this is equal to $\mathbf{q}A\mathbf{p}$. The min-max theorem says:

$$\min_{\substack{\mathbf{p}\in[0,1]^n \\ \sum_i p_i = 1}} \max_{\substack{\mathbf{q}\in[0,1]^m \\ \sum_i q_i = 1}} \mathbf{q}A\mathbf{p} = \max_{\substack{\mathbf{q}\in[0,1]^m \\ \sum_i q_i = 1}} \min_{\substack{\mathbf{p}\in[0,1]^n \\ \sum_i p_i = 1}} \mathbf{q}A\mathbf{p} \tag{7}$$

The min-max theorem can be proven using the following result, known as Farkas' Lemma:[2] if C and D are disjoint convex subsets of $\mathbb{R}^m$, then there is an $m-1$ dimensional hyperplane that separates them. That is, there is a vector $\mathbf{z}$ and a number a such that for every $\mathbf{x} \in C$, $\langle \mathbf{x}, \mathbf{z} \rangle = \sum_i x_i z_i \leq a$ and for every $\mathbf{y} \in D$, $\langle \mathbf{y}, \mathbf{z} \rangle \geq a$. (A subset $C \subseteq \mathbb{R}^m$ is *convex* if whenever it contains a pair of points $\mathbf{x}, \mathbf{y}$, it contains the line segment $\{\alpha\mathbf{x} + (1-\alpha)\mathbf{y} : 0 \leq \alpha \leq 1\}$ that lies between them.) We ask you to prove Farkas' Lemma in Exercise 2 but here is a "proof by picture" for the two dimensional case:



Farkas' Lemma implies the min-max theorem by noting that $\max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{q}A\mathbf{p} \geq c$ if and only if the convex set $D = \{A\mathbf{p} : \mathbf{p} \in [0,1]^n \ \sum_i p_i = 1\}$ does not intersect with the convex set $C = \left\{\mathbf{x} \in \mathbb{R}^m : \forall_{i\in[m]} x_i < c\right\}$ and using the Lemma to show that this implies the existence of a probability vector $\mathbf{q}$ such that $\langle \mathbf{q}, \mathbf{y} \rangle \geq c$ for every $\mathbf{y} \in D$ (see Exercise 3). The Min-Max Theorem is equivalent to another well-known result called *linear programming duality*, that can also be proved using Farkas' Lemma (see Exercise 4).

17.4 Error correcting codes: the intuitive connection to hardness amplification

Now we construct average-case hard functions using functions that are only worst-case hard. To do so, we desire a way to transform any function f to another function g such that if there is a small circuit that computes g approximately (i.e., correctly outputs $g(x)$ for many x) then there is a small circuit that computes f at all points. Taking the contrapositive, we can conclude that if there is no small circuit that computes f then there is no small circuit that computes g approximately.

Let us reason abstractly about how to go about the above task.

View a function $f : \{0,1\}^n \to \{0,1\}$ as its truth table, namely, as a string of length 2^n, and view any circuit C for computing this function as a device that, given any index $x \in [2^n]$, gives the x'th bit in this string. If the circuit only computes g on "average" then this device may be thought of as only partially correct; it gives the right bit only for many indices x's, but not all. Thus we need to show how to turn a partially correct string for g into a completely correct string for f. This is of course reminiscent of *error correcting codes* (ECC), but with a distinct twist involving computational efficiency of decoding, which we will call *local decoding*.

The classical theory of ECC's (invented by Shannon in 1949) concerns the following problem. We want to record some data $x \in \{0,1\}^n$ on a compact disk to retrieve at a later date, but that compact disk might scratched and say 10% of its contents might be corrupted. The idea behind error correcting codes is to encode x using some *redundancy* so that such corruptions do not prevent us from recovering x.

The naive idea of redundancy is to introduce repetitions but that does not work. For example suppose we repeat each bit three times, in other words encode x as the string $y = x_1x_1x_1x_2x_2x_2 \ldots x_nx_nx_n$. But now if the first three coordinates of y are corrupted then we cannot recover x_1, even if all other coordinates of y are intact. (Note that the first three coordinates take only a $1/n \ll 10\%$ fraction of the entire string y.) Clearly, we need a smarter way.

Definition 17.8 (Error Correcting Codes)
For $x, y \in \{0,1\}^m$, the *fractional Hamming distance* of x and y, denoted $\Delta(x,y)$, is equal to $\frac{1}{m}|\{i : x_i \neq y_i\}|$.
For every $\delta \in [0,1]$, a function $E : \{0,1\}^n \to \{0,1\}^m$ is an *error correcting code (ECC) with distance* δ, if for every $x \neq y \in \{0,1\}^n$, $\Delta(E(x), E(y)) \geq \delta$. We call the set $Im(E) = \{E(x) : x \in \{0,1\}^n\}$ the set of *codewords* of E.

Suppose $E : \{0,1\}^n \to \{0,1\}^m$ is an ECC of distance $\delta > 0.2$. Then the encoding $x \to E(x)$ suffices for the CD storage problem (momentarily ignoring issues of computational efficiency). Indeed, if y is obtained by corrupting $0.1m$ coordinates of $E(x)$, then $\Delta(y, E(x)) < \delta/2$ and by the triangle inequality $\Delta(y, E(x')) > \delta/2$ for every $x' \neq x$. Thus, x is the unique string that satisfies

[2]Many texts use the name Farkas' Lemma only to denote a special case of the result stated in Note 17.7. Namely the result that there is a separating hyperplane between any disjoint sets C, D such that C is a single point and D is a set of the form $\{A\mathbf{x} : \forall_i x_i > 0\}$ for some matrix A.

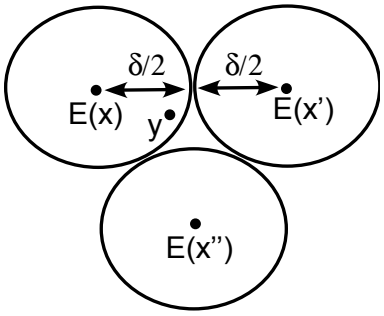


Figure 17.1: In a δ-distance error correcting code, $\Delta(E(x), E(x')) \geq \delta$ for every $x \neq x'$. We can recover x from every string y satisfying $\Delta(y, E(x)) < \delta/2$ since the $\delta/2$-radius ball around every codeword $z = E(x)$ does not contain any other codeword.

$\Delta(y, E(x)) < \delta/2$. (See Figure 17.1.)

Of course, we still need to show that error correcting codes with minimum distance 0.2 actually exist. The following lemma shows this. It introduces $H(\delta)$, the so-called *entropy function*, which lies strictly between 0 and 1 when $\delta \in (0, 1)$.

LEMMA 17.9
For every $\delta < 1/2$ and sufficiently large n, there exists a function $E : \{0,1\}^n \to \{0,1\}^{2n/(1-H(\delta))}$ that is an error correcting code with distance δ, where $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$.

PROOF: We simply choose the function $E : \{0,1\}^n \to \{0,1\}^m$ at random for $m = 2n/(1 - H(\delta)n$. That is, we choose 2^n random strings $y_1, y_2, \ldots, y_{2^n}$ and E will map the input $x \in \{0,1\}^n$ (which we can identify with a number in $[2^n]$) to the string y_x.

It suffices to show that the probability that for some $i < j$ with $i, j \in [2^n]$, $\Delta(y_i, y_j) < \delta$ is less than 1. But for every string y_i, the number of strings that are of distance at most δ to it is $\binom{m}{\lceil \delta m \rceil}$ which at most $0.99 \cdot 2^{H(\delta)m}$ for m sufficiently large (see Appendix A) and so for every $j > i$, the probability that y_j falls in this ball is bounded by $0.99 \cdot 2^{H(\delta)m}/2^m$. Since there are at most 2^{2n} such pairs i, j, we only need to show that

$$0.99 \cdot 2^{2n} \frac{2^{H(\delta)m}}{2^m} < 1 \,.$$

which is indeed the case for our choice of m. ■

REMARK 17.10
By a slightly more clever argument, we can get rid of the constant 2 above, and show that there exists such a code $E : \{0,1\}^n \to \{0,1\}^{n/(1-H(\delta))}$ (see Exercise 6). We do not know whether this is the smallest value of m possible.

Why half? Lemma 17.9 only provides codes of distance δ for $\delta < 1/2$ and you might wonder whether this is inherent or can we have codes of even greater distance. It turns out we can have codes of distance $1/2$ but only if we allow m to be exponentially larger than n (i.e., $m \geq 2^{n/2}$). For

every $\delta > 1/2$, if n is sufficiently large then there is no ECC $E : \{0,1\}^n \to \{0,1\}^m$ that has distance δ, no matter how large m is. Both these bounds are explored in Exercise 7.

