UNIVERSITY OF CALIFORNIA SAN DIEGO

Connections Between Randomness Extractors, Pseudorandom Generators, and Optimal Derandomization

A Thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Computer Science

by

Dustin Lin

Committee in charge:

Professor Russell Impagliazzo, Chair
Professor Daniele Micciancio
Professor Ramamohan Paturi

2023

The Thesis of Dustin Lin is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would first like to acknowledge Russell Impagliazzo for all of his help and guidance. From first agreeing to be my mentor for an independent study project while I was still an undergrad, to being the chair of my thesis committee during my masters program, Russell has always been generous with his time. His enthusiasm for both complexity theory as well as mentorship is unwavering, and he has been a huge inspiration for me to continue to learn more about the field and pursue academia.

In addition I would like to thank Ramamohan Patrui and Daniele Micciancio for being part of my thesis committee, Nadia Heninger and Gerald Soosai Raj for giving me the opportunity to work with them during my time at UCSD, and all the wonderful teachers I had who have helped develop my interest in theoretical computer science.

Lastly I would like to give a big shout-out to all the other students in the UCSD theory group, and my roommates and friends for making my graduate studies more enjoyable. And of course, my parents, David and Jenny, and sister Iris, for their longstanding and continuing support in allowing me to pursue my academic interests.

ABSTRACT OF THE THESIS

Connections Between Randomness Extractors, Pseudorandom Generators, and Optimal Derandomization

by

Dustin Lin

Master of Science in Computer Science

University of California San Diego, 2023

Professor Russell Impagliazzo, Chair

This thesis will be a survey aimed to be an overview of some connections between randomness extractors and pseudorandom generators in complexity theory. Particularly there will be a focus on how such connections are leveraged in achieving faster derandomization and "beating the hybrid argument". We will cover both some classic results and examples, as well as some more recent ones.

Chapter 1

Statistical and Computational Randomness

The use of randomness in computation has been integral in algorithm design, and has led to the creation of simple yet effective algorithms that are common in everyday use (eg: testing if a number is prime, sorting, finding the minimum cut of a graph). In computational complexity theory we look at different resources computers can use, and their power in solving problems. Formally defined computational complexity classes around Turing machines and other computational models with access to random bits have existed for decades, and people have attempted to compare and contrast these classes to others. However, it is not clear the *power* of randomness, and where certain randomized complexity classes stand against others. Is it the case that allowing computers access to random coin tosses strictly increases the number of computational problems it can solve? If so, by how much? Looking from a finer lens, do randomized polynomial time algorithms always have a deterministic/derandomized counterpart?

While specific derandomized algorithms exist (eg: the deterministic AKS algorithm testing if a number is prime [AKS04]), does this statement hold for *every* randomized algorithm? The area of pseudorandomness attempts to look at ways to efficiently generate bits that "look random" using little to no true randomness. Beyond the perhaps philosophical question of if true randomness exists in our universe, generating pseudorandom bits would lead to derandomized algorithms by simply substituting their random bits with pseudorandom ones. The intuition is

that if the computational model has a difficult time distinguishing between pseudorandom and truly random bits, then the accuracy of the randomized algorithm should not be affected by much. This is known as "derandomizing" an algorithm.

In this work we will first go over some notation and facts that may be unknown to some readers, then in the first half of Chapter 1 review definitions related to complexity classes and pseudorandomness. Basic knowledge (such as that acquired from an undergraduate theory of computation or algorithms course) is assumed. In the later half we will cover some classic results about pseudorandom generators and randomness extractors, and show some common proof techniques used. In chapter 2 we will start looking at the question of "optimal" derandomization and how randomness extractors and pseudorandom generators are actually closely related. Finally in chapters 3 and 4 we will go over some recent results relating to just how fast we can derandomize algorithms.

1.0.1 Notation

We represent the set of numbers $\{1,\ldots,n\}$ as $[n]$, and often use capital numbers to represent their lowercase numbers raised to the power of 2 (eg: $N = 2^n$). All logarithms are base 2 unless specified. We consider (unless specified) U_n to be the uniform distribution over n bits. We call the support of a random variable X , $supp(X)$ to be the set of all values X can take with non-zero probability. For a circuit C or boolean function f, we define $supp(C)$ and $supp(f)$ to be the set of all inputs for which the circuit/function accepts respectively (where accept means circuit/function returns 1 and reject 0). We consider $\mathsf{poly}(n)$ to be an arbitrary polynomial with variable(s) n.

We denote the complexity class $DTIME(t(n))$ to represent all languages that are deterministically decidable by a Turing machine in time $t(n)$. For example $\mathsf{P} = \cup_c DTIME(n^c)$ for all constants $c \geq 1$. Below we will state some common inequalities and definitions as a brief refresher:

Theorem 1 *Chernoff Bound: Let $X_1,\ldots,X_t$ be independent random variables taking values in*

the interval $[0,1]$*, and* $X = (\sum_i X_i)/t$ *and* $\mu = \mathbb{E}[X]$*. Then the probability that X deviates from its mean exponentially decays in the number of variables:*

$$\Pr[|X - \mu| \geq \varepsilon] \leq 2^{-t\varepsilon^2/4}$$

Theorem 2 *Union Bound: Let* $E_1, E_2, \ldots, E_n$ *be a collection of events (may or may not be independent), the probability that at least one such event occurs is bounded by the sums of their probabilities:*

$$\Pr[\cup_{i=1}^{n} E_i] \leq \sum_{i=1}^{n} \Pr[E_i]$$

Definition 3 *Let X be the random variable drawn from some distribution D. We say the Shannon entropy of distribution D is:*

$$\sum_{x \in supp(X)} \Pr[X = x] \log \frac{1}{\Pr[X = x]}$$

1.1 Randomness in computation: complexity classes

Randomness can be thought of as an additional resources available to computers. As a quick refresher, we include the definition of BPP (bounded-error probabilistic polynomial time) and probabilistic Turing machines. A probabilistic Turing machine is a Turing machine that has access to a black-box providing it with unbiased and independent random bits. The machine can use this box to "toss a coin" and get it's result in one time step.

One can also think about probabilistic Turing machines as having 2 transition functions which, at each step of computation, run with probability 1/2 each. If the reader is familiar with nondeterministic Turing machines, we are essentially describing the transition function in the same way. One key difference is that while in nondeterminism we only require that *one* branch of computation accepts, for probabilistic Turing machines we want some "majority" to accept.

Additionally, another often useful view of randomized computation is a regular Turing

machine with two input tapes. Besides the tape containing the input string, the Turing machine is also given an infinite *one-way* read-only tape filled with random 1's and 0's generated uniformly at random. At any stage in the Turing machine's computation, it can choose to move the tape-head above the random tape and read in random bits. This view of probabilistic Turing machines is useful because it allows us to "decouple the randomness" from the machine [AB09, Vad12].

This complexity class BPP aims to capture "efficent" randomized computation.

Definition 4 BPP *is a class of languages L for which there exists a probabilistic polynomial-time algorithm A such that:*

- $x \in L \rightarrow \Pr[A(x)\ accepts] \geq 2/3$
- $x \notin L \rightarrow \Pr[A(x)\ accepts] \leq 1/3$

The fact that the probability of false positive and false negatives is bounded significantly (a constant) away from 1/2 allows us to perform *error-reduction* by running the randomized algorithm multiple times on the same input, and simply taking the majority answer. One can show that the error exponentially decreases towards 0 with a polynomial amount of repetition via a Chernoff bound [AB09, Vad12].

Another model of computation tied closely with pseudorandomness is Boolean circuits. For an n-length input, a Boolean circuit is a directed acyclic graph with n sources (which corresponds to the n input bits) and one sink (which we can think of as the output). Between the source and sink we have vertices representing "gates" labeled the logical operators AND, OR, NOT. The AND and OR gates each have in-degree 2 (often called fan-in 2) and out-degree 1. The NOT gates have both in and out degree 1. We measure the *size* of each circuit as the number of vertices. Note that circuits are often referred to as a non-uniform model of computation since each circuit is defined against a specific input length n. Circuit families refer to a collection of circuits, one for each input length. The complexity class of circuits that we will most often refer to is the class $\mathsf{P/poly}()$. $\mathsf{P/poly}()$ represents the class of languages which can be decided by

polynomial sized circuit families. The size of each individual circuit should be polynomial with respect to it's input length.

Definition 5 $\mathsf{P/poly}$ *is a class of languages L that are decidable by polynomial-sized circuit families.*

It turns out that circuit families are essentially equivalent to "Turing machines with advice". In the nonuniform Turing machine model we allow, for each input length, an advice bitstring polynomial in length with respect to the input. It turns out every nonuniform Turing machine running in time $t(n)$ can be simulated by a sequence of boolean circuits of size $\tilde{O}(t(n))$ (where $\tilde{O}$ is simply big-O that ignores logarithmic factors). Likewise every sequence of boolean circuits of size $s(n)$ can be simulated by a nonuniform Turing machine running in time $\tilde{O}(s(n))$ [AB09, Vad12]. In this paper we will often refer to "nonuniform Turing machines", "circuits", and "Turing machines with advice" interchangeably.

This alternative concept of "hard wiring" advice will be useful in particular. A good example is Aldeman's theorem that $\mathsf{BPP} \subset \mathsf{P/poly}$. He reasons that with for a language in BPP you can, through error reduction, obtain an algorithm with error rate exponentially small. This then implies that there exists a random string that, on every input, outputs the correct answer. By encoding this random string as advice, we get a circuit/TM with advice deciding the same language [AB09, Adl78].

1.2 Statistical and Computational Randomness

The type of randomness that most people initially think of is the statistical definition. How close is a distribution to the uniform distribution? This measure is formally defined as the *statistical distance*.

Definition 6 *For random variables X and Y taking values in some domain D, their* statistical distance $\Delta(X,Y) = max_{T \subset D} |\Pr[X \in T] - \Pr[Y \in T]|$. *Additionally, X and Y are "ε-close" if* $\Delta(X,Y) \leq \varepsilon$.

The motivation for randomness extractors is to run randomized algorithms without access to "perfect" randomness. What if there are no true random sources, but only biased weak ones? Or even if there are true random sources, what if they are hard to sample from? Randomness extractors aim to extract the randomness out of weak sources, and produce output that is close to uniform. We define extracting randomness from distributions/random variables where we know a lower bound on their min-entropy [Vad12]. We say a random variable sampling from a distribution is a k-source if it has min entropy *at least* k.

Definition 7 *A random variable X is a k-source if $H_\infty(X) \geq k$, i.e. if $\Pr[X = x] \leq 2^{-k}$*

An example of k-sources (which it turns out to generalize to any k-source) are *flat* k-sources: a uniform distribution on a subset $S \subset \{0,1\}^n$ with $|S| = 2^k$ [Vad12]. Note for the case where $n = k$ we have U_n: the uniform distribution over n bits. We define a seeded randomness extractor as the following:

Definition 8 *A randomness extractor: $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ε)-extractor if for every k-source X on $\{0,1\}^n$, $Ext(X,U_d)$ is ε-close to U_m*

A natural question to ask is the motivation for a randomness extractor to require a truly random seed of length d. It turns out having a k-source of length n be the only input to the extractor (and thus having it be deterministic) would imply that there exists an $(n-1)$-source X such that the first bit of every output is the same (by simply looking at a flat $(n-1)$-source of 2^{n-1} elements). This implies a distribution far from uniform, since the first bit is constant! However, even with a random seed essentially being required for our randomness extractor, as long as it is sufficiently short it is possible for us to enumerate over all possible 2^d seeds and run the randomized algorithm on each one. Then by simply taking a majority vote of the outputs we can derandomize the algorithm [AB09].

We now show that, ignoring computational efficiency, extractors with short seeds exist. Namely with seed *logarithmic* in the input length. The below theorem is proved via the

probabilistic method: by reasoning that some mathematical object has a non-zero probability of occurring. In this case, we will show that picking a random function to be our extractor turns out to be a good strategy.

Theorem 9 *(from [AB09]) For every $k,n \in \mathbb{N}$ and $\varepsilon > 0$, there exists a (k,ε)-extractor Ext : $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^k$ with $d = \log n + 2\log 1/\varepsilon + O(1)$.*

Proof: We will focus on looking at flat sources without loss of generality. For an (n,k)-flat source X, let our extractor be a random function $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^k$. For a function $f : \{0,1\}^k \to \{0,1\}$, the expectation of evaluating f with inputs from our random function extractor $\mathbb{E}[f(Ext(X,u_d))]$ involves evaluating f on all $2^k \cdot 2^d$ inputs.

Since these evaluations are independent, by a Chernoff bound the probability that this expectation deviates from the average $\mathbb{E}[f(U_k)]$ by more than ε is exponentially small (ie: $< 2^{-2^{k+d}\varepsilon^2/4}$). By our choice of d we can simplify the bound to $< 2^{2n(2^k)}$. And finally we know the number of flat distribution is at most $(2^n)^{2^k}$, and the number of functions from k bits to 1 bit is 2^{2^k}. By a union bound, this probability that a random function extractor deviates from the true expectation is $(2^n)^{2^k} \cdot 2^{2^k}/2^{2n(2^k)} < 1$, implying that there exists a randomness extractor Ext working for every $(n.k)$-flat source and function $f : \{0,1\}^k \to \{0,1\}$. $\square$

The probabilistic method is very useful for showing the existence of objects with desirable properties, but it is non-constructive in the sense that the proof gives no insight on what such an object would look like and how to go about constructing it. However, very good explicit extractor constructions have been presented with seed lengths close to optimal [AB09, Vad12].

