Computational Complexity: A Modern Approach

Sanjeev Arora and Boaz Barak
Princeton University

http://www.cs.princeton.edu/theory/complexity/

complexitybook@gmail.com

Chapter 19

Hardness Amplification and Error Correcting Codes

Complexity theory studies the *computational hardness* of functions. In this chapter we are interested in functions that are hard to compute on the "average" instance, continuing a topic that played an important role in Chapters 9 and 18, and will do so again in Chapter 20. The special focus in this chapter is on techniques for *amplifying* hardness, which is useful in a host of contexts. In *cryptography* (see Chapter 9), hard functions are necessary to achieve secure encryption schemes of non-trivial key size. Many conjectured hard functions like factoring are only hard on a few instances, not all. Thus these functions do not suffice for some cryptographic applications, but via hardness amplification we can turn them into functions that do suffice. Another powerful application will be shown in Chapter 20— derandomization of the class **BPP** under worst-case complexity theoretic assumptions. Figure 19.1 contains a schematic view of this chapter's sections and the way their results are related to that result. In addition to their applications in complexity theory, the ideas covered in this chapter have had other uses, including new constructions of error-correcting codes and new algorithms in machine learning.

For simplicity we study hardness amplification in context of Boolean functions though this notion can apply to functions that are not Boolean-valued. Section 19.1 introduces the first technique for hardness amplification, namely, *Yao's XOR Lemma*. It allows us to turn weakly hard functions into strongly hard functions. Roughly speaking, a Boolean function f is said to be *weakly hard* if every moderate-sized circuit fails to compute it on some nonnegligible fraction of inputs, say 0.01 fraction. The function is *strongly hard* if every moderate-sized circuit fails to compute it on almost *half* the inputs, say $1/2 - \epsilon$ fraction of inputs. (Note that every Boolean function can be computed correctly on at least half the inputs by a trivial circuit, namely one that always outputs 1 or always outputs 0.) The Section describes a way to transform every function using a simple "XOR" construction that does not greatly increase the complexity of computing it but has the property that if the function we started with was weakly hard then it becomes strongly hard. This construction is very useful in cryptographic applications, as mentioned in Chapter 9.

We then turn our attention to a different technique for hardness amplification that produces strongly hard functions starting with functions that are merely guaranteed to be hard in the *worst case*. This is highly non-trivial as there is often quite a difference between the worst-case and average-case complexity of computational problems. (For example, while finding the smallest factor of a given integer seems difficult in general, it's trivial to do for half the integers— namely, the even ones.) The main tool we use is *error correcting codes*. We review the basic definition and constructions in sections 19.2 and 19.3, while Section 19.4 covers *local decoding* which is the main notion needed to apply error-correcting codes in our setting. As a result we obtain a way to transform every function f that is hard in the worst case into a function $\hat{f}$ that is mildly hard in the average case.

Combining the transformation of Section 19.4 with Yao's XOR Lemma of Section 19.1, we are able to get functions that are extremely hard on the average case from functions that are

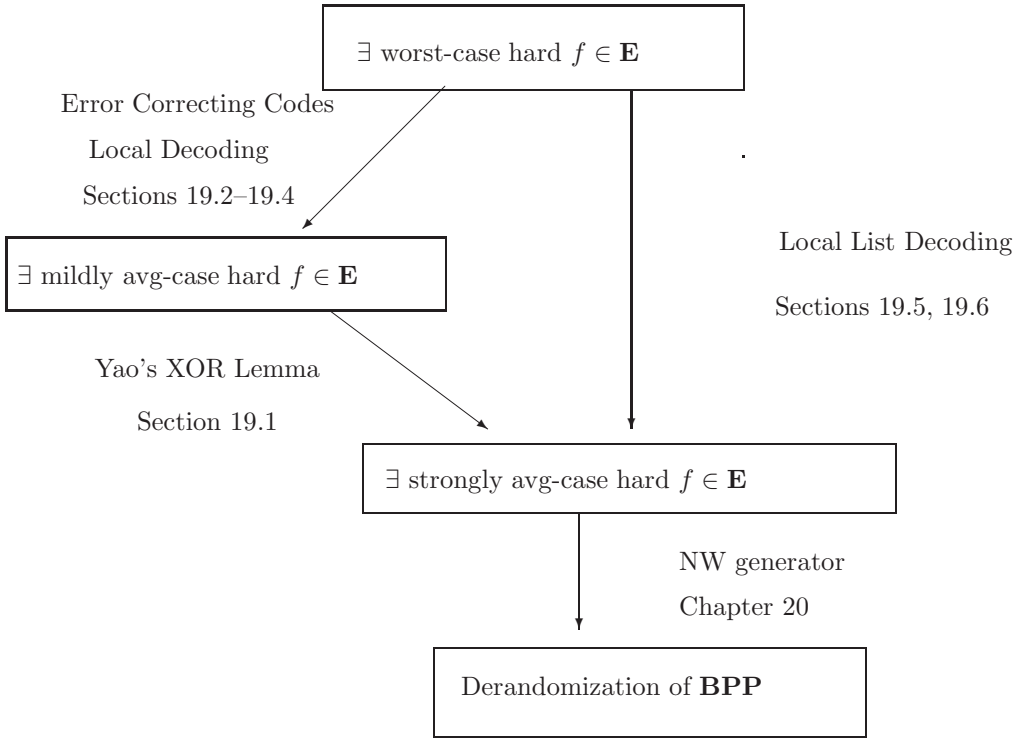


Figure 19.1 Organization of Chapter 19

only hard on the worst case. Alas, quantitatively speaking the above transformation is not optimal, in the sense that even if the original function was worst-case hard for exponential sized (i.e. size $2^{\Omega(n)}$) circuits, we are only able to guarantee that the transformed function will only be hard in the average case for sub-exponential sized (i.e., size $2^{n^{\Omega(1)}}$) circuits. In Sections 19.5 and 19.6 we show a stronger result, that transforms in one fell swoop a function f that is hard on the worst case to a function $\hat{f}$ that is *extremely hard* on the average case. This transformation uses error correcting codes in a more sophisticated way, via an independently interesting notion called *list decoding*. List decoding is covered in Section 19.5 while Section 19.6 describes *local list decoding* which is the extension of list decoding needed for our purposes.

Readers who are familiar with the theory of error-correcting codes can skim through Sections 19.2 and 19.3 in a first reading (pausing to remind themselves of the Reed-Solomon and Reed-Muller codes in Definitions 19.10 and 19.12 and their associated decoding algorithms) and go on to Section 19.4.

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

Yao's XOR Lemma transforms 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). To state the Lemma we need to define precisely the meaning of worst-case hardness and average-case hardness of a function:

Definition 19.1 *(Average-case and worst-case hardness)*
For $f : \{0,1\}^n \to \{0,1\}$ and $\rho \in [0,1]$ we define the *ρ-average case hardness of f*, denoted $\mathsf{H}^{\rho}_{\mathsf{avg}(}(f)$, to be the largest S such that for every circuit C of size at most S, $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < \rho$. For an infinite $f : \{0,1\}^* \to \{0,1\}$, we let $\mathsf{H}^{\rho}_{\mathsf{avg}(}(f)(n)$ denote $\mathsf{H}^{\rho}_{\mathsf{avg}(}(f_n)$ where f_n is the restriction of f to $\{0,1\}^n$.

