12
Derandomization

12.1 Introduction

In this lecture, we will only scratch the surface of a huge topic. There are many directions to explore (see online courses dedicated to this).

The central question is: If there exists a randomized algorithm for a problem (say, a decision problem) that takes polynomial time and is correct with probability at least $3/4$ (on all instances), does there exist a deterministic polynomial-time algorithm as well?

In complexity terms, we are asking:

$$\text{Is } \mathbf{BPP} = \mathbf{P}?$$

(Bounded-error Probabilistic Polynomial time vs. Deterministic Polynomial time.)

There are several major directions used to approach this:

- **Randomness from Hardness:** If a function is hard to compute or hard to invert, its outputs can be used to set up pseudorandom generators (PRGs) for bounded machines. This is often framed as "fooling" computationally restricted algorithms.
- **Limited Independence:** Designing algorithms whose analysis only needs limited independence (e.g. pairwise independence, k-wise independence) so that we can replace truly random bits by a smaller number of pseudorandom bits.
- **Randomness Amplification:** Using expanders and related objects to reduce the amount of randomness required (see notes from the expander lectures).
- **Other methods:** Randomness extractors (information-theoretic pseudorandomness), hash functions, etc.

These different ideas will keep showing up in the course. :contentReference[oaicite:0]index=0

12.2 *Pseudorandom Generators*

We focus on pseudorandom generators (PRGs) for a class of algorithms $\mathcal{A}$.

Let $G : \{0,1\}^m \to \{0,1\}^M$ be a generator, where $r \in \{0,1\}^m$ is a random ***seed*** and M is the number of random bits the algorithm needs ($M \gg m$). We want $G(r)$ to look indistinguishable from a truly random string $R \in \{0,1\}^M$ to any test algorithm $T \in \mathcal{A}$.

Definition 12.1. We say G ϵ-***fools*** a test T if

$$\left| \Pr_{r \in \{0,1\}^m}[T(G(r)) = 1] - \Pr_{R \in \{0,1\}^M}[T(R) = 1] \right| \leq \epsilon.$$

We say G ϵ-fools the class $\mathcal{A}$ if it ϵ-fools every $T \in \mathcal{A}$.

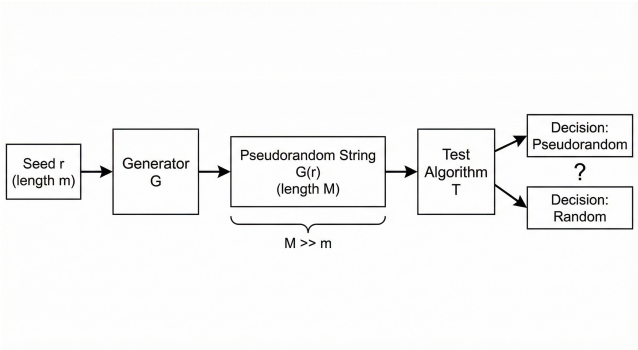

Figure 12.1: A pseudorandom generator G expands a short random seed r into a long pseudorandom string $G(r)$ that fools algorithms in class $\mathcal{A}$.

Intuitively, if we have an algorithm $A \in \mathcal{A}$ that uses M random bits and succeeds (e.g. outputs the correct answer) with probability at least $3/4$ under truly random bits, then using G instead of true randomness changes the success probability by at most ϵ:

$$\Pr_r[A(G(r)) \text{ correct}] \geq \frac{3}{4} - \epsilon.$$

This is already useful: if the seed length m is small, we can ***enumerate*** all seeds:

- If $m = O(\log n)$, then $2^m = \text{poly}(n)$.
- We can run A on $G(r)$ for every $r \in \{0,1\}^m$ and take, say, the majority answer, or the best solution found.

Since the average success probability over a uniformly random seed is at least $3/4 - \epsilon$, at least a $3/4 - \epsilon$ fraction of seeds are good. In particular, there exists at least one seed on which $A(G(r))$ is correct, and we can find it deterministically in time $2^m \cdot \text{poly}(n)$, which is polynomial if $m = O(\log n)$.

So a PRG with logarithmic seed length for the class of polynomial-time algorithms would imply $\mathbf{BPP} = \mathbf{P}$.