The mere existence of an error correcting code is not sufficient for most applications: we need to actually be able to compute them. For this we need to show an *explicit function* $E : \{0,1\}^n \to \{0,1\}^m$ that is an ECC satisfying the following properties:

Efficient encoding There is a polynomial time algorithm to compute $E(x)$ from x.

Efficient decoding There is a polynomial time algorithm to compute x from every y such that $\Delta(y, E(x)) < \rho$ for some ρ. (For this to be possible, the number ρ must be less than $\delta/2$, where δ is the distance of E.)

There is a very rich and still ongoing body of work dedicated to this task, of which Section 17.5 describes a few examples.

17.4.1 Local decoding

For use in hardness amplification, we need ECCs with more than just efficient encoding and decoding algorithms: we need *local decoders*, in other words, decoding algorithms whose running time is polylogarithmic. Let us see why.

Recall that we are viewing a function from $\{0,1\}^n$ to $\{0,1\}$ as a string of length 2^n. To amplify its hardness, we take an *ECC* and map function f to its encoding $E(f)$. To *prove* that this works, it suffices to show how to turn any circuit that correctly computes many bits of $E(f)$ into a circuit that correctly computes all bits of f. This is formalized using a *local decoder*, which is a decoding algorithm that can compute any desired bit in the string for f using a small number of random queries in any string y that has high agreement with (in other words, low hamming distance to) $E(f)$. Since we are interested in the circuits of size poly(n)— in other words, *polylogarithmic* in 2^n —this must also be the running time of the local decoder.

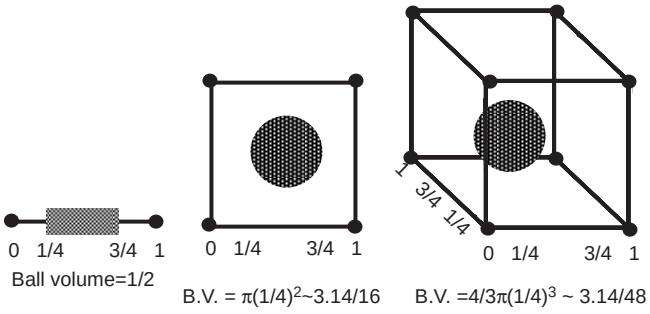
Definition 17.12 (Local decoder)
Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let ρ and q be some numbers. A *local decoder for E handling ρ errors* is an algorithm L that, given random access to a string y such that $\Delta(y, E(x)) < \rho$ for some (unknown) $x \in \{0,1\}^n$, and an index $j \in \mathbb{N}$, runs for polylog(m) time and outputs x_j with probability at least $2/3$.

Remark 17.13
The constant $2/3$ is arbitrary and can be replaced with any constant larger than $1/2$, since the probability of getting a correct answer can be amplified by repetition.

Notice, local decoding may be useful in applications of ECC's that have nothing to do with hardness amplification. Even in context of CD storage, it seems nice if we do not to have to read the entire CD just to recover one bit of x.

Using a local decoder, we can turn our intuition above of hardness amplification into a proof.

NOTE 17.11 (HIGH DIMENSIONAL GEOMETRY)
While we are normally used to geometry in two or three dimensions, we can get some intuition on error correcting codes by considering the geometry of *high dimensional spaces*. Perhaps the strongest effect of high dimension is the following: compare the cube with all sides 1 and the ball of radius 1/4. In one dimension, the ratio between their areas is $1/(1/2) = 2$, in two dimensions it is $1/(\pi 1/4^2) = 16/\pi$, while in three dimensions it is $1/(4/3\pi 1/4^3) = 48/\pi$. Note that as the number of dimension grows, this ratio grows exponentially in the number of dimensions. (Similarly for any two radii $r_1 > r_2$ the volume of the m-dimension ball of radius r_1 is exponentially larger than the volume of the r_2-radius ball.)



This intuition lies behind the existence of an error correcting code with distance 1/4 mapping n bit strings into $m = 5n$ bit strings. We can have $2^{m/5}$ codewords that are all of distance at least 1/4 from one another because, also in the Hamming distance, the volume of the radius 1/4 ball is exponentially smaller than the volume of the cube $\{0,1\}^n$. Therefore, we can "pack" $2^{m/5}$ such balls within the cube.

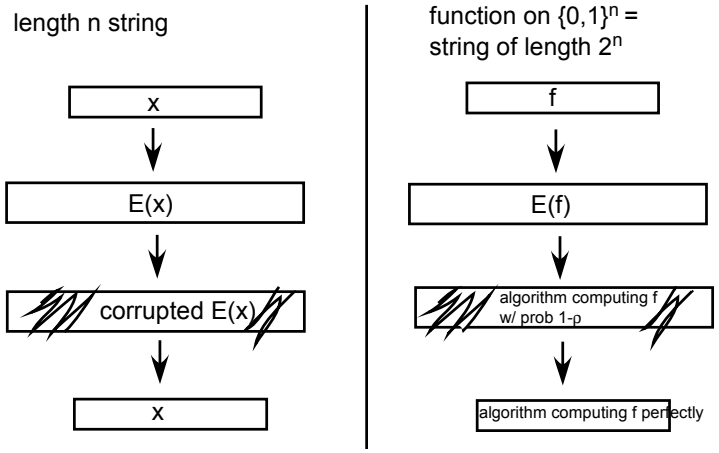


Figure 17.2: An ECC allows to map a string x to $E(x)$ such as x can be reconstructed from a corrupted version of $E(x)$. The idea is to treat a function $f : \{0,1\}^n \to \{0,1\}$ as a string in $\{0,1\}^{2^n}$, encode it using an ECC to a function $\hat{f}$. Intuitively, $\hat{f}$ should be hard on the average case if f was hard on the worst case, since an algorithm to solve $\hat{f}$ with probability $1-\rho$ could be transformed (using the ECC's decoding algorithm) to an algorithm computing f on every input.

THEOREM 17.14
Suppose that there is an ECC with polynomial-time encoding algorithm and a local decoding algorithm handling ρ errors (where ρ is a constant independent of the input length). Suppose also that there is $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for some function $S : \mathbb{N} \to \mathbb{N}$ satisfying $S(n) \geq n$. Then, there exists $\epsilon > 0$ and $g \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(g)(n) \geq S(\epsilon n)^{\epsilon}$

The proof of Theorem 17.14 follows essentially from the definition, and we will prove it for the case of a particular code later on in Theorem 17.24.

17.5 Constructions of Error Correcting Codes

We now describe some explicit functions that are error correcting codes, building up to the construction of an explicit ECC of constant distance with polynomial-time encoding and decoding. Section 17.6 describes *local decoding* algorithms for some of these codes.

17.5.1 Walsh-Hadamard Code.

For two strings $x, y \in \{0,1\}^n$, define $x \odot y$ to be the number $\sum_{i=1}^{n} x_i y_i \pmod 2$. The *Walsh-Hadamard code* is the function $\mathsf{WH} : \{0,1\}^n \to \{0,1\}^{2^n}$ that maps a string $x \in \{0,1\}^n$ into the string $z \in \{0,1\}^{2^n}$ where for every $y \in \{0,1\}^n$, the y^{th} coordinate of z is equal to $x \odot y$ (we identify $\{0,1\}^n$ with $[2^n]$ in the obvious way).

CLAIM 17.15
The function WH is an error correcting code of distance $1/2$.

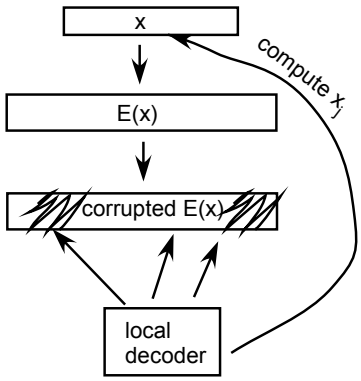


Figure 17.3: A local decoder gets access to a corrupted version of $E(x)$ and an index i and computes from it x_i (with high probability).

PROOF: First, note that WH is a linear function. By this we mean that if we take $x + y$ to be the componentwise addition of x and y modulo 2, then $\mathsf{WH}(x+y) = \mathsf{WH}(x) + \mathsf{WH}(y)$. Now, for every $x \neq y \in \{0,1\}^n$ we have that the number of 1's in the string $\mathsf{WH}(x) + \mathsf{WH}(y) = \mathsf{WH}(x+y)$ is equal to the number of coordinates on which $\mathsf{WH}(x)$ and $\mathsf{WH}(y)$ differ. Thus, it suffices to show that for every $z \neq 0^n$, at least half of the coordinates in $\mathsf{WH}(z)$ are 1. Yet this follows from the random subsum principle (Claim A.5) that says that the probability for $y \in_R \{0,1\}^n$ that $z \odot y = 1$ is exactly $1/2$. ■

17.5.2 Reed-Solomon Code

The Walsh-Hadamard code has a serious drawback: its output size is exponential in the input size. By Lemma 17.9 we know that we can do much better (at least if we're willing to tolerate a distance slightly smaller than $1/2$). To get towards explicit codes with better output, we need to make a detour to codes with *non-binary alphabet*.

