Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–71

DOI: xxxxxx

now
the essence of knowledge

Pseudorandomness II

Salil P. Vadhan[1]

[1] *Harvard UniversityCambridge, MA02138, USA, salil@eecs.harvard.edu*

Abstract

This is the second volume of a 2-part survey on *pseudorandomness,* the theory of efficiently generating objects that "look random" despite being constructed using little or no randomness. The survey places particular emphasis on the intimate connections that have been discovered between a variety of fundamental "pseudorandom objects" that at first seem very different in nature: expander graphs, randomness extractors, list-decodable error-correcting codes, samplers, and pseudorandom generators. The survey also illustrates the significance the theory of pseudorandomness has for the study of computational complexity, algorithms, cryptography, combinatorics, and communications. The structure of the presentation is meant to be suitable for teaching in a graduate-level course, with exercises accompanying each chapter.

To Amari

Acknowledgments

My exploration of pseudorandomness began in my graduate and postdoctoral years at MIT and IAS, under the wonderful guidance of Shafi Goldwasser, Oded Goldreich, Madhu Sudan, and Avi Wigderson. It was initiated by an exciting reading group organized at MIT by Luca Trevisan, which immersed me in the subject and started my extensive collaboration with Luca. Through fortuitous circumstances, I also began to work with Omer Reingold, starting another lifelong collaboration. I am indebted to Shafi, Oded, Madhu, Avi, Luca, and Omer for all the insights and research experiences they have shared with me.

I have also learned a great deal from my other collaborators on pseudorandomness, including Boaz Barak, Eli Ben-Sasson, Michael Capalbo, Kai-Min Chung, Nenad Dedic, Ronen Gradwohl, Dan Gutfreund, Venkat Guruswami, Alex Healy, Jesse Kamp, Danny Lewin, Chi-Jen Lu, Michael Mitzenmacher, Shien Jin Ong, Michael Rabin, Anup Rao, Ran Raz, Leo Reyzin, Eyal Rozenman, Madhur Tulsiani, Chris Umans, Emanuele Viola, and David Zuckerman. Needless to say, this list omits many other researchers in the field with whom I have had stimulating discussions on related topics.

The starting point for this survey was scribe notes taken by students in the 2004 version of my graduate course on Pseudorandomness. I thank those students for their contribution: Alexandr Andoni, Adi Akavia, Yan-Cheng Chang, Denis Chebikin, Hamilton Chong, Vitaly Feldman, Brian Greenberg, Chun-Yun Hsiao, Andrei Jorza, Adam Kirsch, Kevin Matulef, Mihai Pătraşcu, John Provine, Pavlo Pylyavskyy, Arthur Rudolph, Saurabh Sanghvi, Grant Schoenebeck, Jordanna Schutz, Sasha Schwartz, David Troiano, Vinod Vaikuntanathan, Kartik Venkatram, David Woodruff. I also thank the students from the other offerings of the course; Dan Gutfreund, who gave a couple of guest lectures in 2007; and all of my teaching fellows, Johnny Chen, Kai-Min Chung, Minh Nguyen, and Emanuele Viola. Special thanks are due to Greg Price for his extensive feedback on the lecture notes. Helpful comments and corrections have also been given by Trevor Bass, Zhou Fan, Alex Healy, Andrei Jorza, Shira Mitchell, Jelani Nelson, Yakir Reshef, Shrenik Shah, Michael von Korff, Neal Wadhwa, and Hoeteck Wee.

Contents

5

Randomness Extractors

We now move on to the second major pseudorandom object of this survey: randomness extractors. We begin by discussing the original motivation for extractors, which was to simulate randomized algorithms with sources of biased and correlated bits. This motivation is still compelling, but extractors have taken on a much wider significance in the years since they were introduced. They have found numerous applications in theoretical computer science beyond this initial motivating one, in areas from cryptography to distributed algorithms to metric embeddings. More importantly from the perspective of this survey, they have played a major unifying role in the theory of pseudorandomness. Indeed, the links between the various pseudorandom objects we will study in this survey (expander graphs, randomness extractors, list-decodable codes, pseudorandom generators, samplers) were all discovered through work on extractors.

5.1 Motivation and Definition

5.1.1 Deterministic Extractors

Typically, when we design randomized algorithms or protocols, we assume that all algorithms/parties have access to sources of *perfect* randomness, i.e. bits that are unbiased and completely independent. However, when we implement these algorithms, the physical sources of randomness to which we have access may contain biases and correlations. For example, we may use low-order bits of the system clock, the user's mouse movements, or a noisy diode based on quantum effects. While these sources may have some randomness in them, the assumption that the source is perfect is a strong one, and thus it is of interest to try and relax it.

Ideally, what we would like is a compiler that takes any algorithm A that works correctly when fed perfectly random bits U_m, and produces a new algorithm A' that will work even if it is fed random bits $X \in \{0,1\}^n$ that come from a "weak" random source. For example, if A is a **BPP** algorithm, then we would like A' to also run in probabilistic polynomial time. One way to design such compilers is to design a *randomness extractor* $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that $\mathrm{Ext}(X)$ is distributed uniformly in $\{0,1\}^m$.

IID-Bit Sources (aka Von Neumann Sources). A simple version of this question was already considered by von Neumann. He looked at sources that consist of boolean random variables $X_1, X_2, \ldots, X_n \in \{0,1\}$ that are independent but biased. That is, for every i, $\Pr[X_i = 1] = \delta$ for some unknown δ. How can such a source be converted into a source of independent, unbiased bits? Von Neumann proposed the following extractor: Break all the variables in pairs and for each pair output 0 if the outcome was 01, 1 if the outcome was 10 , and skip the pair if the outcome was 00 or 11. This will yield an unbiased random bit after $1/\delta$ pairs on average.

Independent-Bit Sources. Lets now look at a bit more interesting class of sources in which all the variables are still independent but the bias is no longer the same. Specifically, for every i, $\Pr[X_i = 1] = \delta_i$ and $0 < \delta \leq \delta_i \leq 1 - \delta$. How can we deal with such a source?

Let's be more precise about the problems we are studying. A *source* on $\{0,1\}^n$ is simply a random variable X taking values in $\{0,1\}^n$. In each of the above examples, there is an implicit *class* of sources being studied. For example, $\text{IndBits}_{n,\delta}$ is the class of sources X on $\{0,1\}^n$ where the bits X_i are independent and satisfy $\delta \leq \Pr[X_i = 1] \leq 1 - \delta$. We could define $\text{IIDBits}_{n,\delta}$ to be the same with the further restriction that all of the X_i's are identically distributed, i.e. $\Pr[X_i = 1] = \Pr[X_j = 1]$ for all i, j, thereby capturing von Neumann sources.

Definition 5.1 (deterministic extractors). [1] Let $\mathcal{C}$ be a class of sources on $\{0,1\}^n$. An *ε-extractor* for $\mathcal{C}$ is a function $\text{Ext} : \{0,1\}^n \to \{0,1\}^m$ such that for every $X \in \mathcal{C}$, $\text{Ext}(X)$ is "ε-close" to U_m.

Note that we want a *single* function Ext that works for all sources in the class. This captures the idea that we do not want to assume we know the exact distribution of the physical source we are using, but only that it comes from some class. For example, for $\text{IndBits}_{n,\delta}$, we know that the bits are independent and none are too biased, but not the specific bias of each bit. Note also that we only allow the extractor *one* sample from the source X. If we want to allow multiple independent samples, then this should be modelled explicitly in our class of sources; ideally we would like to minimize the independence assumptions used.

We still need to define what we mean for the output to be ε-close to U_m.

Definition 5.2. For random variables X and Y taking values in $\mathcal{U}$, their *statistical difference* (also known as *variation distance*) is $\Delta(X,Y) = \max_{T \subseteq \mathcal{U}} |\Pr[X \in T] - \Pr[Y \in T]|$. We say that X and Y are *ε-close* if $\Delta(X,Y) \leq \varepsilon$.

Intuitively, any event in X happens in Y with the same probability $\pm\varepsilon$. This is really the most natural measure of distance for probability distributions (much more so than the ℓ_2 distance we used in the study of random walks). In particular, it satisfies the following natural properties.

Lemma 5.3 (properties of statistical difference). Let $X, Y, Z, X_1, X_2, Y_1, Y_2$ be random variables taking values in a universe $\mathcal{U}$. Then,

[1] Such extractors are called *deterministic* or *seedless* to contrast with the probabilistic or *seeded* randomness extractors we will see later.

(1) $\Delta(X, Y) \geq 0$, with equality iff X and Y are identically distributed,
(2) $\Delta(X, Y) \leq 1$, with equality iff X and Y have disjoint supports,
(3) $\Delta(X, Y) = \Delta(Y, X)$,
(4) $\Delta(X, Z) \leq \Delta(X, Y) + \Delta(X, Z)$,
(5) for every function f, we have $\Delta(f(X), f(Y)) \leq \Delta(X, Y)$,
(6) $\Delta((X_1, X_2), (Y_1, Y_2)) \leq \Delta(X_1, Y_1) + \Delta(X_2, Y_2)$ if X_1 and X_2, as well as Y_1 and Y_2, are independent, and
(7) $\Delta(X, Y) = \frac{1}{2} \cdot |X - Y|_1$, where $|\cdot|_1$ is the ℓ_1 distance. (Thus, X is ε-close to Y iff we can transform X into Y by "shifting" at most an ε fraction of probability mass.)

We now observe that extractors according to this definition give us the "compilers" we want.

Proposition 5.4. Let $A(w; r)$ be a randomized algorithm such that $A(w; U_m)$ has error probability at most γ, and let $\text{Ext} : \{0,1\}^n \to \{0,1\}^m$ be an ε-extractor for a class $\mathcal{C}$ of sources on $\{0,1\}^n$. Define $A'(w; x) = A(w; \text{Ext}(x))$. Then for every source $X \in \mathcal{C}$, $A'(w; X)$ has error probability at most $\gamma + \varepsilon$.

This application identifies some additional properties we'd like from our extractors. We'd like the extractor itself to be efficiently computable (e.g. polynomial time). In particular, to get m almost-uniform bits out, we should need at most $n = \text{poly}(m)$ bits bits from the weak random source.

We can cast our earlier extractor for sources of independent bits in this language:

Proposition 5.5. For every constant $\delta > 0$, every $n, m \in \mathbb{N}$, there is a polynomial-time computable function $\text{Ext} : \{0,1\}^n \to \{0,1\}^m$ that is an ε-extractor for $\text{IndBits}_{n,\delta}$, with $\varepsilon = m \cdot 2^{-\Omega(n/m)}$.

In particular, taking $n = m^2$, we get exponentially small error with a source of polynomial length.

Proof. Ext breaks the source into m blocks of length $\lfloor n/m \rfloor$ and outputs the parity of each block. □

Unpredictable-Bit Sources (aka Santha–Vazirani Sources). Another interesting class of sources, which looks similar to the previous example is the class $\text{UnpredBits}_{n,\delta}$ of *unpredictable-bit sources*. These are the sources that for every i, every $x_1, \ldots, x_n \in \{0,1\}$ and some constant $\delta > 0$, satisfy

$$\delta \leq \Pr\left[X_i = 1 \mid X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}\right] \leq 1 - \delta$$

The parity extractor used above will be of no help with this source since the next bit could be chosen in a way that the parity will be equal to 1 with probability δ. Problem 5.4 shows that there does not exist any nontrivial extractor for these sources:

Proposition 5.6. For every $n \in \mathbb{N}$, $\delta > 0$, and fixed extraction function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}$ there exists a source $X \in \mathrm{UnpredBits}_{n,\delta}$ such that either $\Pr[\mathrm{Ext}(X) = 1] \leq \delta$ or $\Pr[\mathrm{Ext}(X) = 1] \geq 1 - \delta$. That is, there is no ε-extractor for $\mathrm{UnpredBits}_{n,\delta}$ for $\varepsilon < 1/2 - \delta$.

Nevertheless, as we will see, the answer to the question whether we can simulate **BPP** algorithms with unpredictable sources will be "yes"! Indeed, we will even be able to handle a much more general class of sources, introduced in the next section.

5.1.2 Entropy Measures and General Weak Sources

Intuitively, to extract m almost-uniform bits from a source, the source must have at least "m bits of randomness" in it (e.g. its support cannot be much smaller than 2^m). Ideally, this is all we would like to assume about a source. Thus, we need some measure of how much randomness is in a random variable; this can be done using various notions of *entropy* described below.

Definition 5.7 (entropy measures). Let X be a random variable. Then

- the *Shannon entropy* of X is:
$$\mathrm{H}_{Sh}(X) = \mathop{\mathrm{E}}_{x \xleftarrow{\mathrm{R}} X}\left[\log \frac{1}{\Pr[X = x]}\right].$$
- the *Rényi entropy* of X is:
$$\mathrm{H}_2(X) = \log\left(\frac{1}{\mathrm{E}_{x \xleftarrow{\mathrm{R}} X}[\Pr[X = x]]}\right) = \log \frac{1}{\mathrm{CP}(X)}, \text{ and}$$
- the *min-entropy* of X is:
$$\mathrm{H}_\infty(X) = \min_x \left\{\log \frac{1}{\Pr[X = x]}\right\},$$

where all logs are base 2.

All the three measures satisfy the following properties we would expect from a measure of randomness:

Lemma 5.8 (properties of entropy). For each of of the entropy measures $\mathrm{H} \in \{\mathrm{H}_{Sh}, \mathrm{H}_2, \mathrm{H}_\infty\}$ and random variables X, Y, we have:

- $\mathrm{H}(X) \geq 0$, with equality iff X is supported on a single element,
- $\mathrm{H}(X) \leq \log|\mathrm{Supp}(X)|$, with equality iff X is uniform on $\mathrm{Supp}(X)$,
- if X, Y are independent, then $\mathrm{H}((X,Y)) = \mathrm{H}(X) + \mathrm{H}(Y)$,
- for every deterministic function f, we have $\mathrm{H}(f(X)) \leq \mathrm{H}(X)$, and
- for every X, we have $\mathrm{H}_\infty(X) \leq \mathrm{H}_2(X) \leq \mathrm{H}_{Sh}(X)$.

To illustrate the differences between the three notions, consider a source X such that $X = 0^n$ with probability 0.99 and $X = U_n$ with probability 0.01. Then $\mathrm{H}_{Sh}(X) \geq 0.01n$ (contribution from the uniform distribution), $\mathrm{H}_2(X) \leq \log(1/.99^2) < 1$ and $\mathrm{H}_\infty(X) \leq \log(1/.99) < 1$ (contribution from 0^n). Note that even though X has Shannon entropy linear in n, we cannot expect to extract bits that are close to uniform or carry out any useful randomized computations with one sample from X, because it gives us nothing useful 99% of the time. Thus, we should use the stronger measures of entropy given by H_2 or H_∞.

Then why is Shannon entropy so widely used in information theory results? The reason is that such results typically study what happens when you have many independent samples from the source. In such a case, it turns out that the the source is "close" to one where the min-entropy is roughly equal to the Shannon entropy. Thus the distinction between these entropy measures becomes less significant. (Recall that we only allow one sample from the source.) Moreover, Shannon entropy satisfies many nice identities that make it quite easy to work with. Min-entropy and Renyi entropy are much more delicate.

We will consider the task of extracting randomness from sources where all we know is a lower bound on the min-entropy:

Definition 5.9. X is a *k-source* is $\mathrm{H}_\infty(X) \geq k$, i.e., if $\Pr[X = x] \leq 2^{-k}$.

A typical setting of parameters is $k = \delta n$ for some fixed δ, e.g., 0.01. We call δ the *min-entropy rate*. Some different ranges that are commonly studied (and are useful for different applications): $k = \mathrm{polylog}(n)$, $k = n^\gamma$ for a constant $\gamma \in (0, 1)$, $k = \delta n$ for a constant $\delta \in (0, 1)$, and $k = n - O(1)$. The middle two ($k = n^\gamma$ and $k = \delta n$) are the most natural for simulating randomized algorithms with weak random sources.

Examples of k-sources:

- k random and independent bits, together with $n - k$ fixed bits (in an arbitrary order). These are called *oblivious bit-fixing* sources.
- k random and independent bits, and $n - k$ bits that depend arbitrarily on the first k bits. These are called *adaptive bit-fixing sources*.
- Unpredictable-bit sources with bias parameter δ. These are k-sources with $k = \log(1/(1-\delta)^n) = \Theta(\delta n)$.
- Uniform distribution on $S \subset \{0,1\}^n$ with $|S| = 2^k$. These are called *flat k-sources*.

It turns out that flat k-sources are really representative of general k-sources.

Lemma 5.10. Every k-source is a convex combination of flat k-sources (provided that $2^k \in \mathbb{N}$), i.e., $X = \sum p_i X_i$ with $0 \leq p_i \leq 1$, $\sum p_i = 1$ and all the X_i are flat k-sources.

Proof. [Sketch] Consider each source on $[N]$ (recall that $N = 2^n$) as a vector $X \in \mathbb{R}^N$. Then X is a k-source if and only if $X(i) \in [0, 2^{-k}]$ for every $i \in [N]$ and $\sum_i X(i) = 1$. The set of vectors X satisfying these linear inequalities is a polytope. By basic linear programming theory, all of the points in the polytope are convex combinations of its *vertices*, which are defined to be the points

that make a maximal subset of the inequalities tight. By inspection, the vertices of the polytope of k-sources are those sources where $X(i) = 2^{-k}$ for 2^k values of i and $X(i) = 0$ for the remaining values of i; these are simply the k-sources. □

By Lemma 5.10, we can think of any k-source as being obtained by first selecting a flat k-source X_i according to some distribution (given by the p_i's) and then selecting a random sample from X_i. This means that if we can compile probabilistic algorithms to work with flat k-sources, then we can compile them to work with any k-source.

5.1.3 Seeded Extractors

Proposition 5.6 tell us that it impossible to have deterministic extractors for unpredictable sources. Here we consider k-sources, which are more general than unpredictable sources, and hence it is also impossible to have deterministic extractors for them. The impossibility result for k-sources is stronger and simpler to prove.

Proposition 5.11. For any $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}$ there exists an $(n-1)$-source X so that $\mathrm{Ext}(X)$ is constant.

Proof. There exists $b \in \{0,1\}$ so that $|\mathrm{Ext}^{-1}(b)| \geq 2^n/2 = 2^{n-1}$. Then let X be the uniform distribution on $\mathrm{Ext}^{-1}(b)$. □

On the other hand, if we reverse the order of quantifiers, allowing the extractor to depend on the source, it is easy to see that good extractors exist and in fact a randomly chosen function will be a good extractor with high probability.

Proposition 5.12. For every $n, k, m \in \mathbb{N}$, every $\varepsilon > 0$, and every flat k-source X, if we choose a random function $\mathrm{Ext} : \{0,1\}^n \to \{0,1\}^m$ with $m = k - 2\log(1/\varepsilon) - O(1)$, then $\mathrm{Ext}(X)$ will be ε-close to U_m with probability $1 - 2^{-\Omega(K\varepsilon^2)}$, where $K = 2^k$.

(We will commonly use the convention that capital variables are 2 raised to the power of the corresponding lowercase variable, such as $K = 2^k$ above.)

Proof. Choose our extractor randomly. We want it to have following property: for all $T \subseteq [M]$, $|\Pr[\mathrm{Ext}(X) \subseteq T] - \Pr[U_m \subseteq T| \leq \varepsilon$. Equivalently, $|\{x \in \mathrm{Supp}(X) : \mathrm{Ext}(x) \in T\}|/K$ differs from the density $\mu(T)$ by at most ε. For each point $x \in \mathrm{Supp}(X)$, the probability that $\mathrm{Ext}(x) \in T$ is $\mu(T)$, and these events are independent. By the Chernoff Bound, for each fixed T, this condition holds with probability at least $1 - 2^{-\Omega(K\varepsilon^2)}$. Then the probability that condition is violated for at least one T is at most $2^M 2^{-\Omega(K\varepsilon^2)}$, which is less than 1 for $m = k - 2\log(1/\varepsilon) - O(1)$. □

Note that the failure probability is doubly-exponentially small in k. Naively, one might hope that we could get an extractor that's good for all flat k-sources by a union bound. But the number of flat k-sources is $\binom{N}{K} \approx N^K$ (where $N = 2^n$), which is unfortunately a larger double-exponential in k. We can overcome this gap by allowing the extractor to be "slightly" probabilistic, i.e. allowing the extractor a *seed* consisting of a small number of truly random bits in addition to the weak random source. We can think of this seed of truly random bits as a random choice of an extractor from family of extractors. This leads to the following crucial definition:

Definition 5.13 (seeded extractors). Extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-*extractor* if for every k-source X on $\{0,1\}^n$, $\text{Ext}(X, U_d)$ is ε-close to U_m.

We want to give a construction that minimizes d and maximizes m. We prove the following theorem.

Theorem 5.14. For every $n \in \mathbb{N}$, $k \in [0,n]$ and $\varepsilon > 0$, there exists a (k,ε)-extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k + d - 2\log(1/\varepsilon) - O(1)$ and $d = \log(n-k) + 2\log(1/\varepsilon) + O(1)$.

One setting of parameters to keep in mind (for our application of simulating randomized algorithms with a weak source) is $k = \delta n$, with δ a fixed constant (e.g. $\delta = 0.01$), and ε a fixed constant (e.g. $\varepsilon = 0.01$).

Proof. We use the Probabilistic Method. By Lemma 5.10, it suffices for Ext to work for flat k-sources. Choose the extractor Ext at random. Then the probability that the extractor fails is not more than number of flat k-sources times times the probability Ext fails for a fixed flat k-source. By the above proposition, the probability of failure for a fixed flat k-source is at most $2^{-\Omega(KD\varepsilon^2)}$, since (X, U_d) is a flat $(k+d)$-source) and $m = k + d - 2\log(\frac{1}{\varepsilon}) - O(1)$. Thus the total failure probability is at most

$$\binom{N}{K} \cdot 2^{-\Omega(KD\varepsilon^2)} \leq \left(\frac{Ne}{K}\right)^K 2^{-\Omega(KD\varepsilon^2)}.$$

The latter expression is less than 1 if $D\varepsilon^2 \geq c\log(Ne/K) = c \cdot (n-k) + c'$ for constants c, c'.This is equivalent to $d = \log(n-k) + 2\log(\frac{1}{\varepsilon}) + O(1)$. □

It turns out that both bounds (on m and d) are individually tight up to the $O(1)$ terms.

Recall that our motivation for extractors was to simulate randomized algorithms given *only* a weak random source, so allowing a truly random seed may seem to defeat the purpose. However, if the seed is of logarithmic length as in Theorem 5.14, then instead of selecting it randomly, we can enumerate all possibilities for the seed and take a majority vote.

Proposition 5.15. Let $A(w;r)$ be a randomized algorithm such that $A(w;U_m)$ has error probability at most γ, and let $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ε)-extractor. Define

$$A'(w;x) = \operatorname*{maj}_{y \in \{0,1\}^d} \{A(w; \text{Ext}(x,y))\}.$$

Then for every k-source X on $\{0,1\}^n$, $A'(w;X)$ has error probability at most $2(\gamma + \varepsilon)$.

Proof. The probability that $A(w; \text{Ext}(X, U_d))$ is incorrect is not more than probability $A(w;U_m)$ is incorrect plus ε, i.e. $\gamma + \varepsilon$, by the definition of statistical difference. Then the probability that $\text{maj}_y A(w, \text{Ext}(X,y))$ is incorrect is at most $2 \cdot (\gamma + \varepsilon)$, because each error of $\text{maj}_y A(w; \text{Ext}(x,y))$ corresponds to $A(w; \text{Ext}(x, U_d))$ erring with probability at least $1/2$. □

Note that the enumeration incurs a 2^d factor slowdown in the simulation. Thus, for this application, we want to construct extractors where (a) $d = O(\log n)$; (b) Ext is computable in polynomial time; and (c) $m = k^{\Omega(1)}$.

We remark that the error probability in Proposition 5.15 can actually be made exponentially small (say 2^{-t}) by using an extractor that is designed for min-entropy roughly $k - t$ instead of k.

We note that even though seeded extractors suffice for simulating randomized algorithms with only a weak source, they do not suffice for all applications of randomness in theoretical computer science. The trick of eliminating the random seed by enumeration does not work, for example, in cryptographic applications of randomness. Thus the study of deterministic extractors for restricted classes of sources remains a very interesting and active research direction. We, however, will focus on seeded extractors, due to their many applications and their connections to the other pseudorandom objects we are studying.

5.2 Connections with Hash Functions and Expanders

As mentioned earlier, extractors have played a unifying role in the theory of pseudorandomness, through their close connections with a variety of other pseudorandom objects. In this section, we will see two of these connections. Specifically, how by reinterpreting them appropriately, extractors can be viewed as providing families of hash functions, and as being a certain type of highly expanding graphs.

5.2.1 Extractors as Hash Functions

One of the results we saw last time says that for any subset $S \subseteq [N]$ of size K, if we choose a completely random hash function $h : [N] \to [M]$ for $M \ll K$, then h will map the elements of S almost-uniformly to $[M]$. Equivalently, if we let H be distributed uniformly over all functions $h : [N] \to [M]$ and X be uniform on the set S, then $(H, H(X))$ is statistically close to $(H, U_{[M]})$, where we use the notation U_T to denote the uniform distribution on a set T. Can we use a smaller family of hash functions than the set of all functions $h : [N] \to [M]$? This gives rise to the following variant of extractors.

Definition 5.16 (strong extractors). Extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a *strong* (k, ε)*-extractor* if for every k-source X on $\{0,1\}^n$, $(U_d, \text{Ext}(X, U_d))$ is ε-close to (U_d, U_m). Equivalently, $\text{Ext}'(x, y) = (y, \text{Ext}(x, y))$ is a standard (k, ε)-extractor.

The nonconstructive existence proof from last time can be extended to establish the existence of very good strong extractors.

Theorem 5.17. For every $n, k \in \mathbb{N}$ and $\varepsilon > 0$ there exists a strong (k, ε)-extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k - 2\log(\frac{1}{\varepsilon}) - O(1)$ and $d = \log(n - k) + 2\log(\frac{1}{\varepsilon}) + O(1)$.

Note that the output length $m \approx k$ instead of $m \approx k + d$; intuitively a strong extractor needs to extract randomness that is *independent* of the seed and thus can only get the k bits from the source.

We see that strong extractors can be viewed as very small families of hash functions having the almost-uniform mapping property mentioned above. Indeed, our first explicit construction of extractors is obtained by using pairwise independent hash functions.

Theorem 5.18 (Leftover Hash Lemma). If $\mathcal{H} = \{h : \{0,1\}^n \to \{0,1\}^m\}$ is a pairwise independent (or even 2-universal) family of hash functions where $m = k - 2\log(\frac{1}{\varepsilon})$, then $\mathrm{Ext}(x,h) \stackrel{\text{def}}{=} h(x)$ is a strong (k,ε)-extractor. Equivalently, $\mathrm{Ext}(x,h) = (h,h(x))$ is a standard (k,ε)-extractor.

Note that the seed length is $d = O(n)$, i.e., the number of random bits required to choose $h \stackrel{\text{R}}{\leftarrow} \mathcal{H}$. This is far from optimal; for the purposes of simulating randomized algorithms we would like $d = O(\log n)$. However, the output length of the extractor is $m = k - 2\log(\frac{1}{\varepsilon})$, which is optimal up to an additive constant.

Proof. Let X be an arbitrary k-source on $\{0,1\}^n$, $\mathcal{H}$ as above, and $H \stackrel{\text{R}}{\leftarrow} \mathcal{H}$. Let d be the seed length. We show that $(H, H(X))$ is ε-close to $U_d \times U_m$ in the following three steps:

(1) We show that the collision probability of $(H, H(X))$ is close to that of $U_d \times U_m$.
(2) We note that this is equivalent to saying that the ℓ_2 distance between $(H, H(X))$ and $U_d \times U_m$ is small.
(3) Then we deduce that the statistical difference is small, by recalling that the statistical difference equals half of the ℓ_1 distance, which can be (loosely) bounded by the ℓ_2 distance.