12.3 Derandomization via Limited Independence

12.3.1 The Max-Cut Example

Consider the Max-Cut problem. We are given an undirected weighted graph $G = (V, E)$ with nonnegative edge weights $(w_e)_{e \in E}$. We want to find a subset $S \subseteq V$ that maximizes the total weight of edges going between S and its complement $\bar{S}$:

$$\max_S w(E(S, \bar{S})) \;=\; \max_S \sum_{e \in E(S,\bar{S})} w_e.$$

Theorem 12.2. *There exists a cut S such that*

$$w(E(S, \bar{S})) \;\geq\; \frac{1}{2} \sum_{e \in E} w_e \;\geq\; \frac{1}{2} OPT,$$

where OPT is the value of a maximum cut.

Proof. For each vertex $v \in V$, pick a random sign $x_v \in \{-1, +1\}$ uniformly and independently. Let

$$S = \{v \in V \mid x_v = +1\}.$$

Consider an edge $e = \{u, v\}$. The edge is cut iff $x_u \neq x_v$. Since x_u, x_v are independent and uniformly random,

$$\Pr(x_u \neq x_v) = \frac{1}{2},$$

so the contribution of edge e to the ***expected*** cut weight is $\frac{1}{2} w_e$.

By linearity of expectation,

$$\mathbb{E}[w(E(S, \bar{S}))] \;=\; \sum_{e \in E} \frac{1}{2} w_e \;=\; \frac{1}{2} \sum_{e \in E} w_e.$$

Hence the average cut weight over all $2^{|V|}$ cuts is $W/2$, where $W = \sum_{e \in E} w_e$. Therefore there exists at least one cut with weight at least $W/2$. □

Exercise 12.3. Show that if we repeat this randomized algorithm $O(\log n)$ times and return the best cut found, then with probability at least $1 - 1/\mathrm{poly}(n)$ we obtain a cut of weight at least $(1/2 - \epsilon) \sum_e w_e$ for any fixed constant $\epsilon > 0$.

12.3.2 Pairwise Independence

To derandomize the above algorithm, let us look carefully at what properties of the randomness the proof actually used. The only place we used independence was to conclude that for each edge $\{u, v\}$ we have

$$\Pr(x_u \neq x_v) = \frac{1}{2}.$$

This remains true even if the random variables $(x_v)_{v \in V}$ are only ***pairwise*** independent.

Definition 12.4 (Pairwise Independence)**.** A distribution on $(X_1, \dots, X_n) \in \Omega^n$ is ***pairwise independent*** if for every $i \neq j$ and every $a, b \in \Omega$,

$$\Pr(X_i = a \wedge X_j = b) = \Pr(X_i = a) \Pr(X_j = b).$$

If the distribution is a ***product distribution*** (i.e. the X_i are mutually independent), then it is clearly pairwise independent. But the converse is false.

Example. Let X_1, X_2 be independent unbiased bits, and define

$$X_3 = X_1 \oplus X_2.$$

The joint support is

$$(0,0,0), (0,1,1), (1,0,1), (1,1,0),$$

four strings from $\{0,1\}^3$. This is not a product distribution (knowing X_1 and X_2 determines X_3), but any ***two*** of the variables are independent: the marginal on any pair is uniform on $\{0,1\}^2$.

This raises two questions:

1. How can we generate pairwise independent distributions over $\{-1,+1\}^n$ (or $\{0,1\}^n$) using only a few random bits?
2. What can we do with such distributions?

12.3.3 *A Finite-Field Construction of PI Distributions*

We construct a pairwise independent family of random variables using a finite field. Let $n = 2^b$ and identify the vertex set V with the field $\mathbb{F}_{2^b}$.

Pick a random slope $\alpha \in \mathbb{F}_{2^b}$ and a random intercept $\beta \in \mathbb{F}_{2^b}$, independently and uniformly. For each $z \in \mathbb{F}_{2^b}$ define

$$X_z = \alpha z + \beta \qquad \text{(arithmetic in } \mathbb{F}_{2^b}\text{)}.$$

Thus we obtain a random vector

$$X = (X_z)_{z \in \mathbb{F}_{2^b}}$$

whose coordinates range over $\mathbb{F}_{2^b}$.

Claim 12.5. The collection $(X_z)_{z \in \mathbb{F}_{2^b}}$ is pairwise independent and each X_z is uniform on $\mathbb{F}_{2^b}$.

Proof. Fix $z_1 \neq z_2$ and $a_1, a_2 \in \mathbb{F}_{2^b}$. Consider the event

$$\{X_{z_1} = a_1 \wedge X_{z_2} = a_2\}.$$

This is the event that

$$\begin{pmatrix} z_1 & 1 \\ z_2 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}.$$

The 2×2 matrix is invertible because $z_1 \neq z_2$, so there is ***exactly one*** pair (α, β) giving this outcome. Since (α, β) is uniform on $\mathbb{F}_{2^b}^2$, the probability of this event is $1/|\mathbb{F}_{2^b}|^2$.