DEFINITION 17.16
For every set Σ and $x, y \in \Sigma^m$, we define $\Delta(x,y) = \frac{1}{m}|\{i : x_i \neq y_i\}|$. We say that $E : \Sigma^n \to \Sigma^m$ is an *error correcting code with distance δ over alphabet Σ* if for every $x \neq y \in \Sigma^n$, $\Delta(E(x), E(y)) \geq \delta$.

Allowing a larger alphabet makes the problem of constructing codes easier. For example, every ECC with distance δ over the binary ($\{0,1\}$) alphabet automatically implies an ECC with the same distance over the alphabet $\{0,1,2,3\}$: just encode strings over $\{0,1,2,3\}$ as strings over $\{0,1\}$ in the obvious way. However, the other direction does not work: if we take an ECC over $\{0,1,2,3\}$ and transform it into a code over $\{0,1\}$ in the natural way, the distance might grow from δ to 2δ (Exercise 8).

The Reed-Solomon code is a construction of an error correcting code that can use as its alphabet any field $\mathbb{F}$:

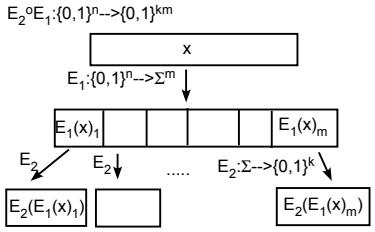


Figure 17.4: If E_1,E_2 are ECC's such that $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \sigma \to \{0,1\}^k$, then the concatenated code $E : \{0,1\}^n \to \{0,1\}^{nk}$ maps x into the sequence of blocks $E_2(E_1(x)_1), \ldots, E_2(E_1(x)_m)$.

DEFINITION 17.17
Let $\mathbb{F}$ be a field and n, m numbers satisfying $n \leq m \leq |\mathbb{F}|$. The *Reed-Solomon code* from $\mathbb{F}^n$ to $\mathbb{F}^m$ is the function $\mathsf{RS} : \mathbb{F}^n \to \mathbb{F}^m$ that on input $a_0, \ldots, a_{n-1} \in \mathbb{F}^n$ outputs the string $z_0, \ldots, z_{m-1}$ where

$$z_j = \sum_{i=0}^{n-1} a_i f_j^i$$

and f_j denotes the j^{th} element of $\mathbb{F}$ under some ordering.

LEMMA 17.18
The Reed-Solomon code $\mathsf{RS} : \mathbb{F}^n \to \mathbb{F}^m$ *has distance* $1 - \frac{n}{m}$.

PROOF: As in the case of Walsh-Hadamard code, the function RS is also linear in the sense that $\mathsf{RS}(a+b) = \mathsf{RS}(a) + \mathsf{RS}(b)$ (where addition is taken to be componentwise addition in $\mathbb{F}$). Thus, as before we only need to show that for every $a \neq 0^n$, $\mathsf{RS}(a)$ has at most n coordinates that are zero. But this immediate from the fact that a nonzero $n-1$ degree polynomial has at most n roots (see Appendix A). ■

17.5.3 Concatenated codes

The Walsh-Hadamard code has the drawback of exponential-sized output and the Reed-Solomon code has the drawback of a non-binary alphabet. We now show we can combine them both to obtain a code without neither of these drawbacks:

DEFINITION 17.19
If RS is the Reed-Solomon code mapping $\mathbb{F}^n$ to $\mathbb{F}^m$ (for some $n, m, \mathbb{F}$) and WH is the Walsh-Hadamard code mapping $\{0,1\}^{\log|\mathbb{F}|}$ to $\{0,1\}^{2^{\log|\mathbb{F}|}} = \{0,1\}^{|\mathbb{F}|}$, then the code $\mathsf{WH} \circ \mathsf{RS}$ maps $\{0,1\}^{n \log|\mathbb{F}|}$ to $\{0,1\}^{m|\mathbb{F}|}$ in the following way:

1. View RS as a code from $\{0,1\}^{n\log|\mathbb{F}|}$ to $\mathbb{F}^m$ and WH as a code from $\mathbb{F}$ to $\{0,1\}^{|\mathbb{F}|}$ using the canonical representation of elements in $\mathbb{F}$ as strings in $\{0,1\}^{\log|\mathbb{F}|}$.
2. For every input $x \in \{0,1\}^{n\log|\mathbb{F}|}$, $\mathsf{WH} \circ \mathsf{RS}(x)$ is equal to $\mathsf{WH}(\mathsf{RS}(x)_1), \ldots, \mathsf{WH}(\mathsf{RS}(x)_m)$ where $\mathsf{RS}(x)_i$ denotes the i^{th} symbol of $\mathsf{RS}(x)$.

Note that the code $\mathsf{WH} \circ \mathsf{RS}$ can be computed in time polynomial in n, m and $|\mathbb{F}|$. We now analyze its distance:

CLAIM 17.20
Let $\delta_1 = 1 - n/m$ be the distance of RS and $\delta_2 = 1/2$ be the distance of WH. Then $\mathsf{WH} \circ \mathsf{RS}$ is an ECC of distance $\delta_1\delta_2$.

PROOF: Let x, y be two distinct strings in $\{0,1\}^{\log|\mathbb{F}|n}$. If we set $x' = \mathsf{RS}(x')$ and $y' = \mathsf{RS}(y')$ then $\Delta(x', y') \geq \delta_1$. If we let x'' (resp. y'') to be the binary string obtained by applying WH to each of these blocks, then whenever two blocks are distinct, the corresponding encoding will have distance δ_2, and so $\delta(x'', y'') \geq \delta_1\delta_2$. ■

REMARK 17.21
Because for every $k \in \mathbb{N}$, there exists a finite field $|\mathbb{F}|$ of size in $[k, 2k]$ (e.g., take a prime in $[k, 2k]$ or a power of two) we can use this construction to obtain, for every n, a polynomial-time computable ECC $E : \{0,1\}^n \to \{0,1\}^{20n^2}$ of distance 0.4.

Both Definition 17.19 and Lemma 17.20 easily generalize for codes other than Reed-Solomon and Hadamard. Thus, for every two ECC's $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ their concatenation $E_2 \circ E_1$ is a code from $\{0,1\}^n$ to $\{0,1\}^{mk}$ that has distance at least $\delta_1\delta_2$ where δ_1 (resp. δ_2) is the distance of E_1 (resp. E_2), see Figure 17.6. In particular, using a different binary code than WH, it is known how to use concatenation to obtain a polynomial-time computable ECC $E : \{0,1\}^n \to \{0,1\}^m$ of constant distance $\delta > 0$ such that $m = O(n)$.

17.5.4 Reed-Muller Codes.

Both the Walsh-Hadamard and and the Reed-Solomon code are special cases of the following family of codes known as Reed-Muller codes:

DEFINITION 17.22 (REED-MULLER CODES)
Let $\mathbb{F}$ be a finite field, and let ℓ, d be numbers with $d < |\mathbb{F}|$. The *Reed Muller* code with parameters $\mathbb{F}, \ell, d$ is the function $\mathsf{RM} : \mathbb{F}^{\binom{\ell+d}{d}} \to \mathbb{F}^{|\mathbb{F}|^\ell}$ that maps every ℓ-variable polynomial P over $\mathbb{F}$ of total degree d to the values of P on all the inputs in $\mathbb{F}^\ell$.

That is, the input is a polynomial of the form

$$g(x_1, \ldots, x_\ell) = \sum_{i_1+i_2+\ldots+i_\ell \leq \ell} c_{i_1,\ldots,i_\ell} x_1^{i_1} x_2^{i_2} \cdots x_\ell^{i_\ell}$$

specified by the vector of $\binom{\ell+d}{d}$ coefficients $\{c_{i_1,\ldots,i_\ell}\}$ and the output is the sequence $\{g(x_1, \ldots, x_\ell)\}$ for every $x_1, \ldots, x_\ell \in \mathbb{F}$.

Setting $\ell = 1$ one obtains the Reed-Solomon code (for $m = |\mathbb{F}|$), while setting $d = 1$ and $\mathbb{F} = \mathrm{GF}(2)$ one obtains a slight variant of the Walsh-Hadamard code. (I.e., the code that maps every $x \in \{0,1\}^n$ into the $2 \cdot 2^n$ long string z such that for every $y \in \{0,1\}^n$,$a \in \{0,1\}$, $z_{y,a} = x \odot y + a \pmod 2$.)

The Schwartz-Zippel Lemma (Lemma A.25 in Appendix A) shows that the Reed-Muller code is an ECC with distance $1 - d/|\mathbb{F}|$. Note that this implies the previously stated bounds for the Walsh-Hadamard and Reed-Solomon codes.