Turning our attention over to *computational* indistinguishability, the difference from the statistical definition is instead of trying to fool every statistical test, only look at fooling specific models of computation.

Definition 10 *Computational indistinguishability (from [Vad12])*

Random variables X and Y taking values in $\{0,1\}^m$ are (t,ε) indistinguishable if for

every nonuniform *algorithm T running in time at most t:*

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

The left hand side of the inequality is often referred to as the *advantage* of a "test" T. We can now define formally a pseudorandom generator.

Definition 11 *A deterministic function $G:\{0,1\}^d \to \{0,1\}^m$ is a (t,ε) pseudorandom generator (PRG) if:*

- $d < m$
- *$G(U_d)$ and U_m are (t,ε) indistinguishable.*

The goal with these definitions is to guarantee that the pseudorandom outputs are as good as truly random bits against all "advarsaries" running in time at most t. This provides us with another way of derandomizing algorithm (in comparison to using randomness extractors). Simply loop through all 2^d seeds of the PRG and take a majority answer of the randomized algorithm with its randomness substituted by the PRG. Interesting to note that while we define the indistinguishability with respect to non-uniform algorithms, for the purposes of derandomization we want the generator to be itself computable uniformly and deterministically. The following theorem highlights the general strategy of using PRGs to derandomize BPP [Vad12].

Theorem 12 *Suppose that for every polynomial m there exists an $(m,1/8)$-PRG $G:\{0,1\}^{d(m)} \to \{0,1\}^m$ (where the seed d is some function dependent on the output length m) computable in time $t(m)$. then* $\mathsf{BPP} \subset \cup_c DTIME(2^{d(n^c)} \cdot (n^c + t(n^c))$.

Proof: We let A be a BPP algorithm that takes in an input x and random string r. From section 1.1 we know that A can be simulated by a family of Boolean circuits of polynomial size (size at most n^c for every input x of length n to A). Without loss of generality assume that on input x, r is

of length at most n^c (if $|r| > |n^c|$ then the machine can't possibly read the entire random string in time). What we wish to do is replace these random bits with pseudorandom ones, and show that A will still be correct "with high probability". The proof of theorem 12 essentially boils down to proving this subclaim: for every x of length n, $A(x, G(U_{d(n^c)}))$ errors with probability smaller than 1/2.

Proof of subclaim: Suppose towards a contradiction that there exists an input x where $A(x, G(U_{d(n^c)}))$ errors with probability at least 1/2. Then we can define a boolean circuit $T(\cdot) = A(x, \cdot)$ of size at most n^c that distinguishes U_{n^c} from $G(U_{d(n^c)})$. T will take in as input either a truly random or pseudorandom string, and use it to run A on input x with the randomness provided as input.

We can reason the advantage of T. From our assumption $A(x, G(U_{d(n^c)})$ errors with probability at least 1/2, and since A is a BPP algorithm it errors with probability at most 1/3 over uniform randomness. The advantage of T is thus $1/2 - 1/3 > 1/8$. This contradicts our assumption that our generator G has $\varepsilon = 1/8$. □

Note that T fixes the particular input x as advice, a reason why we define PRGs to be hard against nonuniform tests. This short proof also showcases the usefulness of reductions and proof by contradiction for proving indistinguishability. With the subclaim being true the runtime of theorem 12 comes from running A and G over all $2^{d(n^c)}$ seeds to the generator and taking a majority answer [Vad12]. For derandomization, since we are running a randomized algorithm $2^{d(n^c)}$ times, we can actually afford to have the PRG run in time $\mathsf{poly}(m, 2^{d(m)})$, exponential in the seed length. Next, just like with theorem 9, we can show via the probabilistic method that good PRGs exist.

Theorem 13 *(from [Vad12]) For all $m \in \mathbb{N}$ and $\varepsilon > 0$, there exists a (m, ε) pseudorandom generator $G : \{0,1\}^d \to \{0,1\}^m$ with seed length $d = O(\log m) + 2\log 1/\varepsilon$.*

Proof: We pick a function $G : \{0,1\}^d \to \{0,1\}^m$ at random, and fix a time m algorithm T. The probability that T distinguishes $G(U_d)$ from U_m with advantage ε is at most $2^{-\Omega(2^d \varepsilon^2)}$ by a

Chernoff bound. We can union bound over all nonuniform algorithms running in time m by counting the number of circuits of size m, which there are $2^{\mathsf{poly}(m)}$ (where the $\mathsf{poly}(m)$ comes from counting the different gates).

We wish for the union bound probability to be less than 1.

$$\frac{2^{\mathsf{poly}(m)}}{2^{\Omega(2^d\varepsilon^2)}} < 1$$

For this to hold we want $d = O(\log m) + 2\log 1/\varepsilon$. □

1.3 Randomness from hard functions

Based on theorem 12, the shorter the seed length, the faster we are able to derandomize a particular algorithm. Looking broadly at BPP, we can reason which deterministic time complexity class it will lie based on the seed length d.

- $d = n^{\varepsilon}$ gets you $\mathsf{BPP} \subset SUBEXP = O(2^{n^{\varepsilon}}) \forall \varepsilon > 0$
- $d = \mathsf{poly}(\log n)$ gets you $\mathsf{BPP} \subset \tilde{P} = O(2^{\log^c n})$
- $d = O(\log n)$ gets you $\mathsf{BPP} = \mathsf{P}$

Note that one can't hope to get a generator with seed length exactly $\log n$, since each function has hardness 2^n trivially. However, unlike the extractor case, we need to be careful in considering the computational randomness that is being output from our PRG. As defined in section 1.2, we have to ensure that our PRG is indistinguishable against nonuniform algorithms running in time at most t.

1.3.1 Randomness as next-bit-unpredictability

The most elementary hard function takes in a string, and outputs 1 "hard" bit. It is useful for some of the future analysis below to establish that indistinguishability is preserved under multiple samples. This extends naturally to taking multiple "hard" bits from a hard function and

showing that that bitstring is also pseudorandom. The theorem and proof below utilize what is often referred to as the "hybrid argument" or "hybrid distributions".

Theorem 14 *(from [Vad12]) If random variables X and Y are (t,ε) indistinguishable, then for every $k \in \mathbb{N}$, X^k and Y^k are $(t,k\varepsilon)$ indistinguishable, where X^k and Y^k represent taking independent copies and concatenating them together.*

Proof: The proof will be of the contrapositive: if there is an efficient distinguisher T for X^k and Y^k with advantage greater than $k\varepsilon$, then there is a efficient distinguisher T' for X and Y with advantage greater than ε (where T' uses T as a subroutine).

We assume a *nonuniform* time T running in time t such that:

$$|\Pr[(T(X^k) = 1] - \Pr[T(Y^k) = 1]| > k\varepsilon$$

Define *hybrid distributions* $H_i = X^{k-i}Y^i$ for $i = 0,\ldots,k$, with $H_0 = X^k$ and $H_k = Y^k$. The inequality from our assumption above can then be rewritten as the following telescoping sum (writing out the sun explicitly you can notice that all but the first and last term cancel). We can also drop the absolute value in the advantage without loss of generality since we can simply negate the output of our nonuniform algorithm/circuit T.

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

Since the sum is over k terms, this implies that there must be some i such that the advantage is $> k$ implying (for that specific i): $\Pr[T(H_{i-1}) = 1] - \Pr[T(H_i) = 1] > \varepsilon$ Writing these hybrid distributions back out in terms of X and Y we get:

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

Now we can focus on the individual random variables X and Y. By the *averaging argument/averaging principle*, which states that any event A which depends on two independent

random variables B, C, then there exists a fixed b in the support of B such that $\Pr[A(b,C)] \geq \Pr[A(B,C)]$, there exists fixed $x_i, \ldots, x_{k-i}$ and $y_{k-i+2}, \ldots, y_k$ such that the above inequality still holds. In other words:

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

Now we have an inequality with just random variables X and Y! Define our distinguisher for X and Y as $T'(z) = T(x_1, \ldots, x_{k-i}, z, y_{k-i+2}, \ldots, y_k)$ with the specific index i and bits $x_1, \ldots, x_{k-i}, y_{k-i+2}, \ldots, y_k$ hardwired as advice. Thus we get a nonuniform algorithm T' running in time t that distinguishes between X and Y with advantage greater than ε. This is a contradiction to the indistinguishability of X and Y, implying that X^k and Y^k must be $(t, k\varepsilon)$ indistinguishable. $\square$

Note a small technicality here that this reduction wouldn't work for cases where $k \geq t$, since then the advice length would exceed the time it takes to read all of it.

With multiple pseudorandom bits, it is sometimes helpful to reformulate the notion of pseudorandomness to that of "next-bit unpredictability": given a prefix of the output, it should be hard to guess the next bit of the string much better than simply random guessing. More formally, we say a random variable X taking values from $\{0,1\}^m$ is (t, ε) next-bit unpredictable if for every nonuniform probabilistic algorithm P running in time t and for each index $i \in [m]$, the probability that given the first $i-1$ indices, $\Pr[P(X_1 X_2 \cdots X_{i-1}) = X_i] \leq 1/2 + \varepsilon$. The following theorem relates this next bit unpredictability property to pseudorandomness.

Theorem 15 *(from [Vad12]) For a random variable X taking values in $\{0,1\}^m$, if X is (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 $(t, m \cdot \varepsilon)$ pseudorandom.*

Proof: The forward direction (that pseudorandomness implies next-bit unpredictability) goes by contradiction. Assume towards a contradiction that X is not $(t-3,\varepsilon)$ next-bit unpredictable, and thus there is a predictor $P:\{0,1\}^{i-1}\to\{0,1\}$ (for fixed i) that succeeds in guessing the ith bit with probability at least $1/2+\varepsilon$. We then construct an algorithm $T:\{0,1\}^m\to\{0,1\}$ that distinguishes X from U_m (using P as a subroutine), contradicting the pseudorandomness of X.

Let T be the function as follows:

$$T(x_1x_2\cdots x_m)=\begin{cases}1 & \text{if } P(x_1x_2\cdots x_{i-1})=x_i\\ 0 & \text{otherwise}\end{cases}$$

Clearly the advantage of T on X is the same as P on a prefix of X, and T can be implemented in the same size as P plus an additional 3 gates to test for the equality of bit x_i.

The converse direction, that next-bit unpredictability implies pseudorandomness uses a similar *hybrid argument*. Assume towards a contradiction that X is not pseudorandom, which implies there exists a nonuniform algorithm T running in time t such that:

$$|\Pr[T(X)=1]-\Pr[T(U_m)=1]|>\varepsilon$$

Similar to theorem 14, we can drop the absolute values without loss of generality, and define the hybrid distribution $H_i=X_1X_2\cdots X_iU_{i+1}U_{i+2}\cdots U_m$ with $H_m=X$ and $H_0=U_m$. (Here we will have to deal with the inconvenience that U_i refers to the ith bit of a string as opposed to the conventional notion that U_i is the uniform distribution on i bits.) We can write the inequality above as a telescoping sum in the same way:

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

There exists an i such that:

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

We can reintroduce X_i's and U_i's:

$$\begin{aligned}&\Pr[T(X_1, \ldots, X_{i-1}, X_i, U_{i+1}, \ldots, U_m) = 1] \\ &- \Pr[T(X_1, \ldots, X_{i-1}, U_i, U_{i+1}, \ldots, U_m) = 1] > \varepsilon/m\end{aligned}$$

Notice that in the second term, we can treat the uniformly random bit U_i as X_i with probability 1/2, and it's negation $\overline{X_i}$ with probability 1/2. This allows us to simplify the expression as:

$$\begin{aligned}\frac{1}{2} \cdot (&\Pr[T(X_1, \ldots, X_{i-1}, X_i, U_{i+1}, \ldots, U_m) = 1] \\ &- \Pr[T(X_1, \ldots, X_{i-1}, \overline{X_i}, U_{i+1}, \ldots, U_m) = 1]) > \varepsilon/m\end{aligned} \tag{1.1}$$

Based on this analysis, we say our next bit predictor $P(x_1, x_2, \cdots, x_{i-1})$ will do the following 3 steps: first choose random bits $u_i, \ldots, u_m$ uniformly at random, then compute $b = T(x_1, \ldots, x_{i-1}, u_i, \cdots, u_m)$, finally output u_i if $b = 1$ else output $\overline{u_i}$

The intuition is because T is more likely to output 1 when $u_i = x_i$ based on equation 1.1. The probability of correctly predicting the next bit is:

$$\Pr[P(X_1, \ldots, X_{i-1}) = X_i]$$

$$
\begin{aligned}
=&\frac{1}{2}\cdot(\Pr[T(X_1,\ldots,X_{i-1},U_i,U_{i+1},\ldots,U_m)=1|U_i=X_i]\\
&+\Pr[T(X_1,\ldots,X_{i-1},U_i,U_{i+1},\ldots,U_m)=0|U_i\neq X_i])\\
=&\frac{1}{2}\cdot(\Pr[T(X_1,\ldots,X_{i-1},X_i,U_{i+1},\ldots,U_m)=1]\\
&+1-\Pr[T(X_1,\ldots,X_{i-1},\overline{X_i},U_{i+1},\ldots,U_m)=1])\\
=&\frac{1}{2}+\frac{1}{2}\cdot(\Pr[T(X_1,\ldots,X_{i-1},X_i,U_{i+1},\ldots,U_m)=1]\\
&+1-\Pr[T(X_1,\ldots,X_{i-1},\overline{X_i},U_{i+1},\ldots,U_m)=1])\\
>&\frac{1}{2}+\frac{\varepsilon}{m}
\end{aligned}
$$

Where the second equality is from rewriting $U_i \neq X_i$ as $\overline{X_i}$, and the last inequality is from applying equation 1.1. Finally the runtime of P is $t + O(m)$ from producing random bits and running T. Note that we can always fix random bits as advice (again via the averaging argument) while preserving the advantage which would bring down the runtime to t. □

1.4 Hardness amplification

As explored in the previous section: the general idea is that to derandomize an algorithm, it is enough to use the outputs of a sufficiently hard function in lieu of truly random bits. Since to achieve the goal of derandomizing an algorithm we want to enumerate over all outputs of the generator we require the hard function to be hard "on average". This is a stricter requirement then worst case hardness: often what people first learn about in their algorithms/data structure courses. The following definition is noted [AB09, Vad12].