We define the *worst-case hardness* of f, denoted $\mathsf{H}_{\mathsf{wrs}(}(f)$, to equal $\mathsf{H}^{1}_{\mathsf{avg}(}(f)$ and define the *average-case hardness of f*, denoted $\mathsf{H}_{\mathsf{avg}(}(f)$, to equal $\max\{S : \mathsf{H}^{1/2+1/S}_{\mathsf{avg}}(f) \geq S\}$. That is, $\mathsf{H}_{\mathsf{avg}(}(f)$ is the largest number S such that $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < 1/2 + 1/S$ for every Boolean circuit C on n inputs with size at most S.

Note that for every function $f : \{0,1\}^n \to \{0,1\}$, $\mathsf{H}_{\mathsf{avg}(}(f) \leq \mathsf{H}_{\mathsf{wrs}(}(f) \leq O(2^n/n)$ (see Exercise 6.1). This definition of average-case hardness is tailored to the application of derandomization, and in particular only deals with the uniform distribution over the inputs. See Chapter 18 for a more general treatment of average-case complexity. We can now state Yao's lemma:

Theorem 19.2 *(Yao's XOR Lemma* [Yao82a]*)*
For every $f : \{0,1\}^n \to \{0,1\}$, $\delta > 0$ and $k \in \mathbb{N}$, if $\epsilon > 2(1-\delta)^k$ then

$$\mathsf{H}^{1/2+\epsilon}_{\mathsf{avg}}(f^{\oplus k}) \geq \tfrac{\epsilon^2}{400n}\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f)\,,$$

where $f^{\oplus k} : \{0,1\}^{nk} \to \{0,1\}$ is defined by $f^{\oplus k}(x_1,\ldots,x_k) = \sum_{i=1}^{k} f(x_i) \pmod 2$.

Yao's Lemma says that if small circuits cannot compute f with probability better than $1-\delta$ then somewhat smaller circuits cannot compute $f^{\oplus k}$ with probability better than $1/2 + 2(1-\delta)^k$. Intuitively, it makes sense that if you can only compute f on a $1-\delta$ fraction of the inputs, then given a random k tuple $x_1,\ldots,x_k$, unless all of these k inputs fall into this "good set" of inputs (which happens with probability $(1-\delta)^k$), you will have to guess the answer to $\sum_{i=1}^{k} f(x_i) \pmod 2$ at random and be successful with probability at most $1/2$; see also Exercise 19.1. But making this intuition into a proof takes some effort. The main step is the following beautiful result of Impagliazzo.

Lemma 19.3 *(Impagliazzo's Hardcore Lemma* [Imp95b]*) Say that a distribution H over $\{0,1\}^n$ has density δ if for every $x \in \{0,1\}^*$, $\Pr[H = x] \leq 1/(\delta 2^n)$. For every $\delta > 0$, $f : \{0,1\}^n \to \{0,1\}$, and $\epsilon > 0$, if $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$ then there exists a density-δ distribution H such that for every circuit C of size at most $\frac{\epsilon^2 S}{100n}$,*

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

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 (different circuits make mistakes on different inputs), and the function 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, it is possible to transform it into a result on sets, see Exercise 19.2.)

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

We now show how to use Impagliazzo's Hardcore Lemma (Lemma 19.3) to prove Yao's XOR Lemma (Theorem 19.2). Let $f : \{0,1\}^n \to \{0,1\}$ be a function such that $\mathsf{H}_{\mathsf{avg}}^{1-\delta}(f) \geq S$, let $k \in \mathbb{N}$ and suppose, towards a contradiction, that there is a circuit C of size $S' = \frac{\epsilon^2}{400n}S$ 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 $\epsilon > 2(1-\delta)^k$. We will first prove the lemma for the case $k = 2$ and then indicate how the proof can be generalized for every k.

Let H be the hardcore density-δ distribution obtained from Lemma 19.3, on which every S'-sized circuit fails to compute f with probability better than $1/2 + \epsilon/2$. We can think of the process of picking a uniform element in $\{0,1\}^n$ as follows: first toss a biased coin that comes up "Heads" with probability δ. Then, if the coin came up "Heads" then pick a random element according to H, and if it came up "Tails" pick an element according to the distribution G which is the "complement" of H. Namely, G is defined by setting $\Pr[G = x] = (2^{-n} - \delta\Pr[H = x])/(1-\delta)$. (Exercise 19.3 asks you to verify that G is indeed a distribution and that this process does indeed yield a uniform element.) 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 independent* random strings and concatenating them, then by (2) we can write

$$(U_n)^2 = (1-\delta)^2G^2 + (1-\delta)\delta GH + \delta(1-\delta)HG + \delta^2H^2\,, \tag{3}$$

where we use G^2 to denote the concatenation of two independent copies of G, GH to denote the concatenation of a string chosen from G and a string chosen independently from H, and so on.

Now for every distribution $\mathcal{D}$ over $\{0,1\}^{2n}$, let $P_{\mathcal{D}}$ be the probability of the event of the left-hand side of (1). That is, $P_{\mathcal{D}}$ is the probability that $C(x_1,x_2) = f(x_1) + f(x_2) \pmod 2$ where x_1, x_2 are chosen from $\mathcal{D}$. Combining (1) and (3) we get

$$1/2 + \epsilon \leq P_{(U_n)^2} = (1-\delta)^2P_{G^2} + (1-\delta)\delta P_{GH} + \delta(1-\delta)P_{HG} + \delta^2P_{H^2} \tag{4}$$

But since $\epsilon > 2(1-\delta)^2$ and $P_{G^2} \leq 1$, (4) implies

$$1/2 + \epsilon/2 \leq (1-\delta)\delta P_{GH} + \delta(1-\delta)P_{HG} + \delta^2P_{H^2}\,. \tag{5}$$

Since the coefficients on the right hand side of (5) sum up to less than 1, the averaging principle implies that at least one of these probabilities must be larger than the left hand side. For example, assume that $P_{HG} \geq 1/2 + \epsilon/2$ (the other cases are symmetrical). This means that

$$\Pr_{x_1\in_R H, x_2\in_R G}[C(x_1,x_2) = f(x_1) + f(x_2) \pmod 2] > 1/2 + \epsilon/2\,.$$

Thus by the averaging principle, there exists a fixed string x_2 such that

$$\Pr_{x_1\in_R H}[C(x_1,x_2) = f(x_1) + f(x_2) \pmod 2] > 1/2 + \epsilon/2\,,$$

or, equivalently,

$$\Pr_{x_1\in_R H}[C(x_1,x_2) + f(x_2) \pmod 2 = f(x_1)] > 1/2 + \epsilon/2\,.$$

But this means that we have an S'-sized circuit D (the circuit computing the mapping $x_1 \mapsto C(x_1,x_2) + f(x_2) \pmod 2$) that computes f on inputs chosen from H with probability better than $1/2 + \epsilon/2$, contradicting the fact that H is hard-core!

This completes the proof for the case $k = 2$. The proof for general k follows along the same lines, using the equation

$$(U_n)^k = (1-\delta)^k G^k + (1-\delta)^{k-1}\delta G^{k-1}H + \cdots + \delta^k H^k$$

in place of (3); we leave verifying the details to the reader as Exercise 19.4. ■

19.1.2 Proof of Impagliazzo's Lemma

We now turn to proving Impagliazzo's Hardcore Lemma (Lemma 19.3). Let f be a function with $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$ and let $\epsilon > 0$. To prove the lemma we need to show a density δ distribution H on which *every* circuit C of size $S' = \frac{\epsilon^2 S}{100n}$ cannot compute f with probability better than $1/2 + \epsilon$.