17.5.5 Decoding Reed-Solomon.

To actually use an error correcting code to store and retrieve information, we need a way to efficiently *decode* a data x from its encoding $E(x)$ even if $E(x)$ has been corrupted in a fraction ρ of its coordinates. We now show this for the Reed-Solomon code, that treats x as a polynomial g, and outputs the values of this polynomial on m inputs.

We know (see Theorem A.24 in Appendix A) that a univariate degree d polynomial can be interpolated from any $d+1$ values. Here we consider a *robust* version of this procedure, whereby we wish to recover the polynomial from m values of which ρm are "faulty" or "noisy".

Let $(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)$ be a sequence of (point, value) pairs. We say that a degree d polynomial $g(x)$ *describes* this (a_i, b_i) if $g(a_i) = b_i$.

We are interested in determining if there is a degree d polynomial g that describes $(1-\rho)m$ of the pairs. If $2\rho m > d$ then this polynomial is unique (exercise). We desire to recover it, in other words, find a degree d polynomial g such that

$$g(a_i) = b_i \qquad \text{for at } (1-\rho)m \text{ least values of } i. \tag{8}$$

The apparent difficulty is in identifying the noisy points; once those points are identified, we can recover the polynomial.

Randomized interpolation: the case of $\rho < 1/(d+1)$

If ρ is very small, say, $\rho < 1/(2d)$ then we can actually use the standard interpolation technique: just select $d+1$ points at random from the set $\{(a_i, b_i)\}$ and use them to interpolate. By the union bound, with probability at least $1 - \rho(d+1) > 0.4$ all these points will be non-corrupted and so we will recover the correct polynomial. (Because the correct polynomial is unique, we can verify that we have obtained it, and if unsuccessful, try again.)

Berlekamp-Welch Procedure: the case of $\rho < (m-d)/(2m)$

The Berlekamp-Welch procedure works when the error rate ρ is bounded away from $1/2$; specifically, $\rho < (m-d)/(2m)$. For concreteness, assume $m = 4d$ and $\rho = 1/4$.

1. We claim that if the polynomial g exists then there is a degree $2d$ polynomial $c(x)$ and a degree d nonzero polynomial $e(x)$ such that

$$c(a_i) = b_i e(a_i) \qquad \text{for all } i. \tag{9}$$

   The reason is that the desired $e(x)$ can be any nonzero degree d polynomial whose roots are precisely the a_i's for which $g(a_i) \neq b_i$, and then just let $c(x) = g(x)e(x)$. (Note that this is just an *existence* argument; we do not know g yet.))

2. Let $c(x) = \sum_{i \leq 2d} c_i x^i$ and $e(x) = \sum_{i \leq d} e_i x^i$. The e_i's and c_i's are our unknowns, and these satisfy $4d$ linear equations given in (**??**), one for each a_i. The number of unknowns is $3d+2$, and our existence argument in part 1 shows that the system is feasible. Solve it using Gaussian elimination to obtain a candidate c, e.

3. Let c, e are *any* polynomials obtained in part 2. Since they satisfy (9) and $b_i = g(a_i)$ for at least $3d$ values of i, we conclude that

$$c(a_i) = g(a_i)e(a_i) \qquad \text{for at least } 3d \text{ values of } i.$$

Hence $c(x) - g(x)e(x)$ is a degree $2d$ polynomial that has at least $3d$ roots, and hence is identically zero. Hence e divides c and that in fact $c(x) = g(x)e(x)$.

4. Divide c by e to recover g.

17.5.6 Decoding concatenated codes.

Decoding concatenated codes can be achieved through the natural algorithm. Recall that if $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ are two ECC's then $E_2 \circ E_1$ maps every string $x \in \{0,1\}^n$ to the string $E_2(E_1(x)_1) \cdots E_2(E_1(x)_n)$. Suppose that we have a decoder for E_1 (resp. E_2) that can handle ρ_1 (resp. ρ_2) errors. Then, we have a decoder for $E_2 \circ E_1$ that can handle $\rho_2\rho_1$ errors. The decoder, given a string $y \in \{0,1\}^{mk}$ composed of m blocks $y_1, \ldots, y_m \in \{0,1\}^k$, first decodes each block y_i to a symbol z_i in Σ, and then uses the decoder of E_1 to decode $z_1, \ldots, z_m$. The decoder can indeed handle $\rho_1\rho_2$ errors since if $\Delta(y, E_2 \circ E_1(x)) \leq \rho_1\rho_2$ then at most ρ_1 of the blocks of y are of distance at least ρ_2 from the corresponding block of $E_2 \circ E_1(x)$.

17.6 Local Decoding of explicit codes.

We now show *local decoder* algorithm (c.f. Definition 17.12) for several explicit codes.

17.6.1 Local decoder for Walsh-Hadamard.

The following is a two-query local decoder for the Walsh-Hadamard code that handles ρ errors for every $\rho < 1/4$. This fraction of errors we handle is best possible, as it can be easily shown that there cannot exist a local (or non-local) decoder for a binary code handling ρ errors for every $\rho \geq 1/4$.

WALSH-HADAMARD LOCAL DECODER for $\rho < 1/4$:

Input: $j \in [n]$, random access to a function $f : \{0,1\}^n \to \{0,1\}$ such that $\Pr_y[g(y) \neq x \odot y] \leq \rho$ for some $\rho < 1/4$ and $x \in \{0,1\}^n$.

Output: A bit $b \in \{0,1\}$. *(Our goal: $x_j = b$.)*

Operation: Let e^j be the vector in $\{0,1\}^n$ that is equal to 0 in all the coordinates except for the j^{th} and equal to 1 on the j^{th} coordinate. The algorithm chooses $y \in_R \{0,1\}^n$ and outputs $f(y) + f(y + e^j) \pmod 2$ (where $y + e^j$ denotes componentwise addition modulo 2, or equivalently, flipping the j^{th} coordinate of y).

Analysis: Since both y and $y+e^j$ are uniformly distributed (even though they are dependent), the union bound implies that with probability $1-2\rho$, $f(y) = x \odot y$ and $f(y+e^j) = x \odot (y+e^j)$. But by the bilinearity of the operation $\odot$, this implies that $f(y)+f(y+e^j) = x \odot y + x \odot (y+e^j) =$

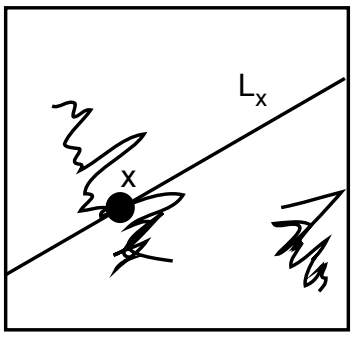


Figure 17.5: Given access to a corrupted version of a polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$, to compute $P(x)$ we pass a random line L_x through x, and use Reed-Solomon decoding to recover the restriction of P to the line L_x.

$2(x \odot y) + x \odot e^j = x \odot e^j \pmod 2$. Yet, $x \odot e^j = x_j$ and so with probability $1 - 2\rho$, the algorithm outputs the right value.

REMARK 17.23
This algorithm can be modified to locally compute not just $x_i = x \odot e^j$ but in fact the value $x \odot z$ for every $z \in \{0,1\}^n$. Thus, we can use it to compute not just every bit of the original message x but also every bit of the uncorrupted codeword $\mathsf{WH}(x)$. This property is sometimes called the *self correction property* of the Walsh-Hadamard code.

17.6.2 Local decoder for Reed-Muller

We now show a local decoder for the Reed-Muller code. (Note that Definition 17.12 can be easily extended to the case of codes, such as Reed-Muller, that use *non-binary* alphabet.) It runs in time polynomial in ℓ and d, which, for an appropriate setting of the parameters, is polylogarithmic in the output length of the code. **Convention:** Recall that the input to a Reed-Muller code is an ℓ-variable d-degree polynomial P over some field $\mathbb{F}$. When we discussed the code before, we assumed that this polynomial is represented as the list of its coefficients. However, below it will be more convenient for us to assume that the polynomial is represented by a list of its values on its first $\binom{d+\ell}{\ell}$ inputs according to some canonical ordering. Using standard interpolation, we still have a polynomial-time encoding algorithm even given this representation. Thus, it suffices to show an algorithm that, given access to a corrupted version of P, computes $P(x)$ for every $x \in \mathbb{F}^\ell$

REED-MULLER LOCAL DECODER for $\rho < (1 - d/|\mathbb{F}|)/4 - 1/|\mathbb{F}|$.

Input: A string $x \in \mathbb{F}^\ell$, random access to a function f such that $\Pr_{x \in \mathbb{F}^\ell}[P(x) \neq f(x)] < \rho$, where $P : \mathbb{F}^\ell \to \mathbb{F}$ is an ℓ-variable degree-d polynomial.