Definition 16 *For a function $f : \{0,1\}^n \to \{0,1\}$ and parameter $\delta \in [0,1]$ we say that f is (t,δ) average case hard for all nonuniform algorithms C running in time t if:*

$$\Pr_{x\sim U_n}[C(x) = f(x)] \leq 1 - \delta$$

We usually refer to "mildly" average-case hard functions for general $\delta \in [0,1]$, and

"strongly" average-case hard functions for when algorithms can't do much better than random guessing the values of the function f. In the later case we would think of setting $\delta = 1/2 - \varepsilon$ for $\varepsilon = 1/s$ Note that worst case hardness is captured above with exponentially small $\delta > 0$, since we only need at least one input x where C and f differ.

A simple pseudorandom generator from a average case hard function $f : \{0,1\}^n \to \{0,1\}$ is the following: given a truly random seed $r \in U_n$, output $r \circ f(r)$. Note that a distinguisher can't distinguish such an output from a random string in U_{n+1} unless it can predict f, and that such a simple construction includes its seed in its output and is only feasible when the generator has more computational resources than the algorithm/distinguisher it is trying to fool (or else the distinguisher can just compute the function itself). Also see that such a construction requires the "average case" hardness of f. If f was only worst case hard, then it could be the case that for many inputs f is easy to compute and a distinguisher would have a greater probability of breaking the PRG given a uniformly random seed r.

1.4.1 Nisan Wigderson Generator

The above example only extends a truly random string into a pseudorandom string that is one bit longer. In a landmark paper, Nisan and Wigderson show how you can do much better (with respect to the output string length) given an average-case hard function. The construction is commonly referred to today as the "Nisan-Wigderson generator". The idea of the generator is similar to the example just covered, except that we want to split up the random seed r into smaller chucks, and evaluate them separately on the hard function f. To get a PRG output that is *exponentially* longer, they allow the split chunks of r to be "slightly dependent" on each other. They accomplish this by coming up with "designs": a family of subsets (of a string) such that for any two subsets their intersection in small.

Definition 17 *Designs*

A group of m subsets from a universe $[d]$ (ie: $S_1, \ldots, S_m \subset [d]$) is a (ℓ, a)-design if:

- $\forall i, |S_i| = \ell$
- $\forall i \neq j, |S_i \cap S_j| \leq a$

Given an (ℓ, a) design from a universe $[d] = [1, \ldots, d]$ of size d, the NW generator essentially evaluates the hard input function f m times on selected indices of the seed, and concats the evaluations together. The notation $x|_{S_i}$ refers to taking the indices of x defined by S_i. For example $x = 1001$ and $S_1 = \{1, 3\}$ yields $x|_{S_1} = 01$ (with 1 indexing).

The more formal definition is as follows.

Definition 18 *Nisan-Wigderson Generator (NW generator)*

Given a function $f : \{0,1\}^\ell \to \{0,1\}$ and an (ℓ, a) design with subsets $S_1, \ldots, S_m \subset [d]$, the generator $G : \{0,1\}^d \to \{0,1\}^m$ is defined as:

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

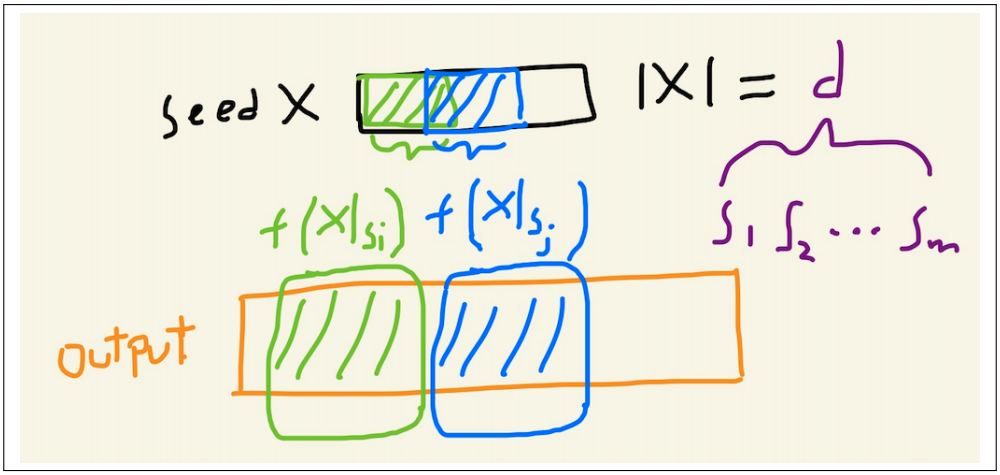

Figure 1.1. A pictorial representation of the "nearly disjoint" inputs to the hard function. Each output bit may share some bounded number of it's bits with the seed. Note the indices in each set S_i need not be contiguous.

The following proof of correctness highlights some concepts such as the hybrid argument result, the averaging principle, and the need for nonuniformity [Vad12, AB09].

Theorem 19 *If $f : \{0,1\}^\ell \to \{0,1\}$ is $(s, 1/2 - \varepsilon/m)$ average-case hard, then the NW generator G defined above is a (t, ε) pseudorandom generator, for $t = s - m \cdot a \cdot 2^a$.*

Proof: Assume towards a contradiction that G is not a (t, ε) PRG. This implies there is a nonuniform distinguisher (and by theorem 15), a next bit predictor P running in time t such that:

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

For random variable X taking values from U_d and $i \in [m]$. The goal will be to construct a nonuniform algorithm A that computes f with probability greater than $1/2 + \varepsilon/m$, which contradicts the assumed hardness of the function.

For each string $X|_{S_j}$ of length ℓ and $j \in [i-1]$ we will try to rewrite them in terms of the indices it shares with $X|_{S_i}$. Let Z_1 and Z_2 be two independent random variables corresponding to values of S_j in S_i and Z_2 be the ones in $[d] \setminus S_i$. We can now rewrite as the following, where each f_j is simply a wrapper function that applies the function f to the restriction of X defined by Z_1 and Z_2 (ie: constructs $X|_{S_j}$ from Z_1, and Z_2). We can now rewrite the above equation as the following:

$$\Pr[P(f_1(Z_1, Z_2)f_2(Z_1, Z_2) \cdot f_{i-1}(Z_1, Z_2) = f(Z_1)] > \frac{1}{2} + \frac{\varepsilon}{m}$$

Note that for bit i, we can simply look at Z_1 since by definition $Z_1 = S_i$. By the *averaging argument* (which was also mentioned in section 1.3.1) there exists a fixed z_2 in the range of Z_2 such that:

$$\Pr[P(f_1(Z_1, z_2)f_2(Z_1, z_2) \cdot f_{i-1}(Z_1, z_2) = f(Z_1)] > \frac{1}{2} + \frac{\varepsilon}{m}$$

Now by the parameters of our (ℓ, a) design we know that for each value z_1 of Z_1, computing any $f_j(z_1)$ only depends on $|S_j \cap S_i| \leq a$ bits of z_1. The rest of the bits can be fixed by z_2. Thus the only bits that are free are the $\leq a$ bits that each S_j shares with S_i. Since every

function on a bits can be computed by a boolean circuit of size $a \cdot 2^a$ naively (which we can give it's truth table as advice), we can define $A(y) = P(f_1(y) \cdot f_{i-1}(y))$ that computes the next bit with probability of error $1 - (1/2 - \varepsilon/m) = 1/2 + \varepsilon/2$. In time $t + m \cdot a \cdot 2^a = s$. This contradicts the hardness of the function f, and therefore it must be the case that G is a (m, ε) generator. □

Finally we will mention constructions of designs. The important point is that we can manage to get intersections a that are only logarithmic in the number of sets m, and the universe d only needs to be at most quadratic in the set size ℓ [Vad12]. A proof sketch is to show that via the probabilistic method that with non-zero probability all sets will have small intersection size, and via brute force find such sets greedily set by set. Notice that Nisan and Wigderson's construction, as well as the analysis, don't assume anything about the input function other than its hardness. This has given rise to the so called "black-box" PRG constructions which we will discuss more in section 4.1.

1.4.2 From worst-case hardness to average-case hardness

The requirement that the input hard function f to the NW generator be "average-case hard" may seem a bit out of the ordinary. Often times in computer science we measure the performance of algorithms in the *worst-case*. A natural question is if there are "hardness amplification" techniques to amplify "hard" functions to "harder" functions in some metric. Turns out we can look at *error correcting codes* to help us achieve this. At a high level, an error correcting code maps strings into slightly longer strings (often called *codewords* which may be over a larger alphabet) such that their "distance" increases. The distance metric that is commonly used is the *hamming distance*: given 2 strings x, y, the relative hamming distance $d_H(x, y) = \Pr_i[x_i \neq y_i]$ where i represents the indices of the strings.

The original motivation for error correcting codes is this: say Alice and Bob want to share information across a noisy channel. A message Alice tries to send to Bob could become corrupted. By first encoding her message using an error correcting code and sending the codeword, since each codeword is "far" apart from each other hopefully Bob will be able

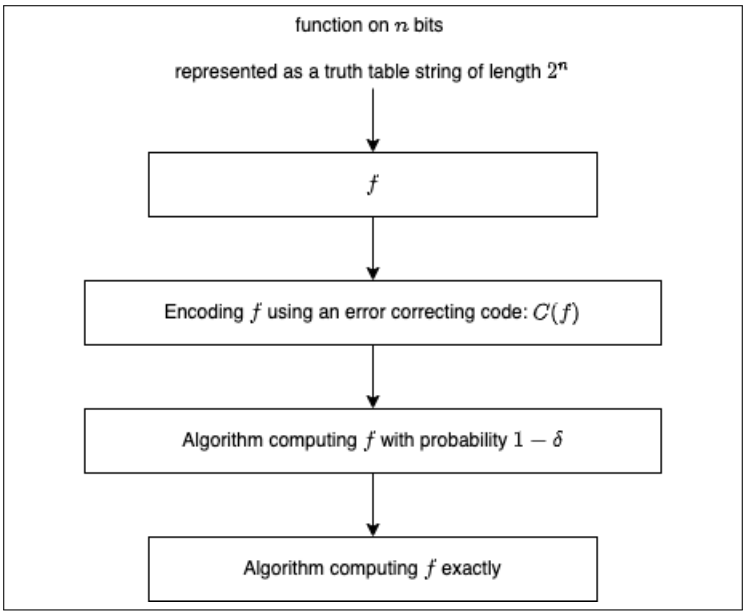

Figure 1.2. Using an error correcting code to encoding the truth table of a function. By properties of the code, being able to compute f approximately should still yield an algorithm computing f exactly

to decode the corrupted codeword back to the original message with high probability. More formally, an encoding function is a bijective mapping between messages and codewords such that the hamming distance between codewords are large.

To think about how error correcting codes could be related to hardness amplification, first think about the representation of a function $f : \{0,1\}^n \to \{0,1\}$ as a string of length $N = 2^n$. Running this "message" through an error correcting code from $\{0,1\}^N$ to $\{0,1\}^M$ we get a codeword representing a function $\hat{f} : \{0,1\}^{\log M} \to \{0,1\}$. Form the properties of an error correcting code, one should be able to recover f even if some portions of $\hat{f}$'s truth table was corrupted. In other words, one should be able to compute f *exactly* if they are able to compute $\hat{f}$ approximately! Taking the contrapositive, this implies that if f is hard to compute in the worst case, then $\hat{f}$ should be hard to compute on average [AB09].

Through the use of error correcting codes, we can turn a worst-case hard function to an average case hard function. This type of $1-\delta$ hardness is often referred to as "mildly" hard. However, we want to do better. Ideally for the NW generator construction we would like to get a

function that is "strongly" hard: one where the error rate is $1/2+\varepsilon$. In other words, a function where one can't do much better than randomly guessing the output. Here we turn to 2 concepts that are related: XOR lemmas and approximate local list decoding.