Proof of 1: By definition, $\mathrm{CP}(H, H(X)) = \Pr\left[(H, H(X)) = (H', H'(X'))\right]$, where (H', X') is independent of and identically distributed to (H, X). Note that $(H, H(X)) = (H', H'(X))$ if and only if $H = H'$ and either $X = X'$ or $X \neq X'$ but $H(X) = H(X')$. Thus

$$\begin{aligned} \mathrm{CP}(H, H(X)) &= \mathrm{CP}(H) \cdot \Big(\mathrm{CP}(X) + \Pr\left[H(X) = H(X') \mid X \neq X'\right]\Big) \\ &\leq \frac{1}{D}\left(\frac{1}{K} + \frac{1}{M}\right) = \frac{1+\varepsilon^2}{DM}. \end{aligned}$$

To see the penultimate inequality, note that $\mathrm{CP}(H) = 1/D$ because there are D hash functions, $\mathrm{CP}(X) \leq 1/K$ because $\mathrm{H}_\infty(X) \geq k$, and $\Pr\left[H(X) = H(X') \,|X \neq X'\right] = 1/M$ by pairwise independence.

Proof of 2:

$$\begin{aligned} \|(H, H(X)) - U_d \times U_m\|^2 &= \mathrm{CP}(H, H(X)) - \frac{1}{DM} \\ &\leq \frac{1+\varepsilon^2}{DM} - \frac{1}{DM} = \frac{\varepsilon^2}{DM}. \end{aligned}$$

Proof of 3: Recalling that the statistical difference between two random variables X and Y is

equal to $\frac{1}{2}|X - Y|_1$, we have:

$$\begin{aligned}\Delta((H, H(X)), U_d \times U_m) &= \frac{1}{2}\,|(H, H(X)) - U_d \times U_m|_1 \\ &\leq \frac{\sqrt{DM}}{2}\,\|(H, H(X)) - U_d \times U_m\| \\ &\leq \frac{\sqrt{DM}}{2} \cdot \sqrt{\frac{\varepsilon^2}{DM}} \\ &= \frac{\varepsilon}{2}.\end{aligned}$$

Thus, we have in fact obtained a strong $(k, \varepsilon/2)$-extractor. □

The proof above actually shows that $\mathrm{Ext}(x, h) = h(x)$ extracts with respect to collision probability, or equivalently, with respect to the ℓ_2-norm. This property may be expressed in terms of Renyi entropy $\mathrm{H}_2(Z) \stackrel{\text{def}}{=} \log(1/\mathrm{CP}(Z))$. Indeed, we can define $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^m$ to be a (k, ε) *Renyi-entropy extractor* if $\mathrm{H}_2(X) \geq k$ implies $\mathrm{H}_2(\mathrm{Ext}(X, U_d)) \geq m - \varepsilon$ (or $\mathrm{H}_2(U_d, \mathrm{Ext}(X, U_d)) \geq m + d - \varepsilon$ for strong Renyi-entropy extractors). Then the above proof shows that pairwise-independent hash functions yield strong Renyi-entropy extractors.

In general, it turns out that an extractor with respect to Renyi entropy must have seed length $d \geq \Omega(\min\{m, n - k\})$ (as opposed to $d = O(\log n)$); this explains why the seed length in the above extractor is large.

5.2.2 Extractors vs. Expanders

Extractors have a natural interpretation as graphs. Specifically, we can interpret an extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^m$ as the neighbor function of a bipartite multigraph $G = ([N], [M], E)$ with $N = 2^n$ left-vertices, $M = 2^m$ right-vertices, and left-degree $D = 2^d$,[2] where the r'th neighbor of left-vertex u is $\mathrm{Ext}(u, r)$. Typically $n \gg m$, so the graph is very unbalanced. It turns out that the extraction property of Ext is related to various "expansion" properties of G. In this section, we explore this relationship.

Let $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^m$ be a (k, ε)-extractor and $G = ([N], [M], E)$ the associated graph. Recall that it suffices to examine Ext with respect to *flat* k-sources: in this case, the extractor property says that given a subset S of size k on the left, a random neighbor of a random element of S should be close to uniform on the right. In particular, if $S \subseteq [N]$ is a subset on the left of size k, then $|N(S)| \geq (1 - \varepsilon)M$. This property is just like vertex expansion, except that it ensures expansion only for sets of size exactly K, not any size $\leq K$. We call such a graph an $(= K, A)$ *vertex expander*. Indeed, this gives rise to the following weaker variant of extractors.

Definition 5.19 (dispersers). A function $\mathrm{Disp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k, ε)-*disperser* if for every k-source X on $\{0,1\}^n$, $\mathrm{Disp}(X, U_d)$ has a support of size at least $(1 - \varepsilon) \cdot 2^m$.

While extractors can be used to simulate **BPP** algorithms with a weak random source, dispersers can be used to simulate **RP** algorithms with a weak random source.

Then, we have:

[2] This connection is the reason we use d to denote the seed length of an extractor.

Proposition 5.20. A function $\text{Disp} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-disperser iff the corresponding bipartite multigraph $G = ([N],[M],E)$ with left-degree D is an $(=K, A)$ vertex expander for $A = (1-\varepsilon)\cdot M/K$.

Note that extractors and dispersers are interesting even when $M \ll K$, so the expansion parameter A may be less than 1. Indeed, $A < 1$ is interesting for vertex "expanders" when the graph is highly imbalanced. Still, for an *optimal* extractor, we have $M = \Theta(\varepsilon^2 KD)$ (because $m = k + d - 2\log(1/\varepsilon) - \Theta(1)$), which corresponds to expansion factor $A = \Theta(\varepsilon^2 D)$. (An optimal disperser actually gives $A = \Theta(D/\log(1/\varepsilon))$.) Note this is smaller than the expansion factor of $D/2$ in Ramanujan graphs and $D - O(1)$ in random graphs; the reason is that those expansion factors are for "small" sets, whereas here we are asking for sets to expand to almost the entire right-hand side.

Now let's look for a graph-theoretic property that is *equivalent* to the extraction property. Ext is a (k,ε)-extractor iff for every set $S \subseteq [N]$ of size K,

$$\Delta(\text{Ext}(U_S, U_{[D]}), U_{[M]}) = \max_{T \subseteq [M]} \left| \Pr\left[\text{Ext}(U_S, U_{[D]}) \in T\right] - \Pr\left[U_{[M]} \in T\right] \right| \le \varepsilon,$$

where U_S denotes the uniform distribution on S. This inequality may be expressed in graph-theoretic terms as follows. For every set $T \subseteq [M]$,

$$\begin{aligned} & \left| \Pr\left[\text{Ext}(U_S, U_{[D]}) \in T\right] - \Pr\left[U_{[M]} \in T\right] \right| \le \varepsilon \\ \Leftrightarrow \quad & \left| \frac{e(S,T)}{|S|D} - \frac{|T|}{M} \right| \le \varepsilon \\ \Leftrightarrow \quad & \left| \frac{e(S,T)}{ND} - \mu(S)\mu(T) \right| \le \varepsilon\mu(S), \end{aligned}$$

where $e(S,T)$ is the number of edges from S to T.

Thus, we have:

Proposition 5.21. Ext is a (k,ε)-extractor iff the corresponding bipartite multigraph $G = ([N],[M],E)$ with left-degree D has the property that $|e(S,T)/ND - \mu(S)\mu(T)| \le \varepsilon\mu(S)$ for every $S \subseteq [N]$ of size K and every $T \subseteq [M]$.

Note that this is very similar to the Expander Mixing Lemma (Lemma 4.15) which states that if a graph G has spectral expansion λ, then for *all* sets $S, T \subseteq [N]$ we have

$$\left| \frac{e(S,T)}{ND} - \mu(S)\mu(T) \right| \le \lambda\sqrt{\mu(S)\mu(T)}.$$

It follows that if $\lambda\sqrt{\mu(S)\mu(T)} \le \varepsilon\mu(S)$ for all $S \subseteq [N]$ of size K and all $T \subseteq [N]$, then G gives rise to a (k,ε)-extractor (by turning G into a D-regular bipartite graph with N vertices on each side in the natural way). It suffices for $\lambda \le \varepsilon \cdot \sqrt{K/N}$ for this to work.

We can use this connection to turn our explicit construction of spectral expanders into an explicit construction of extractors. To achieve $\lambda \le \varepsilon \cdot \sqrt{K/N}$, we can take an appropriate power of

a constant-degree expander. Specifically, if G_0 is a D_0-regular expander on N vertices with bounded second eigenvalue, we can consider the tth power of G_0, $G = G_0^t$, where $t = O(\log((1/\varepsilon)\sqrt{N/K})) = O(n - k + \log(1/\varepsilon))$. The degree of G is $D = D_0^t$, so $d = \log D = O(t)$. This yields the following result:

Theorem 5.22. For every $n, k \in \mathbb{N}$ and $\varepsilon > 0$, there is an explicit (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \longrightarrow \{0,1\}^n$ with $d = O(n - k + \log(1/\varepsilon))$.

Note that the seed length is significantly better than in the construction from pairwise-independent hashing when k is close to n, say $k \geq n - O(\log n)$ (i.e. $K = \Omega(N/\log N)$). The output length is just n, which is much larger than the typical output length for extractors (usually $m \ll n$). Using a Ramanujan graph (rather than an arbitrary constant-degree expander), the seed length can be improved to $d = n - k + 2\log(1/\varepsilon) + O(1)$, which yields an optimal output length $n = k + d - 2\log(1/\varepsilon) - O(1)$.

Another way of proving Theorem 5.22 is to use the fact that a random step on an expanders decreases the ℓ_2 distance to uniform, like in the proof of the Leftover Hash Lemma. This analysis shows that we actually get a Renyi-entropy extractor; and thus explains the large seed length $d \approx n - k$.

The following table summarizes the main differences between "classic" expanders and extractors.

Expanders	**Extractors**
Measured by vertex or spectral expansion	Measured by min-entropy/statistical difference
Typically constant degree	Typically logarithmic or poly-logarithmic degree
All sets of size *at most* K expand	All sets of size *exactly* (or at least) K expand
Typically balanced	Typically unbalanced, bipartite graphs

Fig. 5.1 Differences between "classic" expanders and extractors

5.3 Constructing Extractors

In the previous sections, we have seen that very good extractors exist — extracting almost all of the min-entropy from a source with only a logarithmic seed length. But the explicit constructions we have seen (via pairwise-independent hashing and spectral expanders) are still quite far from optimal in seed length, and in particular cannot be used to give a polynomial-time simulation of **BPP** with a weak random source.

Fortunately, much better extractor constructions are known — ones that extract any constant fraction of the min-entropy using a logarithmic seed length, or extract all of the min-entropy using a polylogarithmic seed length. In this lecture, we will see how to construct such extractors (assuming a certain *condenser* construction that we will see in Chapter 6).

5.3.1 Block Sources

We introduce a useful model of sources that has more structure than an arbitrary k-source:

Definition 5.23. $X = (X_1, X_2, \ldots, X_t)$ is a $(k_1, k_2, \ldots, k_t)$ *block source* if for every $x_1, \ldots, x_{i-1}$, $X_i|_{X_1=x_1,\ldots,X_{i-1}=x_{i-1}}$ is a k_i-source. If $k_1 = k_2 = \cdots = k_t = k$, then we call X a $t \times k$ *block source*.

Note that a $(k_1, k_2, \ldots, k_t)$ block source is also a $(k_1 + \cdots + k_t)$-source, but it comes with additional structure — each block is guaranteed to contribute some min-entropy. Thus, extracting randomness from block sources is an easier task than extracting from general sources.

The study of block sources has a couple of motivations.

- They are a natural and plausible model of sources in their own right. Indeed, they are more general than unpredictable-bit sources of Section 5.1.1: if $X \in \text{UnpredBits}_{n,\delta}$ is broken into t blocks of length $\ell = n/t$, then the result is a $t \times \delta'\ell$ block source, where $\delta' = \log(1/(1-\delta))$.
- We can construct extractors for general weak sources by converting a general weak source into a block source. We will see how to do this later in the lecture.

We now illustrate how extracting from block sources is easier than from general sources. The idea is that we can extract almost-uniform bits from later blocks that are essentially independent of earlier blocks, and hence use these as a seed to extract more bits from the earlier blocks. Specifically, for the case of two blocks we have the following:

Lemma 5.24. Let $\text{Ext}_1 : \{0,1\}^{n_1} \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ be a (k_1, ε_1)-extractor, and $\text{Ext}_2 : \{0,1\}^{n_2} \times \{0,1\}^{d_2} \to \{0,1\}^{m_2}$ be a (k_2, ε_2)-extractor with $m_2 \geq d_1$. Define $\text{Ext}'((x_1, x_2), y_2) = (\text{Ext}_1(x_1, y_1), z_2)$, where (y_1, z_2) is obtained by partitioning $\text{Ext}_2(x_2, y_2)$ into a prefix y_1 of length d_1 and a suffix z_2 of length $m_2 - d_1$.

Then for every (k_1, k_2) block source $X = (X_1, X_2)$ taking values in $\{0,1\}^{n_1} \times \{0,1\}^{n_2}$, it holds that $\text{Ext}'(X, U_{d_2})$ is $(\varepsilon_1 + \varepsilon_2)$-close to $U_{m_1} \times U_{m_2-d_1}$.

Proof. Since X_2 is a k_2-source conditioned on any value of X_1 and Ext_2 is a (k_2, ε_2)-extractor, it follows that $(X_1, Y_1, Z_2) = (X_1, \text{Ext}_2(X_2, U_{d_2}))$ is ε_2-close to $(X_1, U_{m_2}) = (X_1, U_{d_1}, U_{m_2-d_1})$.

Thus, $(\text{Ext}_1(X_1, Y_1), Z_2)$ is ε_2-close to $(\text{Ext}_1(X_1, U_{d_1}), U_{m_2-d_1})$, which is ε_1-close to $(U_{m_1}, U_{m_2-d_1})$ because X_1 is a k_1-source and Ext_1 is a (k_1, ε_1)-extractor.

By the triangle inequality, $\text{Ext}'(X, U_{d_2}) = (\text{Ext}_1(X_1, Y_1), Z_2)$ is $(\varepsilon_1 + \varepsilon_2)$-close to $(U_{m_1}, U_{m_2-d_1})$. □

The benefit of this composition is that the seed length of Ext' depends only one of the extractors (namely Ext_2) rather than being the sum of the seed lengths. (If this is reminiscent of the zig-zag product, it is because they are closely related — see Section 5.4.2). Thus, we get to extract from multiple blocks at the "price of one." Moreover, since we can take $d_1 = m_2$, which is typically much larger than d_2, the seed length of Ext' can even be smaller than that of Ext_1.

The lemma extends naturally to extracting from many blocks:

Lemma 5.25. For $i = 1, \ldots, t$, let $\text{Ext}_i : \{0,1\}^{n_i} \times \{0,1\}^{d_i} \to \{0,1\}^{m_i}$ be a (k_i, ε_i)-extractor, and suppose that $m_i \geq d_{i-1}$ for every $i = 1, \ldots, t$, where we define $d_0 = 0$. Define $\text{Ext}'((x_1, \ldots, x_t), y_t) =$

$(z_1, \ldots, z_t)$, where for $i = t, \ldots, 1$, we inductively define (y_{i-1}, z_i) to be a partition of $\mathrm{Ext}_i(x_i, y_i)$ into a d_{i-1}-bit prefix and a $(m_i - d_{i-1})$-bit suffix.

Then for every $(k_1, \ldots, k_t)$ block source $X = (X_1, \ldots, X_t)$ taking values in $\{0,1\}^{n_1} \times \cdots \{0,1\}^{n_t}$, it holds that $\mathrm{Ext}'(X, U_{d_t})$ is ε-close to U_m for $\varepsilon = \sum_{i=1}^{t} \varepsilon_i$ and $m = \sum_{i=1}^{t}(m_i - d_{i-1})$.

We remark that this composition preserves "strongness." If each of the Ext_i's correspond to strong extractors in the sense that their seeds are prefixes of their outputs, then Ext' will also correspond to a strong extractor. If in addition $d_1 = d_2 = \cdots = d_t$, then this construction can be seen as simply using the same seed to extract from all blocks.

Already with this simple composition, we can simulate **BPP** with an unpredictable-bit source (even though deterministic extraction from such sources is impossible by Proposition 5.6). As noted above, by breaking an unpredictable-bit source X with parameter δ into blocks of length ℓ, we obtain a $t \times k$ block source for $t = n/\ell$, $k = \delta'\ell$, and $\delta' = \log(1/(1-\delta))$.

Suppose that δ is a constant. Set $\ell = (10/\delta')\log n$, so that X is a $t \times k$ block source for $k = 10 \log n$, and define $\varepsilon = n^{-2}$. Letting $\mathrm{Ext} : \{0,1\}^{\ell} \times \{0,1\}^{d} \to \{0,1\}^{d+m}$ be the (k, ε) extractor using pairwise-independent hash functions (Theorem 5.18), we have:

$$\begin{aligned} d &= O(\ell) = O(\log n) \qquad \text{and} \\ m &= k - 2\log\frac{1}{\varepsilon} - O(1) > k/2 \end{aligned}$$

Composing Ext with itself t times as in Lemma 5.24, we obtain $\mathrm{Ext}' : \{0,1\}^{t\cdot\ell} \times \{0,1\}^{d} \to \{0,1\}^{d+t\cdot m}$ such that $\mathrm{Ext}'(X, U_d)$ is ε'-close to uniform, for $\varepsilon' = 1/n$. (Specifically, $\mathrm{Ext}'((x_1, \ldots, x_t), h) = (h, h(x_1), \ldots, h(x_t))$.) This tells us that Ext' essentially extracts half of the min-entropy from X, given a random seed of logarithmic length. Plugging this extractor into the construction of Proposition 5.15 gives us the following result.

Theorem 5.26. For every constant $\delta > 0$, we can simulate **BPP** with an unpredictable-bit source of parameter δ. More precisely, for every $L \in \mathbf{BPP}$ and every constant $\delta > 0$, there is a polynomial-time algorithm A and a polynomial q such that for every $w \in \{0,1\}^*$ and every source $X \in \mathrm{UnpredBits}_{q(|w|),\delta}$, the probability that $A(w; X)$ errs is at most $1/|w|$.

5.3.2 Reducing General Sources to Block Sources

Given the results of the previous section, a common approach to constructing extractors for general k-sources is to reduce the case of general k-sources to that of block sources.

One approach to doing this is as follows. Given a k-source X of length n, where $k = \delta n$, pick a (pseudo)random subset S of the bits of X, and let $W = X|_S$ be the bits of X in those positions. If the set S is of size ℓ, then we expect that W will have at least roughly $\delta\ell$ bits of min-entropy (with high probability over the choice of S). Moreover, W can have at most ℓ bits of min-entropy, so if $\ell < \delta n$, intuitively there must still should be at least min-entropy left in X. (This is justified by Lemma 5.27 below.) Thus, the pair (W, X) should be a block source. This approach can be shown to work for appropriate ways of sampling the set S, and recursive applications of it was the original approach to constructing good extractors (and is still useful in various contexts today). The fact mentioned above, that conditioning on a string of length ℓ reduces min-entropy by at most ℓ bits, is given by the following lemma (which is very useful when working with min-entropy) .

Lemma 5.27 (chain rule for min-entropy). If (W, X) are two jointly distributed random variables, where (W, X) is a k-source and W has length at most ℓ, then for every $\varepsilon > 0$, it holds that with probability at least $1 - \varepsilon$ over $w \stackrel{R}{\leftarrow} W$, $X|_{W=w}$ is a $(k - \ell - \log(1/\varepsilon))$-source.

This is referred to as the "chain rule" for min-entropy by analogy with the chain rule for Shannon entropy, which states that $\mathrm{H}_{Sh}(X|W) = \mathrm{H}_{Sh}(W, X) - \mathrm{H}_{Sh}(W)$, where the conditional Shannon entropy is defined to be $\mathrm{H}_{Sh}(X|W) = \mathrm{E}_{w \stackrel{R}{\leftarrow} W}[\mathrm{H}_{Sh}(X|_{W=w})]$. Thus, if $\mathrm{H}_{Sh}(W, X) \geq k$ and W is of length at most ℓ, we have $\mathrm{H}_{Sh}(X|W) \geq k - \ell$. The chain rule for min-entropy is not quite as clean; we need to assume that W has small support (rather than just small min-entropy) and we lose $\log(1/\varepsilon)$ bits of additional min-entropy.

Another approach, which we will follow, is based on the observation that every source of high min-entropy rate (namely, greater than 1/2) is (close to) a block source, as shown by the lemma below. Thus, we will try to convert arbitrary sources into ones of high min-entropy rate

Lemma 5.28. If X is an $(n - \Delta)$-source of length n, and $X = (X_1, X_2)$ is a partition of X into blocks of lengths n_1 and n_2, then for every $\varepsilon > 0$, (X_1, X_2) is ε-close to some $(n_1 - \Delta, n_2 - \Delta - \log(1/\varepsilon))$ block source.

Consider $\Delta = \alpha n$ for a constant $\alpha < 1/2$, and $n_1 = n_2 = n/2$. Then each block contributes min-entropy at least $(1/2 - \alpha)n$. The proofs of Lemmas 5.27 and 5.28 are left as exercises (Problem 5.1).

We are still left with the question of converting a general k-source into one of high min-entropy rate. We will do this via the following kind of object.

Definition 5.29. A function $\mathrm{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $k \to_\varepsilon k'$ *condenser* if for every k-source X on $\{0,1\}^n$, $\mathrm{Con}(X, U_d)$ is ε-close to some k'-source. Con is *lossless* if $k' = k + d$.

If $k'/m > k/n$, then the condenser increases the min-entropy rate, intuitively making extraction an easier task.

In Chapter 6, we will construct the following kind of condenser, using connections with both expander graphs and list-decodable error-correcting codes:

Theorem 5.30. For every constant $\alpha > 0$, for all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an explicit

$$k \to_\varepsilon k + d$$

lossless condenser $C : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d = O(\log n + \log(1/\varepsilon))$ and $m = (1 + \alpha)k + O(\log(n/\varepsilon))$.

Note that setting α to be a small constant, we obtain an output min-entropy rate arbitrarily close to 1.

5.3.3 The Extractor

In this section, we will use the ideas outlined in the previous section — namely condensing and block-source extraction — to construct an extractor that is optimal up to constant factors (assuming the condenser of Theorem 5.30).

Theorem 5.31. Assume Theorem 5.30. Then for all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an explicit (k, ε) extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq k/2$ and $d = O(\log(n/\varepsilon))$.

We will use the following building block, constructed in Problem 5.5.

Lemma 5.32. Assume Theorem 5.30. Then for every *constant* $t > 0$ and all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an explicit (k, ε) extractor $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq k/2$ and $d = k/t + O(\log(n/\varepsilon))$.

The point is that this extractor has a seed length that is an arbitrarily large constant factor (approximately $t/2$) smaller than its output length. Thus, if we use it as Ext_2 in the block-source extraction of Lemma 5.24, the resulting seed length will be smaller than that of Ext_1 by an arbitrarily large constant factor. (The seed length of the composed extractor Ext' in Lemma 5.24 is the same of that as Ext_2, which will be a constant factor smaller than its output length m_2, which we can take to be equal to the seed length d_1 of Ext_1.)

Overview of the Construction. Note that for small min-entropies k, namely $k = O(\log(n/\varepsilon))$, the extractor we want is already given by Lemma 5.32 with seed length d smaller than the output length m by any constant factor. (If we allow $d \geq m$, then extraction is trivial — just output the seed.) Thus, our goal will be to recursively construct extractors for large min-entropies using extractors for smaller min-entropies. Of course, if $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k_0, ε) extractor, say with $m = k_0/2$, then it is also a (k, ε) extractor for every $k \geq k_0$. The problem is that the output length is only $k_0/2$ rather than $k/2$. Thus, we need to increase the output length. This can be achieved by simply applying extractors for smaller min-entropies several times:

Lemma 5.33. Suppose $\mathrm{Ext}_1 : \{0,1\}^n \times \{0,1\}^{d_1} \to \{0,1\}^{m_1}$ is a (k_1, ε_1) extractor and $\mathrm{Ext}_2 : \{0,1\}^n \times \{0,1\}^{d_2} \to \{0,1\}^{m_2}$ is a (k_2, ε_2) extractor for $k_2 = k_1 - m_1 - \log(1/\varepsilon_3)$. Then $\mathrm{Ext}' : \{0,1\}^n \times \{0,1\}^{d_1+d_2} \to \{0,1\}^{m_1+m_2}$ is a $(k_1, \varepsilon_1 + \varepsilon_2 + \varepsilon_3)$ extractor.

The proof of this lemma follows from Lemma 5.27. After conditioning a k_1-source X on $W = \mathrm{Ext}_1(X, U_{d_1})$, X still has min-entropy at least $k_1 - m_1 - \log(1/\varepsilon_3) = k_2$ (except with probability ε_3), and thus $\mathrm{Ext}_2(X, U_{d_2})$ can extract an additional m_2 almost-uniform bits.

To see how we might apply this, consider setting $k_1 = .8k$ and $m_1 = k_1/2$, $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \varepsilon \geq 2^{-.1k}$, $k_2 = k_1 - m_1 - \log(1/\varepsilon_3) \in [.3k, .4k]$, and $m_2 = k_2/2$. Then we obtain a $(k, 3\varepsilon)$ extractor Ext' with output length $m = m_1 + m_2 > k/2$ from two extractors for min-entropies k_1, k_2 that are smaller than k by a constant factor, and we can hope to construct the latter two extractors recursively via the same construction.

Now, however, the problem is that the seed length grows by a constant factor in each level of recursion (e.g. if $d_1 = d_2 = d$ in Lemma 5.33, we get seed length $2d$ rather than d). Fortunately, as mentioned above, block source extraction using the extractor of Lemma 5.32 gives us a method to reduce the seed length by a constant factor. In order to apply block source extraction, we first need to convert our source to a block source; by Lemma 5.28, we can do this by using our condenser to make its entropy rate close to 1.

One remaining issue is that the error ε still grows by a constant factor in each level of recursion. However, we can start with polynomially small error at the base of the recursion and there are only logarithmically many levels of recursion, so we can afford this blow-up.

We now proceed with the proof details. It will be notationally convenient to do the steps in the reverse order from the description above — first we will reduce the seed length by a constant factor via block-source extraction, and then apply Lemma 5.33 to increase the output length.

Proof. [Theorem 5.31] Fix $n \in \mathbb{N}$ and $\varepsilon_0 > 0$. Set $d = c\log(n/\varepsilon_0)$ for an error parameter ε_0 and a sufficiently large constant c to be determined in the proof below. (To avoid ambiguity, we will keep the dependence on c explicit throughout the proof, and all big-Oh notation hides universal constants independent of c.) For $k \in [0, n]$, let $i(k)$ be the smallest nonnegative integer i such that $k \leq 2^i \cdot 8d$. This will be the level of recursion in which we handle min-entropy k; note that $i(k) \leq \log k \leq \log n$.

For every $k \in [0, n]$, we will construct an explicit $\mathrm{Ext}_k : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{k/2}$ that is a $(k, \varepsilon_{i(k)})$ extractor, for an appropriate sequence $\varepsilon_0 \leq \varepsilon_1 \leq \varepsilon_2 \cdots$. Note that we require the seed length to remain d and the fraction of min-entropy extracted to remain $1/2$ for all values of k. The construction will be by induction on $i(k)$.