On the other hand, for each fixed z, $X_z = \alpha z + \beta$ is a bijection in (α, β), so X_z is uniform on $\mathbb{F}_{2^b}$ and $\Pr(X_z = a) = 1/|\mathbb{F}_{2^b}|$. Therefore

$$\Pr(X_{z_1} = a_1 \wedge X_{z_2} = a_2) = \frac{1}{|\mathbb{F}_{2^b}|^2} = \Pr(X_{z_1} = a_1) \Pr(X_{z_2} = a_2),$$

as desired. □

To get bits from this, we can take for each z a fixed coordinate of X_z (e.g. the first bit in its binary representation) or more generally any nontrivial linear form of the field element. This gives us a distribution on $\{0,1\}^n$ that is pairwise independent. The sample space has size $|\mathbb{F}_{2^b}|^2 = n^2$, and the seed length needed to choose a random (α, β) is $2b = 2 \log n$.

So we have:

An explicit algorithm that converts $2 \log n$ truly random bits into n pairwise independent bits.

12.3.4 *Derandomizing Max-Cut via Pairwise Independence*

Recall that our analysis of the randomized Max-Cut algorithm only used the property

$$\Pr(x_u \neq x_v) = \frac{1}{2}$$

for each edge $\{u, v\}$, which holds whenever the x_v's are pairwise independent unbiased bits. Thus we can replace the fully independent distribution by the finite-field construction above without changing the analysis.

Concretely:

1. Fix an identification of vertices V with $\mathbb{F}_{2^b}$ where $n = |V| = 2^b$.
2. For each pair $(\alpha, \beta) \in \mathbb{F}_{2^b}^2$, generate the sequence $(X_z)_{z \in V}$ as above and then convert each X_z to a bit $x_z \in \{-1, +1\}$ (e.g. take the first bit and map $0 \mapsto -1$, $1 \mapsto +1$).

3. Use x_v as the color of vertex v and compute the resulting cut value.

Because there are only n^2 pairs (α, β), we can deterministically enumerate all of them in polynomial time and pick the best cut obtained. The expected cut value over (α, β) is still $W/2$, so there must be at least one pair giving a cut of weight at least $W/2$. Hence:

Theorem 12.6. *Max-Cut admits a deterministic $1/2$-approximation algorithm running in time $O(n^2 \cdot \mathrm{poly}(n))$ obtained by enumerating all seeds of the pairwise independent distribution.*

The high-level pattern here is important and will reappear:

> If the analysis of a randomized algorithm only uses pairwise independence, then we can often derandomize it by enumerating over an explicit pairwise independent distribution with small support.

12.4 The Method of Conditional Expectations

The technique we used above (enumerating all seeds) can be expensive if the support size is large. Another powerful derandomization method is the ***method of conditional expectations***.

Suppose we have some random process depending on independent random variables $X_1, \dots, X_n$, and some objective function $Y = Y(X_1, \dots, X_n)$ (e.g. the weight of a cut). If we can efficiently compute conditional expectations of the form

$$\mathbb{E}[Y \mid X_1 = x_1, \dots, X_i = x_i],$$

then we can deterministically choose the values of the X_i one by one so as to guarantee that Y is at least as large as its expectation.

12.4.1 The General Template

Let $X_1, \dots, X_n$ be independent unbiased bits (or ± 1 signs) and let Y be a random variable with $\mathbb{E}[Y] = \mu$.

Look at the first variable X_1. We can write

$$\mathbb{E}[Y] = \frac{1}{2}\mathbb{E}[Y \mid X_1 = -1] + \frac{1}{2}\mathbb{E}[Y \mid X_1 = +1].$$

Therefore at least one of the conditional expectations is at least μ:

$$\max\{\mathbb{E}[Y \mid X_1 = -1], \mathbb{E}[Y \mid X_1 = +1]\} \;\geq\; \mu.$$

We fix X_1 to be the value that achieves the maximum, say $X_1 = x_1^\star$, and continue.

Inductively, after we have fixed $X_1 = x_1^\star, \dots, X_{i-1} = x_{i-1}^\star$, we have some conditional expectation

$$\mu_{i-1} = \mathbb{E}[Y \mid X_1 = x_1^\star, \dots, X_{i-1} = x_{i-1}^\star].$$

For the next variable X_i we again have

$$\mu_{i-1} = \frac{1}{2}\mathbb{E}[Y \mid X_1 = x_1^\star, \dots, X_i = -1] + \frac{1}{2}\mathbb{E}[Y \mid X_1 = x_1^\star, \dots, X_i = +1].$$

Thus at least one of the two conditional expectations is at least μ_{i-1}. We set X_i to be that value and proceed.