Let's think of this task as a game between two players named *Russell* and *Noam*. Noam wants to compute the function f and Russell wants Noam to fail. The game proceeds as follows: Russell first chooses a δ-density distribution H, and then Noam chooses a circuit C of size at most S'. At the game's conclusion, Russell pays Noam v dollars, where $v = \Pr_{x \in_R H}[C(x) = f(x)]$. Assume towards a contradiction that the lemma is false, and hence for every δ-density distribution H chosen by Russell, Noam can find an S'-sized circuit C on which $\Pr_{x \in_R H}[C(x) = f(x)] \geq 1/2 + \epsilon$.

Now this game is a zero-sum game, and so we can use von-Neumann's *min-max theorem* (see Note 19.4) that says that if we allow *randomized* (also known as mixed) strategies then Noam can achieve the same value even if he plays first. By randomized strategies we mean that Noam and Russell can also select arbitrary distributions over their choices. In Russell's case this makes no difference as a distribution over density-δ distributions is still a density-δ distribution.[1] However in Noam's case we need to allow him to choose a *distribution* $\mathcal{C}$ over S'-sized circuits. Our assumption, combined with the min-max theorem, means that there exists such a distribution $\mathcal{C}$ satisfying

$$\Pr_{C \in_R \mathcal{C}, x \in_R H}[C(x) = f(x)] \geq 1/2 + \epsilon \tag{6}$$

for *every* δ-density H.

Call a string $x \in \{0,1\}^n$ "bad" if $\Pr_{C \in_R \mathcal{C}}[C(x) = f(x)] < 1/2 + \epsilon$ and call x "good" otherwise. There are less than $\delta 2^n$ bad x's. Indeed, otherwise we could let H be the uniform distribution over the bad x's and it would violate (6). Now let us choose a circuit C as follows: set $t = 50n/\epsilon^2$, pick $C_1, \ldots, C_t$ independently from $\mathcal{C}$, and define $C(x)$ to equal the majority of $C_1(x), \ldots, C(x)$ for every $x \in \{0,1\}^n$. Note that the size of C is $tS' < S$. We claim that if we choose the circuit C in this way then for every good $x \in \{0,1\}^n$, $\Pr[C(x) \neq f(x)] < 2^{-n}$. Indeed, this follows by applying the Chernoff bound (see Corollary A.15). Since there are at most 2^n good x's, we can apply the union bound to deduce that there exists a size S circuit C such that $C(x) = f(x)$ for *every* good x. But since there are less than $\delta 2^n$ bad x's this implies that $\Pr_{x \in_R U_n}[C(x) = f(x)] > 1 - \delta$, contradicting the assumption that $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$. ■

Taken in the contrapositive, Lemma 19.3 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 [KS99].

[1] In fact, the set of density δ distributions can be viewed as the set of distributions over $\delta 2^n$-flat distributions, where a distribution is K-flat if it is uniform over a set of size K (see Exercise 19.7). This fact means that we can think of the game as finite and so use the min-max theorem in the form it is stated in Note 19.4.

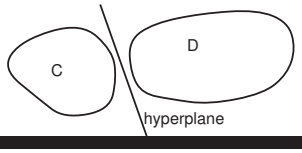
Note 19.4 *(The Min-Max Theorem)*

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 two moves. 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 the row player pays the column player $|a_{i,j}|$ units). Clearly, the *order* in which players make their moves is important. The min-max theorem says that, surprisingly, if we allow the players randomized strategies, then the order of play is immaterial.

By *randomized* (also known as *mixed*) strategies we mean that 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 that

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

As discussed in Exercise 19.6, the Min-Max Theorem can be proven using the following result, known as the *Separating Hyperplane Theorem*: if C and D are disjoint convex subsets of $\mathbb{R}^m$, then there is a hyperplane that separates them. (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 $\{\alpha\mathbf{x} + (1-\alpha)\mathbf{y} : 0 \leq \alpha \leq 1\}$ with $\mathbf{x}$ and $\mathbf{y}$ as its endpoints.) We ask you to prove (a relaxed variant of) the separating hyperplane theorem in Exercise 19.5 but here is a "proof by picture" for the two dimensional case:



19.2 Tool: Error correcting codes

Our next goal will be to construct average-case hard functions using functions that are only worst-case hard. Our main tool will be *error correcting codes*. An error correcting code maps strings into slightly larger strings in a way that "amplifies differences" in the sense that every two distinct strings (even if they differ by just one bit) get mapped into two strings that are "very far" from one another. The formal definition follows:

Definition 19.5 *(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.

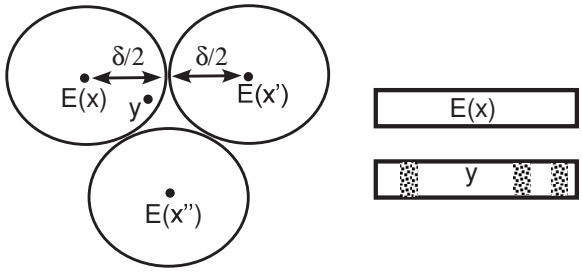


Figure 19.2 In a δ-distance error correcting code, $\Delta(E(x), E(x')) \geq \delta$ for every $x \neq x'$. We can recover x from every string y in which less than $\delta/2$ coordinates were corrupted (i.e., $\Delta(y, E(x)) < \delta/2$) since the $\delta/2$-radius balls around every codeword are disjoint. In the figure above the dotted areas represent corrupted coordinates.

Note that some texts define an error correcting code not as a function $E : \{0,1\}^n \to \{0,1\}^m$ but rather as a 2^n-sized subset of $\{0,1\}^m$ (corresponding to $Im(E)$ in our notation). Error correcting codes have had a vast number of practical and theoretical applications in Computer Science and engineering, but their motivation stems from the following simple application: suppose that Alice wants to transmit a string $x \in \{0,1\}^m$ to Bob, but her channel of communication to Bob is *noisy* and every string y she sends might be corrupted in as many as 10% of its coordinates. That is, her only guarantee is that Bob would receive a string y' satisfying $\Delta(y,y') \leq 0.1$. Alice can perform this task using an error correcting code $E : \{0,1\}^n \to \{0,1\}^m$ of distance $\delta > 0.2$. The idea is that she sends to Bob $y = E(x)$ and Bob receives a string y' satisfying $\Delta(y,y') \leq 0.1$. Since $\Delta(y, E(w)) > 0.2$ for every $w \neq x$, it follows that y is the unique codeword of E that is of distance at most 0.1 from y' and so Bob can find y and from it find x such that $E(x) = y$ (see Figure 19.2). One can see from this example that we'd want codes with as large a distance δ as possible, as small output length m as possible, and of course we'd like both Alice and Bob to be able to carry the encoding and decoding efficiently. The following lemma shows that, ignoring issues of computational efficiency, pretty good error correcting codes exist:

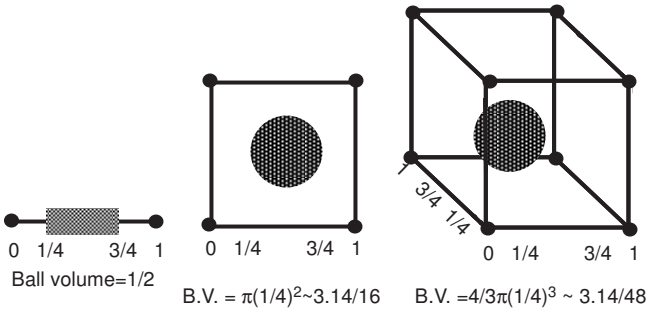
Lemma 19.6 *(Gilbert-Varshamov Bound) For every $\delta < 1/2$ and sufficiently large n, there exists a function $E : \{0,1\}^n \to \{0,1\}^{n/(1-H(\delta))}$ that is an error correcting code with distance δ, where $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$.*[2] ♢

Proof: We prove a slightly weaker statement: the existence of a δ-distance ECC $E : \{0,1\}^n \to \{0,1\}^m$ where $m = 2n/(1-H(\delta))$ instead of $m = n/(1-H(\delta))$. To do so, we

[2] $H(\delta)$ is called the *Shannon entropy function*. It is not hard to see that $H(1/2) = 1, H(0) = 0$, and $H(\delta) \in (0,1)$ for every $\delta \in (0, 1/2)$.