Base Case: $i(k) = 0$, i.e. $k \leq 8d$. The construction of Ext_k follows from Lemma 5.32, setting $t = 9$ and taking c to be a sufficiently large constant.

Inductive Case: We construct Ext_k for $i(k) \geq 1$ from extractors $\mathrm{Ext}_{k'}$ with $i(k') < i(k)$ as follows. Given a k-source X of length n, Ext_k works as follows.

(1) We apply our condenser (Theorem 5.30) to convert X into a source X' that is ε_0-close to a k-source of length $(9/8)k + O(\log(n/\varepsilon_0))$. This requires a seed of length $O(\log(n/\varepsilon_0))$.

(2) We divide X' into two equal-sized halves (X_1, X_2). By Lemma 5.28, (X_1, X_2) is $2\varepsilon_0$-close to a $2 \times k'$ block source for

$$k' = k/2 - k/8 - O(\log(n/\varepsilon_0)) \ .$$

Note that $i(k') < i(k)$. Since $i(k) \geq 1$, we also have $k' \geq 3d - O(\log(n/\varepsilon_0)) \geq 2d$, for a sufficiently large choice of the constant c.

(3) Now we apply block-source extraction as in Lemma 5.24. We take Ext_2 to be a $(2d, \varepsilon_0)$ extractor from Lemma 5.32 with parameter $t = 16$, which will give us $m_2 = d$ output bits using a seed of length $d_2 = (2d)/16 + O(\log(n/\varepsilon_0))$. For Ext_1, we use our recursively constructed $\mathrm{Ext}_{k'}$, which has seed length d, error $\varepsilon_{i(k')}$, and output length $k'/2 \geq k/6$ (where the latter inequality holds for a sufficiently large choice of the constant c, because $k > 8d > 8c\log(1/\varepsilon)$).

All in all, our extractor so far has seed length at most $d/8 + O(\log(n/\varepsilon_0))$, error at most $\varepsilon_{i(k)-1} + O(\varepsilon_0)$, and output length at least $k/6$. This would be sufficient for our induction except that the output length is only $k/6$ rather than $k/2$. We remedy this by applying Lemma 5.33.

With one application of the extractor above, we extract at least $m_1 = k/6$ bits of the source min-entropy. Then with another application of the extractor above for min-entropy threshold $k_2 =$

$k - m_1 - \log(1/\varepsilon) = 5k/6 - \log(1/\varepsilon)$, by Lemma 5.33, we extract another $(5k/6 - \log(1/\varepsilon))/6$ bits and so on. After four applications, we have extracted all but $(5/6)^4 \cdot k + O(\log(1/\varepsilon)) \leq k/2$ bits of the min-entropy. Our seed length is then $4 \cdot (d/8 + O(\log(n/\varepsilon_0))) \leq d$ and the total error is $\varepsilon_{i(k)} = O(\varepsilon_{i(k)-1})$.

Solving the recurrence for the error, we get $\varepsilon_i = 2^{O(i)} \cdot \varepsilon_0 \leq \text{poly}(n) \cdot \varepsilon_0$, so we can obtain error ε by setting $\varepsilon_0 = \varepsilon/\text{poly}(n)$. As far as explicitness, we note that computing Ext_k consists of four evaluations of our condenser from Theorem 5.30, four evaluations of $\text{Ext}_{k'}$ for values of k' such that $i(k') < (i(k) - 1)$, four evaluations of the explicit extractor from Lemma 5.32, and simple string manipulations that can be done in time $\text{poly}(n, d)$. Thus, the total computation time is at most $4^{i(k)} \cdot \text{poly}(n, d) = \text{poly}(n, d)$. □

Repeatedly applying Lemma 5.33 using extractors from Theorem 5.31, we can extract any constant fraction of the min-entropy using a logarithmic seed length, and all the min-entropy using a polylogarithmic seed length.

Corollary 5.34. Assume Theorem 5.30. Then the following holds for every constant $\alpha > 0$. For every $n \in \mathbb{N}$, $k \in [0, n]$, and $\varepsilon > 0$, there is an explicit (k, ε) extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m \geq (1 - \alpha)k$ and $d = O(\log(n/\varepsilon))$.

Corollary 5.35. Assume Theorem 5.30. Then for every $n \in \mathbb{N}$, $k \in [0, n]$, and $\varepsilon > 0$, there is an explicit (k, ε) extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k - O(\log(1/\varepsilon))$ and $d = O(\log k \cdot \log(n/\varepsilon))$.

We remark that the above construction can be modified to yield *strong* extractors achieving the same output lengths as above (so the entropy of the seed need not be lost in Corollary 5.35).

A summary of the extractor parameters we have seen is in Table 5.1.

Method	Seed Length d	Output Length m
Optimal and Nonconstructive	$\log(n-k) + O(1)$	$k + d - O(1)$
Necessary for **BPP** Simulation	$O(\log n)$	$k^{\Omega(1)}$
Spectral Expanders	$O(n - k)$	n
Pairwise Independent Hashing	$O(n)$	$k + d - O(1)$
Corollary 5.34	$O(\log n)$	$(1-\gamma)k$, any constant $\gamma > 0$
Corollary 5.35	$O(\log^2 n)$	$k - O(1)$

Table 5.1 Parameters for some constructions of $(k, .01)$ extractors.

While Theorem 5.31 and Corollary 5.34 give extractors that are optimal up to constant factors in both the seed length and output length, it remains an important open problem to get one or both of these to be optimal to within an *additive* constants while keeping the other optimal to within a constant factor.

Open Problem 5.36. Give an explicit construction of $(k, .01)$ extractors $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with seed length $d = O(\log n)$ and output length $m = k + d - O(1)$ (or even $m = (1 - o(1)) \cdot k$).

By using the condenser of Theorem 5.30, it suffices to solve achieve the above for high min-entropy rate, e.g. $k = .99n$.

Open Problem 5.37. Give an explicit construction of $(k, .01)$ extractors $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with seed length $d = \log n + O(1)$ and $m = \Omega(k)$ (or even $m = k^{\Omega(1)}$).

One of the reasons that these open problems are significant is that, in many applications of extractors, the resulting complexity depends *exponentially* on the seed length d and/or the entropy loss $k + d - m$. (An example is the simulation of **BPP** with weak random sources given by Proposition 5.15.) Thus, additive constants in these parameters corresponds to constant factors in complexity.

Another open problem is more aesthetic in nature. The construction of Theorem 5.31 makes use of the condenser of Theorem 5.31, the Leftover Hash Lemma (Theorem 5.18) and the composition techniques of Lemmas 5.24 and Lemma 5.33 in a somewhat complex recursion. It is of interest to have a construction that is more direct. In addition to the aesthetic appeal, such a construction would likely be more practical to implement and provide more insight into extractors. In Chapter 7, we will see a very direct construction based on a connection between extractors and pseudorandom generators, but its parameters will be somewhat worse than Theorem 5.31. Thus the following remains open:

Theorem 5.38. Give a "direct" construction of (k, ε) extractors $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with seed length $d = O(\log(n/\varepsilon))$ and $m = \Omega(k)$.

5.4 More Connections with Expanders

5.4.1 Lossless Condensers vs. Expanders

In the previous section, we saw the notion of a $k \to_\varepsilon k'$ *condenser* $\text{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where for every k-source X on $\{0,1\}^n$, $\text{Con}(X, U_d)$ is ε-close to some k'-source. We can define the *entropy loss* of a condenser to be $\ell = k + d - k'$. As we have discussed, an extractor (i.e. $m = k'$) must have $\ell \geq 2\log(1/\varepsilon) - O(1)$, whereas if we allow m to be larger than k' (specifically, $m \geq k' + \log(1/\varepsilon) + O(1)$), then it is possible for a condenser to be *lossless* (i.e. have $\ell = 0$).

As we have seen for extractors in Section 5.2.2, every function $\text{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ can be viewed as a bipartite multigraph G with $N = 2^n$ left vertices, left-degree $D = 2^d$, and $M = 2^m$ right-vertices where the y'th neighbor of left-vertex x is $\text{Con}(x, y)$. Generalizing what we showed for extractors, the condenser property implies a vertex-expansion property of the graph G. Specifically, recalling that an $(= K, A)$ vertex expander is one in which sets of size exactly K expand by a factor of A, then we have:

Proposition 5.39. If $\text{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $k \to_\varepsilon k'$ condenser, then the corresponding bipartite graph is a $(= K, A)$ vertex expander for $A = (1-\varepsilon) \cdot K'/K = (1-\varepsilon) \cdot D/L$, where $K = 2^k$, $K' = 2^{k'}$, $D = 2^d$, $L = 2^\ell$, and $\ell = k + d - k'$ is the entropy loss of Con.

Thus, if $L < (1-\varepsilon) \cdot D$, the expansion factor A is in fact bigger than 1 (but the case $A < 1$ is still interesting and nontrivial, because the graph is unbalanced). In general, the vertex expansion property is weaker than the property of being a condenser — for example, when $m = k'$, it corresponds to a *disperser* rather than an extractor (Proposition 5.20). However, for the special case of *lossless* condensers (i.e. $L = 1$), it turns out that the two properties are equivalent.

Proposition 5.40. A function $\text{Con} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $k \to_\varepsilon k+d$ (lossless) condenser *if and only if* the corresponding bipartite graph is a $(= K, A)$ vertex expander for $A = (1-\varepsilon) \cdot D$, where $K = 2^k$, $K' = 2^{k'}$, and $D = 2^d$ (provided $2^k \in \mathbb{N}$).

Proof. The "only if" direction follows from Proposition 5.39, so we only prove the "if" direction. Assume that the bipartite graph corresponding to Con is a $(= K, (1-\varepsilon) \cdot D)$ vertex expander. To show that Con is a lossless condenser, by Lemma 5.10 it suffices to show that $\text{Con}(X, U_d)$ is ε-close to a k'-source for every *flat* k-source X. Let S be the support of X, which is of size K. By the vertex expansion of the graph, $|N(S)| \geq (1-\varepsilon) \cdot DK$. Since there are only DK edges leaving S, we can make all of these edges lead to distinct vertices by shifting an ε fraction of them. Let $T \subset [M]$ be the set of KD vertices hit after this shifting. Then $\text{Con}(X, U_d)$ is ε-close to the uniform distribution on T, which is a $(k+d)$-source. □

Thus, the lossless condenser that we assumed in the previous lecture follows immediately from the following expander:

Theorem 5.41. For every constant $\alpha > 0$, every $N \in \mathbb{N}$, $K \leq N$, and $\varepsilon > 0$, there is an explicit $(K, (1-\varepsilon)D)$ expander with N left-vertices, M right-vertices, left-degree $D = O((\log N)(\log K)/\varepsilon)^{1+1/\alpha}$ and $M \leq D^2 \cdot K^{1+\alpha}$. Moreover, D is a power of 2.

We will construct this expander in Chapter 6, using ideas based on list-decodable error-correcting codes.

Note that the kind of expander given by this theorem can be used for the data structure application in Problem 4.6 — storing a $K/2$-sized subset $S \subseteq [N]$ with $M = K^{1+\alpha} \cdot \text{polylog}(N)$ bits in such a way that membership can be probabilistically tested by reading only 1 bit of the data structure. (An efficient solution to this application actually requires more than the graph being explicit in the usual sense, but also that there are efficient algorithms for finding all left-vertices having at least a δ fraction neighbors in a given set $T \subseteq [M]$ of right vertices, but it turns out that the expanders we will construct have this property.)

A deficiency of the expander of Theorem 5.41 is that the size of the right-hand side is polynomial in K and D (for constant α), whereas the optimal bound is $M = O(KD/\varepsilon)$. Achieving the latter, while keeping the left-degree polylogarithmic, is an open problem:

Open Problem 5.42. Construct $(= K, A)$ bipartite expanders with N left-vertices, degree $D = \text{poly}(\log N)$, expansion $A = .99D$, and $M = O(KD)$ right-hand vertices.

We remark that a construction where D is *quasipolynomial* in $\log N$ is known. By Proposition 5.40, a solution to Open Problem 5.42 would give lossless condensers whose output is of extremely high min-entropy ($k' = m - O(1)$), and thus we could get extractors that extract all the min-entropy by then applying extractors based on spectral expanders (Theorem 5.22), thereby also solving Open Problem 5.36.

5.4.2 Block-Source Extraction vs. the Zig-Zag Product

Recall the block-source extraction method presented last time. We define $\text{Ext}' : \{0,1\}^{n_1+n_2} \times \{0,1\}^{d_2} \to \{0,1\}^{m_1}$ by $\text{Ext}'((x_1, x_2), y_2) = \text{Ext}_1(x_1, \text{Ext}_2(x_2, y_2))$. (Here we consider the special case that $m_2 = d_1$.)

Viewing the extractors as bipartite graphs, the left-vertex set is $[N_1] \times [N_2]$ and the left-degree is D_2. A random step from a vertex $(x_1, x_2) \in [N_1] \times [N_2]$ corresponds to taking a random step from x_2 in G_2 to obtain a right-hand vertex $y_1 \in \{0,1\}^{m_2}$, which we view as an edge label y for G_1. We then move to the y'th neighbor of x_1.

This is just like the first two steps of the zig-zag graph product. Why do we need a third step in the zig-zag product? It is because of the slightly different goals in the two setting. In a (spectral) expander, we consider an arbitrary initial distribution that does not have too much (Renyi) entropy, and need to add entropy to it. In a block-source extractor, our initial distribution is constrained to be a block source (so both blocks have a certain amount of min-entropy), and our goal is to produce an almost-uniform output (even if we end up with less bits than the initial entropy).

Thus, in the zig-zag setting, we must consider the following extreme cases (that are ruled out for block sources):

- The second block has no entropy given the first. Here, the step using G_2 will add entropy, but not enough to make y_1 close to uniform. Thus, we have no guarantees on the behavior of the G_1-step, and we may lose entropy with it. For this reason, we keep track of the edge used in the G_1-step — that is, we remember b_1 such that x_1 is the b_1'th neighbor of $z_1 = \text{Ext}(x_1, y_1)$. This ensures that the (edge-rotation) mapping $(x_1, y_1) \mapsto (z_1, b_1)$ is a permutation and does not lose any entropy. We can think of b_1 as a "buffer" that retains any extra entropy in (x_1, y_1) that did not get extracted into z_1. So a natural idea is to just do block source extraction, but output (z_1, b_1) rather than just z_1. However, this runs into trouble with the next case.
- The first block has no entropy but the second block is completely uniform given the first. In this case, the G_2 step cannot add any entropy and the G_1 step does not add any entropy because it is a permutation. However, the G_1 step transfers entropy into z_1. So if we add another expander-step from b_1 at the end, we can argue that it will add entropy. This gives rise to the 3-step definition of the zig-zag product.

While we analyzed the zig-zag product with respect to spectral expansion (i.e. Rényi entropy), it is also possible to do analyze it in terms of a condenser-like definition (i.e. distributions ε-close

to having some min-entropy). all" It turns out that a variant of the zig-zag product for condensers leads to a construction of constant-degree bipartite expanders with expansion $(1-\varepsilon)\cdot D$ for the balanced ($M = N$) or slightly unbalanced (e.g. $M = \Omega(N)$) case. However, as mentioned in Open Problems 4.39, 4.40, and 5.42, there are still several significant open problems concerning the explicit construction of expanders with vertex expansion close to the degree, involving achieving expansion $D - O(1)$, the non-bipartite case, and achieving a near-optimal number of right-hand vertices.

5.5 Exercises

Problem 5.1. (Min-entropy and Statistical Difference)

(1) Prove that for every two random variables X and Y,

$$\Delta(X,Y) = \max_f |\operatorname{E}[f(X)] - \operatorname{E}[f(Y)]| = \frac{1}{2}\cdot |X-Y|_1,$$

where the maximum is over all $[0,1]$-valued functions f. (Hint: first identify the functions f that maximize $|\operatorname{E}[f(X)] - \operatorname{E}[f(Y)]|$.)

(2) Suppose that (W,X) are jointly distributed random variables where W takes values in $\{0,1\}^\ell$ and (W,X) is a k-source. Show that for every $\varepsilon > 0$, with probability at least $1-\varepsilon$ over $w \stackrel{R}{\leftarrow} W$, we have $X|_{W=w}$ is a $(k-\ell-\log(1/\varepsilon))$-source.

(3) Suppose that X is an $(n-\Delta)$-source taking values in $\{0,1\}^n$, and we let X_1 consist of the first n_1 bits of X and X_2 the remaining $n_2 = n - n_1$ bits. Show that for every $\varepsilon > 0$, (X_1, X_2) is ε-close to some $(n_1 - \Delta, n_2 - \Delta - \log(1/\varepsilon))$ block source.

Problem 5.2. (Extractors vs. Samplers) One of the problems we have revisited several times is that of randomness-efficient sampling: Given oracle access to a function $f : \{0,1\}^m \to [0,1]$, approximate its average value $\mu(f)$ to within some small additive error. Most of the samplers we have seen work as follows: they choose some n random bits, use these to decide on some D samples $z_1, \ldots, z_D \in \{0,1\}^m$, and output the average of $f(z_1), \ldots, f(z_D)$. We call such a procedure a (δ,ε)-*averaging sampler* if, for any function f, the probability that the sampler's output differs from $\mu(f)$ by more than ε is at most δ. In this problem, we will see that averaging samplers are essentially equivalent to extractors.

Given $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, we obtain a sampler Smp which chooses $x \stackrel{R}{\leftarrow} \{0,1\}^n$, and uses $\{\mathrm{Ext}(x,y) : y \in \{0,1\}^d\}$ as its $D = 2^d$ samples. Conversely, every sampler Smp using n random bits to produce $D = 2^d$ samples in $\{0,1\}^m$ defines a function $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$.

(1) Prove that if Ext is a $(k-1,\varepsilon)$-extractor, then Smp is a $(2^k/2^n,\varepsilon)$-averaging sampler.

(2) Prove that if Smp is a $(2^k/2^n,\varepsilon)$-sampler, then Ext is a $(k+\log(1/\varepsilon), 2\varepsilon)$-extractor.

(3) Suppose we are given a constant-error **BPP** algorithm which uses $r = r(n)$ random bits on inputs of length n. Show how, using Part 1 and the extractor of Theorem 5.31, we can reduce its error probability to $2^{-\ell}$ using $O(r) + \ell$ random bits, for any polynomial $\ell = \ell(n)$. (Note that this improves the $r + O(\ell)$ given by expander walks for $\ell \gg r$.) Conclude that every problem in **BPP** has a randomized algorithm which only errs for $2^{q^{0.01}}$ choices of its q random bits!

Problem 5.3. (Encryption and Deterministic Extraction) A (one-time) *encryption scheme* with key length n and message length m consists of an encryption function $\mathrm{Enc}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ and a decryption function $\mathrm{Dec}\colon \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$ such that $\mathrm{Dec}(k, \mathrm{Enc}(k,u)) = u$ for every $k \in \{0,1\}^n$ and $u \in \{0,1\}^m$. Let K be a random variable taking values in $\{0,1\}^n$. We say that $(\mathrm{Enc}, \mathrm{Dec})$ is *(statistically) ε-secure with respect to K* if for every two messages $u, v \in \{0,1\}^m$, we have $\Delta(\mathrm{Enc}(K,u), \mathrm{Enc}(K,v)) \leq \varepsilon$. For example, the *one-time pad*, where $n = m = \ell$ and $\mathrm{Enc}(k,u) = k \oplus u = \mathrm{Dec}(k,u)$ is 0-secure (aka perfectly secure) with respect to the uniform distribution $K = U_m$. For a class $\mathcal{C}$ of sources on $\{0,1\}^n$, we say that the encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ is *ε-secure with respect to $\mathcal{C}$* if Enc is ε-secure with respect to every $K \in \mathcal{C}$.

(1) Show that if there exists a deterministic ε-extractor $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}^m$ for $\mathcal{C}$, then there exists an 2ε-secure encryption scheme with respect to $\mathcal{C}$.

(2) Conversely, use the following steps to show that if there exists an ε-secure encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ with respect to $\mathcal{C}$, where $\mathrm{Enc}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$, then there exists a deterministic 2ε-extractor $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}^{m - 2\log(1/\varepsilon) - O(1)}$ for $\mathcal{C}$, provided $m \geq \log n + 2\log(1/\varepsilon) + O(1)$.

   (a) For each fixed key $k \in \{0,1\}^n$, define a source X_k on $\{0,1\}^\ell$ by $X_k = \mathrm{Enc}(k, U_m)$, and let $\mathcal{C}'$ be the class of all these sources (i.e., $\mathcal{C}' = \{X_k : k \in \{0,1\}^n\}$). Show that there exists a deterministic ε-extractor $\mathrm{Ext}'\colon \{0,1\}^\ell \to \{0,1\}^{m - 2\log(1/\varepsilon) - O(1)}$ for $\mathcal{C}'$, provided $m \geq \log n + 2\log(1/\varepsilon) + O(1)$.

   (b) Show that if Ext' is a deterministic ε-extractor for $\mathcal{C}'$ and Enc is ε-secure with respect to $\mathcal{C}$, then $\mathrm{Ext}(k) = \mathrm{Ext}'(\mathrm{Enc}(k, 0^m))$ is a deterministic 2ε-extractor for $\mathcal{C}$.

Thus, a class of sources can be used for secure encryption iff it is deterministically extractable.

Problem 5.4. (Extracting from Unpredictable-Bit Sources)

(1) Let X be a source taking values in $\{0,1\}^n$ such that for all x, y, $\Pr[X = x]/\Pr[X = y] \leq (1-\delta)/\delta$. Show that $X \in \mathrm{UnpredBits}_{n,\delta}$.

(2) Prove that for every function $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}$ and every $\delta > 0$, there exists a source $X \in \mathrm{UnpredBits}_{n,\delta}$ with parameter δ such that $\Pr[\mathrm{Ext}(X) = 1] \leq \delta$ or $\Pr[\mathrm{Ext}(X) = 0] \geq 1 - \delta$. (Hint: for $b \in \{0,1\}$, consider X that is uniform on $\mathrm{Ext}^{-1}(b)$ with probability $1-\delta$ and is uniform on $\mathrm{Ext}^{-1}(\bar{b})$ with probability δ.)

(3) (*) Show how to extract from sources in $\mathrm{UnpredBits}_{n,\delta}$ using a seeded extractor with a seed of *constant* length. That is, the seed length should not depend on the length n of the source, but only on the bias parameter δ and the statistical difference ε from uniform desired. The number of bits extracted should be $\Omega(\delta n)$.

Problem 5.5. (The Building-Block Extractor) Assume the condenser stated in Theorem 5.30. Show that for every *constant* $t > 0$ and all positive integers $n \geq k$ and all $\varepsilon > 0$, there is an *explicit* (k, ε)-extractor $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $m = k/2$ and $d = k/t + O(\log(n/\varepsilon))$. (Hint: convert the source into a block source with blocks of length $k/O(t) + O(\log(n/\varepsilon))$.)

Problem 5.6. (Extracting from Symbol-Fixing Sources*) A generalization of a bit-fixing source is a *symbol-fixing source* X taking values in Σ^n for some alphabet Σ, where subset of the coordinates of X are fixed and the rest are uniformly distributed and independent elements of Σ. For $\Sigma = \{0,1,2\}$ and $k \in [0,n]$, give an explicit ε-extractor $\mathrm{Ext} : \Sigma^n \to \{0,1\}^m$ for the class of of symbol-fixing sources on Σ^n with min-entropy at least k, with $m = \Omega(k)$ and $\varepsilon = 2^{-\Omega(k)}$. (Hint: use a random walk on a consistently labelled 3-regular expander graph.)

6

List-Decodable Codes

6.1 Definitions and Existence

The field of *coding theory* is motivated by the problem of communicating reliably over noisy channels — where the data sent over the channel may come out corrupted on the other end, but we nevertheless want the receiver to be able to correct the errors and recover the original message. There is a vast literature studying aspects of this problem from the perspectives of electrical engineering (communications and information theory), computer science (algorithms and complexity), and mathematics (combinatorics and algebra). In this survey, we are interested in codes as "pseudorandom objects," ones that are intimately related with the other objects we are studying. In particular, we will see how to use ideas from coding theory to construct the condensers and unbalanced expanders that we assumed in the previous chapter (Theorem 5.41).

The approach to communicating over a noisy channel is to restrict the data we send to be from a certain set of strings that can be easily disambiguated (even after being corrupted).

Definition 6.1. A *q-ary code* is a set $\mathcal{C} \subseteq \Sigma^{\hat{n}}$, where Σ is an *alphabet* of size q. Elements of $\mathcal{C}$ are called *codewords*. Some key parameters:

- $\hat{n}$ is the *block length*.
- $n = \log_2 |\mathcal{C}|$ is the *message length*.
- $\rho = n/(\hat{n} \cdot \log |\Sigma|)$ is the *(relative) rate* of the code.

An *encoding function* for $\mathcal{C}$ is an injective mapping $\mathrm{Enc}\colon \{1, \ldots, |\mathcal{C}|\} \to \mathcal{C}$. Given such an encoding function, we view the elements of $\{1, \ldots, |\mathcal{C}|\}$ as *messages*. When $n = \log |\mathcal{C}|$ is an integer, we often think of messages as strings in $\{0,1\}^n$.

Note that every code $\mathcal{C}$ whose message length is an integer has an encoding function Enc. We view $\mathcal{C}$ and Enc as being essentially the same object (with Enc merely providing a "labelling" of codewords), with the former being useful for studying the combinatorics of codes and the latter for algorithmic purposes.

We remark that our notation differs from the standard notation in coding theory in several ways. Typically in coding theory, the input alphabet is taken to be the same as the output alphabet (rather than $\{0,1\}$ and Σ, respectively), the blocklength is denoted n, and the message length (over Σ) is denoted k and is referred to as the rate.

So far, we haven't talked at all about the error-correcting properties of codes. Here we need to specify two things: the model of errors (as introduced by the noisy channel) and the notion of a successful recovery.

For the errors, the main distinction is between *random errors* and *worst-case errors*. For random errors, one needs to specify a stochastic model of the channel. The most basic one is the *binary symmetric channel* (over alphabet $\Sigma = \{0,1\}$), where each bit is flipped independently with probability δ. People also study more complex channel models, but as usual with stochastic models, there is always the question of how well the theoretical model correctly captures the real-life distribution of errors. We, however, will focus on *worst-case errors*, where we simply assume that fewer than a δ fraction of symbols have been changed. That is, when we send a codeword $c \in \Sigma^{\hat{n}}$ over the channel, the received word $r \in \Sigma^{\hat{n}}$ differs from c in fewer than $\delta\hat{n}$ places. Equivalently, c are r are close in Hamming distance:

Definition 6.2 (Hamming distance). For two strings $x, y \in \Sigma^{\hat{n}}$, their *(relative) Hamming distance* $d_H(x,y)$ equals $\Pr_i[x_i \neq y_i]$. The *agreement* is defined to be $\mathrm{agr}(x,y) = 1 - d_H(x,y)$.

For a string $x \in \Sigma^{\hat{n}}$ and $\delta \in [0,1]$, the *(open) Hamming ball* of radius δ around x is the set $B(x,\delta)$ of strings $y \in \Sigma^{\hat{n}}$ such that $d_H(x,y) < \delta$. Define $H_q(\delta,\hat{n})$ to be such that $|B(x,\delta)| = q^{H_q(\delta,\hat{n})\cdot\hat{n}}$.

For the notion of a successful recovery, the traditional model requires that we can uniquely decode the message from the received word (in the case of random errors, this need only hold with high probability). Our main focus will be on a more relaxed notion which allows us to produce a small list of candidate messages. As we will see, the advantage of such list-decoding is that it allows us to correct a larger fraction of errors.