At a high level, XOR lemmas state that given a function that is mildly hard with respect to a complexity class, by taking multiple evaluations where the inputs are sampled from a distribution, computing the XOR/parity of the outputs should be strongly hard with respect to a *slightly* weaker class. The original construction proven by Yao samples inputs uniformly at random, later Impagliazzo and Wigderson showed that the inputs can be psuedorandomly generated [AB09, IW97, Yao82].

From the view of approximate local list decoding, the idea is very similar to figure 1.2. This time the truth table of a function f that we are sending as out "message" is a mildly hard function, and out encoded function $\hat{f}$ should be average case strongly hard. A proof sketch of its correctness goes by reduction. Suppose that there exists some algorithm $\tilde{C}$ that computes $\hat{f}$ on $> 1/2+\varepsilon$ of the inputs. Then there exists a nonuniform algorithm C with oracle access to $\tilde{C}$ that computes f on $> 1-\delta$ positions, a contradiction to the assumed hardness of f [AB09, Vad12].

With this proof sketch, we can note why we are looking at specifically "approximately local list" decodable codes. Firstly our reduction constructs a C that approximates f on $> 1-\delta$ inputs, just enough to get a contradiction. Therefore we are only approximating the original function f. Second, our decoder must be "local". Even though C has access to $\tilde{C}$ as an oracle, it can't afford the time to query all inputs to $\hat{f}$. This leads to the notion that each output of f should only depend on "few" outputs of the function $\hat{f}$. The final relaxation is the notion of "list" decoding. Whereas in our original error correcting code picture, Bob wishes to recover the exact message that Alice tried to send to him, with list decoding Bob may be content with decoding the corrupted message to a *list* of potential messages that Alice could have sent (with the original message guaranteed to be in the list). This relaxation form exact decoding to list decoding is essential for the reduction to work because we allow for C to be nonuniform. As inline with psuedorandomness's connection to nonuniformity, the reduction only guarantees that

there exists a nonuniform algorithm C with some advice string a that can approximately compute f. The problem is from the perspective of codes, the decoder only receives a corrupted codeword (which is representing a function's truth table). The decoder does not have any information about the advice string a, in fact it doesn't even know what the original function f is (that's what the decoder is trying to figure out)! Thus, the decoder can only hope to output a list of candidate messages (one can also think about it as the decoder trying to guess the advice string).

To connect XOR lemmas to codes, one can also think about the different XOR lemmas as a linear mapping between f and $\hat{f}$ that can act as one such encoding scheme. In this section we have only just scratched the surface of coding theory and it's connections to pseudorandomness from the perspective of hardness amplification. The area of coding theory has numerous other independent interests, so please do not consider what is introduced here to be any representation of the entire field. Nevertheless the link to pseudorandomness may seem non-intuitive at first, which is eye-opening to point out.

In this first chapter we have reviewed definitions of complexity classes closely related to pseudorandomness, as well as the precise notions of statistical and computational indistinguishability. We explored the concept of derandomizing randomized algorithms, as well as proving via the probabilistic method that optimal pseudorandom construction exist. A key topic we explored was that of relating pseudorandomness and next-bit unpredictability, as well as the hardness amplification and the Nisan-Wigderson pseudorandom generator construction.

Also note so far we have been talking about pseudorandom generators in the context of *derandomization*. One can define a pseudorandom generator fit for cryptographic proposes, but their requirements are more strict. For starters, cryptograpghic pseudorandom generators should consider all "efficent" distinguishes, not just those running in some specific time t. Cryptographic generators are also required to be efficiently computable (polynomial time), which is different in the derandomization lens where we allow the generator to run in time exponential in the seed length.

With many definitions and initial intuitions out of the way, we can now build off of

these ideas to look more closely at some classical results surrounding derandomization, as well as explore the surprising relationship between pseudorandom generators and randomness extractors.

Chapter 2

Derandomization and relations between PRGs and Extractors

Since Nisan and Wigderson's construction of a black-box pseudorandom generator and hardness amplification techniques introduced by Yao, there has been much progress in trying to derandomizing BPP [NW88, Yao82]. The connection between randomness and hardness naturally implies that derandomizing BPP gets you circuit lower bounds that are stronger than what we currently know [AB09]. Perhaps one of the most well-known results in derandomization comes from Impagliazzo and Wigderson.

2.1 Getting to $\mathsf{BPP} = \mathsf{P}$

One of main contributions in Impagliazzo and Wigderson's paper is looking more closely at the XOR lemma first presented by Yao and making improvements to this technique [IW97, Yao82]. Recall that Yao's XOR lemma states that given a boolean function f, if a certain model of nonuniform computation has a significant probability of failure at predicting the output of f given a random input, then any predictor (of a slightly smaller complexity) can't predict much better than a coin toss the *parity* of k random instances to f.

The main hurdle that the authors point out is that Yao assumed the k inputs to be independent. This required the hardness amplification (and eventual hard function for the NW generator) to increase its input size by a polynomial amount, which means the simulation will

end up being quasi-polynomial. Impagliazzo and Wigderson showed that it is not necessary (in the context of derandomizing BPP), for these separate inputs to be completely independent from each other. In other words rather that picking the inputs uniformly at random, it suffices for them to be generated pseudorandomly. This improved worst-case to strongly-hard average case reduction leads them to reduce the input size, and led them to the famous result that if some function in E (the class of problems decidable in time $2^{O(n)}$) has worst-case *circuit* size $2^{\Omega(n)}$, then $\mathsf{BPP} = \mathsf{P}$ [IW97].

2.1.1 Limits of the hybrid argument

The analysis of the NW generator in section 1.4.1 relied on that fact that the generator was "next-bit unpredictable". We needed to show that previous evaluations of the hard function don't help you that much in trying to predict future ones. As we will show, this tight relationship between pseudorandomness and next-bit unpredictability pose strict limits on how short of a seed length we can have.

Take the theoretically optimal seed length of PRGs derived in theorem 13. There we showed that the seed length $d = O(\log m) + 2\log 1/\varepsilon$, where m is the output length of the PRG. Just like our analysis of PRGs in section 1.3, if we wish to to merely extend the length of the PRG by a multiple of k to get a output length of km, we would need to show that the PRG needs to be next-bit unpredictable. For k samples, our PRG then needs to be in fact $(m, \varepsilon/k)$ unpredictable (by theorem 14).

Setting $\varepsilon' = \varepsilon/k$, we can see a new lower bound on the seed length. In order to create a next-bit unpredictable (m, ε)-PRG with longer output km, our new optimal seed would be with respect to ε':

$$
\begin{aligned}
O(\log m) + 2\log 1/\varepsilon' & \\
&= O(\log m) - \log[(\varepsilon')^2] \\
&= O(\log m) - \log[(\varepsilon/k)^2] \\
&= O(\log m) - (\log \varepsilon^2 - \log k^2) \\
&= O(\log m) - \log \varepsilon^2 + \log k^2
\end{aligned}
$$

This requires inherently an additional $2\log k$ in seed length of the generator if we are to take the "multiple samples" strategy. It also is often the case that we want the output to be a polynomial amount of bits (take for example the NW generator, or even extending the naive single bit output generator). This in turn gives us a large polynomial slowdown in the derandomization enumerating over all the seeds.

This isn't a big deal for showing that under certain circuit lower bound assumptions $\mathsf{BPP} = \mathsf{P}$, or in general derandomization results that are "efficent" in the polynomial sense. This does however pose an obstacle to answering the question of how *efficiently* we can derandomize algorithms. Taking a closer look at the polynomial slowdown has both practical and theoretical importance. To avoid this obstacle, we would need new types of PRG constructions that don't go from "one-to-many" bits but rather output them "all at once". This wouldn't require the proof of correctness to have to show previous outputs of a hard function give no significant advantage to predicting the next one. To explore this concept, we revisit randomness extractors (defined back in section 1.2) and their surprising relationship to computational randomness.

2.2 Parameters of PRGs and Extractors

With these objects defined in section 1, we can explore the relationships among black-box pseudorandom generators and randomness extractors. A key idea is to view the parameters of an extractor $Ext(x,y)$ where x comes from a k source of n bits (and y from uniform) as a truth table of a boolean function on $\log n$ bits where $f(i) = f_i$. Thus we can treat the input function as the "hard function" used in a black-box pseudorandom generator construction: $Ext(f,y) = G^f(y)$. This leads to the following relationship between these two objects:

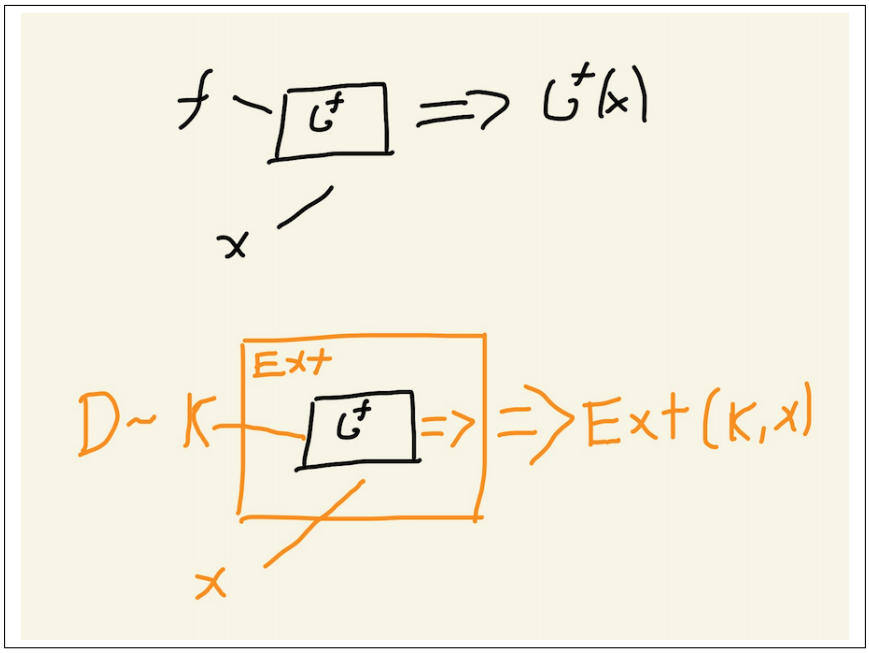

Figure 2.1. Given a string k drawn from a distribution D, and a random seed x, treat k as the truth table of a "hard" function to generator G^f.

Theorem 20 *If $G_f : \{0,1\}^d \to \{0,1\}$ is an ε-black box pseudorandom generator against arbitrary time adversaries with advice string length k, then $Ext(f,y)$ is a $(k + \log 1/\varepsilon, 2\varepsilon)$-extractor. On the flip side, if $Ext(f,y)$ is a (k,ε)-extractor, then G^f is a ε-PRG strong against arbitrary time adversaries with advice length $k+1$.*

Since we are relating to the statistical indistinguishably of randomness extractors, converting between the 2 objects requires a notion of indistinguishability that is strong against

algorithms running in arbitrary time (similar to section 2.2.1).

Some intuition for theorem 20: for any fixed nonuniform distinguisher there are 2^a different advice strings corresponding to 2^a different circuits each able to break a unique function. Therefore we can bound the total number of functions that this distinguisher is capable of breaking with the length of the advice given to it. With a source of the extractor having min-entropy $k + \log 1/\varepsilon$, the chance that a function (of length k) drawn from the distribution is one the distinguisher is capable of breaking is:

$$\frac{2^a}{2^{a+\log 1/\varepsilon}} = \varepsilon$$

Finally, the error bound of 2ε is due to the fact that computational indistinguishability is defined by looking at the absolute value (refer to section 1.2) of the advantage, which corresponds to a double sided error bound that is a factor of 2. Likewise, to convert from an (k, ε)-extractor to a ε-PRG, you would need an advice length of $a = k + 1$ to keep the ε-error rate.

Again this theorem is unfortunately non-constructive. There is nothing about the proof that actually shows how to convert a construction of a PRG into an extractor or vice-versa.

2.2.1 Constructive example of an Extractor from a PRG

In a seminal paper, Trevisan detailed a connection between pseudorandom generators and randomness extractors: namely that every PRG can be thought of as an extractor [Tre01]. He uses the same paradigm of viewing an extractor's 2 inputs as a function sampled from a k-source and evaluation point. Trevisan's lemma, which can be adapted similarly to other explicit PRG constructions, uses the Impagliazzo-Wigderson construction (which modifies the original NW construction) in it's analysis [IW97]. For the purposes of this connection, the IW generator states that if we have a function $f : \{0,1\}^\ell \to \{0,1\}$ that cannot be computed by circuits of size $2^{2\delta\ell}$, then IW_f will be $(2^{\delta\ell}, \varepsilon)$-indistinguishable from uniform.