By induction, at the end we obtain a deterministic assignment $(x_1^\star, \dots, x_n^\star)$ with

$$Y(x_1^\star, \dots, x_n^\star) \;\geq\; \mathbb{E}[Y \mid X_1 = x_1^\star, \dots, X_n = x_n^\star] \;\geq\; \mu_0 \;=\; \mathbb{E}[Y].$$

So we have deterministically found an outcome at least as good as the expected value of the randomized process.

The whole trick is that we need to be able to ***compute*** these conditional expectations efficiently.

12.4.2 *Applying MCE to Max-Cut*

Let us apply this to the same Max-Cut setup. Let $X_v \in \{-1, +1\}$ be the color of vertex v and let Y be the (random) weight of the cut induced by X. We know that

$$\mathbb{E}[Y] = \frac{1}{2}\sum_{e \in E} w_e.$$

We will color the vertices one by one. Suppose we have already assigned colors to vertices $1, \dots, i-1$, and we want to decide the color of vertex i. Condition on the choices so far; formally, we consider

$$\mathbb{E}[Y \mid X_1 = x_1, \dots, X_{i-1} = x_{i-1}, X_i = c]$$

for $c \in \{-1, +1\}$.

We can decompose the contribution of edges into three parts:

- Edges whose endpoints are both in $\{1, \dots, i-1\}$: their contribution to the cut is already completely determined and does not depend on X_i.
- Edges incident to vertex i, i.e. edges $\{i, j\}$. For neighbors $j < i$, the contribution of such an edge ***does*** depend on X_i, deterministically: if $X_j \neq X_i$ the edge is in the cut, otherwise it is not. For neighbors $j > i$, the vertex j is still random, so conditioned on $X_i = c$, X_j is uniform in $\{-1, +1\}$ and $\Pr(X_j \neq c) = 1/2$.

- Edges whose endpoints are both in $\{i+1, \dots, n\}$: neither endpoint has been colored yet, so each such edge contributes an expected $\frac{1}{2} w_e$ regardless of X_i.

Hence:

$$\begin{aligned}\mathbb{E}[Y \mid \text{choices up to } i-1, X_i = c] = {} & (\text{weight of already decided edges}) \\ & + \sum_{\substack{j<i \\ \{i,j\}\in E}} \mathbf{1}[c \neq X_j] \cdot w_{ij} \\ & + \frac{1}{2} \sum_{\substack{j>i \\ \{i,j\}\in E}} w_{ij} \\ & + \frac{1}{2} \sum_{\substack{e\in E \\ e\subseteq\{i+1,\dots,n\}}} w_e.\end{aligned}$$

The last two sums do not depend on c, so the difference between assigning $X_i = +1$ and $X_i = -1$ only comes from the edges between i and already-colored neighbors. Thus we can compute which choice of X_i gives the larger conditional expectation by looking only at the neighbors of i that have already been colored.

Concretely, define

$$A_i = \sum_{\substack{j<i \\ \{i,j\}\in E \\ X_j=-1}} w_{ij}, \qquad B_i = \sum_{\substack{j<i \\ \{i,j\}\in E \\ X_j=+1}} w_{ij}.$$

If we set $X_i = +1$, then the edges from i to neighbors with $X_j = -1$ contribute A_i to the cut, while if we set $X_i = -1$ we get contribution B_i. So we choose

$$X_i = \begin{cases} +1 & \text{if } A_i \geq B_i, \\ -1 & \text{otherwise.} \end{cases}$$

By the argument above, this choice guarantees that the conditional expectation never decreases as we assign more colors, so at the end we obtain a cut of weight at least $\mathbb{E}[Y] = \sum_e w_e/2$.

Remark 12.7. For Max-Cut, computing the conditional expectations is easy and leads to a very simple greedy algorithm. For other problems the conditional expectation can be much more difficult to compute explicitly.

12.5 *Randomness from Hardness*

We now switch viewpoint and ask: can we generate pseudorandomness from computational hardness assumptions?

The guiding cryptographic notion is that of a ***one-way function*** / ***one-way permutation***. Roughly, a one-way permutation is a permutation on $\{0,1\}^n$ that is easy to compute but hard to invert on a random input for any efficient algorithm.