Definition 6.3. Let $\mathrm{Enc}\colon \{0,1\}^n \to \Sigma^{\hat{n}}$ be an encoding algorithm for a code $\mathcal{C}$. A *δ-decoding* algorithm for Enc is a function $\mathrm{Dec} : \Sigma^{\hat{n}} \to \{0,1\}^n$ such that for every $m \in \{0,1\}^n$ and $r \in \Sigma^{\hat{n}}$ satisfying $d_H(\mathrm{Enc}(m), r) < \delta$, we have $\mathrm{Dec}(r) = m$. If such a function Dec exists, we call the code *δ-decodable*.

A *(δ, L)-list-decoding* algorithm for Enc is a function $\mathrm{Dec} : \Sigma^{\hat{n}} \to (\{0,1\}^n)^L$ such that for every $m \in \{0,1\}^n$ and $r \in \Sigma^{\hat{n}}$ satisfying $d_H(\mathrm{Enc}(m), r) < \delta$, we have $m \in \mathrm{Dec}(r)$. If such a function *Dec* exists, we call the code *(δ, L)-list-decodable*.

Note that a δ-decoding algorithm is the same as a $(\delta, 1)$-list-decoding algorithm. It is not hard to see that, if we do not care about computational efficiency, the existence of such decoding algorithms depends only on the combinatorics of the set of codewords.

Definition 6.4. The *(relative) minimum distance* of a code $\mathcal{C} \subseteq \Sigma^{\hat{n}}$ equals $\min_{x \neq y \in \mathcal{C}} d_H(x,y)$.

Proposition 6.5. Let $\mathcal{C} \subseteq \Sigma^{\hat{n}}$ be a code with any encoding function.

(1) For $\delta\hat{n} \in \mathbb{N}$, $\mathcal{C}$ is δ-decodable iff its minimum distance is at least $2\delta - 1/\hat{n}$.
(2) $\mathcal{C}$ is (δ, L)-list-decodable iff for every $r \in \Sigma^{\hat{n}}$, we have $|B(r, \delta) \cap \mathcal{C}| \leq L$.

Because of the factor of 2 in Item 1, unique decoding is only possible at distances up to $1/2$, whereas we will see that list-decoding is possible at distances approaching (with small lists).

The main goals in constructing codes are to have infinite families of codes (e.g. for every message length n) in which we:

- Maximize the fraction δ of errors correctible (e.g. constant independent of n and $\hat{n}$).
- Maximize the rate ρ (e.g. a constant independent of n and $\hat{n}$).
- Minimize the alphabet size q (e.g. a constant, ideally $q = 2$).
- Keep the list size L relatively small (e.g. a constant or poly(n)).
- Have computationally efficient encoding and decoding algorithms.

In particular, coding theorists are very interested in obtaining the optimal tradeoff between the constants δ and ρ with efficiently encodable and decodable codes.

6.1.1 Existential Bounds

The existence of very good codes can be shown using the probabilistic method.

Theorem 6.6. (1) For all $\hat{n}, q \in \mathbb{N}$ and $\delta \in (0, 1 - 1/q)$, there exists a q-ary code of block length $\hat{n}$, minimum distance at least δ, and rate $\rho \geq 1 - H_q(\delta, \hat{n})$.
(2) For all all integers $\hat{n}, q, L \in \mathbb{N}$ and $\delta \in (0, 1 - 1/q)$, there exists a (δ, L)-list-decodable q-ary code of block length $\hat{n}$ and rate $\rho \geq 1 - H_q(\delta, \hat{n}) - 1/(L + 1)$.

Proof. [Sketch]

(1) Pick the codewords $c_1, \ldots, c_N$ in sequence ensuring that c_i is at distance at least δ from $c_1, \ldots, c_{i-1}$. The union of the Hamming balls of radius δ around $c_1, \ldots, c_{i-1}$ contains at most $(i-1) \cdot q^{H_q(\delta,\hat{n})\cdot\hat{n}} < N \cdot q^{H_q(\delta,\hat{n})}$, so there is always a choice of c_i outside these balls provided that $N \cdot q^{H_q(\delta,\hat{n})} \leq q^{\hat{n}}$.
(2) We use the probabilistic method. Choose the $q^{\rho\hat{n}}$ elements of the code randomly and independently from $\Sigma^{\hat{n}}$. The probability that there is a Hamming Ball of radius δ containing $L + 1$ codewords is at most

$$q^{\hat{n}} \cdot \binom{q^{\rho\hat{n}}}{L+1} \cdot \left(\frac{q^{H_q(\delta,\hat{n})\cdot\hat{n}}}{q^{\hat{n}}}\right)^{L+1},$$

which is less than 1 by our setting of parameters.

□

Note that while the rate bounds are essentially the same for achieving minimum distance and the list-decoding radius δ (as we take large list size), recall that minimum distance δ only corresponds to unique decoding up to radius roughly $\delta/2$. Indeed, the bound for list-decoding is known to be tight up to the dependence on L (Problem 6.1, while the bound on minimum distance is not tight in general. Indeed, there are families "algebraic-geometric" codes with constant alphabet size q, constant minimum distance $\delta > 0$, and constant rate $\rho > 0$ where $\rho > 1 - H_q(\delta, \hat{n})$ for sufficiently large $\hat{n}$. (Thus, this is a rare counterexample to the phenomenon "random is best".) Identifying the best tradeoff between ρ and δ, even for binary codes, is a long-standing open problem in coding theory.

Open Problem 6.7. For each constant $\delta \in (0, 1)$, identify the largest $\rho > 0$ such that for every $\varepsilon > 0$, there exists an infinite family of codes $\mathcal{C}_{\hat{n}} \subseteq \{0,1\}^{\hat{n}}$ of rate at least $\rho - \varepsilon$ and minimum distance at least δ.

Let's look at some special cases of the parameters in the above theorem. For binary codes ($q = 2$), it turns out that $H_2(\delta, \hat{n})$ is at most the Shannon entropy $\mathrm{H}_{Sh}(\delta)$ of a $\{0,1\}$-valued random variable B s.t. $\Pr[B = 1] = \delta$, so the rate is roughly $1 - \mathrm{H}_{Sh}(\delta)$ (as we take a large list size). (More generally, $H_q(\delta, \hat{n})$ is bounded by a q-ary analogue of Shannon entropy.) This is known to be tight for list-decoding, but it is unknown whether it is tight for unique decoding. We will be most interested in the case $\delta \to 1/2$, which corresponds to correcting the maximum possible fraction of errors for binary codes. (No nontrivial decoding is possible for binary codes at distance greater than $1/2$, since a completely random received word will be at distance roughly $1/2$ with most codewords.) In which case the rate is $1 - \mathrm{H}_{Sh}(1/2 - \varepsilon) = \Theta(\varepsilon^2)$, i.e. $\hat{n} = \Theta(n/\varepsilon^2)$ (for list size $L = \Theta(1/\varepsilon^2)$).

For large alphabets q, we have $H_q(\delta, \hat{n}) \leq H_2(\delta, \hat{n})/\log q + \delta$, in which case the rate approaches $1 - \delta$ as q grows. We will be most interested in the case $\delta = 1 - \varepsilon$, where the above bounds imply codes where we can correct a $\delta = 1 - \varepsilon$ fraction of errors with a rate of $\rho = .99\varepsilon$, a list size of $L = O(1/\varepsilon)$ and an alphabet of size $\mathrm{poly}(1/\varepsilon)$. Alternatively, it is possible to achieve rate $\rho = (1 + \gamma)\varepsilon$ with an alphabet size of $q = (1/\varepsilon)^{O(1/\gamma)}$.

While we are primarily interested in list-decodable codes, minimum distance is often easier to bound. The following allows us to translate bounds on minimum distance into bounds on list-decodability.

Proposition 6.8 (Johnson Bound).
(1) If $\mathcal{C}$ has minimum distance $1 - \varepsilon$, then it is $(1 - O(\sqrt{\varepsilon}), O(1/\sqrt{\varepsilon}))$-list-decodable.
(2) If a *binary* code $\mathcal{C}$ has minimum distance $1/2 - \varepsilon$, then it is $(1/2 - O(\sqrt{\varepsilon}), O(1/\varepsilon))$-list-decodable.

Proof. We prove Item 1, and leave Item 2 as an exercise. The proof is by inclusion-exclusion. Suppose for contradiction that there are codewords $c_1, \ldots, c_s$ at distance less than $1 - \varepsilon'$ from some

$r \in \Sigma^{\hat{n}}$, for $\varepsilon' = 2\sqrt{\varepsilon}$ and $s = \lceil 2/\varepsilon' \rceil$. Then:

$$\begin{aligned} 1 &\geq \text{fraction of positions where } r \text{ agrees with some } C_i \\ &\geq \sum_i \mathrm{agr}(r, C_i) - \sum_{1 \leq i < j \leq s} \mathrm{agr}(C_i, C_j) \\ &> s\varepsilon' - \binom{s}{2} \cdot \varepsilon \\ &\geq 2 - 1 = 1 \end{aligned}$$

where the last inequality is by our setting of parameters. Contradiction. □

Note the quadratic loss in the distance parameter. This means that optimal codes with respect to minimum distance are not necessarily optimal with respect to list-decoding. Nevertheless, if we do not care about the exact tradeoff between the rate and the decoding radius, the above can yield codes where the decoding radius is as large as possible (approaching 1 for large alphabets and 1/2 for binary alphabets).

6.1.2 Explicit Codes

As usual, most applications of error-correcting codes (in particular the original motivating one) require computationally efficient encoding and decoding. For now, we focus on only the efficiency of encoding.

Definition 6.9. A code $\mathrm{Enc} : \{0,1\}^n \to \Sigma^{\hat{n}}$ is *(fully) explicit* if given a message $m \in \{0,1\}^n$ and an index $i \in \hat{n}$, the i'th symbol of $\mathrm{Enc}(m)$ can be computed in time $\mathrm{poly}(n, \log \hat{n}, \log |\Sigma|)$.

The reason we talk about computing individual symbols of the codeword rather than the entire codeword is to have a meaningful definition even for codes where the blocklength $\hat{n}$ is superpolynomial in the message length n. One can also consider weaker notions of explicitness, where we simply require that the entire codeword $\mathrm{Enc}(m)$ can be computed in time $\mathrm{poly}(\hat{n}, \log |\Sigma|)$.

The constructions of codes that we describe will involve arithmetic over finite fields, so we recall some facts about the complexity of such arithmetic (previously mentioned in Section 3.5.2):

For every prime power $q = p^k$ there is a field $\mathbb{F}_q$ of size q, and this field is unique up to isomorphism (renaming elements). The prime p is called the *characteristic* of the field. $\mathbb{F}_q$ has a description of length $O(\log q)$ enabling addition, multiplication, and division to be formed in polynomial time (i.e. time $\mathrm{poly}(\log q)$). (This description is simply an irreducible polynomial f of degree k over $\mathbb{F}_p = \mathbb{Z}_p$.) If $q = p^k$ for a given prime p and integer k, this description can be found probabilistically in time $\mathrm{poly}(\log p, k) = \mathrm{poly}(\log q)$ and deterministically in time $\mathrm{poly}(p, k)$. Note that for even finding a prime p of a desired bitlength, we only know time $\mathrm{poly}(p)$ deterministic algorithms. Thus, for computational purposes, a convenient choice is often to instead take $p = 2$ and k large, in which case everything can be done deterministically in time $\mathrm{poly}(k) = \mathrm{poly}(\log q)$. With such a choice, all of the constructions below meet our definition of explicitness.

In describing the explicit constructions below, it will be convenient to think of the codewords as functions $c : [\hat{n}] \to \Sigma$ rather than as strings in $\Sigma^{\hat{n}}$.

Construction 6.10 (Hadamard Code). For $n \in \mathbb{N}$, the *(binary) Hadamard code* of message length n is the binary code of blocklength $\hat{n} = 2^n$ consisting of all functions $c : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that are *linear* (modulo 2).

Proposition 6.11. The Hadamard code:

(1) is explicit with respect to the encoding function that takes a message $m \in \mathbb{Z}_2^n$ to the linear function c_m defined by $c_m(x) = \sum_i m_i x_i \bmod 2$.
(2) has minimum distance $1/2$, and
(3) is $(1/2 - \varepsilon, O(1/\varepsilon^2))$ list-decodable for every $\varepsilon > 0$.

Proof. Explicitness is clear by inspection. The minimum distance follows from the fact that for every two distinct linear functions $c, c' : \mathbb{Z}_2^m \to \mathbb{Z}_2$, $\Pr_x[c(x) = c'(x)] = \Pr_x[(c - c')(x) = 0] = 1/2$. The list-decodability follows from the Johnson Bound. □

The advantages of the Hadamard code are its small alphabet (binary) and optimal distance $(1/2)$, but unfortunately its rate is exponentially small ($\rho = n/2^n$). By increasing both the field size and degree, we can obtain complementary properties

Construction 6.12. For a prime power q and $d \in \mathbb{N}$, the q-ary *Reed–Solomon code* of degree d is the code of blocklength $\hat{n} = q$ and message length $n = (d+1) \cdot \log q$ consisting of all polynomials $p : \mathbb{F}_q \to \mathbb{F}_q$ of degree at most d.

Proposition 6.13. The q-ary Reed–Solomon Code of degree d:

(1) is explicit with respect to the encoding function that takes a vector of coefficients $m \in \mathbb{F}_q^{d+1}$ to the polynomial p_m defined by $p_m(x) = \sum_{i=0}^{d} m_i x^i$.
(2) has minimum distance $\delta = 1 - d/q$, and
(3) is $(1/2 - O(\sqrt{d/q}), O(\sqrt{q/d}))$ list-decodable.

Note that by setting $q = O(d)$, Reed-Solomon codes simultaneously achieve constant rate and constant distance, the only disadvantage being that the alphabet is of nonconstant size (namely $q = \hat{n} > n$.)

Another useful setting of parameters for Reed-Solomon codes in complexity theory is $q = \text{poly}(d)$, which gives polynomial rate ($\hat{n} = \text{poly}(n)$) and distance tending to 1 polynomially fast ($\delta = 1 - 1/\text{poly}(n)$).

The following codes "interpolate" between Hadamard and Reed-Solomon codes, by allowing the number of variables, the degree, and field size all to vary.

Construction 6.14. For a prime power q and $d, m \in \mathbb{N}$, the q-ary *Reed–Muller code* of degree d and dimension t is the code of blocklength $\hat{n} = q^m$ and message length $n = \binom{m+d}{d} \cdot \log q$ consisting of all polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ of (total) degree at most d.

Proposition 6.15. The q-ary Reed–Muller Code of degree d and dimension m:

(1) is explicit with respect to the encoding function that takes a vector of coefficients $v \in \mathbb{F}_q^{\binom{m+d}{d}}$ to the corresponding polynomial p_v.
(2) has minimum distance $\delta \geq 1 - d/q$, and
(3) is $(1/2 - O(\sqrt{d/q}), O(\sqrt{q/d}))$ list-decodable.

Note that Reed–Solomon Codes are simply Reed–Muller codes of dimension $m = 1$, and Hadamard codes are essentially Reed–Muller codes of degree $d = 1$ and alphabet size $q = 2$ (except that the Reed–Muller code also contains affine linear functions).

6.2 List-Decoding Algorithms

In this section, we will describe efficient list-decoding algorithms for the Reed–Solomon code and variants. It will be convenient to work with the following notation:

Definition 6.16. Let $\mathcal{C}$ be a code with encoding function $\text{Enc} : \{1, \ldots, N\} \to \Sigma^{\hat{n}}$. For $r \in \Sigma^{\hat{n}}$, define $\text{LIST}(r, \varepsilon) = \{m : \text{agr}(m, r) > \varepsilon\}$.

Then the task of $(1 - \varepsilon)$ list-decoding (according to Definition 6.3) is equivalent to producing the elements of $\text{LIST}(r, \varepsilon)$ given $r \in \Sigma^{\hat{n}}$. In this section, we will see algorithms that do this in time polynomial in the bit-length of r, i.e. time $\text{poly}(\hat{n}, \log |\Sigma|)$.

6.2.1 Review of Algebra

The list-decoding algorithms will require some additional algebraic facts and notation:

- For every field $\mathbb{F}$, $\mathbb{F}[X_1, \ldots, X_n]$ is the integral domain consisting of formal polynomials $Q(X_1, \ldots, X_n)$ with coefficients in $\mathbb{F}$, where addition and multiplication of polynomials is defined in the usual way.
- A nonzero polynomial $Q(X_1, \ldots, X_n)$ is *irreducible* if we cannot write $Q = RS$ where R, S are nonconstant polynomials. For a finite field $\mathbb{F}_q$ of characteristic p and $d \in \mathbb{N}$, a univariate irreducible polynomial of degree d over $\mathbb{F}_q$ can be found in deterministically in time $\text{poly}(p, \log q, d)$.
- $\mathbb{F}[X_1, \ldots, X_n]$ is a *unique factorization domain*. That is, every nonzero polynomial Q can be factored as $Q = Q_1 Q_2 \cdots Q_m$, where each Q_i is irreducible and this factorization is unique up to reordering and multiplication by constants from $\mathbb{F}$. Given the description of a finite field $\mathbb{F}_{p^k}$ and the polynomial Q, this factorization can be done probabilistically in time $\text{poly}(\log p, k, |Q|)$ and deterministically in time $\text{poly}(p, k, |Q|)$.

- For a nonzero polynomial $Q(Y, Z) \in \mathbb{F}[Y, Z]$ and $f(Y) \in \mathbb{F}[Y]$, if $Q(Y, f(Y)) = 0$, then $Z - f(Y)$ is one of the irreducible factors of $Q(Y, Z)$ (and thus $f(Y)$ can be found in polynomial time). This is analogous to the fact that if $c \in \mathbb{Z}$ is a root of an integer polynomial $Q(Z)$, then $Z - c$ is one of the factors of Q (and can be proven in the same way, by long division).

6.2.2 List-Decoding Reed-Solomon Codes

~~**Theorem 6.17.** There is a polynomial-time $(1 - \varepsilon)$ list-decoding algorithm for the q-ary Reed-~~Solomon code of degree d, for $\varepsilon = 2\sqrt{d/q}$. That is, given a function $r : \mathbb{F}_q \to \mathbb{F}_q$ and $d \in \mathbb{N}$, all polynomials of degree at most d that agree with r on more than $\varepsilon q = 2\sqrt{dq}$ inputs can be found in polynomial time.

In fact the constant of 2 can be improved to 1, matching the combinatorial list-decoding radius for Reed–Solomon codes given by an optimized form of the Johnson Bound, but we will not do this optimization here.

Proof. We are given a received word $r : \mathbb{F}_q \to \mathbb{F}_q$, and want to find all elements of $\mathrm{LIST}(r, \varepsilon)$ for $\varepsilon = 2\sqrt{d/q}$.

Step 1: Find a low-degree Q "explaining" r. Specifically, $Q(Y, Z)$ will be a nonzero bivariate polynomial of degree at most d_Y in its first variable Y and d_Z in its second variable, and will satisfy $Q(y, r(y)) = 0$ for all $y \in \mathbb{F}_q$. Each such y imposes a linear constraint on the $(d_Y + 1)(d_Z + 1)$ coefficients of Q. Thus, this system has a nonzero solution provided $(d_Y + 1)(d_Z + 1) > q$, and it can be found in polynomial time by linear algebra (over $\mathbb{F}_q$).

Step 2: Argue that each $f(Y) \in \mathrm{LIST}(r, \varepsilon)$ is a "root" of Q. Specifically, it will be the case that $Q(Y, f(Y)) = 0$ for each $f \in \mathrm{LIST}(r, \varepsilon)$. The reason is that $Q(Y, f(Y))$ is a univariate polynomial of degree at most $d_Y + d \cdot d_Z$, and has more than εq zeroes (one for each place that f and r agree). Thus, we can conclude $Q(Y, f(Y)) = 0$ provided $\varepsilon q \geq d_Y + d \cdot d_Z$. Then we can enumerate all of the elements of $\mathrm{LIST}(r)$ by factoring $Q(Y, Z)$ and taking all the factors of the form $Z - f(Y)$.

For this algorithm to work, the two conditions we need to satisfy are

$$(d_Y + 1)(d_Z + 1) > q,$$

and

$$\varepsilon q \geq d_Y + d \cdot d_Z.$$

These conditions can be satisfied by setting $d_Y = \lfloor \varepsilon q/2 \rfloor$, $d_Z = \lfloor \varepsilon q/(2d) \rfloor$, and $\varepsilon = 2\sqrt{d/q}$. □

Note that the rate of Reed-Solomon codes is $\rho = (d + 1)/q = \Theta(\varepsilon^2)$. The alphabet size is $q = \tilde{\Omega}(n/\rho) = \tilde{\Omega}(n/\varepsilon^2)$. In contrast, the random codes of Theorem 6.6 achieve $\rho \approx \varepsilon$ and $q = \mathrm{poly}(1/\varepsilon)$. It is not known whether the known bounds on the list-decodability of Reed–Solomon codes can be improved, even with inefficient decoding.

Open Problem 6.18. Is the q-ary Reed–Solomon Code of degree d $(1 - eps)$-list-decodable for $\varepsilon < \sqrt{d/q}$?

6.2.3 Parvaresh–Vardy Codes

Our aim is to improve the rate-distance tradeoff to $\rho = \tilde{\Theta}(\varepsilon)$. Intuitively, the power of the Reed–Solomon list-decoding algorithm comes from the fact that we can interpolate the q points $(y, r(y))$ of the received word using a *bivariate* polynomial Q of degree roughly $\sqrt{q}$ in each variable (think of $d = O(1)$ for now). If we could use m variables instead of 2, then the degrees would only have to be around $q^{1/m}$.

First attempt: Replace Step 1 with finding an $(m+1)$-variate polynomial $Q(Y, Z_1, \ldots, Z_m)$ of degree d_Y in Y and d_Z in each Z_i such that $Q(y, r(y), r(y), \ldots, r(y)) = 0$ for every $y \in \mathbb{F}_q$. Then, we will be able to choose a nonzero polynomial Q of degree roughly $q^{1/m}$ such that $Q(Y, f(Y), \ldots, f(Y)) = 0$ for every $f \in \mathrm{LIST}(r, \varepsilon)$, and hence it follows that $Z - f(Y)$ divides the bivariate polynomial $Q^*(Y, Z) = Q(Y, Z, \ldots, Z)$. Unfortunately, Q^* might be the zero polynomial even if Q is nonzero.

Second attempt: Replace Step 1 with finding an $(m + 1)$-variate polynomial $Q(Y, Z_1, \ldots, Z_m)$ of degree d_Y in Y and $d_Z = h - 1$ in each Z_i such that $Q(y, r(y), r(y)^h, r(y)^{h^2}, \ldots, r(y)^{h^{m-1}}) = 0$ for every $y \in \mathbb{F}_q$. Then, it follows that $Q^*(Y, Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}})$ is nonzero if Q is nonzero because every monomial in Q with individual degrees at most $h - 1$ in $Z_1, \cdots, Z_m$ gets mapped to a different power of Z. However, here the difficulty is that the degree of $Q^*(Y, f(Y))$ is too high (roughly $d^* = d_Y + d \cdot h^m > d_Z^m$) for us to satisfy the constraint $\varepsilon q \geq d^*$.

Parvaresh–Vardy codes get the best of both worlds by providing more information with each symbol — not just the evaluation of f at each point, but the evaluation of $m-1$ other polynomials, each of which is still of degree d (as is good for Step 1), but can be viewed as raising f to successive powers of h for the purposes of the getting a nonzero polynomial in one variable Z in Step 2.

To introduce this idea, we need some additional algebra.

- For univariate polynomials $f(Y)$ and $E(Y)$, we define $f(Y)$ mod $E(Y)$ to be the remainder when f is divided by E. If $E(Y)$ is of degree k, then $f(Y)$ mod $E(Y)$ is of degree at most $k - 1$.
- The ring $\mathbb{F}[Y]/E(Y)$ consists of all polynomials of degree at most $k - 1$ with arithmetic modulo $E(Y)$ (analogous to $\mathbb{Z}_n$ consisting integers smaller than n with arithmetic modulo n). If E is irreducible then, $\mathbb{F}[Y]/E(Y)$ is a field (analogous to $\mathbb{Z}_p$ being a field when p is prime). Indeed, this is how the finite field of size p^k is constructed: take $\mathbb{F} = \mathbb{Z}_p$ and $E(Y)$ to be an irreducible polynomial of degree k over $\mathbb{Z}_p$, and then $\mathbb{F}[Y]/E(Y)$ is the (unique) field of size p^k.
- A multivariate polynomial $Q(Y, Z_1, \ldots, Z_m)$ can be reduced modulo $E(Y)$ by writing it as a polynomial in variables $Z_1, \ldots, Z_m$ with coefficients in $\mathbb{F}[Y]$ and then reducing each coefficient modulo $E(Y)$. After reducing Q modulo E, we think of Q as a polynomial in variables $Z_1, \ldots, Z_m$ with coefficients in the field $\mathbb{F}[Y]/E(Y)$.

Construction 6.19 (Parvaresh–Vardy Codes). For a prime power q, $m, d, h \in \mathbb{N}$, and an irreducible polynomial $E(Y)$ of degree larger than d, the q-ary *Parvaresh–Vardy* code of degree d, power h, redundancy m, and irreducible E is defined as follows:

- The alphabet is $\Sigma = \mathbb{F}_q^m$.
- The blocklength is $\hat{n} = q$.
- The message space is $\mathbb{F}_q^{d+1}$, where we view each message as representing a polynomial $f(Y)$ of degree at most d over $\mathbb{F}_q$.
- For $y \in \mathbb{F}_q$, the y'th symbol of the $\mathrm{Enc}(f)$ is

$$[f_0(y), f_1(y), \ldots, f_{m-1}(y)],$$

  where $f_i(Y) = f(Y)^{h^i} \bmod E(Y)$.

Theorem 6.20. For every prime power q, $d \in \mathbb{N}$, and irreducible polynomial E of degree $d+1$, the q-ary Parvaresh–Vardy code of degree d, redundancy $m = \lceil \log(d/q) \rceil$, power $h = 2$, and irreducible E has has rate $\rho = \tilde{\Omega}(d/q)$ and can be list-decoded in polynomial time up to distance $\delta = 1 - \tilde{O}(d/q)$.

Proof. We are given a received word $r : \mathbb{F}_q \to \mathbb{F}_q^m$, and want to find all elements of $\mathrm{LIST}(r, \varepsilon)$, for some $\varepsilon = \tilde{O}(d/q)$.

Step 1: Find a low-degree Q "explaining" r. We find a polynomial $Q(Y, Z_0, \ldots, Z_{m-1})$ of degree at most d_Y in its first variable Y and at most $h-1$ in each of the remaining variables, and satisfying $Q(y, r(y)) = 0$ for all $y \in \mathbb{F}_q$.

This is possible provided

$$d_Y \cdot h^m > q. \tag{6.1}$$

Moreover, we may assume that Q is not divisible by $E(Y)$. If it is, we can divide out all the factors of $E(Y)$, which will not affect the conditions $Q(y, r(y)) = 0$ since E has no roots (being irreducible).

Step 2: Argue that each $f(Y) \in \mathrm{LIST}(r, \varepsilon)$ is a "root" of a related univariate polynomial Q^*. First, we argue as before that for $f \in \mathrm{LIST}(r, \varepsilon)$, we have

$$Q(Y, f_0(Y), \ldots, f_{m-1}(Y)) = 0.$$

Since each f_i has degree at most $\deg(E) - 1 = d$, this will be ensured provided

$$\varepsilon q \geq d_Y + (h-1) \cdot d \cdot m. \tag{6.2}$$

Once we have this, we can reduce both sides modulo $E(Y)$ and deduce

$$\begin{aligned} 0 &= Q(Y, f_0(Y), f_1(Y), \ldots, f_{m-1}(Y)) \bmod E(Y) \\ &= Q(Y, f(Y), f(Y)^h, \ldots, f(Y)^{h^{m-1}}) \bmod E(Y) \end{aligned}$$

Thus, if we define the univariate polynomial

$$Q^*(Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}}) \bmod E(Y),$$

then $f(Y)$ is a root of Q^* over the field $\mathbb{F}_q[Y]/E(Y)$.