For $x \in \{0,1\}^n$ where $n = 2^\ell$ we can think of x as a truth table to a function $\langle x \rangle : \{0,1\}^\ell \to$

$\{0,1\}$. Each bit of x can be thought of as the output bit of one if the 2^ℓ possible inputs to the function (the inputs can be in lexicographical order). Trevisan's lemma states:

Lemma 21 *Fix constants $\varepsilon, \delta > 0$ and an integer ℓ. Let the function $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be defined below for $d = O(\ell) = O(\log n)$ and $m = 2^{\alpha(\delta,\varepsilon)\ell}$. Then Ext is a $(m\delta n^\delta \log n + \log(1/\varepsilon), 2\varepsilon)$-extractor [Tre01].*

$$Ext(x,s) = IW^{(m)}_{\langle x \rangle}(s)$$

Here α is a constant parameter of the IW generator $0 < \alpha < \delta$. The indistinguishability parameter ε is referenced both by the PRG, as well as the extractor which bootstraps it. The proof of the lemma involves a counting argument.

Proof: We want to show for every random variable X of min entropy $k \geq m\delta n^\delta \log n + \log(1/\varepsilon)$ and every statistical test $T : \{0,1\}^m \to \{0,1\}$, the indistinguishability holds.

$$|\Pr[T(Ext(X,U_d)) = 1] - \Pr[T(U_m) = 1]| \leq 2\varepsilon$$

Fixing X and T, define the set of strings x which T succeeds distinguishing on as $B \subset \{0,1\}^n$:

$$|\Pr[T(Ext(X,U_d)) = 1] - \Pr[T(U_m) = 1]| > \varepsilon$$

Recall from our PRG definition this implies that there is a circuit (in this case of the IW generator of size $2^{\delta\ell}$) that uses T as an oracle to compute the original hard function $\langle x \rangle$. We can model oracle calls in a circuit by abstracting them as T-gates. Due to the nonuniformity of circuits, this applies to *each* $x \in B$. We can try to count $|B|$ by counting the number of circuits that compute $\langle x \rangle$ for each $x \in B$. Since any two different functions must be computed by two different circuits, and for the sake of simply counting circuits we can fix T (with fan-in at most m). Therefore the total number of elements of B is at most the number of circuits of size $S = 2^{\delta\ell}$ with gates (AND, OR, NOT, T) of fan-in at most m. This gives us: $|B| \leq 2^{mS\log S} = 2^{m\delta n^\delta \log n}$

(since $|S| = 2^{\delta\ell}$ and $n = 2^{\ell}$).

With a bound on B, we can think of having a source X for the extractor with min-entropy much higher. This means the probability of sampling a random function belonging to B gets smaller and smaller. Since we defined the source X to have min-entropy $k = m\delta n^{\delta}\log n + \log(1/\varepsilon)$ in lemma 21, we bound the following when sampling:

$$\Pr[X \in B] \leq |B| \cdot 2^{-k} \leq 2^{m\delta n^{\delta}\log n} \cdot 2^{-(m\delta n^{\delta}\log n + \log 1/\varepsilon)} = \varepsilon$$

Therefore we can show the indistinguishability of the extractor as for any test T:

$$|\Pr[T((Ext(X,U_t) = 1] - \Pr[T(U_m) = 1]| \tag{2.1}$$

$$\leq \mathop{\mathbb{E}}_{x\in X}[|\Pr[T((EXT(X,U_t) = 1] - \Pr[T(U_m) = 1]|] \tag{2.2}$$

$$= \sum_{x\in B} \Pr[X = x] \cdot |\Pr[T(Ext(x,U_t) = 1] - \Pr[T(U_m) = 1]| \tag{2.3}$$

$$+ \sum_{x\notin B} \Pr[X = x] \cdot |\Pr[T(Ext(x,U_t) = 1] - \Pr[T(U_m) = 1]| \tag{2.4}$$

$$\leq 2\varepsilon \tag{2.5}$$

Where line 2.3 comes from an application of the triangle inequality to split up the probability to an expectation over all the elements in the support of X. The conclusion on line 2.5 comes from the proof above that $\Pr[X \in B] = \varepsilon$ in line 2.4 , and that $\forall x \notin B$ can be thought of as the set of *good* strings such that the advantage $\leq \varepsilon$.

By counting the number of fixed size circuits, Trevisan was able to think of pseudo-random generators in a classical, statistical way. This argument is dependent on the particular parameters of the Impagliazzo-Wigderson generator, and this counting argument seems to work independently on the assumed hardness of the function f being used (up to $2^{n/2}$). Since harder functions yield larger circuits, and the hardness parameter δ is included in the min-entropy of the

source, assuming harder functions doesn't seem to get you better parameters. While we aren't particularly interested in fine tuning the performance of the extractor, Trevisan does later in his work describe alternate constructions that yield better performance [Tre01].

The key insight in sections 2.2 and 2.2.1 is the difference between the adversaries for pseudorandom generators and randomness extractors. For PRGs the set of possible adversaries is limited computationally, where as the extractor is tasked with producing a distribution that look uniform for any function.

2.3 Hardness amplification without the XOR lemma

Another result that leveraged this connection between PRGs and Extractors was interestingly enough a construction of a PRG that composed of a modified NW generator discussed in section 1.4.1 as well as the extractor just discussed in section 2.2.1. Sudan, Trevisan, and Vadhan came up with an alternative view of constructing pseudorandom generators not with average case strongly hard functions (derived from worst case hard functions and using hardness amplification described in 1.4.2), but with trying to see if just a mildly hard function is enough.

The authors were able to show that simply using a mildly hard function, one is able to generate bits with the the weaker notion that the output distribution is indistinguishable from having high Shannon entropy. Through some additional modifications they turn this distribution into one having high min-entropy, which they in turn are able to use an extractor to obtain a PRG. A motivation was to try to understand more deeply the intuition that generating the weaker notion of "pseudoentropy" should be cheaper than pseudorandomness. They credit Håstad et al. with the original idea of using pseudoentropy as a stepping stone to pseudorandomness via an extractor [STV99, HILL99].

Given the intuition as to why the NW generator requires a function that is average case extremely hard, it would make sense that if the hard function were only $1-\delta$ mildly hard then a distinguisher should only be able to correctly guess a small (but noticeable)δ-fraction of the

inputs. This the authors make formal via the hardcore set lemma proven by Impagliazzo [Imp95]:

Theorem 22 *(from [STV99, Imp95]) Suppose no circuit of size s can compute a function $P : \{0,1\}^\ell \to \{0,1\}$ on more than a $1-\delta$ fraction of the inputs. Then, for $\varepsilon > 0$, there exist an ε-hardcore set $H \subset \{0,1\}^\ell$ such that $|H| = \delta \cdot 2^\ell$ and no circuit of size $s' = \Omega(\varepsilon^3\delta^2 s)$ can compute P correctly on more than a $1/2+\varepsilon$ fraction of the inputs in the hardcore set H.*

Using this theorem it can be shown that the NW generator with only a $(1-\delta)$ average case hard function P should have $\delta \cdot m$ number of inputs land in H (in expectation if the inputs are sampled uniformly at random). This corresponds to $\delta \cdot m$ bits of pseudorandomness. A technicality they point out is since we are considering the output distribution *on average* to have $\delta \cdot m$ bits of pseudorandomness. This corresponds to having Shannon entropy (where as the application of a randomness extractor needs min entropy). To get a distribution having high min-entropy, they reason that it must be with high-probability that inputs land in the hard set H for P. We will not go into much detail on how they manage to accomplish this transformation, but at a high level they modify the NW generator by adding an additional seed that they use to generate *pairwise* independent strings which they XOR with the eventual inputs to the function P (recall the these inputs were originally computed via designs from a random seed, the authors simply compute the XOR of them with the pairwise independent strings mentioned above) [STV99].

Thus they are able to compute what they refer to as a pseudoentropy generator:

Definition 23 *Pseudoentropy generator (PEG)([STV99]) A generator $G : \{0,1\}^d \to \{0,1\}^m$ is a (k,s,ε) pseudoentropy generator if there is a distribution D on $\{0,1\}^m$ of min-entropy k such that no circuits of size s can distinguish the output of G from D with advantage greater than ε.*

With a pseudoentropy generator construction, they finally prove that using an extractor yields a computationally pseudorandom distribution:

Theorem 24 *(from [STV01]) Suppose $G_{PEG} : \{0,1\}^{d_1} \to \{0,1\}^m$ is a (k,s,ε_1) pseudoentropy generator and $Ext : \{0,1\}^m \times \{0,1\}^{d_2} \to \{0,1\}^n$ is a (k,ε)-extractor computable by circuits of size t. then $G_{PRG} : \{0,1\}^{d_1+d_2} \to \{0,1\}^n$ defined by $G_{PRG}(u,v) = Ext(G_{PEG}(u),v)$ is a $(s-t,\varepsilon_1+\varepsilon_2)$ pseudorandom generator.*

Proof: This proof goes by contradiction. Let D be the distribution of min-entropy k that cannot be distinguished from $G_{PEG}(U_{d_1})$. Suppose towards a contradiction there is a circuit $C_{PRG} : \{0,1\}^n \to \{0,1\}$ of size $s-t$ that distinguishes $G_{PRG}(U_{d_1},U_{d_2})$ from the uniform distribution with advantage greater than $\varepsilon_1+\varepsilon_2$. Similar to the reason why we can drop the absolute values in section 1.3.1, we can write the advantage of C_{PRG} as:

$$\Pr[C_{PRG}(G_{PRG}(U_{d_1},U_{d_2})) = 1] - \Pr[C_{PRG}(U_n) = 1] > \varepsilon_1 + \varepsilon_2$$

Define a "wrapper" circuit $C' : \{0,1\}^m \times \{0,1\}^{d_2} \to \{0,1\}$ to be the circuit of size s (since Ext can be computed by circuits of size t and out assumption that C_{prg} is of size $s-t$) $C'(x,v) = C_{PRG}(Ext(x,v))$. We can then express the advantage of C' in terms of the advantage of C_{PRG}:

$$\begin{aligned}
&\Pr[C'(G_{PEG}(U_{d_1}),U_{d_2}) = 1] - \Pr[C'(D,U_{d_2}) = 1] \\
&\qquad = \Pr[C_{PRG}(G_{PRG}(U_{d_1},U_{d_2})) = 1] - \Pr[C_{PRG}(Ext(D,U_{d_2})) = 1] \\
&\qquad \geq \Pr[C_{PRG}(G_{PRG}(U_{d_1},U_{d_2})) = 1] - \Pr[C_{PRG}(U_n) = 1] - \varepsilon_2 \\
&\qquad > \varepsilon_1 + \varepsilon_2 - \varepsilon_2 \\
&\qquad > \varepsilon_1
\end{aligned}$$

Where the first equality comes from the fact that $Ext(D,U_{d_2})$ and U_n have *statistical distance* at most ε_2. The second to last inequality comes from the assumption of the advantage

of C_{PRG}. Similar to the proofs in section 1.3.1, we can hard wire some string $v \in \{0,1\}^{d_2}$ to get a circuit $C''(x) = C'(x,v)$ of size at most s which distinguishes $G_{PEG}(U_{d_1})$ from D with advantage greater than ε_1. This is a contradiction to the parameters of the generator G_{PEG}. $\square$

Thus their final PRG construction is actually a composition of a PEG and an extractor. The final seed length is the length of the seed for the PEG combined with the seed length of the extractor. The goal is for this combined length to be short for the purposes of derandomization. Note that this construction is only based on average mildly hard functions, and does not rely on the hardness amplification/XOR lemmas that were previously discussed in section 1.4.2. Additionally this generator also avoids the hybrid argument, by outputting "imperfect" bits only having the guarantee that they have high entropy. These results help motivate future work in beating the hybrid argument and its limitations to the generator's seed length .

In this chapter we explored the conditional results that $\mathsf{BPP} = \mathsf{P}$ assuming "plausible" lower bounds on complexity classes. Additionally we covered some relationships between pseudorandom generators and randomness extractors, and both constructions of a extractor from a PRG, as well as the idea of using an extractor to extract pseudorandomness. This latter idea will be key going forward as we survey a result that tries to achieve optimal derandomization by "extracting from a pseudoentropic source".

Chapter 3

Beating the Hybrid Argument

From the earlier section, we mentioned the limitations of the hybrid argument, and explored the result of Sudan, Trevisan, and Vadhan generating pseudoentropy as a stepping stone to pseudorandomness. This allows them to avoid the next-bit unpredictability requirement of PRGs, and thus avoid the 'hardness amplification + hybrid argument". In this chapter we will explore the results of a recent paper by Doron, Moshkovitz, Oh, and Zuckerman which follow the same paradigm of generating pseudorandom bits using a "PEG + Extractor" approach [DMOZ22]. Their results focuses on using this paradigm to achieve better polynomial slowdowns for randomization.

They assume quantitatively harder assumptions than what is required by Impagliazzo and Wigderson: namely that there exists a function $f : \{0,1\}^\ell \to \{0,1\}$ that can't be computed by circuits of size $2^{(1-o(1))\cdot\ell}$ but able to be computed uniformly in time $2^{O(\ell)}$ (as opposed to functions not computable by size $2^{\Omega(\ell)}$ circuits but computable in time $2^{O(\ell)}$). In addition to qualitative assumptions that require the functions be also hard against these sized circuits with access to randomness and nondeterminism. Through these additional assumptions, they show how to construct a PRG surpassing the barrier set by the hybrid argument.