Note 19.7 *(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 volume of the cube with all sides 1 and the ball of radius 1/4. In one dimension, the ratio between these volumes is $1/(1/2) = 2$, in two dimensions it is $1/(\pi/4^2) = 16/\pi$, while in three dimensions it is $1/(4/3\pi/4^3) = 48/\pi$. As the number of dimension grows, this ratio grows exponentially in the number of dimensions. (The volume of a ball of radius r in m dimensions is roughly $\frac{\pi^{m/2}}{\lfloor m/2 \rfloor !} r^m$.) 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, say, 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 discrete setting the volume (i.e., number of points contained) 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.

simply choose E at random. That is, we choose 2^n random strings $y_1, y_2, \ldots, y_{2^n} \in \{0,1\}^m$ and E maps 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 is less than $0.99 \cdot 2^{H(\delta)m}$ 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. By a slightly more clever argument, we can prove the lemma as stated: see Exercise 19.9. It turns out that as δ tends to zero, there do exist codes with smaller values of m than $n/(1 - H(\delta))$, but it is not known whether or not Lemma 19.6 is optimal for δ tending to $1/2$. ■

Why half? Lemma 19.6 only provides codes of distance δ for $\delta < 1/2$ and you might wonder whether this is inherent or perhaps codes of even greater distance exist. 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., $m \geq 2^{n-1}$). 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 19.10.

19.2.1 Explicit codes

The mere existence of an error correcting code is not sufficient for most applications: we need to be able to actually compute them. For this we need to show an *explicit function*

$E : \{0,1\}^n \to \{0,1\}^m$ that is an error correcting satisfying the following properties:

Efficient encoding There is a poly(m) 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: see Exercise 19.11.

We now describe some explicit functions that are error correcting codes.

19.2.2 Walsh-Hadamard Code.

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

Claim 19.8 *The function* WH *is an error correcting code of distance* 1/2. ♢

Proof: First, note that WH is a linear function. That is, $\mathsf{WH}(x+y) = \mathsf{WH}(x) + \mathsf{WH}(y)$, where $x + y$ denotes the componentwise addition of x and y modulo 2 (i.e., bitwise XOR). Thus, for every $x \neq y \in \{0,1\}^n$ 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 $w \neq 0^n$, at least half of the coordinates in $\mathsf{WH}(w)$ are 1. Yet this follows from the random subsum principle (Claim A.31) that says that the probability that $w \odot y = 1$ for $y \in_R \{0,1\}^n$ is exactly 1/2. ■

19.2.3 Reed-Solomon Code

The Walsh-Hadamard code has a serious drawback: its output size is exponential in the input size. By Lemma 19.6 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'll make a detour via codes with *non-binary* alphabet.

Definition 19.9 For every finite 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 (see Exercise 19.12). The Reed-Solomon code is a construction of an error correcting code that can use as its alphabet any sufficiently large field $\mathbb{F}$:

Definition 19.10 *(Reed-Solomon code)* 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. ♢

Note that an equivalent way of defining the Reed Solomon code is that it takes as input a description of the $n-1$ degree polynomial $A(x) = \sum_{i=1}^{n-1} a_i x^i$ and outputs the evaluation of A on the points $f_0, \ldots, f_{m-1}$.

Lemma 19.11 *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). ■

19.2.4 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 19.12 *(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

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

specified by the vector of $\binom{\ell+d}{d}$ coefficients $\{c_{i_1,\ldots,i_\ell}\}$ and the output is the sequence $\{P(x_1,\ldots,x_\ell)\}$ 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 a $2 \cdot 2^n$ long string z satisfying $z_{y,a} = x \odot y + a \pmod 2$ for every $y \in \{0,1\}^n$,$a \in \{0,1\}$). The Schwartz-Zippel Lemma (Lemma A.36 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.

19.2.5 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 19.13 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 19.14 *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

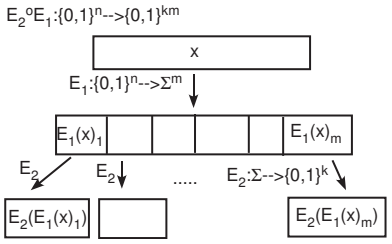


Figure 19.3 If E_1,E_2 are ECC's such that $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \sigma \to \{0,1\}^k$, then the concatenated code $E : \{0,1\}^n \to \{0,1\}^{nk}$ maps x into the sequence of blocks $E_2(E_1(x)_1), \ldots, E_2(E_1(x)_m)$.

to each of these blocks, then whenever two blocks are distinct, the corresponding encoding will have distance δ_2, and so $\delta(x'', y'') \geq \delta_1\delta_2$. ■

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 19.13 and Lemma 19.14 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 19.3. 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 $m = O(n)$, see Exercise 19.18.

19.3 Efficient decoding.

To actually use an error correcting code to store and retrieve information, we need a way to efficiently *decode* a message x from its encoding $E(x)$ even if this encoding has been corrupted in some fraction ρ of its coordinates. We now show how to do this for the Reed-Solomon code and for concatenated codes.

19.3.1 Decoding Reed-Solomon

Recall that the Reed-Solomon treats its input as describing a polynomial and outputs the values of this polynomial on m inputs. We know (see Theorem A.35 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".

Theorem 19.15 *(Unique decoding for Reed-Solomon* [BW86]*)*
There is a polynomial-time algorithm that given a list $(a_1, b_1), \ldots, (a_m, b_m)$ of pairs of elements of a finite field $\mathbb{F}$ such that there is a d-degree polynomial $G : \mathbb{F} \to \mathbb{F}$ satisfying $G(a_i) = b_i$ for t of the numbers $i \in [m]$, with $t > \frac{m}{2} + \frac{d}{2}$, recovers G.

Since Reed-Solomon is an ECC with distance $1 - \frac{d}{m}$, Theorem 19.15 means that we can efficiently recover the correct polynomial from a version corrupted in ρ places as long as ρ is smaller than half the distance. This is optimal in the sense that once the fraction of

errors is larger than half the distance we are no longer guaranteed the existence of a unique solution.

PROOF OF THEOREM 19.15.: As a warmup, we start by considering the case that the number of errors is very small. (This setting is still sufficiently strong for many applications.)

RANDOMIZED INTERPOLATION: THE CASE OF $t \geq (1 - \frac{1}{2(d+1)})m$

Assume that t is quite large: $t > (1 - \frac{1}{2(d+1)})m$. In this case, we can just select $d+1$ pairs $(x_1, y_1), \ldots, (x_{d+1}, y_{d+1})$ at random from the set $\{(a_i, b_i)\}$ and use standard polynomial interpolation to compute the unique a d-degree polynomial P such that $P(x_j) = y_j$ for all $j \in [d+1]$. We then check whether P agrees with at least t pairs of the entire sequence and if so we output P (otherwise we try again). By the union bound, the probability that $x_j \neq G(y_j)$ for one of the $d+1$ chosen pairs is at most $(d+1)\frac{m-t}{t} \leq 1/2$, and hence with probability at least $1/2$ it will be the case that $P = G$.