Observe that Q^* is nonzero because Q is not divisible by $E(Y)$ and has degree at most $h-1$ in each Z_i. Thus, we can find all elements of $\mathrm{LIST}(r, \varepsilon)$ by factoring $Q^*(Z)$.

For this algorithm to work, we need to satisfy Conditions (6.1) and (6.2). We can satisfy Condition (6.2) by setting $d_Y = \lceil \varepsilon q - dhm \rceil$, in which case Condition (6.2) is satisfied for

$$\varepsilon = \frac{1}{h^m} + \frac{dhm}{q} = \tilde{O}(d/q), \tag{6.3}$$

for $h = 2$ and $m = \lceil \log(q/d) \rceil$. Observing that the rate is $\rho = d/(mq) = \tilde{\Omega}(d/q)$, this completes the proof of the theorem. Note that the obstacles to obtaining a tradeoff that is optimal to within constant factors (i.e. $\rho = \Theta(\varepsilon)$) are the factors of m appearing in both the expressions for ε and ρ. □

6.2.4 Folded Reed–Solomon Codes

We now sketch how to further improve the rate-distance tradeoff to be near-optimal.

Theorem 6.21. The following holds for all *constants* $\varepsilon > \rho > 0$. For every $n \in \mathbb{N}$, there is an explicit code of message length n and rate ρ that can be list-decoded in polynomial time from distance $1 - \varepsilon$, with alphabet size $q = \mathrm{poly}(n)$.

We will sketch how to achieve $\varepsilon = O(\rho)$. Consider the Parvaresh–Vardy construction with irreducible polynomial $E(Y) = Y^{q-1} - \gamma$, where γ is a generator of the multiplicative group $\mathbb{F}_q^*$. (That is, $\{\gamma, \gamma^2, \ldots, \gamma^{q-1}\} = \mathbb{F}_q \setminus \{0\}$.) Then it turns out that $f^q(Y) = f(\gamma Y) \bmod E(Y)$. So, we set $h = q$ and the degree of $f_i(Y) = f^{h^i}(Y) \bmod E(Y) = f(\gamma^i Y)$ is d even though E has degree $q - 1$. For each nonzero element y of $\mathbb{F}_q$, the y'th symbol of the PV encoding of $f(Y)$ is then

$$[f(y), f(\gamma y), \ldots, f(\gamma^{m-1} y)] = [f(\gamma^j), f(\gamma^{j+1}), \ldots, f(\gamma^{j+m-1})],$$

where we write $y = \gamma^j$.

Thus, the symbols of the PV encoding have a lot of overlap. For example, the γ^j'th symbol and the γ^{j+1}'th symbol share all but one component. Intuitively, this means that we should only have to send roughly a $1/m$ fraction of the symbols of the codeword, saving us a factor of m in the rate. (The other symbols can be automatically filled in by the receiver.) Thus, the rate becomes $\rho \approx d/q$, just like in Reed–Solomon codes.

However, there is still an extra factor m in the agreement ε we needed when analyzing the PV codes (Equation 6.3), which prohibits us from achieving $\rho = \Theta(\varepsilon)$. To deal with this, we don't just require that $Q(y, r(y)) = 0$ for each y, but instead require that Q has a root of multiplicity s at each point $(y, r(y))$. Formally, this means that the polynomial $Q(Y + y, Z_0 + r(y)_0, \ldots, Z_{m-1} + r(y)_{m-1})$ has no monomials of degree smaller than s.

Then Condition (6.2) becomes

$$\varepsilon q s \geq d_Y + (h-1) \cdot d \cdot m.$$

However, we pay a price in Condition (6.1), because asking for a root of multiplicity s amounts to $\binom{m+s}{s-1}$ constraints on the coefficients of Q (one for each monomial of degree smaller than s). Thus Condition (6.1) becomes:

$$d_Y \cdot h^m > q \cdot \binom{m+s-1}{m}.$$

Putting the two together, we can decode from agreement

$$\varepsilon = \frac{\binom{m+s-1}{m}}{sh^m} + \frac{dhm}{sq}.$$

Recalling that $h = q$, setting $s = (h-1)(m+1)/4$ and $m = O(\log(q/d)) \leq q/4$, we have

$$\varepsilon \leq \frac{((s+m+1)e/(m+1))^{m+1}}{s \cdot h^m} + \frac{4d}{q} \leq \frac{(he/4)^{m+1}}{h^{m+1}m/4} + \frac{4d}{q} = O(d/q) = O(\rho).$$

Further optimizations along the lines of Problem 6.4 can eliminate this constant factor and make $\varepsilon = \rho + \gamma$ for an arbitrarily small γ, which is optimal. Also, with a more sophisticated variant of code concatenation (see Problem 6.2) it is is possible achieve $\varepsilon = \rho + \gamma$ with an alphabet size that is independent of n, namely $q = 2^{\text{poly}(1/\gamma)}$. However, for a fixed constant-sized alphabet, e.g. $q = 2$, it is still not known how to achieve list-decoding capacity.

Open Problem 6.22. For any desired constants $\rho, \delta > 0$ such that $\rho > 1 - \mathrm{H}_{Sh}(\delta)$, construct an explicit family of of codes $\mathrm{Enc}_n : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ that have rate at least ρ and are (δ, L) list-decodable for $L = \text{poly}(n)$.

6.2.5 List-decoding views of expanders and extractors

Previously, we have seen close connections between expanders and extractors (and related objects, such as condensers). In this section, we will see how these objects are also closely related to list-decodable codes, by presenting all of them in a single, list-decoding-like framework. We begin with the syntactic correspondence.

Construction 6.23 (Syntactic Correspondence of Codes, Extractors, and Expanders). Given a code $\mathrm{Enc} : [N] \to [M]^D$, we define the corresponding extractor $\mathrm{Ext} : [N] \times [D] \to [D] \times [M]$ and the neighbor function of the corresponding expander $\Gamma : [N] \times [D] \to [D] \times [M]$ via the correspondence:

$$\mathrm{Ext}(x, y) = \Gamma(x, y) = (y, \mathrm{Enc}(x)_y).$$

Note that this correspondence yields extractors and expanders with output/right-hand-side $[D] \times [M]$ and where the first component equals the seed/edge-label. (Recall that for such an extractor Ext, the second component is called a *strong* extractor.) Conversely, any such extractor or expander yields a code Enc.

Definition 6.24. Let Enc, Ext, and Γ be the corresponding code, extractor, and expander as in Construction 6.23. For a subset $T \subseteq [D] \times [M]$ and $\varepsilon \in [0,1)$, we define

$$\begin{aligned} \mathrm{LIST}(T,\varepsilon) &\stackrel{\text{def}}{=} \{x : \Pr_y[(y, \mathrm{Enc}(x)_y) \in T] > \varepsilon\} \\ &= \{x : \Pr_y[\mathrm{Ext}(x,y) \in T] > \varepsilon\} \\ &= \{x : \Pr_y[\Gamma(x,y) \in T] > \varepsilon\} \end{aligned}$$

We define $\mathrm{LIST}(T,1)$ analogously, except that replace "$> \varepsilon$" with "$= 1$".

Now we can formulate the standard list-decoding property of codes in this language as follows:

Proposition 6.25. $\mathrm{Enc} : [N] \to [M]^D$ is $(1 - 1/M - \varepsilon, K)$ list-decodable iff for every $r \in [M]^D$, we have

$$|\mathrm{LIST}(T_r, 1/M + \varepsilon)| \le K,$$

where $T_r = \{(y, r_y) : y \in [D]\}$.

Now let's look at extractors.

Proposition 6.26. If $\mathrm{Ext} : [N] \times [D] \to [M]$ is a (k,ε) extractor then for every $T \subseteq [D] \times [M]$, we have

$$|\mathrm{LIST}(T, \mu(T) + \varepsilon)| < K, \tag{6.4}$$

where $K = 2^k$ and $\mu(T) = |T|/M$.

Conversely, if (6.4) holds for every $T \subseteq [D] \times [M]$, then Ext is a $(k + \log(1/\varepsilon), 2\varepsilon)$ extractor.

This lemma says that the extractor property is equivalent to a "list-decoding-like property," up to a factor of 2 in the error ε and an extra additive entropy loss of $\log(1/\varepsilon)$ (both of which are usually considered insignificant).

Let's compare this to the standard list-decoding property of codes as formulated in Proposition 6.25. Note that the only difference between the condition in Lemma 6.25 and the one in Proposition 6.26 is that in the former, we restrict to sets T of the form T_r. That is, we restrict to sets $T \subseteq [D] \times [M]$ that contain exactly one element of the form $(y, \cdot)$ for each y.

Corollary 6.27. If $\mathrm{Ext} : [N] \times [D] \to [D] \times [M]$ is a (k,ε) extractor (satisfying $\mathrm{Ext}(x,y) = (y, \mathrm{Ext}'(x,y))$), then the corresponding code Enc is $(1 - 1/M - \varepsilon, K)$ list-decodable.

A converse holds when the alphabet size is small.

Proposition 6.28. If $\mathrm{Enc} : [N] \to [M]^D$ is $(1 - 1/M - \varepsilon, K)$ list-decodable, then the corresponding function $\mathrm{Ext} : [N] \times [D] \to [D] \times [M]$ given by $\mathrm{Ext}(x,y) = (y, \mathrm{Enc}(x)_y)$ is a $(k + \log(1/\varepsilon), M \cdot \varepsilon)$ extractor.

Proof. Let X be a $(k + \log(1/\varepsilon))$-source and $Y = U_{[D]}$. Then the statistical difference between $\mathrm{Ext}(X, Y)$ and $Y \times U_{[M]}$ equals

$$\begin{aligned}\Delta(\mathrm{Ext}(X,Y), Y \times U_{[M]}) &= \mathop{\mathrm{E}}_{y \xleftarrow{R} Y}\left[\Delta(\mathrm{Enc}(X)_y, U_{[M]})\right] \\ &\leq \frac{M}{2} \mathop{\mathrm{E}}_{y \xleftarrow{R} Y}\left[\max_z \Pr[\mathrm{Enc}(X)_y = z] - 1/M\right]\end{aligned}$$

where the last inequality follows from the ℓ_1 formulation of statistical difference.

So if we define $r \in [M]^D$ by setting r_y to be the value z maximizing $\Pr[\mathrm{Enc}(X)_y = z] - 1/M$, we have:

$$\begin{aligned}\Delta(\mathrm{Ext}(X,Y), Y \times U_{[M]}) &\leq \frac{M}{2} \cdot \left(\Pr[(Y, \mathrm{Enc}(X)_Y) \in T_r] - 1/M\right), \\ &\leq \frac{M}{2} \cdot \left(\Pr[X \in \mathrm{LIST}(T_r, 1/M + \varepsilon)] + \varepsilon\right) \\ &\leq \frac{M}{2} \cdot \left(2^{-(k+\log(1/\varepsilon)} \cdot K + \varepsilon\right) \\ &\leq M \cdot \varepsilon.\end{aligned}$$

□

Thus, the quantitative relationship between extractors and list-decodable codes deteriorates extremely fast as the output length/alphabet size increases. Nevertheless, the list-decoding view of extractors as given in Proposition 6.26 turns out to be quite useful (as we will see later in the course).

For expanders, the list-decoding view is quite simple to state and prove.

Lemma 6.29. For $K \in \mathbb{N}$, $\Gamma : [N] \times [D] \to [D] \times [M]$ is an $(= K, A)$ expander iff for every set $T \subseteq [D] \times [M]$ such that $|T| < KA$, we have:

$$|\mathrm{LIST}(T, 1)| < K.$$

Proof.

$$\begin{aligned}&\Gamma \text{ not an } (= K, A) \text{ expander} \\ &\quad\Leftrightarrow\ \exists S \subseteq [N] \text{ s.t. } |S| = K \text{ and } |N(S)| < KA \\ &\quad\Leftrightarrow\ \exists S \subseteq [N] \text{ s.t. } |S| \geq K \text{ and } |N(S)| < KA \\ &\quad\Leftrightarrow\ \exists T \subseteq [D] \times [M] \text{ s.t. } |\mathrm{LIST}(T, 1)| \geq K \text{ and } |T| < KA,\end{aligned}$$

where the last equivalence follows because if $T = N(S)$, then $S \subseteq \mathrm{LIST}(T, 1)$, and conversely if $S = \mathrm{LIST}(T, 1)$ then $N(S) \subseteq T$. □

On one hand, this list-decoding property seems easier to establish than the ones for codes and extractors because we look at $\mathrm{LIST}(T, 1)$ instead of $\mathrm{LIST}(T, \mu(T) + \varepsilon)$. On the other hand, to get expansion (i.e. $A > 1$), we require a very tight relationship between $|T|$ and $|\mathrm{LIST}(T, 1)|$. In the setting of extractors or codes, we would not care much about a factor of 2 loss in $|\mathrm{LIST}(T)|$, as this corresponds to 1 bit of entropy loss for extractors or just a slightly larger list size for codes. But here it corresponds to a factor 2 loss in expansion, which can be quite significant. In particular, we cannot afford it if we are trying to get $A = (1 - \varepsilon) \cdot D$, as we will be in the next section.

6.2.6 Expanders from Parvaresh–Vardy Codes

Consider the bipartite multigraph obtained from the Parvaresh–Vardy codes (Construction 6.19) via the correspondence of Construction 6.23. That is, we define $\Gamma : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q \times \mathbb{F}_q^m$

$$\Gamma(f, y) = [y, f_0(y), f_1(y), \ldots, f_{m-1}(y)], \tag{6.5}$$

where $f(Y)$ is a polynomial of degree at most $n-1$ over $\mathbb{F}_q$, and we define $f_i(Y) = f(Y)^{h^i} \bmod E(Y)$, where E is a fixed irreducible polynomial of degree n over $\mathbb{F}_q$. (Note that we are using $n-1$ instead of d to denote degree of f.)

Theorem 6.30. Let $\Gamma : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q^{m+1}$ be the neighbor function of the bipartite multigraph corresponding to the q-ary Parvaresh–Vardy code of degree d, power h, and redundancy m via Construction 6.23. Then Γ is a (K_{max}, A) expander for $K_{max} = h^m$ and $A = q - nhm$.

Proof. Let K be any integer less than or equal to $K_{max} = h^m$, and let $A = q - nmh$. By Lemma 6.29, it suffices to show that for every set $T \subseteq \mathbb{F}_q^{m+1}$ of size at most $AK - 1$, we have $|\mathrm{LIST}(T)| \leq K - 1$.

We begin by doing the proof for $K = K_{max} = h^m$, and later describe the modifications to handle smaller values of K. The proof goes along the same lines as the list-decoding algorithm for the Parvaresh–Vardy codes from Section 6.2.3.

Step 1: Find a low-degree Q vanishing on T. We find a nonzero polynomial $Q(Y, Z_0, \ldots, Z_{m-1})$ of degree at most $d_Y = A - 1$ in its first variable Y and at most $h - 1$ in each of the remaining variables such that $Q(z) = 0$ for all $z \in T$. (Compare this to $Q(r, r(y)) = 0$ for all $y \in \mathbb{F}_q$ in the list-decoding algorithm, which corresponds to taking $T = T_r$.)

This is possible because

$$A \cdot h^m = AK > |T|.$$

Moreover, we may assume that Q is not divisible by $E(Y)$. If it is, we can divide out all the factors of $E(Y)$, which will not affect the conditions $Q(z) = 0$ since E has no roots (being irreducible).

Step 2: Argue that each $f(Y) \in \mathrm{LIST}(r)$ is a "root" of a related univariate polynomial Q^*. First, we argue as in the list-decoding algorithm that if $f \in \mathrm{LIST}(r, 1)$, we have

$$Q(Y, f_0(Y), \ldots, f_{m-1}(Y)) = 0.$$

This is ensured because

$$q > A - 1 + nmh.$$

(In the list-decoding algorithm, the left-hand side of this inequality was εq, since we were bounding $|\mathrm{LIST}(T_r, \varepsilon)|$.)

Once we have this, we can reduce both sides modulo $E(Y)$ and deduce

$$\begin{aligned} 0 &= Q(Y, f_0(Y), f_2(Y), \ldots, f_{m-1}(Y)) \bmod E(Y) \\ &= Q(Y, f(Y), f(Y)^2, \ldots, f(Y)^{m-1}) \bmod E(Y) \end{aligned}$$

Thus, if we define the univariate polynomial

$$Q^*(Z) = Q(Y, Z, Z^h, \ldots, Z^{h^{m-1}}) \bmod E(Y),$$

then $f(Y)$ is a root of Q^* over the field $\mathbb{F}_q[Y]/E(Y)$.

Observe that Q^* is nonzero because Q is not divisible by $E(Y)$ and has degree at most $h-1$ in each Z_i. Thus,

$$|\mathrm{LIST}(T,1)| \le \deg(Q^*) \le h-1+(h-1)\cdot h+(h-1)\cdot h^2+\cdots+(h-1)\cdot h^{m-1} = K-1.$$

(Compare this to the list-decoding algorithm, where our primary goal was to efficiently enumerate the elements of $\mathrm{LIST}(T,\varepsilon)$, as opposed to bound its size.)

Handling smaller values of K. We further restrict $Q(Y,Z_1,\ldots,Z_m)$ to only have nonzero coefficients on form $Y^i\mathrm{Mon}_j(Z_1,\ldots,Z_m)$ for $0 \le i \le A-1$ and $0 \le j \le K-1 \le h^m-1$, where $\mathrm{Mon}_j(Z_1,\ldots,Z_m) = Z_1^{j_0}\cdots Z_m^{j_{m-1}}$ and $j = j_0 + j_1 h + \cdots + j_{m-1}h^{m-1}$ is the base-h representation of j. Note that this gives us $AK > |T|$ monomials, so Step 1 is possible. Moreover $M_j(Z, Z^h, Z^{h^2}, \ldots, Z^{h^{m-1}}) = Z^j$, so the degree of Q^* is at most $K-1$, and we get the desired list-size bound in Step 3. □

We now set parameters to deduce the expander we used in the previous chapter (to get a condenser).

Theorem 6.31 (Theorem 5.41, restated). For every constant $\alpha > 0$, every $N \in \mathbb{N}$, $K \le N$, and $\varepsilon > 0$, there is an explicit $(K, (1-\varepsilon)D)$ expander with N left-vertices, M right-vertices, left-degree $D = O((\log N)(\log K)/\epsilon)^{1+1/\alpha}$ and $M \le D^2 \cdot K^{1+\alpha}$. Moreover, D is a power of 2.

Proof. Let $n = \log N$ and $k = \log K_{\max}$. Let $h = \lceil (2nk/\varepsilon)^{1/\alpha} \rceil$ and let q be the power of 2 in the interval $(h^{1+\alpha}/2, h^{1+\alpha}]$.

Set $m = \lceil (\log K_{\max})/(\log h) \rceil$, so that $h^{m-1} \le K_{\max} \le h^m$. Then, by Theorem 6.30, the graph $\Gamma : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q^{m+1}$ defined in (6.5) is an (h^m, A) expander for $A = q - nhm$. Since $K_{\max} \le h^m$, it is also a (K_{max}, A) expander.

Note that the number of left-vertices in Γ is $q^n \ge N$, and the number of right-vertices is

$$M = q^{m+1} \le q^2 \cdot h^{(1+\alpha)\cdot(m-1)} \le q^2 \cdot K_{\max}^{1+\alpha}.$$

The degree is

$$D = q \le h^{1+\alpha} = O(nk/\epsilon)^{1+1/\alpha} = O((\log N)(\log K_{\max})/\epsilon)^{1+1/\alpha}.$$

To see that the expansion factor $A = q - nhm \ge q - nhk$ is at least $(1-\varepsilon)D = (1-\varepsilon)q$, note that

$$nhk \le (\varepsilon/2)\cdot h^{1+\alpha} \le \varepsilon q,$$

where the first inequality holds because $h^\alpha \ge 2nk/\varepsilon$.

Finally, the construction is explicit because a description of $\mathbb{F}_q$ for q a power of 2 (i.e. an irreducible polynomial of degree $\log q$ over $\mathbb{F}_2$) as well as an irreducible polynomial $E(Y)$ of degree n over $\mathbb{F}_q$ can be found in time $\mathrm{poly}(n, \log q) = \mathrm{poly}(\log N, \log D)$. □

6.3 Exercises

Problem 6.1 (Limits of List Decoding). Show that if there exists a q-ary code $\mathcal{C} \subseteq \Sigma^{\hat{n}}$ of rate ρ that is (δ, L) list-decodable, then $\rho \leq 1 - H_q(\delta, \hat{n}) + (\log_q L)/\hat{n}$

Problem 6.2. (Concatenated Codes) For codes $\mathrm{Enc}_1 : \{1, \ldots, N\} \to \Sigma_1^{n_1}$ and $\mathrm{Enc}_2 : \Sigma_1 \to \Sigma_2^{n_2}$, their *concatenation* $\mathrm{Enc} : \{1, \ldots, N\} \to \Sigma_2^{n_1 n_2}$ is defined by

$$\mathrm{Enc}(m) = \mathrm{Enc}_2(\mathrm{Enc}_1(m)_1)\mathrm{Enc}_2(\mathrm{Enc}_1(m)_2) \cdots \mathrm{Enc}_2(\mathrm{Enc}_1(m)_{n_1}).$$

This is typically used as a tool for reducing alphabet size, e.g. with $\Sigma_2 = \{0, 1\}$.

(1) Prove that if Enc_1 has minimum distance δ_1 and Enc_2 has minimum distance δ_2, then Enc has minimum distance at least $\delta_1\delta_2$.
(2) Prove that if Enc_1 is $(1 - \varepsilon_1, \ell_1)$ list-decodable and Enc_2 is (δ_2, ℓ_2) list-decodable, then Enc is $((1 - \varepsilon_1\ell_2) \cdot \delta_2, \ell_1\ell_2)$ list-decodable.
(3) By concatenating a Reed–Solomon code and a Hadamard code, show that for every $n \in \mathbb{N}$ and $\varepsilon > 0$, there is a (fully) explicit code $\mathrm{Enc} : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ with blocklength $\hat{n} = O(n^2/\varepsilon^2)$ with minimum distance at least $1/2 - \varepsilon$. Furthermore, show that with blocklength $\hat{n} = \mathrm{poly}(n, 1/\varepsilon)$, we can obtain a code that is $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$ list-decodable in *polynomial time.* (Hint: the inner code can be decoded by brute force.)

Problem 6.3. (List Decoding implies Unique Decoding for Random Errors)

(1) Suppose that $\mathcal{C} \subseteq \{0,1\}^{\hat{n}}$ is a code with minimum distance at least $1/4$ and rate at most $\alpha\varepsilon^2$ for a fixed constant $\alpha > 0$ be determined below, and we transmit a codeword $c \in \mathcal{C}$ over a channel in which each bit is flipped with probability $1/2 - 2\varepsilon$. Show that if α is sufficiently small, then all but exponentially small probability over the errors, c will be the unique codeword at distance at most $1/2 - \varepsilon$ from the received word r.
(2) Using Problem 6.2, deduce that for every $\varepsilon > 0$ and $n \in \mathbb{N}$, there is an explicit code of blocklength $\hat{n} = \mathrm{poly}(n, 1/\varepsilon)$ that can be uniquely decoded from $(1/2 - 2\varepsilon)$ random errors as above in polynomial time.

Problem 6.4. (List-decoding Reed–Solomon Codes)

(1) Show that there is a polynomial-time algorithm for list-decoding the Reed-Solomon codes of degree d over $\mathbb{F}_q$ up to distance $1 - \sqrt{2d/q}$, improving the $1 - 2\sqrt{d/q}$ bound from lecture. (Hint: do not use fixed upper bounds on the individual degrees of the interpolating polynomial $Q(X, Y)$, but rather allow as many monomials as possible.)
(2) (*) Improve the list-decoding radius further to $1 - \sqrt{d/q}$ by using the 'multiple-roots' trick used in Section 6.2.4.

Problem 6.5. (Codes vs. Hashing) Given any code $\mathrm{Enc} : [N] \to [M]^{\hat{n}}$, we can obtain a family of hash functions $\mathcal{H} = \{h_i : [N] \to [M]\}_{i \in [\hat{n}]}$ defined by $h_i(x) = \mathrm{Enc}(x)_i$, and conversely.

(1) Show that Enc has minimum distance at least δ iff $\mathcal{H}$ has collision probability at most $1 - \delta$. That is, for all $x \neq y \in [N]$, we have $\Pr_i[h_i(x) = h_i(y)] \leq 1 - \delta$. (This is a generalization of the definition of universal hash functions, which correspond to the case that $\delta = 1 - 1/M$.)

(2) The Leftover Hash Lemma extends to families of functions with low collision probability; specifically if a family $\mathcal{H}$ with range $[M]$ has collision probability at most $(1+\varepsilon^2)/M$, then $\mathrm{Ext}(x, h) = (h, h(x))$ is a (k, ε) extractor for $k = m + 2\log(1/\varepsilon) + O(1)$, where $m = \log M$. Use this to prove the Johnson Bound for small alphabets: if a code $\mathrm{Enc} : [N] \to [M]^{\hat{n}}$ has minimum distance at least $1 - 1/M - \gamma/M$, then it is $(1 - 1/M - \sqrt{\gamma}, O(M/\gamma))$ list-decodable.

Problem 6.6. (Twenty Questions) In the game of 20 questions, an oracle has an arbitrary secret $s \in \{0,1\}^n$ and the aim is to determine the secret by asking the oracle as few yes/no questions about s as possible. It is easy to see that n questions are necessary and sufficient. Here we consider a variant where the oracle has two secrets $s_1, s_2 \in \{0,1\}^n$, and can adversarially decide to answer each question according to either s_1 or s_2. That is, for a question $f : \{0,1\}^n \to \{0,1\}$, the oracle may answer with either $f(s_1)$ or $f(s_2)$. Here it turns out to be impossible to pin down either of the secrets with certainty, no matter how many questions we ask, but we can hope to compute a small list L of secrets such that $|L \cap \{s_1, s_2\}| \neq \emptyset$. (In fact, $|L|$ can be made as small as 2.) This variant of twenty questions apparently arose from Internet routing algorithms used by Akamai.

(1) Let $\mathrm{Enc} : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ be a code such that that every two codewords in Enc *agree* in at least a $1/2 - \varepsilon$ fraction of positions and that Enc has a polynomial-time $(1/4 + \varepsilon, \ell)$ list-decoding algorithm. Show how to solve the above problem in polynomial time by asking the $\hat{n}$ questions $\{f_i\}$ defined by $f_i(x) = \mathrm{Enc}(x)_i$.

(2) Taking Enc to be the code constructed in Problem 1, deduce that $\hat{n} = \mathrm{poly}(n)$ questions suffices.

7

Pseudorandom Generators

7.1 Motivation and Definition

Our accomplishments in derandomization from the previous chapters include the following:

- Derandomizing specific algorithms, such as the ones for MaxCut and Undirected S-T Connectivity;
- Giving explicit (efficient, deterministic) constructions of various pseudorandom objects, such as expanders, extractors, and list-decodable codes, as well as showing various relations between them;
- Reducing the randomness needed for certain tasks, such as error reduction of randomized algorithms and sampling; and
- Simulating **BPP** with any weak random source.

However, all of these still fall short of answering our original motivating question, of whether *every* randomized algorithm can be efficiently derandomized. That is, does $\mathbf{BPP} = \mathbf{P}$?