3.1 Extracting from a Pseudoentropic Source

Motivated by Sudan, Trevisan, and Vadhan, the authors of the paper "Nearly Optimal Pseudorandomness from Hardness" looked at how avoiding hardness amplification/XOR lemmas can get better derandomization results [DMOZ22]. We know from section 2.1 that under circuit lower bound assumptions $\mathsf{P} = \mathsf{BPP}$, and randomized algorithms can be converted into deterministic ones with a polynomial slowdown. Looking at the general class of polynomial time more closely, an interesting question is to ask if there are more "optimal" derandomization techniques. Are there ways to trim the polynomial slowdown to smaller and smaller powers? One of the barriers standing in the way is the hybrid argument, which as explained in section 1.3.1, is often used to show the hardness of multi-bit output PRGs. The authors in [DMOZ22] look at derandomizations that incur a *nearly quadratic* slowdown. To achieve this goal, and in general to get these types of optimal derandomziation slowdowns, it is important to beat the hybrid argument.

It was important for the authors to avoid this "one-to-many bits" type of construction, and instead focus their attention on other paradigms. The pseudoentropy generator + extractor approach introduced section 2.3 is a paradigm that allows you to avoid the hybrid argument [STV99]. By no longer requiring uniform looking bits, there no longer needs to be a cumulative error of $\frac{1}{t}$ per bit (where t is the number of output bits of the generator). Hence the problem transforms into generating bits that are allowed to be imperfect, in which a smaller seed length may suffice.

The explicit construction given by Doron, Moshkovitz, Oh, and Zuckerman utilize a combination of a pseudoentropy generator, and randomness extractor. This composition leaves the overall seed to be the seed length of the PEG plus the seed length of the randomness extractor. The relaxed notion of pseudoentropy rather than pseudorandomness allows the authors to construct a generator that fools size s circuits with seed length $(1+\alpha)\log s$ for a constant α. This result converts any randomized algorithm of input length n running in time $t \geq n$ to a

deterministic one running in time $t^{2+\alpha}$ for $a > 0$. We will discuss further below their increased assumptions, and how this enables their construction to achieve this result.

3.1.1 Additional Hardness Assumptions of [DMOZ22]

The strengthened hardness assumption that the authors make is that there is a function $f \in \mathsf{E}$ that is hard for circuits of size at least $2^{(1-\alpha')n}$ with access to both nondeterminism and randomness. They formally define what it means for a circuit to have access to both as "single-value nondeterministic circuits (SVN circuits)" [DMOZ22]:

Definition 25 *Let $f : \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$ be a partial function. a SVN circuit computing f is a pair of circuits $C : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C_{\mathsf{check}} : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ such that for input string $x \in \{0,1\}^n$ and witness string $y \in \{0,1\}^w$ the following holds:*

- *$\forall x \in \{0,1\}^n$, $f(x) \neq \perp$ iff $\exists y \in \{0,1\}^w$ such that $C_{\mathsf{check}}(x,y) = 1$. This can be called the witness circuit.*
- *$\forall x \in \{0,1\}^n$, and $y \in \{0,1\}^w$ such that $C_{\mathsf{check}}(x,y) = 1$, it is true that $C(x,y) = f(x)$ This can be thought of as the computing circuit.*

They note that the total size of an SVN circuit is the size of the 2 circuits defined above. And we can think of a combined circuit combining both C and C_{check} by simply running them both and returning $m+1$ output bits. This is in essence a nonuniform analogue of $\mathsf{NP} \cap \mathsf{coNP}$ (though note that NPand coNPare defined against total functions, rather than partial functions). Allowing the checking circuit to use randomness transforms the analogue from $\mathsf{NP} \cap \mathsf{coNP}$ to $\mathsf{MA} \cap \mathsf{coMA}$ (where the class MA stands for Merlin-Arthur, the complexity class similar to NP, except the verifier gets access to randomness).

Definition 26 *Let $f : \{0,1\}^n \to \{0,1\}^m \cup \{\perp\}$ be a partial function.*

A randomized SVN circuit computing f with probability $1-\delta$ for $0 \leq \delta < \frac{1}{2}$ is a pair of circuits $C : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}^m$ and $C_{\mathsf{check}} : \{0,1\}^n \times \{0,1\}^w \to \{0,1\}$ such that for

input string $x \in \{0,1\}^n$, witness string $y \in \{0,1\}^w$, and random string $r \in \{0,1\}^d$ the following holds (note that the random string is only used in the checking phase, and is not shared to the computing circuit):

- $\forall x \in \{0,1\}^n$, $f(x) \neq \perp$ *iff* $\exists y \in \{0,1\}^w$ *such that with probability greater than* $1-\delta$ *the witness circuit will be correct:*

$$\Pr_{r \sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \geq 1 - \delta$$

- $\forall x \in \{0,1\}^n$ *where* $f(x) \neq \perp$ *and* $y \in \{0,1\}^w$, *either* $\Pr_{r \sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \geq 1-\delta$, *or* $\Pr_{r \sim U_d}[C_{\mathsf{check}}(x,y,r) = 1] \leq \delta$.

- $\forall x \in \{0,1\}^n$, *and* $y \in \{0,1\}^w$ *such that* $\Pr_{r \sim U_d}[C_{\mathsf{check}}(x,y) = 1] \geq 1-\delta$, *it is true that* $C(x,y) = f(x)$.

These bullets ensure the probability the witness circuit is correct is significantly bounded from each other (by a constant). Finally, the authors define a $DensityApprox_\eta$-gate, which they will use later in their construction to perform extraction of a string with high metric pseudoentropy. In a nutshell, the gate outputs an additive η-approximation of the fraction of accepting inputs to a circuit [DMOZ22].

Definition 27 *For a fixed error parameter $\eta > 0$, a* $\text{DensityApprox}_\eta$ *gate is an oracle gate that takes as input an encoding of a circuit C and outputs $p \in \{0,1\}^{\lceil log \frac{1}{\eta} \rceil}$, such that $|\mathbb{E}[C] - p| \leq \eta$.*

These definitions allow us to define functions that are hard against these types of circuit classes in a similar way.

3.1.2 Multiple notions of Pseudoentropy

In contrast to statistical min-entropy, the authors focus on pseudoentropy: the computational analogue of statistical entropy. In particular they use the definitions of metric pseudoentropy, and Yao's pseudoentropy from Barak, Shaltiel, Wigderson, and Yao [BSW03, Yao82].

Definition 28 *(metric pseudoentropy) Let X be a random variable distributed over* $\{0,1\}^n$*, s be a positive integer, and* $\varepsilon > 0$*. We say that a random varialbe X has metric pseudoentropy k,* $H^{metric}_{s,\varepsilon}(X) \geq k$ *iff for every circuit* $D : \{0,1\}^n \to \{0,1\}$ *of size s there exists a random variable Y distributed over* $\{0,1\}^n$ *such that* $H_\infty(Y) \geq k$ *(Y has min-entropy at least k) and* $|\mathbb{E}[D(X)] - \mathbb{E}[D(Y)]| \leq \varepsilon$

Alternatively, we can definite metric pseudoentropy of X as having metric pseudoentropy at least k against circuits of size s and ε *error if it holds that* $\Pr[D(X) = 1] \leq \frac{|Supp(D)|}{2^k} + \varepsilon$ *[BSW03]. Recall* $Supp(D)$ *is the set of all inputs for which the circuit accepts.*

Definition 29 *(from [DMOZ22]) Yao pseudoentropy: For some positive integers* s, n, ℓ*, and* $A \subset \{0,1\}^n$*, we say that* $A \in C^{Yao}_{\ell,s}$ *if there exists a circuit* $c : \{0,1\}^n \to \{0,1\}^\ell$ *and a circuit* $d : \{0,1\}^\ell \to \{0,1\}^n$ *both of size s such that:*

$$A = \{x : d(c(x)) = x\}$$

Now let X be a random variable over $\{0,1\}^n$*, and* $\varepsilon > 0$*.* $H^{Yao}_{s,\varepsilon} \geq k$ *if for every* $\ell < k$ *and subset* $A \in C^{Yao}_{\ell,s}$ *it holds that* $\Pr[X \in A] \leq 2^{\ell-k} + \varepsilon$

Yao's pseudoentropy can be thought of as "compressing" sets, with the circuit c often called the compressing circuit, and d the decompressing circuit. The authors extend Yao's definition to allow the decompressing circuits to be SVN circuits that allow nondeterminism. They refer to this as *NYao*-pseudoentropy. Note that in this case the parameters of the decompressing circuit become $d : \{0,1\}^\ell \times \{0,1\}^{w'} \to \{0,1\}^n$ computing a partial function $f_d : \{0,1\}^\ell \to \{0,1\}^n \cup \{\perp\}$ such that $A = \{x : f_d(c(x)) = x\}$.

For those familiar with the more common (and more powerful) $Hill$-pseudoentropy, metric-pseudoentropy is simply a weaker form where the distribution Y can be tailored to each circuit D (the quantifiers are reversed), whereas $Hill$-type mandates a Y that holds for all circuits of size s. Note that in the case of $k = n$, the *HILL* type definition is the same as the standard

definition of pseudorandomness. This is because there is only one such distribution Y over $\{0,1\}^n$ such that $H_\infty(Y) \geq n$ (the uniform distribution) [BSW03].

A result that the authors show (based on the proof ideas of the paper by Barak Shaltiel, and Wigderson) is that Yao's compression type pseudoentropy implies metric pseudoentropy when considering hardness against polynomial sized circuits with nondeterminism. The following theorem is shown with SVN circuits and NYao that is defined above [DMOZ22, BSW03] .

Lemma 30 *(from [DMOZ22]) There exists a constant* $0 < \gamma < 1$ *such that the following holds. Let* X *be a random variable distributed over* $\{0,1\}^n$ *and* $\varepsilon > 0$*. There exists* $s_0 = \tilde{O}(n)$ *such that for every* $s \geq s_0$*,* $H^{NYao}_{s,\varepsilon}$ *implies* $H^{metric}_{\gamma \cdot s,\varepsilon} \geq \frac{k}{2}$

A reminder that $\tilde{O}(n)$ simply hides all the logarithmic factors when looking at the asymptotic runtime. At a high level, to prove the lemma, the authors consider hashing the support of X (definition of metric pseudoentropy) using a two-universal family of hash functions. Taking an efficient implementation of computing two-universal hash functions requires circuits of size $\tilde{O}(n)$. Using a probabilistic argument, it can be shown that there exists some hash function h^* within the function family that is one to one with the support of D. To relate it to NYao pseudoentropy, they let the compression circuit simply compute h^*, while the decompression circuits maps elements from the hash set back to the support of X. Note that allowing the latter circuit to be SVN here seems key, as it needs to be able to compute a partial function. The support of X may be smaller than the input space, so there are inputs with no defined answer. One can also think about this construction as the SVN circuit nondeterministically guessing h^* to compute elements from the image of h^* back to its preimage.

3.1.3 Pseudoentropy from Hardness

One of the authors main results is showing how to construct a metric pseudoentropy generator from a function f that is hard for SVN circuits [DMOZ22]. Their outline is to show that if there is a code C that is *locally list recoverable*, then the symbol at a random coordinate

of the code corresponding to the truth table of f has high Yao pseudoentropy (which implies metric pseudoentropy via lemma 30). Note that in the below definition, $k = m$ would imply it's a pseudorandom generator.

Definition 31 *(from [DMOZ22]) A function $G : \{0,1\}^d \to \{0,1\}^m$ is a (k, s, ε) metric pseudoentropy generator (metric PEG) if*

$$H^{metric}_{s,\varepsilon}(G(U_d)) \geq k$$

In section 1.4.2, we discussed briefly error correcting codes and their relationship with hardness amplification. These codes are in a nutshell a set of "well-separated" strings called codewords of some alphabet Σ and of length n such that if we flip some bounded number of positions in the codeword, the original codeword is still the "closest" to the corrupted string. This allows us to decode the original message. A relaxed notion of retrieving the original codeword is that of "list" decoding, where one is allowed to output a number of candidate messages given a corrupted codeword and is only required that the actual original message be among them. In the "local" list decoding setting perhaps the received corrupted string is too long for us to read in entirety, so we can only read a bounded number of positions in the string to recover the original codeword at some position. Here we are operating index-by-index, and allow the decoding circuits to use randomness in which index of the received string they want to read. In this work, the authors utilize a "locally list recoverable code", which is similar to the "locally list decodable code" discussed in section 1.4.2. In this setting the corrupted string received by the other party contains, at each index of the corrupted string, a list of possible characters that the original message can be. The decoder has access to an oracle that, given a coordinate and index, returns a character from the respective list. The idea of a locally list recoverable code is the same: use few queries overall try to recover the ith character of the original message.

To get a pseudoentropy generator from a worst-case hard function, they show if a function f is hard for SVN circuits, and for some locally list recoverable code C, then the symbol at a random coordinate of the code of the truth table of f has high Yao psuedoentropy. Here we are

doing the same thing as in section 1.4.2 where we pretend the "message" we wish to encode is the truth table of a function. This in turn implies high metric pseudoentropy by lemma 30. Note that their construction yields a *strong* PRG, which refers to generators that prepend the random seed as part of their output.