BERLEKAMP-WELCH PROCEDURE: THE CASE OF $t \geq \frac{m}{2} + \frac{d}{2} + 1$

We now prove Theorem 19.15 using a procedure known as the *Berlekamp-Welch decoding*. For simplicity of notations, we assume that $m = 4d$ and $t = 3d$. However, the proof generalizes to any parameters m, d, t satisfying $t > \frac{m}{2} + \frac{d}{2}$, see Exercise 19.13. Thus, we assume that there exists a d-degree polynomial G such that

$$G(a_i) = b_i \text{ for at least } 3d \text{ of } i\text{'s in } [m] = [4d]. \tag{8}$$

We will use the following decoding procedure:

1. Find a degree $2d$ polynomial $C(x)$ and a degree-d nonzero polynomial $E(x)$ such that:

$$C(a_i) = b_i E(a_i) \text{ for every } i \in [m] \tag{9}$$

   This can be done by considering (9) as a set of $4d$ linear equations with the unknowns being the $2d+1$ coefficients of $C(x)$ and the $d+1$ coefficients of E. These equations have a solution with nonzero $E(x)$ since one can define $E(x)$ to a nonzero polynomial that is equal to zero on every a_i such that $G(a_i) \neq b_i$ (under our assumption (8) there are at most d such places).[3]

2. Divide C by E: get a polynomial P such that $C(x) = E(x)P(x)$ (we will show that E divides C without remainder). Output P.

   We know by (8) and (9) that $C(x) = G(x)E(x)$ for at least $3d$ values, meaning that $C(x) - G(x)E(x)$ is a degree $2d$ polynomial with at least $3d$ roots. This means that this polynomial is identically zero (i.e., $C(x) = G(x)E(x)$ for every $x \in \mathbb{F}$). Thus it does indeed hold that $G = C/E$.

■

19.3.2 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 $E_2 \circ E_1(x)$.

[3] One can efficiently find such a solution by trying to solve the equations after adding to them an equation of the form $E_j = e_j$ where E_j is the j^{th} coefficient of $E(x)$ and e_j is a nonzero element of $\mathbb{F}$. The number of such possible equations is polynomial and at least one of them will result in a satisfiable set of equations.

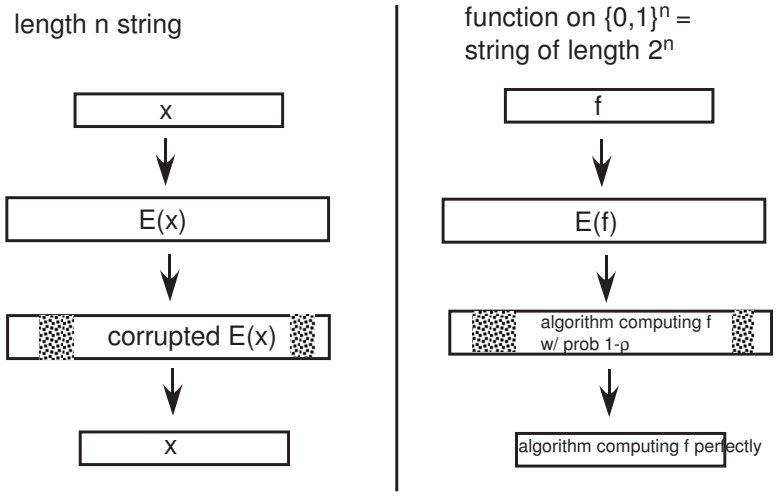


Figure 19.4 An ECC allows us to map a string x to $E(x)$ such as x can be reconstructed from a corrupted version of $E(x)$. The idea is to treat a function $f : \{0,1\}^n \to \{0,1\}$ as a string in $\{0,1\}^{2^n}$, encode it using an ECC to a function $\hat{f}$. Intuitively, $\hat{f}$ should be hard 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.

19.4 Local decoding and hardness amplification

We now show the connection between error correcting codes and hardness amplification. The idea is actually quite simple (see also Figure 19.4). A function $f : \{0,1\}^n \to \{0,1\}$ can be viewed as a binary string of length $N = 2^n$. Suppose we encode f to a string $\hat{f} \in \{0,1\}^M$ using an ECC mapping $\{0,1\}^N$ to $\{0,1\}^M$ with distance larger than, say, 0.2. Then we can view $\hat{f}$ as a function from $\{0,1\}^{\log M}$ to $\{0,1\}$ and at least in principle it should be possible to recover f from a corrupted version of $\hat{f}$ where, say, at most 10% of the locations have been modified. In other words, if it is possible to compute $\hat{f}$ with probability at least 0.9 then it should be possible to compute f exactly. Taking the contrapositive this means that if f is hard to compute in the worst-case then $\hat{f}$ is hard to compute in the average case!

To make this idea work we need to show we can transform every circuit that correctly computes many bits of $\hat{f}$ into a circuit that correctly computes all the bits of f. This is formalized using a *local decoder* (see Figure 19.5), which is a decoding algorithm that given *random access* to a (possibly corrupted) codeword y' close to $E(x)$ can compute any any desired bit of the original input x. Since we are interested in the circuits that could be of size as small as poly(n)— in other words, *polylogarithmic* in $N = 2^n$ —this must also be the running time of the local decoder.

Definition 19.16 *(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 D that, given random access to a string y such that $\Delta(y, E(x)) < \rho$ for some (unknown) $x \in [n]$, and an index $j \in \mathbb{N}$, runs for polylog(m) time and outputs x_j with probability at least $2/3$.

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. We also note that Definition 19.16 can be easily generalized for codes with larger (i.e., non binary) alphabet. Local decoding may also be useful in applications of ECC's that have nothing to do with hardness amplification (e.g., if we use ECC's to encode a huge file, we may want to be able

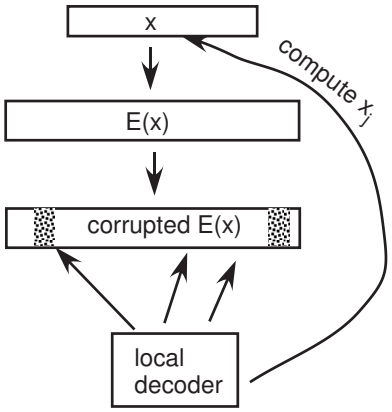


Figure 19.5 A local decoder gets access to a corrupted version of $E(x)$ and an index i and computes from it x_i (with high probability).

to efficiently recover part of the file without decoding it in its entirety). The connection between local decoders and hardness amplification is encapsulated in the following theorem:

Theorem 19.17 *(Hardness amplification from local decoding)*
Suppose that there exists an ECC with polynomial-time encoding algorithm and a local decoding algorithm handling ρ errors. Suppose also that there is $f \in \mathbf{E}$ with $\mathsf{H}_{wrs}(f)(n) \geq S(n)$ for some function $S : \mathbb{N} \to \mathbb{N}$ satisfying $S(n) \geq n$. Then, there exists $\epsilon > 0$ and $\hat{f} \in \mathbf{E}$ with $\mathsf{H}_{avg}^{1-\rho}(\hat{f})(n) \geq S(\epsilon n)^{\epsilon}$

We leave the proof of Theorem 19.17, which follows the ideas described above, as Exercise 19.14. We now show *local decoder* algorithms for several explicit codes.

19.4.1 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 $\rho \geq 1/4$.