Output: $y \in \mathbb{F}$ *(Goal: $y = P(x)$.)*

Operation: 1. Let L_x, be a random line passing through x. That is $L_x = \{x + ty : t \in \mathbb{F}\}$ for a random $y \in \mathbb{F}^\ell$.

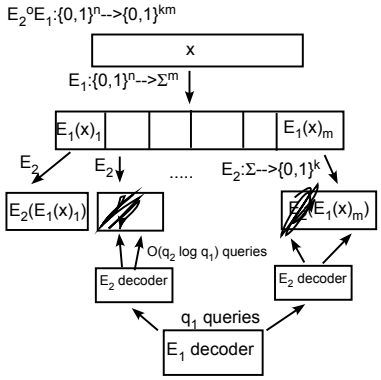


Figure 17.6: To locally decode a concatenated code $E_2 \circ E_1$ we run the decoder for E_1 using the decoder for E_2. The crucial observation is that if y is within $\rho_1\rho_2$ distance to $E_2 \circ E_1(x)$ then at most a ρ_1 fraction of the blocks in y are of distance more than ρ_2 the corresponding block in $E_2 \circ E_1(x)$.

2. Query f on all the $|\mathbb{F}|$ points of L_x to obtain a set of points $\{(t, f(x+ty))\}$ for every $t \in \mathbb{F}$.
3. Run the Reed-Solomon decoding algorithm to obtain the univariate polynomial $Q : \mathbb{F} \to \mathbb{F}$ such that $Q(t) = f(x+ty)$ for the largest number of t's (see Figure 17.5).[3]
4. Output $Q(0)$.

Analysis: For every d-degree ℓ-variable polynomial P, the univariate polynomial $Q(t) = P(x+ty)$ has degree at most d. Thus, to show that the Reed-Solomon decoding works, it suffices to show that with probability at least $1/2$, the number of points on $z \in L_x$ for which $f(z) \neq P(z)$ is less than $(1 - d/|\mathbb{F}|)/2$. Yet, for every $t \neq 0$, the point $x + ty$ is uniformly distributed (independently of x), and so the expected number of points on L_x for which f and P differ is at most $\rho|\mathbb{F}| + 1$. By Markov inequality, the probability that there will be more than $2\rho|\mathbb{F}| + 2 < (1 - d/|\mathbb{F}|)|\mathbb{F}|/2$ such points is at most $1/2$ and hence Reed-Solomon decoding will be successful with probability $1/2$. In this case, we obtain the correct polynomial q that is the restriction of Q to the line L_x and hence $q(0) = P(x)$.

17.6.3 Local decoding of concatenated codes.

Given two locally decodable ECC's E_1 and E_2, we can locally decode their concatenation $E_1 \circ E_2$ by the natural algorithm. Namely, we run the decoder for E_1, but answer its queries using the decoder for E_2 (see Figure 17.6).

LOCAL DECODER FOR CONCATENATED CODE: $\rho < \rho_1\rho_2$

The code: If $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ are codes with decoders of q_1 (resp. q_2) queries with respect to ρ_1 (resp. ρ_2) errors, let $E = E_2 \circ E_1$ be the concatenated code mapping $\{0,1\}^n$ to $\{0,1\}^{mk}$.

[3] If ρ is sufficiently small, (e.g., $\rho < 1/(10d)$), then we can use the simpler randomized Reed-Solomon decoding procedure described in Section 17.5.5.

Input: An index $i \in [n]$, random access to a string $y \in \{0,1\}^{km}$ such that $\Delta(y, E_1 \circ E_2(x)) < \rho_1\rho_2$ for some $x \in \{0,1\}^n$.

Output: $b \in \{0,1\}^n$ *(Goal: $b = x_i$)*

Operation: Simulate the actions of the decoder for E_1, whenever the decoder needs access to the j^{th} symbol of $E_1(x)$, use the decoder of E_2 with $O(q_2 \log q_1 \log |\Sigma|)$ queries applied to the j^{th} block of y to recover all the bits of this symbol with probability at least $1 - 1/(2q_1)$.

Analysis: The crucial observation is that at most a ρ_1 fraction of the length k blocks in y can be of distance more than ρ_2 from the corresponding blocks in $E_2 \circ E_1(x)$. Therefore, with probability at least 0.9, all our q_1 answers to the decoder of E_1 are consistent with the answer it would receive when accessing a string that is of distance at most ρ_1 from a codeword of E_1.

17.6.4 Putting it all together.

We now have the ingredients to prove our second main theorem of this chapter: transformation of a hard-on-the-worst-case function into a function that is "mildly" hard on the average case.

THEOREM 17.24 (WORST-CASE HARDNESS TO MILD HARDNESS)
Let $S : \mathbb{N} \to \mathbb{N}$ and $f \in \mathbf{E}$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n. Then there exists a function $g \in \mathbf{E}$ and a constant $c > 0$ such that $\mathsf{H}^{0.99}_{\mathsf{avg}}(g)(n) \geq S(n/c)/n^c$ for every sufficiently large n.

PROOF: For every n, we treat the restriction of f to $\{0,1\}^n$ as a string $f' \in \{0,1\}^N$ where $N = 2^n$. We then encode this string f' using a suitable error correcting code $E : \{0,1\}^N \to \{0,1\}^{N^C}$ for some constant $C > 1$. We will define the function g on every input $x \in \{0,1\}^{Cn}$ to output the x^{th} coordinate of $E(f')$.[4] For the function g to satisfy the conclusion of the theorem, all we need is for the code E to satisfy the following properties:

1. For every $x \in \{0,1\}^N$, $E(x)$ can be computed in $\mathrm{poly}(N)$ time.
2. There is a local decoding algorithm for E that uses $\mathrm{polylog}(N)$ running time and queries and can handle a 0.01 fraction of errors.

But this can be achieved using a concatenation of a Walsh-Hadamard code with a Reed-Muller code of appropriate parameters:

1. Let RM denote the Reed-Muller code with the following parameters:
   - The field $\mathbb{F}$ is of size $\log^5 N$.
   - The number of variables ℓ is equal to $\log N / \log\log N$.

[4] By padding with zeros as necessary, we can assume that all the inputs to g are of length that is a multiple of C.

- The degree is equal to $\log^2 N$.

RM takes an input of length at least $(\frac{d}{\ell})^\ell > N$ (and so using padding we can assume its input is $\{0,1\}^n$). Its output is of size $|\mathbb{F}|^\ell \leq \text{poly}(n)$. Its distance is at least $1 - 1/\log N$.

2. Let WH denote the Walsh-Hadamard code from $\{0,1\}^{\log F} = \{0,1\}^{5 \log\log N}$ to $\{0,1\}^{|\mathbb{F}|} = \{0,1\}^{\log^5 N}$.

Our code will be WH$\circ$RM. Combining the local decoders for Walsh-Hadamard and Reed-Muller we get the desired result. ■

Combining Theorem 17.24 with Yao's XOR Lemma (Theorem 17.2), we get the following corollary:

COROLLARY 17.25
Let $S : \mathbb{N} \to \mathbb{N}$ and $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n. Then, there exists an $S(\sqrt{\ell})^\epsilon$-pseudorandom generator for some constant $\epsilon > 0$.

PROOF: By Theorem 17.24, under this assumption there exists a function $g \in \mathbf{E}$ with $\mathsf{H}^{0.99}_{\mathsf{avg}}(g)(n) \geq S'(n) = S(n)/\text{poly}(n)$, where we can assume $S'(n) \geq \sqrt{S(n)}$ for sufficiently large n (otherwise S is polynomial and the theorem is trivial). Consider the function $g^{\oplus k}$ where $k = c \log S'(n)$ for a sufficiently small constant c. By Yao's XOR Lemma, on inputs of length kn, it cannot be computed with probability better than $1/2 + 2^{-cS'(n)/1000}$ by circuits of size $S'(n)$. Since $S(n) \leq 2^n$, $kn < \sqrt{n}$, and hence we get that $\mathsf{H}_{\mathsf{avg}}(g^{\oplus k}) \geq S^{c/2000}$. ■

As already mentioned, this implies the following corollaries:

1. If there exists $f \in \mathbf{E}$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq 2^{n^{\Omega(1)}}$ then **BPP** $\subseteq$ **QuasiP**.
2. If there exists $f \in \mathbf{E}$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq n^{\omega(1)}$ then **BPP** $\subseteq$ **SUBEXP**.

However, Corollary 17.25 is still not sufficient to show that **BPP** $=$ **P** under any assumption on the worst-case hardness of some function in **E**. It only yields an $S(\sqrt{\ell})^{\Omega(1)}$-pseudorandom generator, while what we need is an $S(\Omega(\ell))^{\Omega(1)}$-pseudorandom generator.