More formally: say $f : \{0,1\}^{\log n} \to \{0,1\}$ is hard for SVN circuits such that their size needs to be at least $n^{1-\alpha_0}$ (exponential in the input length of f) for some constant $\alpha_0 < \frac{1}{6}$ and the locally list recoverable code $\mathsf{C} : \{0,1\}^n \to \Sigma^m$. The metric PEG will take as input an index of the codeword, and output the corresponding character: $G^f : [m] \to \{0,1\}^{\log|\Sigma|}$ It turns out out that this construction (with the tuning of some parameters) gives a metric PEG. The main idea behind their proof is to have SVN circuits simulate the oracle that the locally list decodable algorithm calls.

Here is a sketch of their proof idea: they assume towards a contradiction that the generator above does not produce high enough *NYao* pseudoentropy, and use that assumption to construct a decoder for the locally list recoverable code. This allows one to compute f, which contradicts its hardness. Formally the assumption is:

$$H^{NYao}_{s',\varepsilon}(Z \circ G^f(Z)) < k'$$

For some s', ε, k' let Z be the random variable uniformly distributed over U_m. For the proof overview we don't care about the particular parameter values s', ε and k'. The above assumption implies that there exists a set $A \in C^{NYao}_{k',s'}$, and a compression and decompression SVN circuit pair c, f_d respectively such that for every pair $(z,\sigma) \in A$ the circuits correctly computes $f_d(c(z,\sigma)) = (z,\sigma)$. The authors define the subsets $S_1, \ldots, S_m \subseteq \Sigma$ as the following: every S_z defines the set of pairs (z,σ) for a fixed z in A. In other words: $S_z = \{\sigma \in \Sigma \ : \ (z,\sigma) \in A\}$. It is true that $\sum_{z\in[m]} |S_z| \leq 2^{k'}$, since the compression circuit f_d outputs strings of length k'. They define the following function $f'_d : \{0,1\}^{k'} \to \Sigma \cup \{\perp\}$ that computes f_d but simply omits z from the output (only returning a character $\sigma \in \Sigma$).

Their next observation is that f_d can act as the oracle for $\bar{S} = (S_1, \ldots, S_m)$ for a locally list recoverable code, and thus f_d can simply replace a circuit's oracle queries with a circuit computing f_d. This gets an overall decoding circuit for the locally list recoverable code, and thus a circuit that contradicts the hardness of the original input function f. Showing that the generator produces high NYao pseudoentropy leads them to a generator for metric pseudoentropy via lemma 30 (with increased hardness assumptions).

Finally the authors mention how to explicitly construct such a locally list recoverable code. Since for their application they wish for the alphabet size of the code as well as the list size to be very large (for higher entropy), they aim to have them both *exponential* in terms of n while still keeping the ε agreement term the same. Particularly they want a locally list recoverable code $\mathsf{C} : \{0,1\}^n \to \Sigma^m$ such that $|\Sigma| = 2^{n^{1-O(\alpha)}}$, each index is recovered with $\approx n^\alpha$ queries to the oracle, and gives pseudoentropy for circuits of size $n^{1-O(\alpha)}$. Note that in the setting above a truth table of length n represents a function on input length $\log n$. Since the seed length of the generator is $\log m$ (dependent on the length of the codewords), they ensure that $m = n^{O(\alpha)}$ which implies a seed length of $O(\alpha) \cdot \log n$.

3.1.4 Extracting from a pseudoentropic source

Finally with a pseudoentropy generator, we still wish to get a pseudorandom generator that is useful for derandomizing BPP. To accomplish this the authors apply a randomness extractor to the output of the metric PEG defined above. The problem is that the output of the above PEG is measured in metric pseudoentropy, which is quite different from the statistical min-entropy source that extractors are typically defined for. To resolve this issue, the authors show that under some additional hardness assumptions it is possible to extract pseudorandom bits from metric pseudoentropy.

As a warm-up, a quick proof overview as to why Hill pseudoentropy is much nicer to extract from and the workaround for metric pseudoentropy: recall from section 3.1.2 that a distribution X has Hill type pseudoentropy if there is another distribution X_0 that is compu-

tationally indistinguishable from X. If we assume towards a contradiction that $Ext(X,y)$ (for random seed $y \in \{0,1\}^d$) is not indistinguishable from uniform then we would be able to use such a distinguisher (lets call it C) and extractor Ext to distinguish X and X_0 by letting such a distinguisher $D(x) = C(Ext(x,y))$ by fixing the particular y. (since it should be the case that $Ext(X_0,y)$ should be close to uniform). This would be a contradiction to the Hill pseudoentropy of X.

Since the quantifiers are reversed for metric pseudoentropy, then the distribution X_0 may be different for each distinguisher on X. The framework above would fall apart since we can't claim a single extractor, distinguisher construction $D(x) = C(Ext(x,y))$ to use to get our contradiction. To resolve this issue they instead try to estimate $\mathbb{E}[C(Ext(x,U_d))]$, which would avoid the dependence on seed y (where y may vary for each different distribution). Here they use the $DensityApprox_\eta$-gates defined in section 3.1.2 to help estimate (on input x) $\mathbb{E}[C(Ext(x,U_d))]$, and compare it to $\mathbb{E}[C(U_m)]$ which will be hard wired into the distinguishing circuit. Since these expectations will be real number, they will need to truncate it to some error to hard-wire it as advice to a circuit trying to distinguish the extractor. For this proof to follow through we need to assume hardness for these specialized $DensityApprox_\eta$ gates, which the authors reduce to hardness against randomness since one can approximate the expectation by random sampling inputs (accomplishing the same role as the aforementioned gates).

Note that in the paper [BSW03], the authors have outlined other methods to extract and convert different forms of pseudoentropy. However [DMOZ22] can't afford the parameter loss outlined in their methods. An open question is to improve parameters of pseudoentropy conversion with the same hardness assumptions. In addition, it should be mentioned that the authors also needed to be careful about the *runtime* of the eventual PRG construction, their eventual construction relies on existing code constructions, and results on algorithms for encoding/decoding quickly [DMOZ22, vdHS13].

It is interesting that the authors made both qualitative and quantitative hardness assumptions in order to achieve their derandomization results. Not only did they need to assume harder

circuit lower bounds but also additional hardness assumptions against randomness and nondeterminism. Naturally a question to ask is if these extra assumptions are necessary. Is it possible to get similar results with less assumptions? In the next chapter we will look at other papers that build off these ideas.

Chapter 4

Lower Bounds on Derandomization

In the previous chapter we looked at a way to achieve faster derandomization of algorithms in BPP. It's methods required both quantitatively harder assumptions than those originally proposed by Impagliazzo and Wigderson [DMOZ22, IW97]. While the additional assumptions against nondeterminism and hardness were essential to the authors proofs, a question to ask if these qualitative assumptions are necessary. Is it possible to simply use the harder circuit lower bound assumptions to achieve the same results? In the following paper by Shatiel and Viola, they aim to answer some of these questions by formally defining what it means to have a black-box pseudorandom generator [SV22]. In a follow up paper, Chen and Tell also provide an alternative "very fast" derandomization assuming the existence of one-way functions: polynomial time computable functions that are hard to invert for all polynomial time adversaries.

4.1 Limits to Black Box Constructions

As we have seen before, the original Nisan-Wigderson generator is a "black-box" PRG: nothing is assumed about the underlying hard function other than its hardness (against nonuniform models of computation).

In the quest to achieve faster derandomization (by necessarily beating the hybrid argument), new construction were introduced [DMOZ22, STV99, CT21]. However, questions still remained after their result as to whether or not their additional assumptions were necessary

(such as hardness against nondeterminism, randomness, or the existence of one-way functions). In a general result, Shaltiel and Viola prove lower bounds on these black-box constructions of generators in regards to nearly optimal derandomization (which they refer to as "extreme high-end" PRGs) [SV22]. They formalize what it means to have a "black box proof" for PRG constructions and hardness amplification, and are able to show that for these general constructions, "it is possible to fix many output bits of the [PRG] construction while fixing few bits of the hard function" [SV22].

First they differentiate between the "high-end hardness assumption" and the recently discussed "extreme-high end hardness assumptions". The former is simply a restatement of the one made by Impagliazzo and Wigderson, while the latter is necessary (but may not be sufficient) for the faster derandomization. This "extreme high-end" hardness assumption is simply one of the assumptions made by Doron, Moshkovitz, Oh, and Zuckerman, as well as Chen and Tell [DMOZ22, CT21].

Definition 32 *The high-end hardness assumption: There exist constants* $0 < \beta < 1 < B$*, and a function* $f : \{0,1\}^\ell \to \{0,1\}$ *such that:*

- f *cannot be computed by circuits of size* $2^{\beta \cdot \ell}$
- *the time it takes to compute* f *is* $\leq 2^{B \cdot \ell}$

Definition 33 *The extreme high-end hardness assumption: There exists a function* $f : \{0,1\}^\ell \to \{0,1\}$ *such that:*

- f *cannot be computed by circuits of size* $2^{(1-o(1))\ell}$
- *the time it takes to compute* f *is* $\leq 2^{(1+o(1))\ell}$

The high-end assumption gives rise to PRGs with seed length $r = O(\log m)$ and the time it takes to run the PRG $\mathsf{poly}(m)$. Not caring about the exact values of the constants β, B as well as the constant hidden in the big-O notation, we get the result $\mathsf{BPP} = \mathsf{P}$. Impagliazzo and

Wigderson showed that under this hardness assumption it was possible to construct such a PRG [IW97]. However if one does start worrying about the constants, and want a seed length r as close to $\log m$ as possible we must rely of functions that satisfy the *extreme* high-end hardness assumption. This would yield a seed length $r = (1 + o(1)) \cdot \log m$, and thus a better guarantee of a *nearly quadratic* slowdown (as opposed to a general polynomial slowdown). As stated before, the classical methods of converting hardness to randomness reply on the hybrid argument, which doesn't follow through in the context of optimal derandomization. It is still an open question as to whether such a PRG construction exists that is able to convert hardness to randomness at this nearly optimal rate with simply the extreme high-end hardness assumption.

In Shaltiel and Viola's work they formally define the black-box proofs and constructions of PRGs. A black-box proof that converts hard functions into PRGs (and PEGs) consists of 2 steps:

- a "construction map" that takes a ρ-hard function $f : \{0,1\}^\ell \to \{0,1\}$ and outputs a candidate ε-PRG $\mathsf{Con}(f) = \mathsf{Con}_f : \{0,1\}^r \to \{0,1\}^m$.
- a "reduction" that shows the correctness of the construction. The reduction is represented as an oracle function $\mathsf{Red}^{(D)}$ that has oracle access to a distinguisher D breaking the security of $\mathsf{Con}(f)$ (ie: distinguishing with advantage $> \varepsilon$). This oracle access, combined with a piece of nonuniform advice $\alpha \in \{0,1\}^a$ (which can depend on f and D) gives a function $C(x) = \mathsf{Red}^D(x, \alpha)$ that computes $f(x)$ for uniformly drawn x with probability $\geq \rho$. This contradicts the hardness of f.

Similarly they define that a black-box proof converting hard functions into pseudoentropy generators with parameters $\ell, r, m, a, k, \varepsilon, \rho, a$ (note the only new parameter k) as 2 functions $(\mathsf{Con}, \mathsf{Red})$ transforming a ρ-hard function to a $(k, \varepsilon) - PEG$. Another important parameter the authors take into account is the *number* of queries q that $\mathsf{Red}^{\mathsf{D}}(\mathsf{x}, \alpha)$ makes to its oracle when trying to compute $f(x)$. In a sense, the number of queries that a reduction makes determines the "hardness-loss" between the original function f an the PRG Con_f. Say that f cannot be

computed by circuits of size s. For a $C(\cdot) = \mathsf{Red}^D(\cdot, \alpha)$ to establish correctness, it needs to be able to compute f (greater than some probability) having size s. Since C has access to advice and makes q query calls, then the PRG must only have hardness against circuits of size $\frac{s-a}{q} \leq \frac{s}{q}$. We can think of the advice and queries as "adding" to C's circuit size, with the advice length being an additive cost to the size and the queries (which can be thought of as a circuit having size at most s instead of an oracle call) contributing a multiplicative factor for each oracle call. A PRG with stronger hardness would result in the C having size greater than s which would not contradict the hardness of f.

4.1.1 Measuring queries - "Useful" PRG Constructions

Since every function on ℓ bits has circuits of size 2^ℓ, for the construction/reduction to be useful the amount of queries $q \leq \frac{s-a}{m} \leq \frac{2^\ell - a}{m}$. A result the authors show is that if some reduction Red makes $\leq 2^\ell$ queries, then Con must allow "fixing many outputs with small information". The authors define Fix_j below aiming to measure the amount of "information" about the hard function f is lost when fixing outputs of the candidate PRG Con_f. Below we will use the notation that a function f takes in a bitstring of length n and outputs a bitstring of length m as $f_{n,m}$. As described below, the construction function Con takes in as input a function, and outputs another.