As we have seen, one way to resolve this question in the positive is to use the following two-step process: First show that the number of random bits for any **BPP** algorithm can be reduced from $\text{poly}(n)$ to $O(\log n)$, and then eliminate the randomness entirely by enumeration.

Thus, we would like to have a function G that stretches a seed of $O(\log n)$ truly random bits into $\text{poly}(n)$ bits that "look random". Such a function is called a *pseudorandom generator.* The question is how we can formalize the requirement that the output should "look random" in such a way that (a) the output can be used in place of the truly random bits in *any* **BPP** algorithm, and (b) such a generator exists.

Some candidate definitions for "looks random" include the following:

- Information-theoretic or statistical measures: e.g., entropy, statistical difference from uniform distribution, pairwise independence. All of these fail one of the two criteria. For example, it is impossible for a deterministic function to increase entropy from $O(\log n)$

to n. And it is easy to construct algorithms that fail when run using random bits that are only guaranteed to be pairwise independent.

- Kolmogorov complexity, which is defined as follows: a string "looks random" if it is incompressible (cannot be generated by a Turing machine with a representation of length less than n). An appealing aspect of this notion is that it makes sense of the randomness in a fixed string (rather than a distribution). Unfortunately, it is not suitable for our purposes. Specifically, if the function G is computable (which we certainly want!) then all of its outputs have Kolmogorov complexity $O(\log n)$ (just hardwire the seed into the TM computing G), and hence are very compressible.
- Computational indistinguishability: this is the measure we will use. Intuitively, we say that a random variable X "looks random" if no *efficient* algorithm can distinguish X from a truly uniform random variable. Another way to look at it is as follows. Recall the definition of statistical difference:

  $$\Delta(X,Y) = \max_T |Pr[X \in T] - Pr[Y \in T]|.$$

  With computational indistinguishability, we simply restrict the max to be taken only over "efficient" statistical tests T (T's for which membership can be efficiently tested).

7.1.1 Computational Indistinguishability

~~Definition 7.1 (computational indistinguishability).~~ ~~Random variables X and Y taking values in $\{0,1\}^n$ are (t,ε) *indistinguishable* if for every *nonuniform* algorithm running in time t, we have~~

$$|\Pr[T(X)=1] - \Pr[T(Y)=1]| \leq \varepsilon$$

The left-hand side above is called also the *advantage* of T.

Recall that a nonuniform algorithm is an algorithm that may have some nonuniform advice hardwired in. If the algorithm runs in time t we require that the advice string is of length at most t. Typically, to make sense of complexity measures like running time, it is necessary to use asymptotic notions (e.g. because a Turing machine can encode a huge lookup table for inputs of any bounded size in its transition function). However, for nonuniform algorithms, we can avoid doing so by using Boolean circuits as our nonuniform model of computation. Similarly to Fact 3.11, every nonuniform Turing machine algorithm running in time $t(n)$ can be simulated by a sequence of Boolean circuit C_n of size $\tilde{O}(t(n))$ and conversely every sequence of Boolean circuits of size $s(n)$ can be simulated by a nonuniform Turing machine running in time $\tilde{O}(s(n))$. Thus, to make our notation cleaner, by "nonuniform algorithm running in time t', we mean "Boolean circuit of size t' (where the size is measured by the number of AND and OR gates in the circuit). Note also that we have not specified whether the the distinguisher is deterministic or randomized; this is because a probabilistic distinguisher achieving advantage greater than ε can be turned into a deterministic distinguisher achieving advantage greater than ε by nonuniformly fixing the randomness.

While we won't do so, it is also of interest to study computational indistinguishability and pseudorandomness against uniform algorithms.

Definition 7.2 (uniform computational indistinguishability). Let X_n, Y_n be some sequences of random variables on $\{0,1\}^n$ (or $\{0,1\}^{\mathrm{poly}(n)}$). For functions $t : \mathbb{N} \to \mathbb{N}$ and $\varepsilon : \mathbb{N} \to [0,1]$, we say that $\{X_n\}$ and $\{Y_n\}$ are $(t(n), \varepsilon(n))$ *indistinguishable for uniform algorithms* if for all probabilistic algorithms T running in time $t(n)$, we have that

$$|\Pr[T(X_n) = 1] - \Pr[T(Y_n) = 1]| \leq \varepsilon(n)$$

for all sufficiently large n, where the probabilities are taken over X_n, Y_n and the random coin tosses of T.

We will focus on the nonuniform definition, but will mention results about the uniform definition as well.

7.1.2 Pseudorandom Generators

~~**Definition 7.3.** A deterministic function $G : \{0,1\}^\ell \to \{0,1\}^n$ is a (t, ε) *pseudorandom generator (PRG)*~~ if

(1) $\ell < n$, and
(2) $G(U_\ell)$ and U_n are (t, ε) indistinguishable.

Also, note that we have formulated the definition with respect to nonuniform computational indistinguishability, but there is a natural uniform analogue of this definition.

People attempted to construct pseudorandom generators long before this definition was formulated. Their generators were tested against a battery of statistical tests (e.g. the number of 1's and 0's are approximately the same, the longest run is of length $O(\log n)$, etc.), but these fixed set of tests provided no guarantee that the generators will perform well in an arbitrary application (e.g. in cryptography or derandomization). Indeed, most classical constructions (e.g. linear congruential generators, as implemented in the standard C library) are known to fail in some applications.

Intuitively, the above definition guarantees that the pseudorandom bits produced by the generator are as good as truly random bits for *all* efficient purposes (where efficient means time at most t). In particular, we can use such a generator for derandomizing any algorithm of running time less than t. For the derandomization to be efficient, we will also need the generator to be efficiently computable.

Definition 7.4. We say a sequence of generators $\{G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^n\}$ is *computable in time* $t(n)$ if there is a *uniform and deterministic* algorithm M such that for every $n \in \mathbb{N}$ and $y \in \{0,1\}^{\ell(n)}$, we have $M(1^n, x) = G_n(x)$ and $M(1^n, x)$ runs in time at most $t(n)$.

Note that even when though we define the pseudorandomness property of the generator with respect to nonuniform algorithms, the efficiency requirement refers to uniform algorithms. As usual, for readability, we will usually refer to a single generator $G = G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^n$, with it being implicit that we are discussing a family $\{G_n\}$.

Theorem 7.5. Suppose that for all n there exists an $(n, 1/8)$ pseudorandom generator $G : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ computable in time $t(n)$. Then $\mathbf{BPP} \subseteq \bigcup_c \mathbf{DTIME}(2^{\ell(n^c)} \cdot (n^c + t(n^c)))$.

Proof. A **BPP** algorithm starts with some algorithm A taking an input x of length n and a sequence of random bits r, and then accepting or rejecting after at most n^c steps for some c. We can of course assume that the algorithm uses at most n^c random bits.

The idea will be to plug in the pseudorandom generator G_{n^c} to produce this sequence of random bits, then to use the pseudorandomness assumption to show that the algorithm will do just as well with that sequence, and then to enumerate over all possible seeds to produce a derandomization.

Claim 7.6. For every x of length n, $A(x; G_{n^c}(U_{\ell(n^c)}))$ errs with probability smaller than $1/2$.

> **Proof of claim:** Suppose that there exists some x on which $A(x; G_{n^c}(U_{\ell(n^c)}))$ errs with probability at least $1/2$. Then $T(\cdot) = A(x, \cdot)$ is a nonuniform algorithm running in time n^c that distinguishes $G_{n^c}(U_{\ell(n^c)})$ from U_{n^c} with advantage at least $1/2 - 1/3 > 1/8$. Notice that we are using x here as nonuniform advice; this is why we need the PRG to be robust against nonuniform tests. □

Now, enumerate over all seeds of length $\ell(n^c)$ and take a majority vote. There are $2^{\ell(n^c)}$ of them, and for each we have to run both G and A. □

Notice that we can afford for the generator G_n have running time $t(n) = \text{poly}(n)$ or even $t(n) = \text{poly}(n) \cdot 2^{O(\ell(n))}$ without affecting the time of the derandomization by than more than a polynomial amount. In particular, for this application, it is OK if the generator runs in more time than the tests it fools (which are time n in this theorem).

The theorem provides a mechanism to produce various different theorems, relating the existence of PRGs for certain seed lengths with the ability to derandomize. Let's look at some typical settings of parameters to see what we might imagine proving with this theorem. Assuming throughout that $t(n) \leq \text{poly}(n) \cdot 2^{O(\ell(n))}$, PRGs of various seed lengths can be used to simulate **BPP** in the following deterministic time classes (see Definition 3.1):

(1) Suppose that for every ε you can create a PRG with $\ell(n) = n^\varepsilon$. Then $\mathbf{BPP} \subseteq \bigcap_{\varepsilon>0} \mathbf{DTIME}(2^{n^\varepsilon}) \stackrel{\text{def}}{=} \mathbf{SUBEXP}$. Since we know that **SUBEXP** is a proper subset of **EXP**, this would be a nontrivial improvement on the current inclusion $\mathbf{BPP} \subseteq \mathbf{EXP}$ (Proposition 3.2).
(2) Suppose we had a PRG with $\ell(n) = \text{polylog}(n)$. Then $\mathbf{BPP} \subseteq \bigcup_c \mathbf{DTIME}(2^{\log^c n}) \stackrel{\text{def}}{=} \tilde{\mathbf{P}}$.
(3) Suppose we had a PRG with $\ell(n) = O(\log n)$. Then $\mathbf{BPP} = \mathbf{P}$.

Of course, all of these derandomizations are contingent on the question of whether PRGs exist. As usual, our first answer is yes but the proof is not very helpful—it is nonconstructive and thus does not provide for an efficiently computable PRG.

Proposition 7.7. For all $t \in \mathbb{N}$ and $\varepsilon > 0$, there exists a (t, ε) pseudorandom generator $G : \{0,1\}^\ell \to \{0,1\}^t$ with seed length $O(\log t + \log(1/\varepsilon))$.

Proof. The proof is by the probabilistic method. Choose $G : \{0,1\}^\ell \to \{0,1\}^n$ at random. Now, fix a time t algorithm, T. The probability (over choice of G) that T distinguishes $G(U_\ell)$ from U_n with advantage ε is at most $2^{-\Omega(2^\ell \varepsilon^2)}$, by a Chernoff bound argument. There are $2^{\text{poly}(t)}$ nonuniform algorithms running in time t (i.e. circuits of size t). Thus, union-bounding over all possible T, and setting $\varepsilon = 1/t$, we get that the probability that there exists a T breaking G is at most $2^{\text{poly}(t)} 2^{-\Omega(2^\ell \varepsilon^2)}$, which is less than 1 for ℓ being $O(\log t + \log(1/\varepsilon))$. □

Note that putting together Proposition 7.7 and Theorem 7.5 gives us another way to prove that **BPP** $\subseteq$ **P/poly** (Corollary 3.12). Just let the advice string be the truth table of the PRG for the proper length, and then one can use that PRG and the proof of Theorem 7.5 to derandomize **BPP**. However, if you unfold both this proof and our previous proof (where we do error reduction and then fix the coin tosses), you will see that both proofs amount to exactly the same "construction".

7.2 Cryptographic PRGs

The theory of computational pseudorandomness developed in this chapter emerged from cryptography, where researchers sought a definition that would ensure that using pseudorandom bits instead of truly random bits (e.g. when encrypting a message) would retain security against all computationally feasible attacks. In this setting, the generator G is used by the honest parties and thus should be very efficient to compute. On the other hand, the distinguisher T corresponds to an attack carried about by an adversary, and we want to protect against adversaries that invest a lot of computational resources into trying to break the system. Thus, one is led to require that the pseudorandom generators be secure even against adversaries with greater running time. The most common setting of parameters in the theoretical literature is that the generator should run in a fixed polynomial time, but the adversary can run in an arbitrary polynomial time.

Definition 7.8. A generator $G_n : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ is a *cryptographic pseudorandom generator* if

- G_n is computable in polynomial time. That is, there is a constant c such that G_n is computable in time n^c.
- G_n is an $(n^{\omega(1)}, 1/n^{\omega(1)})$ PRG. That is, for every constant d, G_n is an $(n^d, 1/n^d)$ pseudorandom generator for all sufficiently large n.

Due to time constraints and the fact that such generators are covered in other texts (see the Chapter Notes and References), we will not do an in-depth study of cryptographic generators, but just survey what is known about them.

The first question to ask is whether such generators exist at all. It is not hard to show that cryptographic pseudorandom generators cannot exist unless **P** $\neq$ **NP**, indeed unless **NP** $\not\subseteq$ **P/poly**. Thus, we do not expect to establish the existence of such generators unconditionally, and instead

need to make some complexity assumption. While it would be wonderful to show that **NP** $\not\subseteq$ **P/poly** implies that existence of cryptographic pseudorandom generators, that too seems out of reach. However, we can base them on the very plausible assumption that there are functions that are easy to evaluate but hard to invert.

Definition 7.9. $f_n : \{0,1\}^n \to \{0,1\}^n$ is a *one-way function* if:

(1) There is a constant c such that f is computable in time n^c.
(2) For every constant d and every nonuniform algorithm A running in time n^d:

$$\Pr[A(f(U_n)) \in f^{-1}(f(U_n))] \leq \frac{1}{n^d}$$

for all sufficiently large n.

Assuming the existence of one-way functions seems stronger than the assumption **NP** $\not\subseteq$ **BPP**. For example, it is an average-case complexity assumption, as it requires that f is hard to invert when evaluated on *random* inputs. Nevertheless, there are a number of candidate functions believed to be one-way. The simplest is integer multiplication: $f_n(x,y) = x \cdot y$, where x and y are $n/2$-bit numbers. Inverting this function is equivalent to the integer factorization problem, for which no efficient algorithm is known.

A classic and celebrated result in the foundations of cryptography is that cryptographic pseudorandom generators can be constructed from any one-way function:

Theorem 7.10. The following are equivalent:

(1) One-way functions exist.
(2) Cryptographic pseudorandom generators exist with seed length $\ell(n) = n - 1$.
(3) For every constant $\varepsilon > 0$, there exist cryptographic pseudorandom generators with seed length $\ell(n) = n^\varepsilon$.

Corollary 7.11. If one-way functions exist, then **BPP** $\subseteq$ **SUBEXP**.

What about getting a better derandomization? The proof of the above theorem is more general quantitatively. It takes any one-way function $f_\ell : \{0,1\}^\ell \to \{0,1\}^\ell$ and a parameter m, and constructs a generator $G_m : \{0,1\}^{\mathrm{poly}(\ell)} \to \{0,1\}^m$. The proof that G_m is pseudorandom is proven by a reduction as follows. Given any algorithm T that runs in time t and distinguishes G_m from uniform with advantage ε, we construct an algorithm T' running in time $t' = t \cdot (m/\varepsilon)^{O(1)}$ inverting f_ℓ (say with probability $1/2$).

Thus if f_ℓ is hard to invert by algorithms running in time $s(\ell)$, we can set $t = m = 1/\varepsilon = s(\ell)^{1/c}$ for a constant c. That is, viewing the seed length ℓ' of G_m as a function of m, we have $\ell'(m) = \mathrm{poly}(s^{-1}(m^c))$.

Thus:

- If $s(\ell)$ can be taken to be an arbitrarily large polynomial (as the definition of one-way function above), we get seed length $\ell'(m) = m^{\varepsilon}$ and $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$ (as discussed above).
- If $s(\ell) = 2^{\ell^{\Omega(1)}}$ (as is plausible for the factoring one-way function), then we get seed length $\ell'(m) = \text{poly}(\log m)$ and $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$.

But we cannot get seed length $\ell'(m) = O(\log m)$, as needed for concluding $\mathbf{BPP} = \mathbf{P}$, from this result. Even for the maximum possible hardness $s(\ell) = 2^{\Omega(\ell)}$, we get $\ell'(m) = \text{poly}(\log m)$. In fact, Problem **??** shows that it is impossible to have a cryptographic PRG with seed length $O(\log m)$ meeting Definition **??**, where we require that G_m be pseudorandom against all poly(m)-time algorithms. However, for derandomization we only need G_m to be pseudorandom against a fixed poly-time algorithm, e.g. running in time $t = m$, and we would get such generators with seed length $O(\log m)$ if the above construction could be improved to yield seed length $\ell' = O(\ell)$ instead of $\ell' = \text{poly}(\ell)$.

Open Problem 7.12. Given a one-way function $f : \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ that is hard to invert by algorithms running in time $s = 2^{\Omega(\ell)}$, is it possible to construct a poly(m)-time computable $(m, 1/8)$ pseudorandom generator $G : \{0,1\}^{\ell'} \to \{0,1\}^{m}$ with seed length $\ell' = O(\ell)$ and output length $m = 2^{\Omega}(\ell)$?

It is known how to do this from any one-way *permutation* $f : \{0,1\}^{\ell} \to \{0,1\}^{\ell}$. In fact, the construction of pseudorandom generators from one-way permutations has a particularly simple description:

$$G_m(x, r) = (\langle x, r\rangle, \langle f(x), r\rangle, \langle f(f(x)), r\rangle, \ldots, \langle f^{(m-1)}(x), r\rangle),$$

where $|r| = |x| = \ell$ and $\langle \cdot, \cdot \rangle$ denotes inner product modulo 2. One intuition for this construction is the following. Consider the sequence $(f^{(m-1)}(U_n), f^{(m-2)}(U_n), \ldots, f(U_n), U_n)$. By the fact that f is hard to invert (but easy to evaluate) it can be argued that the $i+1$'st component of this sequence is infeasible to predict from the first i components except with negligible probability. Thus, it is the computational analogue of a block source. The pseudorandom generator then is obtained by a computational analogue of block-source extraction, using the strong extractor $\text{Ext}(x, r) = \langle x, r\rangle$. The fact that the extraction works in the computational setting, however, is much more delicate and complex to prove than in the setting of extractors, and relies on a "local list-decoding algorithm" for the corresponding (Hadamard) code. (We will discuss local list decoding in Section **??**.)

Pseudorandom Functions. It turns out that a cryptographic pseudorandom generator can be used to build an even more powerful object — a family of *pseudorandom functions*. This is a family of functions $\{f_s : \{0,1\}^{\ell} \to \{0,1\}\}_{s \in \{0,1\}^{\ell}}$ such that (a) given the seed s, the function f_s can be evaluated in polynomial time, but (b) without the seed, it is infeasible to distinguish an oracle for f_s from an oracle to a truly random function. Thus in some sense, the ℓ-bit truly random seed s is stretched to 2^{ℓ} pseudorandom bits (namely the truth table of f_s)!

Pseudorandom functions have applications in several domains:

- Cryptography

When two parties share a seed s to a PRF, they effectively share a random function $f : \{0,1\}^\ell \to \{0,1\}$ (by definition, the function they share is indistinguishable from random by any poly-time 3rd party). Thus, in order for one party to send an encrypted message m to the other, they can simply choose a random $r \xleftarrow{R} \{0,1\}^\ell$, and send $(r, f_s(r) \oplus m)$. With knowledge of s, decryption is easy; simply calculate $f_s(r)$ and XOR it to the second part of the received message. However, the value $f_s(r) \oplus m$ would look essentially random to anyone without knowledge of s.

This is just one example; pseudorandom functions have vast applicability in cryptography.

- Learning Theory
  Here, PRFs are used mainly to prove negative results. The basic paradigm in computational learning theory is that we are given a list of examples of a function's behavior, $(x_1, f(x_2)), (x_2, f(x_2)), ..., (x_k, f(x_k)))$, and we would like to predict what the function's value will be on a new data point x_{k+1} coming from the same distribution. Information-theoretically, it should be possible to predict after a small number of samples assuming that the function has a small description (e.g. is computable by a poly-sized circuit). However, essentially by definition, it should be computationally hard to predict the output of PRFs. Thus, PRFs provide examples of functions that are efficiently computable yet hard to learn (even with membership queries).
- Hardness of Proving Circuit Lower Bounds.
  One main approach to proving $\mathbf{P} \neq \mathbf{NP}$ is to show that some $f \in \mathbf{NP}$ doesn't have polynomial size circuits (equivalently, $\mathbf{NP} \not\subseteq \mathbf{P/poly}$). This approach has had very limited success- the only superpolynomial lower bounds that have been achieved have been using very restricted classes of circuits (monotone circuits, constant depth circuits, etc). For general circuits, the best lower bound that has been achieved for a problem in **NP** is roughly $4.5n$.
  Pseudorandom functions have been used to help explain why existing lower-bound techniques have so far not yielded superpolynomial circuit lower bounds. Specifically, it has been shown that any sufficiently "constructive" proof of superpolynomial circuit lower bounds (one that would allow us to certify that a randomly chosen function has no small circuits) could be used to distinguish a pseudorandom function from truly random in subexponential time and thus invert any one-way function in subexponential time.

7.3 Hybrid Arguments

In this section, we introduce a very useful proof method for working with computational indistinguishability, known as the *hybrid argument*. We use it to establish two important facts — that computational indistinguishability is preserved under taking multiple samples, and that pseudorandomness is equivalent to next-bit unpredictability.

7.3.1 Indistinguishability of Multiple Samples

The following proposition illustrates that computational indistinguishability behaves like statistical difference when taking many independent repetitions; the distance ε multiplies by the number of copies. Proving it will introduce useful techniques for reasoning about computational indistinguishability, and will also illustrate how working with such computational notions can be more

subtle than working with statistical notions.

Proposition 7.13. If X and Y are (t, ε) indistinguishable, then for every k, X^k and Y^k are $(t, k\varepsilon)$ indistinguishable (where X^k represents k independent copies of X).

Proof. We will prove the contrapositive: if there is an efficient algorithm T distinguishing X^k and Y^k with advantage greater than $k\varepsilon$, then there is an efficient algorithm T' distinguishing X and Y with advantage greater than ε. The algorithm T' will naturally use the algorithm T as a subroutine. Thus this is a *reduction* in the same spirit as reductions used elsewhere in complexity theory (**NP**-completeness). The difference in this proof from the corresponding result about statistical difference is that we need to preserve efficiency when going from T to T'.

Suppose that there exists a nonuniform algorithm T such that

$$\left|\Pr[T(X^k) = 1] - \Pr[T(Y^k) = 1]\right| > k\varepsilon \tag{7.1}$$

We can drop the absolute value in the above expression without loss of generality. (Otherwise we can replace T with its negation; recall that negations are free in our measure of circuit size.)

Now we will use a "hybrid argument." Consider the hybrid distributions $H_i = X^{k-i}Y^i$, for $i = 0, \ldots, k$. Note that $H_0 = X^k$ and $H_k = Y^k$.

Then Inequality (7.1) is equivalent to

$$\sum_{i=1}^{k} \Pr[T(H_{i-1}) = 1] - \Pr[T(H_i) = 1] > k\varepsilon,$$

meaning that there exists some i such that $\Pr[T(H_{i-1}) = 1] - \Pr[T(H_i) = 1] > \varepsilon$. The latter simply says that

$$\Pr[T(X^{k-i}XY^{i-1}) = 1] - \Pr[T(X^{k-i}YY^{i-1}) = 1] > \varepsilon.$$

By averaging, there exists some $x_1, \ldots x_{k-i}$ and $y_{k-i+2}, \ldots y_k$ such that

$$\Pr[T(x_1, \ldots x_{k-i}, X, y_{k-i+2}, \ldots y_k) = 1] - \Pr[T(x_1, \ldots x_{k-i}, Y, y_{k-i+2}, \ldots y_k) = 1] > \varepsilon.$$

Then, define $T'(z) = T(x_1, \ldots x_{k-i}, z, y_{k-i+2}, \ldots, y_k)$. Note that T' is a nonuniform algorithm with advice i, $x_1, \ldots, x_{k-i}$, $y_{k-i+2}, \ldots y_k$ hardwired in. Hardwiring these things actually costs nothing in terms of circuit size (because constant inputs can be propagated through the circuit, only eliminating gates). Thus T' is a time t algorithm such that

$$\Pr[T'(X) = 1] - \Pr[T'(Y) = 1] > \varepsilon,$$

contradicting the indistinguishability of X and Y. □

While the parameters in the above result seem to behave nicely, with (t, ε) going to $(t, k\varepsilon)$, it is actually more costly than the corresponding result for statistical difference. First, the amount of nonuniform advice used by T' is larger than that used by T. This is hidden by the fact that we are using the same measure t (namely circuit size) to bound both the time and the advice length. Second, the result is meaningless for large values of k (e.g. $k = t$), because a time t algorithm cannot read more than t bits of the input distribution X^k and Y^k.

We note that there is an analogue of the above result for computational indistinguishability against *uniform* algorithms (Definition 7.2), but it is more delicate, because we cannot simply hardwire i, $x_1, \ldots, x_{k-i}$, $y_{k-i+2}, \ldots, y_k$ as advice. Indeed, the proposition as stated is known to be false. We need to add the additional condition that the distributions X and Y are efficiently samplable. Then T' can choose $i \xleftarrow{R} [k]$ at random, and randomly sample $x_1, \ldots, x_{k-i} \xleftarrow{R} X$, $y_{k-i+2}, \ldots, y_k \xleftarrow{R} Y$.

7.3.2 Next-Bit Unpredictability

In analyzing the pseudorandom generators that we construct, it will be useful to work with a reformulation of the pseudorandomness property, which says that, given a prefix of the output, it should be hard to predict the next bit.

For notational convenience, we deviate from our usual conventions use X to refer to an r.v. on $\{0,1\}^n$ which is part of an ensemble, and we use X_i for some $i \in [n] = \{1, \ldots, n\}$ to denote the ith bit of X. We have:

Definition 7.14. Let X be a random variable distributed on $\{0,1\}^n$. For $t \in \mathbb{N}$ and $\varepsilon \in [0,1]$, we say that X is (t, ε) *next-bit unpredictable* if for every nonuniform probabilistic algorithm P running in time $t(n)$ and every $i \in [n]$, we have:

$$\Pr\left[P(X_1 X_2 \cdots X_{i-1}) = X_i\right] \leq \frac{1}{2} + \varepsilon,$$

where the probability is taken over X and the coin tosses of P.

Note that the uniform distribution $X \equiv U_n$ is $(t, 0)$ next-bit unpredictable for every t. Intuitively, if X is pseudorandom, it must be next-bit unpredictable, as this is just one specific test one can perform on X. In fact the converse also holds, and this is the direction we will use.

Proposition 7.15. Let X be a random variable distributed on $\{0,1\}^n$. If X is a (t, ε) pseudorandom, then X is $(t - O(1), \varepsilon)$ next-bit unpredictable. Conversely, if X is (t, ε) next-bit unpredictable, then it is $(t, n \cdot \varepsilon)$ pseudorandom.

Proof. Here U denotes an r.v. uniformly distributed on $\{0,1\}^n$ and U_i denotes the i'th bit of U.

pseudorandom ⇒ next-bit unpredictable. The proof is by reduction. Suppose for contradiction that X is not $(t - O(n), \varepsilon)$ next-bit unpredictable, so we have a predictor $P : \{0,1\}^{i-1} \to \{0,1\}$ that succeeds with probability at least $1/2 + \varepsilon$. We construct an algorithm $T : \{0,1\}^n \to \{0,1\}$ that distinguishes X from U_n as follows:

$$T(x_1 x_2 \cdots x_n) = \begin{cases} 1 & \text{if } P(x_1 x_2 \cdots x_{i-1}) = x_i \\ 0 & \text{otherwise.} \end{cases}$$

next-bit unpredictable ⇒ pseudorandom. Also by reduction. Suppose X is not pseudorandom, so we have a nonuniform algorithm T running in time t s.t.

$$\Pr[T(X) = 1] - \Pr[T(U) = 1] > \varepsilon,$$

where we have dropped the absolute values without loss of generality as in the proof of Proposition 7.13.