17.7 List decoding

Our approach to obtain stronger worst-case to average-case reduction will be to bypass the XOR Lemma, and use error correcting codes to get directly from worst-case hardness to a function that is hard to compute with probability slightly better than 1/2. However, this idea seems to run into a fundamental difficulty: if f is worst-case hard, then it seems hard to argue that the encoding of f, *under any error correcting code* is hard to compute with probability 0.6. The reason is that any error-correcting code has to have distant at most 1/2, which implies that there is no decoding algorithm that can recover x from $E(x)$ if the latter was corrupted in more than a 1/4 of its locations. Indeed, in this case there is not necessarily a unique codeword closest to the corrupted word. For example, if $E(x)$ and $E(x')$ are two codewords of distance 1/2, let y be the string that is equal to

$E(x)$ on the first half of the coordinates and equal to $E(x')$ on the second half. Given y, how can a decoding algorithm know whether to return x or x'?

This seems like a real obstacle, and indeed was considered as such in many contexts where ECC's were used, until the realization of the importance of the following insight: "If y is obtained by corrupting $E(x)$ in, say, a 0.4 fraction of the coordinates (where E is some ECC with good enough distance) then, while there may be more than one codeword within distance 0.4 to y, *there can not be too many such codewords*."

THEOREM 17.26 (JOHNSON BOUND)
If $E : \{0,1\}^n \to \{0,1\}^m$ is an ECC with distance at least $1/2 - \epsilon$, then for every $x \in \{0,1\}^m$, and $\delta \geq \sqrt{\epsilon}$, there exist at most $1/(2\delta^2)$ vectors $y_1, \ldots, y_\ell$ such that $\Delta(x, y_i) \leq 1/2 - \delta$ for every $i \in [\ell]$.

PROOF: Suppose that $x, y_1, \ldots, y_\ell$ satisfy this condition, and define ℓ vectors $z_1, \ldots, z_\ell$ in $\mathbb{R}^m$ as follows: for every $i \in [\ell]$ and $k \in [m]$, set $z_{i,k}$ to equal $+1$ if $y_k = x_k$ and set it to equal -1 otherwise. Under our assumptions, for every $i \in [\ell]$,

$$\sum_{k=1}^{m} z_{i,k} \geq 2\delta m \,, \tag{10}$$

since z_i agrees with x on an $1/2 + \delta$ fraction of its coordinates. Also, for every $i \neq j \in [\ell]$,

$$\langle z_i, z_j \rangle = \sum_{k=1}^{m} z_{i,k} z_{j,k} \leq 2\epsilon m \leq 2\delta^2 m \tag{11}$$

since E is a code of distance at least $1/2 - \epsilon$. We will show that (10) and (11) together imply that $\ell \leq 1/(2\delta^2)$.

Indeed, set $w = \sum_{i=1}^{\ell} z_i$. On one hand, by (11)

$$\langle w, w \rangle = \sum_{i=1}^{\ell} \langle z_i, z_i \rangle + \sum_{i \neq j} \langle z_i, z_j \rangle \leq \ell m + \ell^2 2\delta^2 m \,.$$

On the other hand, by (10), $\sum_k w_k = \sum_{i,j} z_{i,j} \geq 2\delta m \ell$ and hence

$$\langle w, w \rangle \geq |\sum_k w_k|^2 / m \geq 4\delta^2 m \ell^2 \,,$$

since for every c, the vector $w \in \mathbb{R}^m$ with minimal two-norm satisfying $\sum_k w_k = c$ is the uniform vector $(c/m, c/m, \ldots, c/m)$. Thus $4\delta^2 m\ell^2 \leq \ell m + 2\ell^2\delta^2 m$, implying that $\ell \leq 1/(2\delta^2)$. ■

17.7.1 List decoding the Reed-Solomon code

In many contexts, obtaining a list of candidate messages from a corrupted codeword can be just as good as unique decoding. For example, we may have some outside information on which messages are likely to appear, allowing us to know which of the messages in the list is the correct one.

However, to take advantage of this we need an efficient algorithm that computes this list. Such an algorithm was discovered in 1996 by Sudan for the popular and important Reed-Solomon code. It can recover a polynomial size list of candidate codewords given a Reed-Solomon codeword that was corrupted in up to a $1 - 2\sqrt{d/|\mathbb{F}|}$ fraction of the coordinates. Note that this tends to 1 as $|\mathbb{F}|/d$ grows, whereas the Berlekamp-Welch unique decoding algorithm of Section 17.5.5 gets "stuck" when the fraction of errors surpasses $1/2$.

On input a set of data points $\{(a_i, b_i)\}_{i=1}^m$ in $\mathbb{F}^2$, Sudan's algorithm returns *all* degree d polynomials g such that the number of i's for which $g(a_i) = b_i$ is at least $2\sqrt{d/|\mathbb{F}|}m$. It relies on the following observation:

LEMMA 17.27
For every set of m data pairs $(a_1, b_1), \ldots, (a_m, b_m)$, there is a bivariate polynomial $Q(z, x)$ of degree at most $\lceil\sqrt{m}\rceil + 1$ in z and x such that $Q(b_i, a_i) = 0$ for each $i = 1, \ldots, m$. Furthermore, there is a polynomial-time algorithm to construct such a Q.

PROOF: Let $k = \lceil\sqrt{m}\rceil + 1$. Then the unknown bivariate polynomial $Q = \sum_{i=0}^k \sum_{j=0}^k Q_{ij} z^i x^j$ has $(k+1)^2$ coefficients and these coefficients are required to satisfy m linear equations of the form:

$$\sum_{i=0}^{k} \sum_{j=0}^{k} Q_{ij} (b_t)^i (a_t)^j \qquad \text{for } t = 1, 2, \ldots, m.$$

Note that the a_t's, b_t's are known and so we can write down these equations.

Since the system is homogeneous and the number of unknowns exceeds the number of constraints, it has a nonzero solution. Furthermore this solution can be found in polynomial time. ■

LEMMA 17.28
Let d be any integer and $k > (d+1)(\lceil\sqrt{m}\rceil + 1)$. If $p(x)$ is a degree d polynomial that describes k of the data pairs, then $z - p(x)$ divides the bivariate polynomial $Q(z, x)$ described in Lemma 17.27.

PROOF: By construction, $Q(b_t, a_t) = 0$ for every data pair (a_t, b_t). If $p(x)$ describes this data pair, then $Q(p(a_t), a_t) = 0$. We conclude that the univariate polynomial $Q(p(x), x)$ has at least k roots, whereas its degree is $d(\lceil\sqrt{n}\rceil + 1) < k$. Hence $Q(p(x), x) = 0$. By the division algorithm for polynomials, $Q(p(x), x)$ is exactly the remainder when $Q(z, x)$ is divided by $(z - p(x))$. We conclude that $z - p(x)$ divides $Q(z, x)$. ■

Now it is straightforward to describe Sudan's list decoding algorithm. First, find $Q(z, x)$ by the algorithm of Lemma 17.27. Then, factor it using a standard algorithm for bivariate factoring (see [**?**]). For every factor of the form $(z - p(x))$, check by direct substitution whether or not $p(x)$ describes $2\sqrt{d/|\mathbb{F}|}m$ data pairs. Output all such polynomials.

17.8 Local list decoding: getting to BPP = P.

Analogously to Section 17.4.1, to actually use list decoding for hardness amplification, we need to provide *local* list decoding algorithms for the codes we use. Fortunately, such algorithms are known for the Walsh-Hadamard code, the Reed-Muller code, and their concatenation.

DEFINITION 17.29 (LOCAL LIST DECODER)
Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let $\rho > 0$ and q be some numbers. An algorithm L is called a *local list decoder for E handling ρ errors*, if for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ satisfying $\Delta(E(x), y) \leq \rho$, *there exists a number* $i_0 \in [poly(n/\epsilon)]$ such that for every $j \in [m]$, on inputs i_0, j and with random access to y, L runs for $\mathrm{poly}(\log(m)/\epsilon)$ time and outputs x_j with probability at least $2/3$.

REMARK 17.30
One can think of the number i_0 as the index of x in the list of $\mathrm{poly}(n/\epsilon)$ candidate messages output by L. Definition 17.29 can be easily generalized to codes with non-binary alphabet.

17.8.1 Local list decoding of the Walsh-Hadamard code.

It turns out we already encountered a local list decoder for the Walsh-Hadamard code: the proof of the Goldreich-Levin Theorem (Theorem 10.14) provided an an algorithm that given access to a "black box" that computes the function $y \mapsto x \odot y$ (for $x, y \in \{0,1\}^n$) with probability $1/2 + \epsilon$, computes a list of values $x_1, \ldots, x_{\mathrm{poly}(n/\epsilon)}$ such that $x_{i_0} = x$ for some i_0. In the context of that theorem, we could find the right value of x from that list by checking it against the value $f(x)$ (where f is a one-way permutation). This is a good example for how once we have a list decoding algorithm, we can use outside information to narrow the list down.