Definition 34 *Given* $\mathsf{Con} : f_{\ell,1} \to f_{r,m}$*, define* $Fix_j(\mathsf{Con})$ *to be the minimal number* h *such that there exist* j *distinct outputs* $z_1, \ldots, z_j \in \{0,1\}^m$ *such that:*

$$\Pr_{f \leftarrow \mathscr{F}_{\ell,1}} [\forall i \in [j] : \exists y_i \in \{0,1\}^r \; s.t. \; \mathsf{Con}_f(y_i = z_i)] \geq 2^{-h}$$

Where the probability is over the choice of function $f_{\ell,1}$ from the set of all functions from ℓ-bits to 1 bit (defined as the set $\mathscr{F}_{\ell,1}$). This Fix_j is with respect to a PRG construction function Con (which takes in as input a hard function and outputs a PRG function), and fixed values z_i. Intuitively Fix_j is trying to measure the probability that, given a random function f, the constructed PRG will "hit" the outputs z_i. The authors refer to the h as the "bits of information

lost". Viewing from the information theoretic perspective, if we let $j = 2^r$ (largest j possible), this implies that we want to fully describe Con_F of which there are 2^{2^r} possible functions.

The theorem below Shatiel and Viola prove and utilize to show two noteworthy results. The first is that while extractors can be constructed from using random functions as input to a PRG (see section 2.2.1), random functions often don't yield "useful" PRGs (which Red) makes $q \leq 2^\ell$ queries. The second results is lower bounding the derandomization rate of the "PEG + Extractor" approach of [DMOZ22] by bounding the seed length of black box pseudoentropy generators.

Theorem 35 *(from [SV22]) There exists a constant ν such that for every black-box $\rho-\mathsf{hard}-\mathsf{function}$ to $(k,\varepsilon)-\mathsf{PEG}$ proof $(\mathsf{Con},\mathsf{Red})$ with parameters $\ell, r, m, a, k, \varepsilon, \rho$ such that Red makes at most $q \leq 2^\ell$ queries. If $\rho = \frac{1}{2}+\eta, \eta \geq 2^{-\ell}, k \geq r, \varepsilon \leq 1-2^{r-m}$ and $a \leq \nu \cdot \eta^2 \cdot 2^\ell$, then for $j_{max} = \nu \cdot \frac{\eta^2 \cdot 2^\ell}{\ell}$. and for every $j \leq j_{max}$:*

$$Fix_j(\mathsf{Con}) \leq a + j \cdot (\log q + \log \frac{1}{\eta})$$

Proof overview: To begin proving the above theorem 35, the authors first define a simple distinguisher that, given a string of length m in the output set of the PRG, outputs 1 if there exists a seed that generates that output and a 0 otherwise. Since a reduction Red has oracle access to both a fixed distinguisher and advice of length a, by an averaging argument there exists a subset of hard functions $A_0 \subset \mathscr{F}_{\ell,1}$ and an advice string α' of length a such that for all $f \in A_0$, the reduction $C_f(x) = \mathsf{Red}^{D_f}(x, \alpha')$ ρ-hard function breaks f (where $x \in \{0,1\}^\ell$ and C and D are both with respect to some function $f \in A_0$). From fixing this reduction function C_f, an A_0 which defines these functions f, the advice given to C_f, and the distinguisher oracle which C_f calls, the authors compare how a general function $C : \{0,1\}^\ell \to \{0,1\}$ trying to also be a distinguisher hopes to simulate the reduction function.

To compare a function C that does not rely on an oracle D_f, the authors consider the case where all q queries $z_1^x, \ldots, z_q^x \in \{0,1\}^m$ made by some Red^{D_f} on input x are all answered by 0

(meaning that all queries are not in the range of the PRG). They define $C(x)$ to match the output of $\mathsf{Red}^{D_f}(x)$. For these x's, this further fixes the oracle calls. From the definition of D_f, we know that the number of queries in $\{0,1\}^m$ in which D_f answers 1 is small (only 2^r out of 2^m). They define a particular query $z \in \{0,1\}^m$ to be *t-weak* with respect to some set of functions $A \subset \mathscr{F}_{\ell,1}$ if given a random function from A, $\Pr_{f \leftarrow A}[D_f(z) = 1] \geq 2^{-t}$.

The authors then iteratively fix outputs z_i for Con that are $t = \log \frac{q}{\eta}$-weak. For the j outputs to fix, they define a set W of size j containing the t-weak outputs $z_1, \ldots, z_j$, and $A_j \subset A_0$ such that for every increment $j = 0, \ldots, j_{max}$ the following is true about A_j and W_j, with $W_0 = \emptyset$

1. $\Pr_{f \leftarrow \mathscr{F}_{\ell,1}}[f \in A_j] \geq 2^{-(a+jt)}$
2. $A_j \subset A_0$
3. $|W_j| = j$
4. for every $z \in W_j$ and $f \in A_j$, $D_f(z) = 1$

To incrementally build these 2 sets, for each step they build W_j and A_j by first trying to find some $z \notin W_{j-1}$ that is t-weak w.r.t. A_{j-1} (and if there doesn't exist such a z then stop), and let $W_j = W_{j-1} \cup \{z\}$, and $A_j = \{f \in A_{j-1} : D_f(z) = 1\}$. This they prove maintains the following four invariants above. They further show that this iteration will actually not stop until step j_{max}, implying that there are j_{max} such z that are t-weak against a set of functions $A_{j_{max}}$.

The above result allows us to conclude that the invariant holds for all $j \leq j_{max}$. Recall that we set $t = \log \frac{q}{\eta}$, and we get the following:

$$\Pr_{f \leftarrow \mathscr{F}_{\ell,1}}[f \in A_j] \geq 2^{-1(a+j \cdot t)} = 2^{-(a+t \cdot (\log q + \log \frac{1}{\eta}))}$$

Since we know that the invariant holds for all $j \leq j_{max}$, it implies that for each $f \in A_j$ and $z \in W_j$, there exists a seed $y \in \{0,1\}^r$ such that $\mathsf{Con}_f(y) = z$. We can rewrite the above expression as:

$$\Pr_{f \leftarrow \mathscr{F}_{\ell,1}}[\forall i \in [j] \ : \ \exists y_i \in \{0,1\}^r \text{ s.t. } \mathsf{Con}_f(y_i) = z_i] \geq 2^{-(a+t\cdot(\log q + \log \frac{1}{\eta}))}$$

The event being described above is the same as Fix_j: with respect to some hard function f, given j outputs $z_1, \ldots, z_j$ do there exists seeds $y_1, \ldots y_j$ such that $\mathsf{Con}_f(y_i) = z_i$? Thus a lower bound on this probability becomes an upper bound on the *minimal h* defined by Fix_j.

To note the parameters of theorem 35, the restriction that $k \geq r$ and $\varepsilon \leq 1 - 2^{r-m}$ is required to perform the first averaging argument to fix an advice string α' and A_0. In the proof that their iterative process continues until the j_{max}-step that we have glossed over, the constant η is introduced and can be made sufficiently small such that their proof follows through that their iterative process doesn't stop until j_{max}

4.1.2 Applications of Shaltiel and Viola's Theorem 35

With theorem 35 we can see why this theorem also applies to PRGs. At a high level, we have seen before that any PRG is also in a sense a PEG with optimal parameters. With the theorem also holding for PRGs, then it is with low probability that a randomly chosen chosen function will be a useful PRG construction where the reduction makes $q \leq 2^\ell$ queries. Whereas we have already shown (in theorem 9) that a random function with high probability is a good extractor.

Shaltiel and Viola also use theorem 35 to lower bound the performance of the "PEG + extractor" approach of used to construct a PRG [DMOZ22]. They show that such an approach can not yield an extreme high-end PRG from the "extreme high-end hardness assumption" when the PEG is constructed by a conventional black-box hard-function $\Rightarrow$ PEG approach. Since the overall seed length of the construction is the seed length of the PEG plus the extractor (for which the latter we know lower bounds for [SV22]). Not surprisingly, the following theorem they show follows from theorem 35 stated above:

Theorem 36 *(from [SV22]) There exists a constant $\nu > 0$ and $c > 1$ such that for every black-box ρ-hard-function $\Rightarrow (k,\varepsilon) - \mathsf{PEG}$ proof $(\mathsf{Con},\mathsf{Red})$ with parameters $\ell, r, m, a, k, \varepsilon, \rho$ such that*

$$r < \ell - \log \ell - 2 \cdot \log \frac{1}{\eta} - c$$

If $\rho = \frac{1}{2} + \eta$, $\eta \geq 2^{-\ell}$, $k \geq r$, $\varepsilon \leq 1 - 2^{r-m}$, and $a \leq \nu \cdot \eta^2 \cdot 2^{\ell}$, then Red *must make at least $q > 2^{\ell}$ queries.*

Notice that the requirements for the parameters in theorem 36 almost match theorem 35, except for the fact that theorem 35 states a result for reductions that make $\leq 2^{\ell}$ queries. Since we know theorem 35 holds, then it must be the case that for black box ρ-hard-function to $(k,\varepsilon) - \mathsf{PEG}$ proofs, it is either the case that $q > 2^{\ell}$, or the result of theorem 35 is true. Because of the restrictions on r defined by theorem 36, we have that $r < \ell - \log \ell - 2 \cdot \log \frac{1}{\eta} - c$, which implies that (For constants ν and c):

$$2^r < \frac{2^{\ell} \cdot \eta^2}{\ell \cdot 2^c} < j_{max} = \nu \frac{\eta^2 \cdot 2^{\ell}}{\ell}$$

This implies that it can't be the case that such a black-box ρ-hard function to $(k,\varepsilon) - \mathsf{PEG}$ proof $(\mathsf{Con},\mathsf{Red})$ can have j_{max} outputs fixed if it only has at most r outputs. Therefore a construction of this type must make $q > 2^{\ell}$ number of queries. Remember that black-box generator constructions should have a "useful" property that the reduction make less than 2^{ℓ} queries. Shatiel and Viola can therefore conclude that "useful" black-box PEGs should have seed length $r \geq \ell - \log \ell - O(1)$. Combining this lower bound on the seed length of the PEG with an extractor's seed length gives us the total seed length. From known bounds on extractor seed lengths we know that $r_{ext} \geq \log(m_{PEG} - k_{PEG}) \geq (1 - o(1)) \cdot \ell$ [Vad12]. This lower bound comes from the case where we are able to get "maximal" extraction where the output length of the generator $m_{PEG} = 2^{(1-o(1)) \cdot \ell}$ for some very hard function of ℓ-bits, and the entropy of that result $k_{PEG} = 2^{(1-o(1)) \cdot \ell}$ as well. Therefore we can lower bound the seed length of the "PEG +

extractor" by lower bounding the seed lengths of both the black-box PEG as well as the extractor:

$$\begin{aligned} r_{PRG} = r_{PEG} + r_{EXT} &\geq (\ell - \log \ell - O(1)) + \log (m_{PEG} - k_{PEG}) \\ &\geq (\ell - o(1)) + (\ell - o(1)) = 2\ell - o(1) > (2 - o(1)) \cdot \log m \end{aligned}$$

The lower bound on the seed length of this approach means that to derandomize an algorithm we would have to cycle through all seed lengths, of which there are $2^{(2-o(1))\cdot \log m} = m^{2-o(1))}$ of them. It shows that even in the extreme high-end assumption, using the approach of [DMOZ22] still gives a quadratic slowdown. It is worth reminding that Shaltiel and Viola's proof only assume circuit hardness, and not against stronger classes of circuits that are allowed nondeterminism and randomness. This lower bound is only in the case of "traditional" circuit classes.

We will note another recent result by Chen and Tell that also achieves "optimal" derandomization [CT21]. They also make more than just the "extreme" hardness assumption defined by Shaltiel and Viola, namely the existence of one way functions. One way functions are often defined to be a function that is uniformly polynomial time computable, which is hard to invert for any polynomial time adversary. In the context of pseudorandom generators, the existence of one-way functions naturally implies that there exists a polynomial time computable generator such that *all* polynomial time adversaries have small advantage in distinguishing its outputs from uniform. This is clearly a harder assumption than a function that is hard against a single class of circuits. Chen and Tell's result is a "PRG-composition" approach: composing a NW generator with a generator based on one-way functions. This construction, and Shatiel and Viola's lower bound on this model of PRG is a bit out of scope as its focus shifts away from the relationship between PRGs and extractors. We chose not to cover it in detail and opt to merely mention it here [SV22, CT21].

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, SFCS '78, page 75–83, USA, 1978. IEEE Computer Society.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. Math. (2)*, 160(2):781–793, 2004.

[BSW03] Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 200–215, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CT21] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 283–291, New York, NY, USA, 2021. Association for Computing Machinery.

[DMOZ22] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *J. ACM*, 69(6), nov 2022.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[Imp95] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, page 538, USA, 1995. IEEE Computer Society.

[IW97] Russell Impagliazzo and Avi Wigderson. P = bpp if e requires exponential circuits: Derandomizing the xor lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 220–229, New York, NY, USA, 1997. Association for Computing Machinery.

[NW88] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 2–11, 1988.

[STV99] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the xor lemma (extended abstract). In *Symposium on the Theory of Computing*, 1999.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SV22] Ronen Shaltiel and Emanuele Viola. On Hardness Assumptions Needed for "Extreme High-End" PRGs and Fast Derandomization. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 116:1–116:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, jul 2001.

[Vad12] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

[vdHS13] Joris van der Hoeven and Éric Schost. Multi-point evaluation in higher dimensions. In *Applicable Algebra in Engineering, Communication and Computing*, volume 24, pages 37–52, 2013.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91, 1982.