Theorem 19.18 *For every $\rho < 1/4$, the walsh-Hadamard code has a local decoder handling ρ errors.* ♢

Proof: Theorem 19.18 is proven by the following algorithm:

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

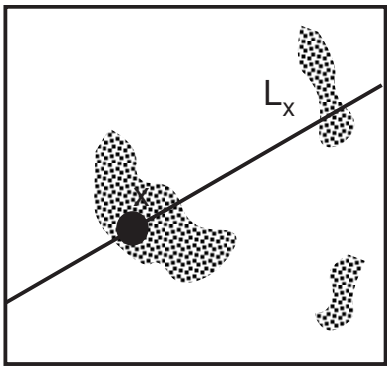


Figure 19.6 Given access to a corrupted version of a polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$, to compute $P(x)$ we pass a random line L_x through x, and use Reed-Solomon decoding to recover the restriction of P to the line L_x.

$f(y + e^j) = x \odot (y + e^j)$. But by the bilinearity of the operation $\odot$, this implies that $f(y) + f(y + e^j) = x \odot y + x \odot (y + e^j) = 2(x \odot y) + x \odot e^j = x \odot e^j \pmod 2$. Yet, $x \odot e^j = x_j$ and so with probability $1 - 2\rho$, the algorithm outputs the right value. (The success probability can be amplified by repetition.) ■

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. ■

19.4.2 Local decoder for Reed-Muller

We now show a local decoder for the Reed-Muller code. 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:

Theorem 19.19 *For every field $|\mathbb{F}|$, numbers d, ℓ and there is a $\mathrm{poly}(|\mathbb{F}|, \ell, d)$-time local decoder for the Reed-Muller code with parameters $\mathbb{F}, d, \ell$ handling $(1 - \frac{d}{|\mathbb{F}|})/6$ errors.*

That is, there is a $\mathrm{poly}(|\mathbb{F}|, \ell, d)$-time algorithm D that given random access to a function $f : \mathbb{F}^\ell \to \mathbb{F}$ that agrees with some degree d polynomial P on a $1 - (1 - \frac{d}{|\mathbb{F}|})/6$ fraction of the inputs and $x \in \mathbb{F}^\ell$ outputs $P(x)$ with probability at least $2/3$. ♢

Proof: 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 $x \in \mathbb{F}^\ell$. We now show such an algorithm:

Reed-Muller Local Decoder for $\rho \leq (1 - \frac{d}{|\mathbb{F}|})/6$.

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 $L_x = \{x + tz : t \in \mathbb{F}\}$ for a random $z \in \mathbb{F}^\ell$.

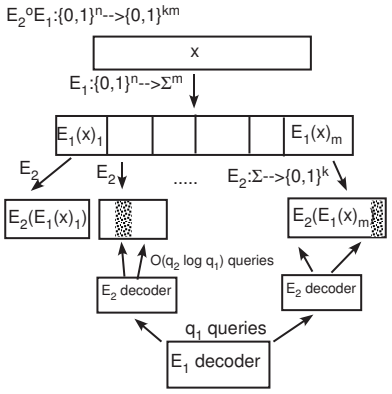


Figure 19.7 To locally decode a concatenated code $E_2 \circ E_1$ we run the decoder for E_1 using the decoder for E_2. The crucial observation is that if y is within $\rho_1\rho_2$ distance to $E_2 \circ E_1(x)$ then at most a ρ_1 fraction of the blocks in y are of distance more than ρ_2 the corresponding 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 $\{(t, f(x+tz))\}$ 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 $Q(t) = f(x+tz)$ for the largest number of t's (see Figure 19.6).[4]
4. Output $Q(0)$.

Analysis: For every d-degree ℓ-variable polynomial P, the univariate polynomial $Q(t) = P(x+tz)$ has degree at most d. Thus, to show that the Reed-Solomon decoding works, it suffices to show that with probability at least $2/3$, the number of points on $w \in L_x$ for which $f(w) \neq P(w)$ is less than $(1 - d/|\mathbb{F}|)/2$. Yet, for every $t \neq 0$, the point $x+tz$ where z is chosen at random in $\mathbb{F}^\ell$ 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 $\rho|\mathbb{F}|$. By the Markov inequality, the probability that there will be more than $3\rho|\mathbb{F}| < (1 - d/|\mathbb{F}|)|\mathbb{F}|/2$ such points is at most $2/3$ and hence Reed-Solomon decoding will be successful with probability $2/3$. In this case, we obtain the correct polynomial q that is the restriction of Q to the line L_x and hence $q(0) = P(x)$.

■

19.4.3 Local decoding of concatenated codes.

As the following lemma shows, given two locally decodable ECC's E_1 and E_2, we can locally decode their concatenation $E_1 \circ E_2$:

Lemma 19.20 *Let $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ be two ECC's with local decoders of q_1 (resp. q_2) queries with respect to ρ_1 (resp. ρ_2) errors. Then there is an $O(q_1 q_2 \log q_1 \log|\Sigma|)$-query local decoder handling $\rho_1\rho_2$ errors for the concatenated code $E = E_2 \circ E_1 : \{0,1\}^n \to \{0,1\}^{mk}$.* ♢

Proof: We prove the lemma using the natural algorithm. Namely, we run the decoder for E_1, but answer its queries using the decoder for E_2 (see Figure 19.7).

Local decoder for concatenated code: $\rho < \rho_1\rho_2$

[4] If ρ is sufficiently small, (e.g., $\rho < 1/(10d)$), then we can use the simpler randomized Reed-Solomon decoding procedure described in Section 19.3.

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 $1 - 1/(10q_1)$.

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.

■

19.4.4 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 19.21 *(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 $E(f')$.[5] 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 $\log N / \log\log N$.
   - 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 $|\mathbb{F}|^\ell \leq \mathrm{poly}(n)$. Its distance is at least $1 - 1/\log N$.

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

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 $\mathsf{WH} \circ \mathsf{RM}$. Combining the local decoders for Walsh-Hadamard and Reed-Muller we get the desired result. ■

Combining Theorem 19.21 with Yao's XOR Lemma (Theorem 19.2), we get the following corollary:

Corollary 19.22 *Let $S : \mathbb{N} \to \mathbb{N}$ be a monotone and time-constructible function. Then there is some $\epsilon > 0$ such that if there exists $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n then there exists $\hat{f} \in \mathbf{E}$ with $ACH(f)(n) \geq S(\sqrt{n})^{\epsilon}$.* ♢

Proof: By Theorem 19.21, under this assumption there exists a function $g \in \mathbf{E}$ with $\mathsf{H}^{0.99}_{\mathsf{avg}}(g)(n) \geq S'(n) = S(n)/\operatorname{poly}(n)$, 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 $\mathsf{H}_{\mathsf{avg}}(g^{\oplus k}) \geq S^{c/2000}$. ■

19.5 List decoding

While Corollary 19.22 is extremely surprising in the qualitative sense (transforming worst-case hardness to average-case hardness) it is still not fully satisfying quantitatively because it loses quite a bit in the circuit size when moving from a worst-case hard to an average-case hard function. In particular, even if we start with a function f that is hard in the worst-case for $2^{\Omega(n)}$-sized circuits, we only end up with a function $\hat{f}$ that is hard on the average case for $2^{\Omega(\sqrt{n})}$-sized circuits. This can make a difference in some applications, and in particular it falls short of what we will need to fully derandomize **BPP** under worst-case assumptions in Chapter 20.

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 binary error-correcting code has to have distance at most $1/2$ but the decoding algorithms work for at most half the distance and hence cannot recover a string f from $E(f)$ if the latter was corrupted in more than a $1/4$ of its locations (i.e., from a string with less than 0.75 agreement with $E(f)$).