17.8.2 Local list decoding of the Reed-Muller code

We now present an algorithm for local list decoding of the Reed-Muller code. Recall that the codeword of this code is the list of evaluations of a d-degree ℓ-variable polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$. The local decoder for Reed-Muller gets random access to a corrupted version of P and two inputs: an index i and $x \in \mathbb{F}^\ell$. Below we describe such a decoder that runs in $\mathrm{poly}(d, \ell, |\mathbb{F}|)$ and outputs $P(x)$ with probability at least 0.9 assuming that i is equal to the "right" index i_0. **Note:** To be a valid local list decoder, given the index i_0, the algorithm should output $P(x)$ with high probability for *every* $x \in \mathbb{F}^\ell$. The algorithm described below is only guaranteed to output the right value for *most* (i.e., a 0.9 fraction) of the x's in $\mathbb{F}^\ell$. We transform this algorithm to a valid local list decoder by combining it with the Reed-Muller local decoder described in Section 17.6.2.

REED-MULLER LOCAL LIST DECODER for $\rho < 1 - 10\sqrt{d/|\mathbb{F}|}$

Inputs:
- Random access to a function f such that $\Pr_{x \in \mathbb{F}^\ell}[P(x) = f(x)] > 10\sqrt{d/|\mathbb{F}|}$ where $P : \mathbb{F}^\ell \to \mathbb{F}$ is an ℓ-variable d-degree polynomial. We assume $|\mathbb{F}| > d^4$ and that both $d > 1000$. (This can always be ensured in our applications.)
- An index $i_0 \in [|\mathbb{F}|^{\ell+1}]$ which we interpret as a pair (x_0, y_0) with $x_0 \in \mathbb{F}^\ell$, $y_0 \in \mathbb{F}$,
- A string $x \in \mathbb{F}^\ell$.

Output: $y \in \mathbb{F}$ *(For some pair (x_0, y_0), it should hold that $P(x) = y$ with probability at least 0.9 over the algorithm's coins and x chosen at random from $\mathbb{F}^\ell$.)*

Operation: 1. Let L_{x,x_0} be a random degree 3 curve passing through x, x_0. That is, we find a random degree 3 univariate polynomial $q : \mathbb{F} \to \mathbb{F}^\ell$ such that $q(0) = x$ and $q(r) = x_0$ for some random $r \in \mathbb{F}$. (See Figure 17.7.)

2. Query f on all the $|\mathbb{F}|$ points of L_{x,x_0} to obtain the set $\mathcal{S}$ of the $|\mathbb{F}|$ pairs $\{(t, f(q(t)) : t \in \mathbb{F})\}$.

3. Run Sudan's Reed-Solomon list decoding algorithm to obtain a list $g_1, \ldots, g_k$ of all degree $3d$ polynomials that have at least $8\sqrt{d|\mathbb{F}|}$ agreement with the pairs in $\mathcal{S}$.

4. If there is a unique i such that $g_i(r) = y_0$ then output $g_i(0)$. Otherwise, halt without outputting anything.

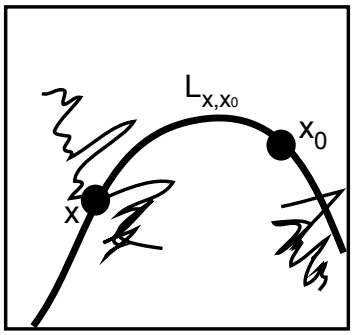


Figure 17.7: Given access to a corrupted version of a polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$ and some index (x_0, y_0), to compute $P(x)$ we pass a random degree-3 curve L_{x,x_0} through x and x_0, and use Reed-Solomon list decoding to recover a list of candidates for the restriction of P to the curve L_{x,x_0}. If only one candidate satisfies that its value on x_0 is y_0, then we use this candidate to compute $P(x)$.

We will show that for every $f : \mathbb{F}^\ell \to \mathbb{F}$ that agrees with an ℓ-variable degree d polynomial on a $10\sqrt{d/|\mathbb{F}|}$ fraction of its input, and every $x \in \mathbb{F}^\ell$, if x_0 is chosen at random from $\mathbb{F}^\ell$ and $y_0 = P(x_0)$, then with probability at least 0.9 (over the choice of x_0 and the algorithm's coins) the above decoder will output $P(x)$. By a standard averaging argument, this implies that there exist a pair (x_0, y_0) such that given this pair, the algorithm outputs $P(x)$ for a 0.9 fraction of the x's in $\mathbb{F}^\ell$.

Let $x \in \mathbb{F}^\ell$, if x_0 is chosen randomly in $\mathbb{F}^\ell$ and $y_0 = P(x_0)$ then the following

For every $x \in \mathbb{F}^\ell$, the following fictitious algorithm can be easily seen to have an identical output to the output of our decoder on the inputs x, a random $x_0 \in_R \mathbb{F}^\ell$ and $y_0 = P(x_0)$:

1. Choose a random degree 3 curve L that passes through x. That is, $L = \{q(t) : t \in \mathbb{F}\}$ where $q : \mathbb{F} \to \mathbb{F}^\ell$ is a random degree 3 polynomial satisfying $q(0) = x$.

2. Obtain the list $g_1, \ldots, g_m$ of all univariate polynomials over $\mathbb{F}$ such that for every i, there are at least $6\sqrt{d|\mathbb{F}|}$ values of t such that $g_i(t) = f(q(t))$.

3. Choose a random $r \in \mathbb{F}$. Assume that you are given the value $y_0 = P(q(r))$.

4. If there exists a unique i such that $g_i(r) = y_0$ then output $g_i(0)$. Otherwise, halt without an input.

Yet, this fictitious algorithm will output $P(x)$ with probability at least 0.9. Indeed, since all the points other than x on a random degree 3 curve passing through x are pairwise independent, Chebyshev's inequality implies that with probability at least 0.99, the function f will agree with the polynomial P on at least $8\sqrt{d|\mathbb{F}|}$ points on this curve (this uses the fact that $\sqrt{d/|\mathbb{F}|}$ is smaller than 10^{-6}). Thus the list $g_1, \ldots, g_m$ we obtain in Step 2 contains the polynomial $g : \mathbb{F} \to \mathbb{F}$ defined as $g(t) = P(q(t))$. We leave it as Exercise 9 to show that there can not be more than $\sqrt{|F|}/4d$ polynomials in this list. Since two $3d$-degree polynomials can agree on at most $3d+1$ points, with probability at least $\frac{(3d+1)\sqrt{|F|}/4d}{|\mathbb{F}|} < 0.01$, if we choose a random $r \in \mathbb{F}$, then $g(r) \neq g_i(r)$ for every $g_i \neq g$ in this list. Thus, with this probability, we will identify the polynomial g and output the value $g(0) = P(x)$. ■

17.8.3 Local list decoding of concatenated codes.

If $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ are two codes that are locally list decodable then so is the concatenated code $E_2 \circ E_1 : \{0,1\}^n \to \{0,1\}^{mk}$. As in Section 17.6.3, the idea is to simply run the local decoder for E_1 while answering its queries using the decoder of E_2. More concretely, assume that the decoder for E_1 takes an index in the set I_1, uses q_1 queries, and can handle $1 - \epsilon_1$ errors, and that I_2, q_2 and ϵ_2 are defined analogously. Our decoder for $E_2 \circ E_1$ will take a pair of indices $i_1 \in I_1$ and $i_2 \in I_2$ and run the decoder for E_1 with the index i_1, and whenever this decoder makes a query answer it using the decoder E_2 with the index i_2. (See Section 17.6.3.) We claim that this decoder can handle $1/2 - \epsilon_1\epsilon_2|I_2|$ number of errors. Indeed, if y agrees with some codeword $E_2 \circ E_1(x)$ on an $\epsilon_1\epsilon_2|I_2|$ fraction of the coordinates then there are $\epsilon_1|I_2|$ blocks on which it has at least $1/2 + \epsilon_2$ agreement with the blocks this codeword. Thus, by an averaging argument, there exists an index i_2 such that given i_2, the output of the E_2 decoder agrees with $E_1(x)$ on ϵ_1 symbols, implying that there exists an index i_1 such that given (i_1, i_2) and every coordinate j, the combined decoder will output x_j with high probability.

17.8.4 Putting it all together.

As promised, we can use local list decoding to transform a function that is merely worst-case hard into a function that cannot be computed with probability significantly better than $1/2$:

THEOREM 17.31 (WORST-CASE HARDNESS TO STRONG HARDNESS)
Let $S : \mathbb{N} \to \mathbb{N}$ and $f \in \mathbf{E}$ such that $\mathsf{H}_{wrs}(f)(n) \geq S(n)$ for every n. Then there exists a function $g \in \mathbf{E}$ and a constant $c > 0$ such that $\mathsf{H}_{avg}(g)(n) \geq S(n/c)^{1/c}$ for every sufficiently large n.