We now use a hybrid argument. Define $H_i = X_1 \circ X_2 \circ \cdots \circ X_i \circ U_{i+1} \circ U_{i+2} \circ \cdots \circ U_n$. Then $H_n = X$ and $H_0 = U$. We have:

$$\sum_{i=1}^{n} (\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1]) > \varepsilon,$$

since the sum telescopes. Thus, there must exist an i such that

$$\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1] > \varepsilon/n.$$

This says that T is more likely to output 1 when we put X_i in the i'th bit than when we put a random bit U_i. We can view U_i as being X_i with probability $1/2$ and being $\overline{X_i}$ with probability $1/2$. The only advantage T has must be coming from the latter case, because in the former case, the two distributions are identical. Formally,

$$\begin{aligned} &\Pr[T(X_1 \cdots X_{i-1} X_i U_{i+1} \cdots U_n) = 1] + 1 - \Pr[T(X_1 \cdots X_{i-1} \overline{X_i} U_{i+1} \cdots U_n) = 1] \\ &= \quad 2 \cdot (\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1]) > \frac{2\varepsilon}{n}. \end{aligned}$$

This motivates the following next-bit predictor:

$P(x_1 x_2 \cdots x_{i-1})$:

(1) Choose random bits $u_i, \ldots, u_n \xleftarrow{R} \{0,1\}$.
(2) Compute $b = T(x_1 \cdots x_{i-1} u_i \cdots u_n)$.
(3) If $b = 1$, output u_i, otherwise output $\overline{u_i}$.

The intuition is that T is more likely to output 1 when $u_i = x_i$ than when $u_i = \overline{x_i}$. Formally, we have:

$$\begin{aligned} &\Pr[P(X_1 \cdots X_{i-1}) = X_i] \\ &= \quad \frac{1}{2} \cdot (\Pr[T(X_1 \cdots X_{i-1} U_i U_{i+1} \cdots U_n) = 1 | U_i = X_i] + \Pr[T(X_1 \cdots X_{i-1} U_i U_{i+1} \cdots U_n) = 0 | U_i \neq X_i]) \\ &= \quad \frac{1}{2} \cdot \left(\Pr[T(X_1 \cdots X_{i-1} X_i U_{i+1} \cdots U_n) = 1] + 1 - \Pr[T(X_1 \cdots X_{i-1} \overline{X_i} U_{i+1} \cdots U_n) = 1]\right) \\ &> \quad \frac{1}{2} + \frac{\varepsilon}{n}. \end{aligned}$$

Note that as described P runs in time $t + O(n)$. Using circuit size as our measure of nonuniform time, we can reduce its running time to t as follows. First, we may nonuniformly fix the coin tosses $u_i, \ldots, u_n$ of P while preserving its advantage. Then all P does is run T on $x_1 \cdots x_{i-1}$ concatenated with some fixed bits and and either output what T does or its negation (depending on the fixed value of u_i). Fixing some input bits and negation can be done without increasing circuit size. Thus we contradict the next-bit unpredictability of X. □

We note that an analogue of this result holds for uniform distinguishers and predictors, provided that we change the definition of next-bit predictor to involve a random choice of $i \xleftarrow{R} [n]$ instead

of a fixed value of i, and change the time bounds in the conclusions to be $t - O(n)$ rather than $t - O(1)$ and t (we can't do tricks like in the final paragraph of the proof). In contrast to the multiple-sample indistinguishability result of Proposition 7.13, this result does not need X to be efficiently samplable for the uniform version.

7.4 Pseudorandom Generators from Average-Case Hardness

In Section 7.2, we surveyed cryptographic pseudorandom generators, which numerous applications within and outside cryptography, including to derandomizing **BPP**. However, for derandomization, we can use generators with weaker properties. Specifically, we only need $G : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ such that:

(1) G fools (nonuniform) distinguishers running in time n (as opposed to all poly(n)-time distinguishers).
(2) G is computable in time poly$(n, 2^\ell)$. In particular, *the PRG may take more time than the distinguishers it is trying to fool.*

Such a generator implies that every **BPP** algorithm can be derandomized in time poly$(n, 2^{\ell(n)})$.

The benefit of studying such generators is that we can hope to construct them under weaker assumptions than used for cryptographic generators. In particular, a generator with the properties above no longer implies $\mathbf{P} \neq \mathbf{NP}$, much less the existence of one-way functions. (Testing whether a string is an output of the generator is still an **NP** search problem, but even if we guess the seed properly, testing may take more time than the distinguishers are allowed.) However, as shown in Problem **??**, such generators still imply nonuniform circuit lower bounds for exponential time, something that is still beyond the state of the art in complexity theory. Our goal in the next couple of sections is to construct generators as above from assumptions that are as weak as possible. In this section, we will construct them from boolean functions computable in exponential time that are hard on average for nonuniform algorithms, and in the next section we will relax this to only require worst-case hardness.

7.4.1 Average-Case Hardness

A function is *hard on average* if it is hard to compute correctly on randomly chosen inputs. Formally:

Definition 7.16. For $t \in \mathbb{N}$ and $\delta \in [0,1]$, we say that a Boolean function $f : \{0,1\}^\ell \to \{0,1\}$ is (t, δ) *average-case hard* if for all nonuniform probabilistic algorithm A running in time t,

$$\Pr[A(X) = f(X)] \leq 1 - \delta.$$

Note that saying that f is (t, δ) hard for some $\delta > 0$ (possibly exponentially small) amounts to saying that f is *worst-case* hard (at least for deterministic algorithms; defining worst-case hardness for probabilistic algorithms is a bit more delicate). Thus, average-case hardness corresponds to δ that are noticeably larger than zero, e.g. $1/t^{.1}$ or constant. Indeed, in this section we will $\alpha = 1/2 - \varepsilon$ for $\varepsilon = 1/t$. That is, no efficient algorithm can compute f much better than random guessing. A typical setting of parameters we use is $t = t(\ell)$ somewhere in range from $\ell^{\omega(1)}$ (slightly superpolynomial)

to $t(\ell) = 2^{\alpha\ell}$ for a constant $\alpha > 0$. (Note that every function is computable by a nonuniform algorithm running in time roughly 2^ℓ, so we cannot take $t(\ell)$ to be any larger.) We will also require f is computable in (uniform) time $2^{O(\ell)}$ so that our pseudorandom generator will be computable in time exponential in its seed length. The existence of such an average-case hard function is quite a strong assumption, but in Section **??** we will see how to relax it to a worst-case hardness assumption.

Now we show how to obtain a pseudorandom generator from average-case hardness.

Proposition 7.17. If $f : \{0,1\}^\ell \to \{0,1\}$ is $(t, 1/2 - \varepsilon)$ average-case hard, then $G(x) = x \circ f(x)$ is a (t, ε) pseudorandom generator.

Proof. This follows from the equivalence of pseudorandomness and next-bit unpredictability. Considering uniformly random seed X, we certainly can't predict the first ℓ bits with any advantage whatsoever, so the only hope is to predict $f(x)$ from x, but f is $(\frac{1}{2} - \varepsilon)$-hard. A black-box application of Theorem **??** would lose a factor of $\ell + 1$ in the advantage ε, but we do not need to pay it here because the first ℓ bits are perfectly uniform. (Following the proof of Theorem **??**, we would have $\Pr[T(H_i) = 1] - \Pr[T(H_{i-1}) = 1] = 0$ for $i = 1, \ldots, \ell$.) □

Note that this generator includes its seed in its output. This is impossible for cryptographic pseudorandom generators, but is feasible (as shown above) when the generator can have more resources than the distinguishers it is trying to fool.

Of course, this generator is quite weak, stretching by only one bit. We would like to get many bits out. Here are two attempts:

- Define $G(x_1 \cdots x_k) = x_1 \cdots x_k f(x_1) \cdots f(x_k)$. This is a $(t, k\varepsilon)$ pseudorandom generator because we have k independent samples of a pseudorandom distribution so nonuniform computational indistinguishability is preserved. Note that already here we are relying on *nonuniform* indistinguishability, because the distribution $(U_\ell, f(U_\ell))$ is not samplable (in time that is feasible for the distinguishers). Unfortunately, however, this construction does not improve the ratio between output length and seed length, which remains very close to 1.
- Use composition. For example, try to get two bits out using the same seed length by defining $G'(x) = G(G(x)_1 \cdots G(x)_\ell)G(x)_\ell$. This works for cryptographic pseudorandom generators, but not for the generators we are considering here. Indeed, for the generator $G(x) = xf(x)$ of Proposition 7.17, we would get $G'(x) = xf(x)f(x) \cdots f(x)$, which is clearly not pseudorandom.

7.4.2 The Nisan–Wigderson Generator

Our goal now is to show the following:

Theorem 7.18. For $t : \mathbb{N} \to \mathbb{N}$, suppose that there is a function $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(\ell)})$ [1] such that for every input length $\ell \in \mathbb{N}$, f is $(1/2 - 1/t(\ell))$-hard for nonuniform time $t(\ell)$. Then for every

[1] $\mathbf{E}$ should be contrasted with the larger class $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(\ell)})$

$m \in \mathbb{N}$, there is an $(m, 1/m)$ pseudorandom generator $G : \{0,1\}^{\ell'(m)} \to \{0,1\}^m$ with seed length $\ell'(m) = O(t^{-1}(\text{poly}(m))^2/\log m)$ that is computable in time $2^{O(\ell'(m))}$.

Note that this is similar to the seed length $\ell'(m) = \text{poly}(s^{-1}(\text{poly}(m)))$ mentioned in Section 7.2 for constructing cryptographic pseudorandom generators from one-way functions, but the assumption is incomparable (and will be weakened further in the next section). In fact, it is known how to achieve a seed length $\ell(m) = O(t^{-1}(\text{poly}(m)))$, which matches what is known for constructing pseudorandom generators from one-way permutations as well as the converse implication of Problem **??**. We will not cover this improvement here, but note that for the important case of hardness $t(\ell) = 2^{\Omega(\ell)}$, we still achieve seed length $\ell(m) = O(O(\log m)^2/\log m) = O(\log m)$ and thus $\mathbf{P} = \mathbf{BPP}$. More generally, we have:

Corollary 7.19. Suppose that $\mathbf{E}$ has a $(t(\ell), 1/2 - 1/t(\ell))$ average-case hard function $f : \{0,1\}^\ell \to \{0,1\}$.

(1) If $t(\ell) = 2^{\Omega(\ell)}$, then $\mathbf{BPP} = \mathbf{P}$.
(2) If $t(\ell) = 2^{\ell^{\Omega(1)}}$, then $\mathbf{BPP} \subseteq \tilde{\mathbf{P}}$.
(3) If $t(\ell) = \ell^{\omega(1)}$, then $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$.

The idea is to apply f on *slightly dependent* inputs, i.e. x_i and x_j share very few bits. The sets of seed bits used for each output bit will be given by a *design*:

Definition 7.20. $S_1, \cdots, S_m \subseteq [d]$ is an (ℓ, a)-design if

(1) $\forall i, |S_i| = \ell$
(2) $\forall i \neq j, |S_i \cap S_j| \leq a$

We want lots of sets having small intersections over a small universe. We will use the designs established by Problem 3.2:[2]

Lemma 7.21. For every constant $\gamma > 0$ and every $\ell, m \in \mathbb{N}$, there exists an (ℓ, a)-design $S_1, \cdots, S_m \subseteq [d]$ with $d = O\left(\frac{\ell^2}{a}\right)$ and $a = \gamma \cdot \log m$. Such a design can be constructed deterministically in time $\text{poly}(m, d)$.

Construction 7.22 (Nisan–Wigderson Generator). Given an (ℓ, a)-design $S_1, \cdots, S_m \subseteq [d]$ and a function $f : \{0,1\}^\ell \to \{0,1\}$, define the *Nisan–Wigderson generator* $G : \{0,1\}^d \to \{0,1\}^m$ as

$$G(x) = f(x|_{S_1}) f(x|_{S_2}) \cdots f(x|_{S_m})$$

where if x is a string in $\{0,1\}^d$ and $S \subseteq [d]$, $|S| = \ell$, $x|_S$ is the string of length ℓ obtained from x by selecting the bits indexed by S.

---

[2] Problem 3.2 was for the special case $\gamma = 1$, but the same proof yields the lemma for all γ.

Theorem 7.23. Let $G : \{0,1\}^d \to \{0,1\}^m$ be the Nisan–Wigderson generator based on an (ℓ, a) design and a function $f : \{0,1\}^\ell \to \{0,1\}$. If f is $(1/2 - \varepsilon/m)$-hard for nonuniform time t, then G is a (t', ε) pseudorandom generator, for $t' = t - m \cdot a \cdot 2^a$.

Theorem 7.18 follows from Theorem 7.23 by setting $\varepsilon = 1/m$ and $a = \log m$, and observing that if for $\ell = t^{-1}(m^3)$, then $t' = t(\ell) - O(m \cdot a \cdot 2^a) \geq m$, so we have an $(m, 1/m)$ pseudorandom generator. The seed length is $\ell'(m) = d = O(\ell^2/\log m) = O(t^{-1}(\mathrm{poly}(m))^2/\log m)$.

Proof. Suppose G is not an (t', ε) pseudorandom generator. By Theorem **??**, there is a nonuniform time t' next-bit predictor P such that

$$\Pr[P(f(X|_{S_1})f(X|_{S_2}) \cdots f(X|_{S_{i-1}})) = f(X|_{S_i})] > \frac{1}{2} + \frac{\varepsilon}{m}, \tag{7.2}$$

for some $i \in [m]$. From P, we construct A that computes f with probability greater than $1/2 + \varepsilon/m$.

Let $Y = X|_{S_i}$. By averaging, we can fix all bits of $X|_{\bar{S}_i} = z$ such that the prediction probability is at least $1/2 + \varepsilon/m$ (over Y and the coin tosses of the predictor P). Define $f_j(y) = f(x|_{S_j})$ for $j \in \{1, \cdots, i-1\}$. (That is, $f_j(y)$ forms x by placing y in the positions in S_i and z in the others, and then applies f to $x|_{S_i}$). Then

$$\Pr_Y[P(f_1(Y) \cdots f_{i-1}(Y)) = f(Y)] > \frac{1}{2} + \frac{\varepsilon}{m}.$$

Note that $f_j(y)$ depends only on $|S_i \cap S_j| \leq a$ bits of y. Thus, we can compute each f_j with a look-up table, which we can include in the advice to our nonuniform algorithm. Indeed, every function on a bits can be computed by a boolean circuit of size at most $a \cdot 2^a$. (In fact, size at most $O(2^a/a)$ suffices.)

Then, defining $A(y) = P(f_1(y) \cdots f_{i-1}(y))$, we deduce that $A(y)$ can be computed with error probability smaller than $1/2 - \varepsilon/m$ in nonuniform time less than $t' + m \cdot a \cdot 2^a = t$. This contradicts the hardness of f. Thus, we conclude G is an (m, ε) pseudorandom generator. □

We make the following additional remarks:

(1) This is a very general construction that works for any average-case hard function f. We only used $f \in \mathbf{E}$ to deduce G is computable in $\mathbf{E}$.
(2) The reduction works for any nonuniform class of algorithms $\mathcal{C}$ where functions of logarithmically many bits can be computed efficiently.

Indeed, we will now use the same construction to obtain an *unconditional* pseudorandom generator following constant-depth circuits.

7.4.3 Derandomizing Constant-depth circuits

~~**Definition 7.24.** An *unbounded fan-in circuit* $C(x_1, \ldots, x_n)$ has input gates consisting of variables~~ x_i, their negations $\neg x_i$, and the constants 0 and 1, as well as computation gates, which can compute the AND or OR of an unbounded number of other gates (rather than just 2, as in usual Boolean circuits).[3] The *size* of such a circuit is the number of computation gates, and the *depth* is the

[3] Note that it is unnecessary to allow internal NOT gates, as these can always be pushed to the inputs via DeMorgan's Laws at no increase in size or depth.

maximum of length of a path from an input gate to the output gate.

$\mathbf{AC^0}$ is the class of functions $f : \{0,1\}^* \to \{0,1\}$ for which there exist constants c and d and a uniformly constructible sequence of unbounded fan-in circuits $(C_n)_{n\in\mathbb{N}}$ such that for all n, C_n has size at most n^c and depth at most d, and for all $x \in \{0,1\}^n$, $C_n(x) = f(x)$. $\mathbf{BPAC^0}$ defined analogously, except that C_n may have extra inputs, which are interpreted as random bits, and we we require $\Pr_r[C_n(x,r) = f(x)] \geq 2/3$.

$\mathbf{AC^0}$ is one of the richest circuit classes for which we have superpolynomial lower bounds:

Theorem 7.25. For all constant $d \in \mathbb{N}$ and every $\ell \in \mathbb{N}$, the function $\textsc{Par}_\ell : \{0,1\}^\ell \to \{0,1\}$ defined by $\textsc{Par}_\ell(x_1,\ldots,x_\ell) = \bigoplus_{i=1}^{\ell} x_i$ is $(t_d(\ell), 1/2 - 1/t_d(\ell))$-average-case hard for nonuniform unbounded fan-in circuits of depth d and size $t_d(\ell) = 2^{\Omega(\ell^{1/d})}$.

In addition to having an average-case hard function against $\mathbf{AC^0}$, we also need that $\mathbf{AC^0}$ can compute arbitrary functions on a logarithmic number of bits.

Lemma 7.26. Every function $g : \{0,1\}^a \to \{0,1\}$ can be computed by a depth 2 circuit of size 2^a.

Using the two facts with the Nisan–Wigderson pseudorandom generator construction, we obtain the following pseudorandom generator for constant-depth circuits.

Theorem 7.27. For every constant d and every m, there exists a $\mathrm{poly}(m)$-time computable $(m, 1/m)$-pseudorandom generator $G_m : \{0,1\}^{\log^{O(d)} m} \to \{0,1\}^m$ fooling nonuniform unbounded fan-in circuits of depth d (and size m).

Proof. This is proven similarly to Theorems 7.18 and 7.23, except that we take $f = \textsc{Par}_\ell$ rather than a hard function in $\mathbf{E}$, and we observe that the reduction can be implemented in a way that increases the depth by only an additive constant. Specifically, to obtain a pseudorandom generator fooling circuits of depth d, we use the hardness of $\textsc{Par}_\ell$ against unbounded fan-in circuits of depth $d' = d+2$ and size $\mathrm{poly}(m)$. Then the seed length of G $O(\ell^2/a) < O(\ell^2) = O(\log^{d'} m)^2 = \log^{O(d)} m$.

We now follow the steps of the proof of Theorem 7.18 to go from an adversary T of depth d breaking the pseudorandomness of G to a circuit A of depth d' calculating the parity function $\textsc{Par}_\ell$.

If T has depth d, then it can be verified that the next-bit predictor P constructed in the proof of Proposition 7.15 also has depth d. (Recall that negations and constants can be propagated to the inputs so they do not contribute to the depth.) Next, in the proof of Theorem 7.23, we obtain A from P by $A(y) = P(f_1(y)f_2(y)\cdots f_{i-1}(y))$ for some $i \in \{1,\ldots,m\}$ and where each f_i depends on at most a bits of y. Now we observe that A can be computed by a small constant-depth circuit (if P can). Specifically, applying Lemma **??** to each f_i, the size of A is at most $O(m \cdot 2^a) = O(m^2)$ plus the size of P and the depth of A is at most $d+2$. This contradicts the hardness of $\textsc{Par}_\ell$. □

Corollary 7.28. $\mathbf{BPAC^0} \subseteq \tilde{\mathbf{P}}$.

With more work, this can be strengthened to actually put $\mathbf{BPAC^0}$ in $\widetilde{\mathbf{AC}}^{\mathbf{0}}$. (The difficulty is that we use majority voting in the derandomization, but small constant-depth circuits cannot compute majority. However, they can compute an "approximate" majority, and this suffices.)

The above pseudorandom generator can also be used to give a quasipolynomial-time derandomization of the randomized algorithm we saw for approximately counting the number of satisfying assignments to a DNF formula (Theorem 2.34); see Problem **??**.

Improving the running time of either of these derandomizations to polynomial is an intriguing open problem.

Open Problem 7.29. Show that $\mathbf{BPAC^0} = \mathbf{AC^0}$ or even $\mathbf{BPAC^0} \subseteq \mathbf{P}$.

Open Problem 7.30 (Open Problem 2.36, restated). Give a deterministic polynomial-time algorithm for approximately counting the number of satisfying assignments to a DNF formula.

7.5 Worst-Case/Average-Case Reductions and Locally Decodable Codes

In the previous section, we saw how to construct pseudorandom generators from boolean functions that are very hard on average, where every nonuniform algorithm running in time t must err with probability greater than $1/2 - 1/t$ on a random input. Now we want to relax the assumption to refer to worst-case hardness, as captured by the following definition.

Definition 7.31. A function $f : \{0,1\}^\ell \to \{0,1\}$ is *worst-case hard for (nonuniform) time t* if, for all (nonuniform) probabilistic algorithms A running in time t, there exists $x \in \{0,1\}^\ell$ such that $\Pr[A(x) \neq f(x)] > 1/3$, where the probability is over the coin tosses of A.

Note that, for *deterministic* algorithms A, the definition simply says $\exists x\ A(x) \neq f(x)$. In the nonuniform case, restricting to deterministic algorithms is without loss of generality because we can always derandomize the algorithm using (additional) nonuniformity. Specifically, following the proof that $\mathbf{BPP} \subseteq \mathbf{P/poly}$, it can be shown that if f is worst-case hard for nonuniform deterministic algorithms running in time t, then it is worst-case hard for nonuniform probabilistic algorithms running in time t' for $t' = \Omega(t/\ell)$.

A natural goal is to be able to construct an average-case hard function from a worst-case hard function. More formally, given a function $f : \{0,1\}^\ell \to \{0,1\}$ vs. time $t = t(\ell)$, construct a function $\hat{f} : \{0,1\}^{O(\ell)} \to \{0,1\}$ such that $\hat{f}$ is average-case hard vs. time $t' = t^{\Omega(1)}$. Moreover, we would like $\hat{f}$ to be in $\mathbf{E}$ if f is in $\mathbf{E}$. (Whether we can obtain a similar result for $\mathbf{NP}$ is a major open problem, and indeed there are negative results ruling out natural approaches to doing so.)

Our approach to doing this will be via error-correcting codes. Specifically, we will show that if $\hat{f}$ is the encoding of f in an appropriate kind of error-correcting code, then worst-case hardness of f implies average-case hardness of $\hat{f}$.

Specifically, we view f as a message of length $L = 2^\ell$, and apply an error-correcting code $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$ to obtain $\hat{f} = \mathrm{Enc}(f)$, which we view as a function $\hat{f} : \{0,1\}^{\hat{\ell}} \to \Sigma$, where $\hat{\ell} = \log \hat{L}$. Pictorially:

$$\boxed{\text{message } f : \{0,1\}^\ell \to \{0,1\}} \longrightarrow \boxed{\mathrm{Enc}} \longrightarrow \boxed{\text{codeword } \hat{f} : \{0,1\}^{\hat{\ell}} \to \Sigma}.$$

(Ultimately, we would like $\Sigma = \{0,1\}$, but along the way we will discuss larger alphabets.)

Now we argue the average-case hardness of $\hat{f}$ as follows. Suppose, for contradiction, that $\hat{f}$ is not δ average-case hard. By definition, there exists an efficient algorithm A with $\Pr[A(x) = \hat{f}(x)] > 1-\delta$. We may assume that A is deterministic by fixing its coins. Then A may be viewed as a received word in $\Sigma^{\hat{L}}$, and our condition on A becomes $\Delta(A, \hat{f}) < \delta$. So if Dec is a δ-decoding algorithm for Enc, then $\mathrm{Dec}(A) = f$. By assumption A is efficient, so if Dec is efficient, then f may be efficiently computed everywhere. This would contradict our worst-case hardness assumption, assuming that $\mathrm{Dec}(A)$ gives a time $t(\ell)$ algorithm for f. However, the standard notion of decoding requires reading all $2^{\hat{\ell}}$ values of the received word A and writing all 2^{ℓ} values of the message $\mathrm{Dec}(A)$, and thus $\mathrm{Time}(\mathrm{Dec}(A)) \gg 2^{\ell}$. Everything is easy for nonuniform time 2^{ℓ}, and even in the uniform case we are mostly interested in $t(\ell) \ll 2^{\ell}$. To solve this problem we introduce the notion of local decoding.

Definition 7.32. A *local δ-decoding algorithm* for some $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$ is a probabilistic oracle algorithm Dec with the following property. Let $f : \{0,1\}^{\ell} \to \{0,1\}$ be any message with associated codeword $\hat{f} = \mathrm{Enc}(f)$, and let $g : \{0,1\}^{\hat{\ell}} \to \Sigma$ be such that $\Delta(g, \hat{f}) < \delta$. Then for all $x \in \{0,1\}^{\ell}$ we have $\Pr[\mathrm{Dec}^g(x) = f(x)] \geq 2/3$, where the probability is taken over the coins flips of Dec.

In other words, given oracle access to g, we want to efficiently compute any desired bit of f with high probability. So both the input (namely, g) and the output (namely, f) are treated implicitly; the decoding algorithm does not need to read/write either in its entirety. Pictorially:

This makes it possible to have sublinear-time (or even polylogarithmic-time) decoding. Also, we note that the bound of $2/3$ in the definition can be amplified in the usual way. Having formalized a notion of local decoding, we can now make our earlier intuition precise.

Proposition 7.33. Let Enc be an error-correcting code with local δ-decoding algorithm Dec that runs in time at most t_{Dec}, and let f be worst-case hard for nonuniform time t. Then $\hat{f} = \mathrm{Enc}(f)$ is (t', δ) average-case hard, where $t' = t/t_{\mathrm{Dec}}$.

Proof. We do everything as explained before except with Dec^A in place of $\mathrm{Dec}(A)$, and now the running time is at most $\mathrm{Time}(\mathrm{Dec}) \cdot \mathrm{Time}(A)$, since Dec can make at most $\mathrm{Time}(\mathrm{Dec})$ calls to A. □

We note that the reduction in this proof does not use nonuniformity in an essential way. We used nonuniformity to fix the coin tosses of A, making it deterministic. To obtain a version for uniform hardness, the coin tosses of A can be chosen randomly instead, which with high probability will not increase A's error by more than a constant factor, which we can compensate for by replacing the (t', δ) average-case hard in the conclusion with, say, $(t', \delta/3)$ average-case hard.

In light of the above proposition, our task is now to find an error-correcting code with a local decoding algorithm. Specifically, we would like the follows parameters.

(1) We want $\hat{\ell} = O(\ell)$, or equivalently $\hat{L} = \mathrm{poly}(L)$.

(2) We would like Enc to be computable in time $2^{O(\ell)} = \text{poly}(L)$. This is because we want $f \in \mathbf{E}$ to imply $\hat{f} \in \mathbf{E}$.
(3) We would like $\Sigma = \{0,1\}$ so that $\hat{f}$ is a boolean function, and have $\delta = 1/2 - \varepsilon$ so that $\hat{f}$ has sufficient average-case hardness for the pseudorandom generator construction of Theorem 7.23.
(4) Since $\hat{f}$ will be average-case hard against time $t' = t/t_{\text{Dec}}$, we would want the running time of Dec to be $t_{\text{Dec}} = \text{poly}(\ell, 1/\varepsilon)$ so that we can take $\varepsilon = t^{\Omega(1)}$ and still have $t = t^{\Omega(1)}/\text{poly}(\ell)$.

Of course, achieving $\delta = 1/2 - \varepsilon$ is not possible with our current notion of local *unique* decoding (which is only harder than the standard notion of unique decoding), and thus in the next section we will focus on getting δ to be just a fixed constant. In Section **??**, we will introduce a notion of local *list* decoding, which will enable decoding from distance $\delta = 1/2 - \varepsilon$.