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*." Formally, we have the following theorem:

Theorem 19.23 *(Johnson Bound* [Joh62]*) 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 $\ell \leq 1/(2\delta^2)$. ■

19.5.1 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 length m Reed-Solomon codeword that is corrupted in up to a $1 - 2\sqrt{\frac{d}{m}}$ fraction of the coordinates. Note that this tends to 1 as m/d grows, whereas the Berlekamp-Welch algorithm of Section 19.3 (as is the case with any other unique decoding algorithm) cannot handle a fraction of errors that is more than half the distance.

Theorem 19.24 *(List decoding for the Reed-Solomon code* [Sud96]*)*
There is a polynomial-time algorithm that given a set $\{(a_i, b_i)\}_{i=1}^m$ of pairs in $\mathbb{F}^2$, returns the list of all *degree d polynomials G such that the number of i's for which $g(a_i) = b_i$ is more than $2\sqrt{dm}$.*

Proof: We prove Theorem 19.24 via the following algorithm:

Reed-Solomon List Decoding: $t > 2\sqrt{dm}$.

1. Find a nonzero bivariate polynomial $Q(x, y)$ of degree at most $\sqrt{dm}$ in x and at most $\sqrt{m/d}$ in y such that $Q(b_i, a_i) = 0$ for every $i \in [m]$.

   We can express this condition as m linear equations in the $(\sqrt{dm}+1)(\sqrt{m/d}+1) > m$ coefficients of Q. Since these equations are homogeneous (right side equalling zero) and there are more unknowns than equations, this system has a nonzero solution that can be found using gaussian elimination.

2. Factor $Q(x, y)$ using an efficient polynomial factorization algorithm (see [VG99]). For every factor of the form $(P(x) - y)$ check whether $P(x)$ has degree at most d and agrees with $\{(a_i, b_i)\}_{i=1}^m$ in at least t places. If so, output P.

   Indeed, if $G(x)$ agrees with $\{(a_i, b_i)\}_{i=1}^m$ in more than t places then $(G(x) - y)$ is a factor of $Q(x, y)$. To see this note that $Q(G(x), x)$ is a univariate polynomial of degree at most $\sqrt{dm} + d\sqrt{m/d} = 2\sqrt{dm} < t$ which has at least t zeroes and hence it is identically zero. It follows that $G(x) - y$ divides $Q(x, y)$ (see Exercise 19.16).

■

19.6 Local list decoding: getting to $\mathrm{BPP} = \mathrm{P}$.

Analogously to Section 19.4, 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. The definition of local list decoding below is somewhat subtle, and deserves a careful reading.

Definition 19.25 *(Local list decoder)* Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let $\rho = 1-\epsilon$ for $\epsilon > 0$. An algorithm D 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* $i_0 \in [\mathrm{poly}(n/\epsilon)]$ such that for every $j \in [m]$, on inputs i_0, j and with random access to y, D runs for $\mathrm{poly}(\log(m)/\epsilon)$ time and outputs x_j with probability at least $2/3$. ♢

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. As is the case for Definition 19.16, Definition 19.25 can be easily generalized to codes with non-binary alphabet.

19.6.1 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 9.12) 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 Chapter 9 we used this algorithm to find the correct 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 showing how we can use outside information to narrow the list of candidates codewords obtained from a list-decoding algorithm.

19.6.2 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 $P : \mathbb{F}^\ell \to \mathbb{F}$ and the task of the local decoder is to compute $P(x)$ on a given point $x \in \mathbb{F}^\ell$.

Theorem 19.26 *(Reed-Muller local list decoder* [BF90, Lip91, BFNW93, STV99]*) The Reed-Muller code has a local list decoder handling $1 - 10\sqrt{d/|\mathbb{F}|}$ errors.*

That is, for every $\mathbb{F}, \ell, d$ there is a $\mathrm{poly}(|\mathbb{F}|, d, \ell)$-time algorithm D that given random access to a function $f : \mathbb{F}^\ell \to \mathbb{F}$, an index $i \in \mathbb{F}^{\ell+1}$ and an input $x \in \mathbb{F}^\ell$ satisfies: if f agrees with a degree-d polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$ on $10\sqrt{d/|\mathbb{F}|}$ fraction of the inputs then there exists $i_0 \in \mathbb{F}^{\ell+1}$ such that $\Pr[D^f(i_0, x) = P(x)] \geq 2/3$ for every x. ♢

Proof: To be a valid local list decoder, given the index i_0, the algorithm should output $P(x)$ with high probability for *every* $x \in \mathbb{F}^\ell$. Below we describe a relaxed decoder that is only guaranteed to output the right value for *most* (i.e., a 0.9 fraction) of the x's in $\mathbb{F}^\ell$. One can transform this algorithm to a valid local list decoder by combining it with the Reed-Muller local decoder described in Section 19.4.2. Thus, Theorem 19.26 is proven via the following algorithm:

Reed-Muller Local List Decoder for $\rho \leq 1 - 10\sqrt{d/|\mathbb{F}|}$

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 that $|\mathbb{F}| > d^4$ and d is sufficiently large (e.g., $d > 1000$ will do). 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 $r \in \mathbb{F}$, and set $L_{x,x_0} = \{q(t) : t \in \mathbb{F}\}$. (See Figure 19.8.)
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.

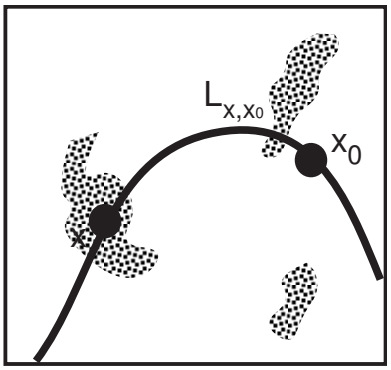


Figure 19.8 Given access to a corrupted version of a polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$ and some index (x_0, y_0), to compute $P(x)$ we pass a random degree-3 curve L_{x,x_0} through x and x_0, and use Reed-Solomon list decoding to recover a list of candidates for the restriction of P to 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 $\mathbb{F}^\ell$.

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. 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 19.15 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 $1 - \frac{(3d+1)\sqrt{|F|}/4d}{|\mathbb{F}|} > 0.99$, 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 $g(0) = P(x)$. ■

19.6.3 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 19.4.3, 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 and can handle $1 - \epsilon_1$ errors, and that E_2 takes an index in I_2 and can handle $1/2 - \epsilon_2$ errors. 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 19.4.3.) 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.

19.6.4 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 19.27 *(Worst-case hardness to strong hardness)*
Let $f \in \mathbf{E}$ be such that $\mathsf{H}_{wrs}(f)(n) \geq S(n)$ for some time-constructible non-decreasing $S : \mathbb{N} \to \mathbb{N}$. 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 19.4.4, 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 $S(n)^{1/100}$. (We may assume without loss of generality that $S(n) > n^{1000}$ as otherwise the theorem is trivial.)
- 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 us 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

Yao's XOR Lemma was first stated and proven by Yao in oral presentations of his paper [Yao82a]. Since then several proofs have been published with the first one by Levin in [Lev87] (see the survey [GNW95]). Russell Impagliazzo's hardcore lemma was proven in [Imp95b]; the proof of Section 19.1.2 is due to Noam Nisan.

The study of error correcting codes is an extremely beautiful and useful field, and we have barely scratched its surface here. This field was initiated by two roughly concurrent seminal papers of Shannon [Sha48] and Hamming [Ham50]. The lecture notes of Madhu Sudan (available from his home page) provide a good starting point for theoretical computer scientists; see also the survey [Sud01].