PROOF SKETCH: As in Section 17.6.4, for every n, we treat the restriction of f to $\{0,1\}^n$ as a

string $f' \in \{0,1\}^N$ where $N = 2^n$ and encode it using the concatenation of a Reed-Muller code with the Walsh-Hadamard code. For the Reed-Muller code we use the following parameters:

- The field $\mathbb{F}$ is of size $S(n)^{1/100}$. [5]
- The degree d is of size $\log^2 N$.
- The number of variables ℓ is $2 \log N / \log S(n)$.

The function g is obtained by applying this encoding to f. Given a circuit of size $S(n)^{1/100}$ that computes g with probability better than $1/2 + 1/S(n)^{1/50}$, we will be able to transform it, in $S(n)^{O(1)}$ time, to a circuit computing f perfectly. We hardwire the index i_0 to this circuit as part of its description. ■

What have we learned?

- Yao's XOR Lemma allows to amplify hardness by transforming a Boolean function with only mild hardness (cannot be computed with say 0.99 success) into a Boolean function with strong hardness (cannot be computed with 0.51 success).
- An *error correcting code* is a function that maps every two strings into a pair of strings that differ on many of their coordinates. An error correcting code with a *local decoding* algorithm can be used to transform a function hard in the worst-case into a function that is mildly hard on the average case.
- A code over the binary alphabet can have distance at most $1/2$. A code with distance δ can be uniquely decoded up to $\delta/2$ errors. *List decoding* allows to a decoder to handle almost a δ fraction of errors, at the expense of returning not a single message but a short list of candidate messages.
- We can transform a function that is merely hard in the worst case to a function that is strongly hard in the average case using the notion of *local list decoding* of error correcting codes.

Chapter notes and history

MANY ATTRIBUTIONS STILL MISSING.

Impagliazzo and Wigderson [**?**] were the first to prove that **BPP** = **P** if there exists $f \in \mathbf{E}$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq 2^{\Omega(n)}$ using a *derandomized* version of Yao's XOR Lemma. However, the presentation

[5] We assume here that $S(n) > \log N^{1000}$ and that it can be computed in $2^{O(n)}$ time. These assumptions can be removed by slightly complicating the construction (namely, executing it while guessing that $S(n) = 2^k$, and concatenating all the results.)

here follows Sudan, Trevisan, and Vadhan [**?**], who were the first to point the connection between local list decoding and hardness amplification, and gave (a variant of) the Reed-Muller local list decoding algorithm described in Section 17.8. They also showed a different approach to achieve the same result, by first showing that the NW generator and a mildly hard function can be used to obtain from a short random seed a distribution that has high *pseudoentropy*, which is then converted to a pseudorandom distribution via a randomness extractor (see Chapter 16).

The question raised in Problem 5 is treated in O'Donnell [**?**], where a hardness amplification lemma is given for **NP**. For a sharper result, see Healy, Vadhan, and Viola [**?**].

Exercises

§1 Let $X_1, \ldots, X_n$ be independent random variables such that X_i is equal to 1 with probability $1-\delta$ and equal to 0 with probability δ. Let $X = \sum_{i=1}^{k} X_i \pmod 2$. Prove that $\Pr[X = 1] = 1/2 + (1-2\delta)^k$.

Hint: Define $Y_i = (-1)^{X_i}$ and $Y = \prod_{i=1}^{k} Y_i$. Then, use the fact that the expectation of a product of independent random variables is the product of their expectations.

§2 Prove Farkas' Lemma: if $C, D \subseteq \mathbb{R}^m$ are two convex sets then there exists a vector $\mathbf{z} \in \mathbb{R}^m$ and a number $a \in \mathbb{R}$ such that

$$\mathbf{x} \in C \Rightarrow \langle \mathbf{x}, \mathbf{z} \rangle \geq a$$
$$\mathbf{y} \in D \Rightarrow \langle \mathbf{y}, \mathbf{z} \rangle \leq a$$

Hint: Start by proving this in the case that C and D are separated, which means that for some $\epsilon > 0$, $\|\mathbf{x}-\mathbf{y}\|_2 \geq \epsilon$ for every $\mathbf{x} \in C$ and $\mathbf{y} \in D$. In this case you can take $\mathbf{z}$ to be the shortest vector of the form $\mathbf{x} - \mathbf{y}$ for $\mathbf{x} \in C$ and $\mathbf{y} \in D$.

§3 Prove the Min-Max Theorem (see Note 17.7) using Farkas' Lemma.

§4 Prove the duality theorem for linear programming using Farkas' Lemma. That is, prove that for every $m \times n$ matrix A, and vectors $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^n$,

$$\max_{\substack{\mathbf{x} \in \mathbb{R}^n \text{s.t.} \\ A\mathbf{x} \leq \mathbf{b} \\ \mathbf{x} \geq \mathbf{0}}} \langle \mathbf{x}, \mathbf{c} \rangle = \min_{\substack{\mathbf{y} \in \mathbb{R}^m \text{s.t.} \\ A^\dagger \mathbf{y} \geq \mathbf{c} \\ \mathbf{y} \geq \mathbf{0}}} \langle \mathbf{y}, \mathbf{b} \rangle$$

where $A^\dagger$ denotes the transpose of A and for two vectors $\mathbf{u}, \mathbf{v}$ we say that $\mathbf{u} \geq \mathbf{v}$ if $u_i \geq v_i$ for every i.

§5 Suppose we know that **NP** contains a function that is weakly hard for all polynomial-size circuits. Can we use the XOR Lemma to infer the existence of a strongly hard function in **NP**? Why or why not?

§6 For every $\delta < 1/2$ and sufficiently large n, prove that there exists a function $E : \{0,1\}^n \to \{0,1\}^{n/(1-H(\delta))}$ that is an error correcting code with distance δ, where $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$.

Hint: Use a greedy strategy, to select the codewords of E one by one, never adding a codeword that is within distance δ to previous ones. When will you get stuck?

§7 Show that for every $E : \{0,1\}^n \to \{0,1\}^m$ that is an error correcting code of distance $1/2$, $2^n < 10\sqrt{n}$. Show if E is an error correcting code of distance $\delta > 1/2$, then $2^n < 10/(\delta - 1/2)$.

§8 Let $E : \{0,1\}^n \to \{0,1\}^m$ be a δ-distance ECC. Transform E to a code $E' : \{0,1,2,3\}^{n/2} \to \{0,1,2,3\}^{m/2}$ in the obvious way. Show that E' has distance δ. Show that the opposite direction is not true: show an example of a δ-distance ECC $E' : \{0,1,2,3\}^{n/2} \to \{0,1,2,3\}^{m/2}$ such that the corresponding binary code has distance 2δ.

§9 Let $f : \mathbb{F} \to \mathbb{F}$ be any function. Suppose integer $d \geq 0$ and number ϵ satisfy $\epsilon > 2\sqrt{\frac{d}{|\mathbb{F}|}}$. Prove that there are at most $2/\epsilon$ degree d polynomials that agree with f on at least an ϵ fraction of its coordinates.

Hint: The first polynomial describes f in an ϵ fraction of points say S_1, the second polynomial describes f in $\epsilon - d/|\mathbb{F}|$ fraction of points S_2 where $S_1 \cap S_2 = \emptyset$, etc.

§10 *(Linear codes)* We say that an ECC $E : \{0,1\}^n \to \{0,1\}^m$ is *linear* if for every $x, x' \in \{0,1\}^n$, $E(x+x') = E(x) + E(x')$ where $+$ denotes componentwise addition modulo 2. A linear ECC E can be described by an $m \times n$ matrix A such that (thinking of x as a column vector) $E(x) = Ax$ for every $x \in \{0,1\}^n$.

(a) Prove that the distance of a linear ECC E is equal to the minimum over all nonzero $x \in \{0,1\}^n$ of the fraction of 1's in $E(x)$.

(b) Prove that for every $\delta > 0$, there exists a linear ECC $E : \{0,1\}^n \to \{0,1\}^{1.1n/(1-H(\delta))}$ with distance δ, where $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$?

Hint: Use the probabilistic method - show this holds for a random matrix.

(c) Prove that for some $\delta > 0$ there is an ECC $E : \{0,1\}^n \to \{0,1\}^{\mathrm{poly}(n)}$ of distance δ with polynomial-time encoding and decoding mechanisms. (You need to know about the field GF(2^k) to solve this, see Appendix A.)

Hint: Use the concatenation of Reed-Solomon over GF(2^k) with the Walsh-Hadamard code.

(d) We say that a linear code $E : \{0,1\}^n \to \{0,1\}^m$ is ϵ-*biased* if for every non-zero $x \in \{0,1\}^n$, the fraction of 1's in $E(x)$ is between $1/2 - \epsilon$ and $1/2 + \epsilon$. Prove that for every $\epsilon > 0$, there exists an ϵ-biased linear code $E : \{0,1\}^n \to \{0,1\}^{\mathrm{poly}(n/\epsilon)}$ with a polynomial-time encoding algorithm.