In our constructions, it will be more natural to focus on the task of decoding *codeword* symbols rather than message symbols:

Definition 7.34 (locally correctible codes[4]). A *local δ-correcting algorithms* for a code $\mathcal{C} \subset \Sigma^{\hat{L}}$ is a probabilistic oracle algorithm Dec with the following property. Let $\hat{f} \in \mathcal{C}$ be any codeword, and let $g : \{0,1\}^{\hat{\ell}} \to \Sigma$ be such that $\Delta(g, \hat{f}) < \delta$. Then for all $x \in [\hat{L}]$ we have $\Pr[\text{Dec}^g(x) = \hat{f}(x)] \geq 2/3$, where the probability is taken over the coins flips of Dec.

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

Definition 7.35 (systematic encodings). An encoding algorithm $\text{Enc} : \{0,1\}^L \to \mathcal{C}$ for a code $\mathcal{C} \subseteq \Sigma^{\hat{L}}$ is *systematic* if there is a polynomial-time computable function $I : [L] \to [\hat{L}]$ such that for all $f \in \{0,1\}^L$, $\hat{f} = \text{Enc}(f)$, and all $x \in [L]$, we have $\hat{f}(I(x)) = f(x)$, where we interpret 0 and 1 as elements of Σ in some canonical way.

Lemma 7.36. If $\text{Enc} : \{0,1\}^L \to \mathcal{C}$ is systematic and $\mathcal{C}$ has a local δ-correcting algorithm running in time t, then Enc has a local δ-decoding algorithm (in the standard sense) running in time $t + \text{poly}(\log L)$.

Proof. If Dec_1 is the local corrector for $\mathcal{C}$ and I the mapping in the definition of systematic encoding, then $\text{Dec}_2^g(x) = \text{Dec}_1^g(I(x))$ is a local decoder for Enc. □

7.5.1 Local Decoding Algorithms

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

Proposition 7.37. The Hadamard code $\mathcal{C} \subseteq \{0,1\}^{2^m}$ of message length m has a local $(1/4-\varepsilon)$-correcting algorithm running in time $\mathrm{poly}(m, 1/\varepsilon)$.

Proof. We are given oracle access to $g : \{0,1\}^m \to \{0,1\}$ that is at distance less than $1/4-\varepsilon$ from some (unknown) linear function c, and we want to compute $c(x)$ at an arbitrary point $x \in \{0,1\}^m$. The idea is *random self-reducibility*: we can reduce computing c at an arbitrary point to computing c at uniformly random points, where g is likely to give the correct answer. Specifically, $c(x) = c(x \oplus r) \oplus c(r)$ for every r, and both $x \oplus r$ and r are uniformly distributed if we choose $r \xleftarrow{R} \{0,1\}^m$. The probability that g differs from c at either of these points is less than $2 \cdot (1/4-\varepsilon) = 1/2 - 2\varepsilon$. Thus $g(x \oplus r) \oplus g(r)$ gives the correct answer with probability noticeably larger than $1/2$. We can amplify this success probability by repetition. Specifically, we obtain the following local corrector:

Algorithm 7.38 (Local Corrector for Hadamard Code).
Input: an oracle $g : \{0,1\}^m \to \{0,1\}$, $x \in \{0,1\}^m$, and a parameter $\varepsilon > 0$

(1) Choose $r_1, \ldots, r_t \xleftarrow{R} \{0,1\}^m$, for $t = O(1/\varepsilon^2)$.
(2) Query $g(r_i)$ for each $i = 1, \ldots, t$.
(3) Output $\mathrm{maj}_{1 \le j \le t}\{g(r_j) \oplus g(r_j \oplus x)\}$.

If $\Delta(g, c) < 1/4 - \varepsilon$, then this algorithm will output $c(x)$ with probability at least $2/3$. □

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

Reed–Muller Code. Recall that the q-ary Reed–Muller code of degree d and dimension m consists of all multivariate polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ of total degree at most d. (Construction 6.14.) This code has minimum distance $\delta = 1 - d/q$. Reed–Muller Codes are a common generalization of both Hadamard and Reed–Solomon codes, and thus we can hope that for an appropriate setting of parameters, we will be able to get the best of both kinds of codes. That is, we want to combine the efficient local decoding of the Hadamard code with the good rate of Reed-Solomon codes.

Theorem 7.39. The q-ary Reed-Muller Code of degree d and dimension m has a local $1/12$-correcting algorithm running in time $\mathrm{poly}(m, q)$ provided $d \le q/9$ and $q \ge 36$.

Note the running time of the decoder is roughly the m'th root of the block length $\hat{L} = q^m$. When $m = 1$, our decoder can query the entire string and we simply obtain a global decoding algorithm for Reed-Solomon Codes (which we already know how to achieve from Theorem 6.17). But for large enough m, the decoder can only access a small fraction of the received word. In fact, one can improve the running time to $\mathrm{poly}(m, d, \log q)$, but the weaker result is sufficient for our purposes.

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

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

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

Algorithm 7.40 (Local Corrector for Reed–Muller Code I).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, and a degree parameter d

(1) Choose $y \overset{R}{\leftarrow} \mathbb{F}^m$. Let $\ell = \ell_{x,y} : \mathbb{F} \to \mathbb{F}^m$ be the line through x in direction y.
(2) Query g to obtain $\beta_0 = g|_\ell(\alpha_0) = g(\ell(a_0)), ..., \beta_d = g|_\ell(\alpha_d) = g(\ell(a_d))$ where $\alpha_0, ..., \alpha_d \in \mathbb{F} \setminus \{0\}$ are any fixed points
(3) Interpolate to find a unique univariate polynomial q of degree at most d s.t. $\forall i, q(\alpha_i) = \beta_i$
(4) Output $q(0)$

Claim 7.41. If g has distance less than $\delta = 1/3(d+1)$ from some polynomial p of degree at most d, then Algorithm 7.40 will output $p(x)$ with probability greater than $2/3$.

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

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

By a union bound,

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

Thus, with probability greater than $2/3$, we have $\forall i, q(\alpha_i) = p|_\ell(\alpha_i)$ and hence $q(0) = p(x)$. The running time of the algorithm is $\text{poly}(m, q)$. □

We now show how to improve the decoder to handle a larger fraction of errors, up to distance $\delta = 1/12$. We alter steps 2 and 3 in the above algorithm. In step 2, instead of querying only $d+1$ points, we query over *all* points in ℓ. In step 3, instead of interpolation, we use a *global* decoding algorithm for Reed–Solomon codes to decode the univariate polynomial $p|_\ell$. Formally, the algorithm proceeds as follows.

Algorithm 7.42 (Local Corrector for Reed–Muller Codes II).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, and a degree parameter d, where $q = |\mathbb{F}| \geq 36$ and $d \leq q/9$.

(1) Choose $y \overset{R}{\leftarrow} \mathbb{F}^m$. Let $\ell = \ell_{x,y} : \mathbb{F} \to \mathbb{F}^m$ be the line through x in direction y.
(2) Query g at all points on ℓ to obtain $g|\ell : \mathbb{F} \to \mathbb{F}$.
(3) Run the 1/3-decoder for the q-ary Reed–Solomon codes of degree d on $g|_\ell$ to obtain the (unique) polynomial q at distance less than 1/3 from $g|_\ell$ (if one exists).[5]
(4) Output $q(0)$.

Claim 7.43. If g has distance less than $\delta = 1/12$ from some polynomial p of degree at most d, and the parameters satisfy $q = |\mathbb{F}| \geq 36$, $d \leq q/9$, then Algorithm 7.42 will output $p(x)$ with probability greater than 2/3.

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

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

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

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

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

7.5.2 Low-Degree Extensions

Recall that to obtain locally decodable codes from locally correctible codes (as constructed above), we need to exhibit systematic encoding (Definition 7.35.) Thus, given $f\colon \{0,1\}^\ell \to \{0,1\}$, we want to encode it as a Reed–Muller codeword $\hat{f}\colon \{0,1\}^{\hat{\ell}} \to \Sigma$ s.t.:

- The encoding time is $2^{O(\ell)}$
- $\hat{\ell} = O(\ell)$
- The code is systematic in the sense of Definition 7.35. Informally, this means that f should be a "restriction" of $\hat{f}$.

Note that the usual encoding for Reed–Muller codes, where the message gives the coefficients of the polynomial, is not systematic. Instead the message should correspond to evaluations of the

[5] A 1/3-decoder for Reed–Solomon codes follows from the $(1 - 2\sqrt{d/q})$ list-decoding algorithm of Theorem 6.17. Since $1/3 \leq 1 - 2\sqrt{d/q}$, the list-decoder will produce a list containing all univariate polynomials at distance less than 1/3, and since 1/3 is smaller than half the minimum distance $(1 - d/q)$, there will be only one good decoding.

polynomial at certain points. Once we settle on the set of evaluation points, the task becomes one of interpolating the values at these points (given by the message) to a low-degree polynomial defined everywhere. does not suffice, since this encoding is not systematic.

The simplest approach is to use the boolean hypercube as the set of evaluation points.

Lemma 7.44 (multilinear extension). For every $f\colon \{0,1\}^\ell \to \{0,1\}$ and every finite field $\mathbb{F}$, there exists a (unique) polynomial $\hat{f}\colon \mathbb{F}^\ell \to \mathbb{F}$ s.t. $\hat{f}_{|\{0,1\}^\ell} \equiv f$ of degree at most 1 in each variable. (and hence total degree at most ℓ).

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

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

for

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

Note that for $x \in \{0,1\}^\ell$, $\delta_\alpha(x) = 1$ only when $\alpha = x$, therefore $\hat{f}_{|\{0,1\}^\ell} \equiv f$. We omit the proof of uniqueness. The bound on the individual degrees is by inspection. □

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

- Since the total degree of the multilinear extension can be as large as ℓ, we need $q \geq 9\ell$ for the local corrector of Theorem 7.39 to apply.
- The encoding time is $2^{O(\ell)}$, as computing a single point of $\hat{f}$ requires summing over $2^{O(\ell)}$ elements.
- The code is systematic, since $\hat{f}$ is an extension of f.
- However, the input length is $\hat{\ell} = \ell \log q = \Theta(\ell \log \ell)$, which is slightly larger than our target of $\hat{\ell} = O(\ell)$.

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

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

Analogously to before, we have the following.

Lemma 7.45 (low-degree extension). For every finite field $\mathbb{F}$, $\mathbb{H} \subseteq \mathbb{F}$, $m \in \mathbb{N}$, and function $f : \mathbb{H}^m \to \mathbb{F}$, there exists a (unique) $\hat{f}\colon \mathbb{F}^m \to \mathbb{F}$ of degree at most $|\mathbb{H}| - 1$ in each variable (and hence total degree at most $m \cdot (|\mathbb{H}| - 1)$) s.t. $\hat{f}|_{H^m} \equiv f$.

Using $|\mathbb{H}| = \lceil\sqrt{q}\rceil$, the total degree of $\hat{f}$ is at most $d = \ell\sqrt{q}$. So we can apply the local corrector of Theorem 7.39, as long as $q \geq 81\ell^2$ (so that $d \leq q/9$). Inspecting the properties of $\hat{f}$ as an encoding of f, we have:

- The input length is $\hat{\ell} = m \cdot \log q = \lceil \ell / \log |\mathbb{H}| \rceil \cdot \log q = O(\ell)$, as desired.
- The code is systematic as long as our mapping from $\{0,1\}^\ell$ to $\mathbb{H}^m$ is efficient.

7.5.3 Putting It Together

Combining Theorem 7.39 with Lemmas 7.45, and 7.36, we obtain the following locally decodable code:

Proposition 7.46. For every $L \in \mathbb{N}$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$, with blocklength $\hat{L} = \mathrm{poly}(L)$ and alphabet size $|\Sigma| = \mathrm{poly}(\log L)$, that has a local $(1/12)$-decoder running in time $\mathrm{poly}(\log L)$.

Using Proposition 7.33, we obtain the following conversion from worst-case hardness to average-case hardness:

Proposition 7.47. If there exists $f \colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against (non-uniform) time $t(\ell)$, then there exists $\hat{f} \colon \{0,1\}^{O(\ell)} \to \{0,1\}^{O(\log \ell)}$ in $\mathbf{E}$ that is $(t'(\ell), 1/12)$ average-case hard for $t'(\ell) = t(\ell)/\mathit{poly}(\ell)$.

This differs from our original goal in two ways: $\hat{f}$ is not Boolean, and we only get hardness $1/12$ (instead of $1/2 - \varepsilon$). The former concern can be remedied by concatenating the code with a Hadamard code, similarly to Problem 6.2. Note that the Hadamard code is for message space $[q]$, so it can be $1/4$-decoded by brute-force in time $\mathrm{poly}(q)$ (which is the amount of time already taken by our decoder).[6] Using this, we obtain:

Theorem 7.48. For every $L \in \mathbb{N}$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \{0,1\}^{\hat{L}}$ with blocklength $\hat{L} = \mathrm{poly}(L)$ that has a local $(1/48)$-decoder running in time $\mathrm{poly}(\log L)$.

Theorem 7.49. If there exists $f \colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against (non-uniform) time $t(\ell)$, then there exists $\hat{f} \colon \{0,1\}^{O(\ell)} \to \{0,1\}$ in $\mathbf{E}$ that is $1/48$ average-case hard against (non-uniform) time $t(\ell)/\mathit{poly}(\ell)$.

An improved decoding distance can be obtained using Problem **??**.

We note that the local decoder of Theorem 7.48 not only runs in time $\mathrm{poly}(\log L)$, but also makes $\mathrm{poly}(\log L)$ queries. For some applications (such as Private Information Retrieval, see Problem **??**), it is important to have the number q of queries be as small as possible, ideally a constant. Using Reed–Muller codes of constant degree, it is possible to obtain constant-query locally decodable codes, but the blocklength will be $\hat{L} = \exp(L^{1/(q-1)})$. In a recent breakthrough, it was shown how to obtain constant-query locally decodable codes with blocklength $\hat{L} = \exp(L^{o(1)})$. Obtaining polynomial blocklength remains open.

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

Open Problem 7.50. Are there binary codes that are locally decodable with a constant number of queries (from constant distance $\delta > 0$) and blocklength polynomial in the message length?

7.5.4 Other Connections

As shown in Problem **??**, locally decodable codes are closely related to protocols for *private information retrieval*. Another connection, and actually the setting in which these local decoding algorithms were first discovered, is to *program self-correctors*. Suppose you have a program for computing a function, such as the DETERMINANT, which happens to be a codeword in a locally decodable code (e.g. the determinant is a low-degree multivariate polynomial). Then, even if this program has some bugs and gives the wrong answer on some small fraction of inputs, you can use the local decoding algorithm to obtain the correct answer on *all* inputs with high probability.

7.6 Local List Decoding

7.6.1 Hardness Amplification

In the previous section, we saw how to use locally decodable codes to convert a worst-case hard function into one with constant average-case hardness (Theorem 7.49). Now our goal is to boost this constant hardness to $1/2 - \varepsilon$.

There are some generic techniques for doing this, known as Direct Product Theorems or the XOR Lemma (for Boolean functions). In those methods we use independent copies of the function at hand. For example, in the XOR lemma, we let f' consist of k independent copies of $\hat{f}$,

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

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

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

Thus, we will take a different approach, namely to generalize our notion and algorithms for locally decodable codes to locally *list*-decodable codes. Nevertheless, the study of hardness amplification is still of great interest, because it (or variants) can be employed in settings where doing a global encoding of the function is infeasible (e.g. for amplifying the average-case hardness of functions in complexity classes lower than **E**, such as **NP**, and for amplifying the security of cryptographic primitives). We remark that results on hardness amplification can be interpreted in a coding-theoretic language as well, as converting locally decodable codes with a small decoding distance into locally list-decodable codes with a large decoding distance.

7.6.2 Definition

We would like to formulate a notion of local *list*-decoding to enable us to have binary codes that are locally decodable from distances close to $1/2$. This is slightly trickier to define, since for any

function g, there may be several codewords $\hat{f}_1, \hat{f}_2, ..., \hat{f}_s$ that are close to g. So what should our decoding algorithm do? One option would be for the decoding algorithm, on input x, to output a set of values $\mathrm{Dec}^g(x) \subset \Sigma$ that is guaranteed to contain $\hat{f}_1(x), \hat{f_2}(x), ...\hat{f}_s(x)$ with high probability. However, this is not very useful, e.g the list could always be $\mathrm{Dec}^g(x) = \Sigma$. However, rather than outputting each of these values, we want to be able to specify to our decoder *which* $\hat{f}_i(x)$ to output. We do this with a two-phase decoding algorithm. The probabilistic algorithms that accomplish these phases will be referred to as Dec_1 and Dec_2:

(1) Dec_1, using g as an oracle, returns a list of advice strings $a_1, a_2, ..., a_2$, which can be thought of as "labels" for each of the codewords close to g.
(2) Dec_2 (again, using oracle access to g), takes input x and a_i, and outputs $\hat{f}_i(x)$.

The picture for Dec_2 is much like our old decoder, but it takes an extra input a_i corresponding to one of the outputs of Dec_1:

More formally:

Definition 7.51. A *local δ list-decoding algorithm* for a code Enc is a pair of probabilistic oracle algorithms $(\mathrm{Dec}_1, \mathrm{Dec}_2)$ such that for all received words g and all codewords $\hat{f} = \mathrm{Enc}(f)$ with $\Delta(\hat{f}, g) < \delta$, the following holds. With probability at least $1/2$ over $(a_1, ..., a_s) \leftarrow \mathrm{Dec}_1^g$, there exists an $i \in [s]$ such that

$$\forall x, \Pr[\mathrm{Dec}_2^g(x, a_i) = f(x)] \geq 2/3.$$

To help clarify this definition, we make the following remarks. First, we don't require that for all $j, \mathrm{Dec}_2^g(x, a_j)$ are codewords, or even that they're close to s; in other words some of the a_j's may be junk. Second, we don't explicitly require a bound on the list size s, but certainly it cannot be larger than the running time of Dec_1.

As we did for locally (unique-)decodable codes, we can define a *local δ list-correcting algorithm*, where Dec^2 should recover arbitrary symbols of the codeword $\hat{f}$ rather than the message f. Analogously to Lemma 7.36, this implies the above definition if the code is systematic.

Proposition 7.33 shows how locally decodable codes converts functions that are hard in the worst case to ones that are hard on average. The same is true for local list-decoding:

Proposition 7.52. Let Enc be an error-correcting code with local δ-list-decoding algorithm $(\mathrm{Dec}_1, \mathrm{Dec}_2)$ where Dec_2 runs in time at most t_{Dec}, and let f be worst-case hard for non-uniform time t. Then $\hat{f} = \mathrm{Enc}(f)$ is (t', δ) average-case hard, where $t' = t/t_{\mathrm{Dec}}$.

Proof. Suppose for contradiction that $\hat{f}$ is not (t', δ)-hard. Then some algorithm A running in time t' computes $\hat{f}$ with error probability smaller than δ. But if Enc has a local δ list-decoding algorithm, then (with A playing the role of g) that means there exists a_i (one of the possible outputs of Dec_1^A), such that $\mathrm{Dec}_2^A(\cdot, a_i)$ computes $f(\cdot)$ everywhere. Hardwiring a_i as advice, $\mathrm{Dec}_2^A(\cdot, a_i)$ is a nonuniform algorithm running in time at most $\mathrm{time}(A) \cdot \mathrm{time}(\mathrm{Dec}_2) \leq t$. □

Note that, in contrast to Proposition 7.33, here we are using nonuniformity more crucially, in order to select the right function from the list of possible decodings.

7.6.3 Local List-Decoding Reed–Muller Codes

Theorem 7.53. There is a universal constant c such that the q-ary Reed–Muller code of degree d and dimension m over can be locally $(1-\varepsilon)$-list-corrected in time $\text{poly}(q,m)$ for $\varepsilon = c\sqrt{d/q}$.

Note that the distance at which list-decoding can be done approaches 1 as $q/d \to \infty$. It matches the bound for list-decoding Reed–Solomon codes (Theorem **??**) up to the constant c. However, as m increases, the running time of the decoder ($\text{poly}(q,m)$) becomes much smaller than the block length ($q^m \cdot \log q$), at the price of a reduced rate ($\binom{m+d}{m}/q^m$).

Proof. Suppose we are given an oracle $g : \mathbb{F}^m \to \mathbb{F}$ that is $(1-\varepsilon)$ close to some unkown polynomial $p : \mathbb{F}^m \to \mathbb{F}$, and that we are given an $x \in \mathbb{F}^m$. Our goal is is describe two algorithms, Dec_1 and Dec_2, where Dec_2 is able to compute $p(x)$ using a piece of Dec_1's output (i.e. advice).

The advice that we will give to Dec_2 is the value of p on a single point. Dec_1 can easily generate a (reasonably small) list that contains one such point by choosing a random $y \in \mathbb{F}^m$, and outputting all pairs (y,z), for $z \in \mathbb{F}$. More formally:

Algorithm 7.54 (Reed–Muller Local List-Decoder Dec_1).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, and a degree parameter d

(1) Choose $y \xleftarrow{R} \mathbb{F}^m$
(2) Output $\{(y,z) \ : \ z \in \mathbb{F}\}$

Now, the task of Dec_2 is to calculate $p(x)$, given the value of p on some point y. Dec_2 does this by looking at g restricted to the line through x and y, and using the RS list-decoding algorithm to find the univariate polynomials $q_1, q_2, ..., q_t$ that are close to g. If exactly one of these polynomials q_i agrees with p on the test point y, then we can be reasonably confident that $q_i(x) = p(x)$. Specifically:

Algorithm 7.55 (Reed–Muller Local List-Corrector Dec_2).
Input: An oracle $g\colon \mathbb{F}^m \to \mathbb{F}$, an input $x \in \mathbb{F}^m$, advice $(y,z) \in \mathbb{F}^m \times \mathbb{F}$, and a degree parameter d

(1) Let $\ell = \ell_{x,y-x} : \mathbb{F} \to \mathbb{F}^m$ be the line through x and y (so that $\ell(0) = x$ and $\ell(1) = y$).
(2) Run the $(1-\varepsilon/2)$-list-decoder for Reed–Solomon Codes (Theorem 6.17) on $g|_\ell$ to get all univariate polys $q_1 ... q_t$ that agree with $g|_\ell$ in greater than an $\varepsilon/2$ fraction of points.
(3) If there exists a unique i such that $q_i(1) = z$, output $q_i(0)$. Otherwise, fail.

Now that we have fully specified the algorithms, it remains to analyze them and show that they work with the desired probabilities. Observe that it suffices to compute p on at $> 11/12$ of the points x, because then we can apply the unique local decoding algorithm from last time. Therefore, to finish the proof of the theorem we must prove the following lemma

---

Claim 7.56. Suppose that $g : \mathbb{F}^m \to \mathbb{F}$ has agreement greater than ε with a polynomial p (i.e. g has distance less than $1 - \varepsilon$ from p) of degree at most d. For at least $1/2$ of the points $y \in \mathbb{F}^m$ the following holds for greater than an $11/12$ fraction of lines ℓ going through y:

(1) $\mathrm{agr}(g|_\ell, p|_\ell) > \varepsilon/2$.
(2) There does not exist any univariate polynomial q of degree at most d other than $p|_\ell$ such that $\mathrm{agr}(g|_\ell, q) > \varepsilon/2$ and $q(y) = p(y)$.

Proof of claim: It suffices to show that Items 1 and 2 hold with probability 0.99 over random y, ℓ; then we can apply Markov's inequality to finish the job.

Item 1 holds by pairwise independence. If the line ℓ is chosen randomly, then the q points on ℓ are pairwise independent samples of $\mathbb{F}^m$. Note that the expected agreement between $g|_\ell$ and $p|_\ell$ is simply the agreement between $g|_\ell$ and $p|_\ell$, which is greater than ε by hypothesis. So by the Pairwise-Independent Tail Inequality (Prop. 3.27), $\Pr[\mathrm{agr}(g|_\ell, p|_\ell) \le \varepsilon/2] < (1/q \cdot (\varepsilon/2)^2)$, which can be made < 0.01 for a large enough choice of the constant c in $\varepsilon = c\sqrt{d/q}$.

To prove Item 2, we imagine first choosing the line ℓ uniformly at random from all lines in $\mathbb{F}^m$, and then choosing y uniformly at random from the points on ℓ (reparametrizing ℓ so that $\ell(1) = y$). Once we choose ℓ, we can let $q_1, ...q_t$ be all polynomials of degree at most d, other than $p|_\ell$, that have agreement greater than $\varepsilon/2$ with $g|_\ell$. (Note that this list is independent of the parametrization of ℓ, i.e. if $\ell'(t) = \ell(at + b)$ for $a \neq 0$ then $p|_{\ell'}$ and $q_i'(t) = q_i(at + b)$ have agreement equal to $\mathrm{agr}(p|_\ell, q_i)$.) By the list-decodability of Reed–Solomon Codes (Proposition 6.13), we have $t = O(\sqrt{q/d})$.

Now, since two distinct polynomials can agree in at most d points, when we choose a random point $y \xleftarrow{R} \ell$, the probability that q_i and p agree at y is at most d/q. After reparameterization of ℓ so that $\ell(1) = y$, this gives

$$\Pr_y[\exists i \ : \ q_i(1) = p(1)] \le t \cdot \frac{d}{q} = O\left(\sqrt{\frac{d}{q}}\right).$$

This can also be made < 0.01 for large enough choice of the constant c (since we may assume $q/d > c^2$, else $\varepsilon = 1$ and the result is trivial). □

□

7.6.4 Putting it Together

To obtain a locally list-*decodable* (rather than list-*correctible*) code, we again use the low-degree extension (Lemma 7.45) to obtain a systematic encoding. As before, to encode messages of length

$\ell = \log L$, we apply Lemma 7.45 with $|\mathbb{H}| = \lceil \sqrt{q} \rceil$ and $m = \lceil \ell / \log |\mathbb{H}|$, for total degree $d \leq \sqrt{q} \cdot \ell$. To decode from a $1 - \varepsilon$ fraction of errors using Theorem 7.53, we need $c\sqrt{d/q} \leq \varepsilon$, which follows if $q \geq c^2 \ell^2 / \varepsilon^4$. This yields the following locally list-decodable codes:

Theorem 7.57. For every $L \in \mathbb{N}$ and $\varepsilon > 0$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \Sigma^{\hat{L}}$, with blocklength $\hat{L} = \mathrm{poly}(L, 1/\varepsilon)$ and alphabet size $|\Sigma| = \mathrm{poly}(\log L, 1/\varepsilon)$, that has a local $(1-\varepsilon)$-list-decoder running in time $\mathrm{poly}(\log L, 1/\varepsilon)$.

Concatenating the code with a Hadamard code, similarly to Problem 6.2, we obtain:

Theorem 7.58. For every $L \in \mathbb{N}$ and $\varepsilon > 0$, there is an explicit code $\mathrm{Enc} : \{0,1\}^L \to \{0,1\}^{\hat{L}}$ with blocklength $\hat{L} = \mathrm{poly}(L, 1/\varepsilon)$ that has a local $(1/2 - \varepsilon)$-list-decoder running in time $\mathrm{poly}(\log L, 1/\varepsilon)$.

Using Proposition 7.52, we get the following hardness amplification result:

Theorem 7.59. If there exists $f \colon \{0,1\}^\ell \to \{0,1\}$ in $\mathbf{E}$ that is worst-case hard against non-uniform time $t(\ell)$, then there exists $\hat{f} \colon \{0,1\}^{O(\ell)} \to \{0,1\}$ in $\mathbf{E}$ that is $(1/2 - 1/t'(\ell))$ average-case hard against (non-uniform) time $t'(\ell)$ for $t'(\ell) = t(\ell)^{\Omega(1)}/\mathrm{poly}(\ell)$.