Reed-Solomon codes were invented in 1960 by Irving Reed and Gustave Solomon [RS60]. The first efficient decoding algorithm for Reed-Solomon codes was by Peterson [Pet60]. (Interestingly, this algorithm is one of the first non-trivial polynomial-time algorithms invented, preceding even the formal definition of the class **P**.) The algorithm presented in Section 19.3 is a simplification due to Gemmell and Sudan [GS92] of the Berlekamp-Welch decoder [BW86].

Reed-Muller codes were invented by Muller [Mul54] with the first decoder given by Reed [Ree54]. The first Reed-Muller local decoders were given by Beaver and Feigenbaum [BF90] and Lipton [Lip91], who observed this implies a worst-case to average-case connection for the *Permanent* (see also Section 8.6.2). Babai, Fortnow, and Lund [BFL90] observed that by taking multilinear extensions, such connections also hold for **PSPACE** and **EXP**, and Babai et al [BFNW93] showed that this allows for derandomization from worst-case assumptions. The Reed-Muller local decoding algorithm of Section 19.4.2 is due to Gemmell et al [GLR$^+$91].

The first list-decoding algorithm for Reed-Solomon codes was given by Sudan [Sud96] and was subsequently improved by Guruswami and Sudan [GS98]. Recently, Parvaresh and Vardy [PV05] showed a list-decoding algorithm handling even more errors for a variant of the Reed-Solomon code, a result that was further improved by Guruswami and Rudra [GR06], achieving an optimal tradeoff between rate and list decoding radius for large alphabets.

The quantitatively strong hardness amplification (Theorem 19.27) was first shown by Impagliazzo and Wigderson [IW97] that gave a *derandomized* version of Yao's XOR Lemma. Our presentation follows the alternative proof by Sudan, Trevisan and Vadhan [STV99] who were the first to make an explicit connection between error correcting codes and hardness amplification, and also the first to explicitly define local list decoding and use it for hardness amplification. The first local list decoding algorithm for the Walsh-Hadamard code was given by Goldreich and Levin [GL89] (although the result is not explicitly described in these terms there). The Reed-Muller local list decoding algorithm of Section 19.6 is a variant of the algorithm of [STV99].

The question raised in Problem 19.8 is treated in O'Donnell [O'D04], where a hardness amplification lemma is given for **NP**. For a sharper result, see Healy, Vadhan, and Viola [HVV04].

Exercises

19.1 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 $\Pr[X = 1] = 1/2 + (1-2\delta)^k$. H457

19.2 Prove that if there exists a δ-density distribution H such that $\Pr_{x \in_R H}[C(x) = f(x)] \leq 1/2 + e$ for every circuit C of size at most S with $S \leq \sqrt{\epsilon^2 \delta 2^n / 100}$, then there exists a subset $I \subseteq \{0,1\}^n$ of size at least $\frac{\delta}{2}2^n$ such that

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

for every circuit C of size at most S. H458

19.3 Let H be an δ-density distribution over $\{0,1\}^n$ (i.e., $\Pr[H = x] \leq 1/(\delta 2^n)$ for every $x \in \{0,1\}^n$).

(a) Let G be the distribution defined by $\Pr[G = x] = (2^{-n} - \delta \Pr[H = x])/(1-\delta)$ for every $x \in \{0,1\}^n$. Prove that G is indeed a distribution (i.e., all probabilities are non-negative and sum up to 1).

(b) Let U be the following distribution: with probability δ pick an element from H and with probability $1-\delta$ pick an element from G. Prove that U is the uniform distribution.

H458

19.4 Complete the proof of Impagliazzo's Hardcore Lemma (Lemma 19.3) for general k.

19.5 Prove the hyperplane separation theorem in the following form: If $C, D \subseteq \mathbb{R}^m$ are two disjoint convex set with C closed and D compact (i.e., closed and bounded) then there exists a nonzero vector $\mathbf{z} \in \mathbb{R}^m$ and a number $a \in \mathbb{R}$ such that

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

H458

19.6 Prove the Min-Max Theorem (see Note 19.4) using the hyperplane separation theorem as stated in Exercise 19.5. H458

19.7 ([CG85]) We say that a distribution D over $\{0,1\}^n$ is K-flat if D is the uniform distribution over a subset of $\{0,1\}^n$ with size at least K. Prove that for every k, every 2^{-k}-density distribution H is a convex combination of 2^{n-k}-flat distributions. That is, there are N 2^{n-k}-flat distributions $D_1, \ldots, D_N$ and non-negative numbers $\alpha_1, \ldots, \alpha_N$ such that $\sum_i \alpha_i = 1$ and H is equivalent to the distribution obtained by picking i with probability α_i and then picking a random element from D_i. H458

19.8 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?

19.9 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 $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$. H458

19.10 Show that for every $E : \{0,1\}^n \to \{0,1\}^m$ that is an error correcting code of distance $1/2$, $2^n < 10m$. Show if E is an error correcting code of distance $\delta > 1/2$, then $2^n < 10/(\delta - 1/2)$. H458

19.11 Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC such that there exist two distinct strings $x^1, x^2 \in \{0,1\}^m$ with $\Delta(E(x^1), E(x^2)) \leq \delta$. Prove that there's no decoder for E handling $\delta/2$ or more errors. That is, show that there is no function $D : \{0,1\}^m \to \{0,1\}^n$ such that for every $x \in \{0,1\}^m$ and y with $\Delta(y, E(x)) \leq \delta/2$, $D(y) = x$.

19.12 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.

19.13 Prove Theorem 19.15 as stated. That is show how to recover a d-degree polynomial G from a sequence of pairs $(a_1, b_1), \ldots, (a_m, b_m)$ agreeing with G in t places whenever $t \geq \frac{m}{2} + \frac{d}{2} + 1$.

19.14 Prove Theorem 19.17. H458

19.15 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. H458

19.16 Prove that if $Q(x, y)$ is a bivariate polynomial over some field $\mathbb{F}$ and $P(x)$ is a univariate polynomial over $\mathbb{F}$ such that $Q(P(x), x)$ is the zero polynomial then $Q(x, y) = (y - P(x))A(x, y)$ for some polynomial $A(x, y)$. H458

19.17 *(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 $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$. H458

(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 $\mathrm{GF}(2^k)$ to solve this, see Appendix A.) H458

(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.

19.18 Recall that for every m there is field $\mathbb{F} = \mathrm{GF}(2^m)$ of 2^m elements such that we can represent each element of $\mathbb{F}$ as a vector in $\mathrm{GF}(2)^m$ with addition in $\mathbb{F}$ corresponding to bitwise XOR (see Appendix A). Thus for every $a \in \mathbb{F}$, the operation $x \mapsto a \times x$ (where $\times$ denotes multiplication in $\mathbb{F}$) is a linear operation in $\mathrm{GF}(2)^m$. Moreover, this operation is efficiently computable given the description of a.

(a) Prove that for every nonzero $x \in F$, if we choose a uniformly in $\mathbb{F}$ then $a \times x$ is distributed uniformly over $\mathbb{F}$.

(b) Prove that for every nonzero $x \in F$, the probability over a random $a \in_R \mathbb{F}$ that $a \times x$ has at most $m/10$ ones in its representation as an m-bit vector is less than $2^{-m/10}$. Conclude that there exists $a \in \mathbb{F}$ such that the function that maps $x \in \{0,1\}^{m/10}$ to $a \times (x \circ 0^{0.9m})$ (where $\circ$ denotes concatenation) is an error correcting code with distance at least 0.1.

(c) Show that there exists constants $c, \delta > 0$ such that for every n there is an explicit error correcting code $E : \{0,1\}^n \to \{0,1\}^{cn}$ of distance at least δ. H458