12.5.1 One-Way Permutations

Fix a length n and let

$$f : \{0,1\}^n \to \{0,1\}^n$$

be a permutation.

Definition 12.8. For an (possibly randomized) algorithm A attempting to invert f, its ***success probability*** on inputs of length n is

$$\delta_A(n) \;=\; \Pr_{x \in \{0,1\}^n}[A(f(x)) = x].$$

We say f is ***one-way*** (or $s(n)$-***secure***) if for every probabilistic algorithm A running in time at most $s(n)$ (say, $s(n) = n^c$ for all c), the success probability $\delta_A(n)$ is negligible (e.g. $\delta_A(n) \le 1/s(n)$).

Loosely speaking:

> Any efficient algorithm has only a tiny probability of inverting f on a random input.

The ambitious goal is to show:

$$\text{existence of one-way permutations} \quad \Rightarrow \quad \text{existence of PRGs.}$$

Even more strongly, one can construct PRGs from ***any*** one-way function (not necessarily a permutation); this was shown by Håstad, Impagliazzo, Levin, and Luby (HILL, 1999).

We will focus on an intermediate step:

> Given a one-way permutation f, construct a single bit of output that looks random even given $f(x)$.

This bit is called a ***hard-core bit***. Chaining many such bits together gives a full-fledged PRG.

12.5.2 Hard-Core Bits and a Simple PRG

Let f be a one-way permutation on $\{0,1\}^n$. For $x, r \in \{0,1\}^n$ define the inner product

$$\langle x, r \rangle = \sum_{i=1}^{n} x_i r_i \pmod 2.$$

Consider the following mapping from $2n$ input bits (x, r) to $2n+1$ output bits:

$$G(x, r) = (f(x), r, \langle x, r \rangle).$$

If we knew that the last bit $\langle x, r\rangle$ is computationally unpredictable given $(f(x), r)$, then G would be a PRG with stretch 1:

- (x, r) uniform implies $(f(x), r)$ is uniform (since f is a permutation).
- If no efficient algorithm can distinguish $(f(x), r, \langle x, r\rangle)$ from $(f(x), r, U)$, where U is an independent uniform bit, then G fools all polynomial-time tests.

The bit $\langle x, r\rangle$ is called the ***hard-core bit***. Our main theorem shows that if this bit were efficiently predictable from $(f(x), r)$, then we could invert f with non-negligible probability, which contradicts the one-wayness of f.

We remark that if we wished to get a PRG with larger stretch, we could output $r, \langle x, r\rangle, \langle f(x), r\rangle, \langle f(2)(x), r\rangle, \ldots, \langle f^{(t)}(x), r\rangle$, where $f^{(i)}(x) = f(f^{(i-1)}(x))$ and $f^{(0)}(x) = x$. We refer the interested reader to an in-depth treatment in more standard texts on cryptography/complexity.

12.6 The Goldreich–Levin Theorem

Theorem 12.9 (Goldreich–Levin / Hidden Bit Theorem)**.** *Let f be a permutation on $\{0,1\}^n$. Suppose there is a probabilistic polynomial-time algorithm A and an $\epsilon > 0$ such that*

$$\Pr_{x,r\in\{0,1\}^n}[A(f(x), r) = \langle x, r\rangle] \;\geq\; \frac{1}{2} + \epsilon.$$

Then there is a probabilistic polynomial-time algorithm B that **inverts** *f on a non-negligible fraction of inputs:*

$$\Pr_{x\in\{0,1\}^n}[B(f(x)) = x] \;\geq\; \mathit{poly}(\epsilon).$$

In particular, if f is one-way then the predicate $b(x, r) = \langle x, r\rangle$ is hard to predict from $(f(x), r)$.

From now on, fix an x and write

$$H(r) = A(f(x), r).$$

For this ***unknown*** x, the assumption says

$$\Pr_r[H(r) = \langle x, r\rangle] \;\geq\; \frac{1}{2} + \epsilon.$$

Our goal is to reconstruct x given oracle access to H.

Warm-Up 0: Perfect Prediction

First, suppose $H(r) = \langle x, r\rangle$ for ***all*** r (no errors). Then we can recover x easily by querying H on the standard unit vectors $e_1, \ldots, e_n \in \{0,1\}^n$:

$$H(e_i) = \langle x, e_i\rangle = x_i.$$

So we get every bit of x exactly.

Warm-Up 1: High Success Probability

Now suppose

$$\Pr_r[H(r) = \langle x, r\rangle] \geq \frac{15}{16}.$$

We can no longer just query $H(e_i)$, because e_i might be one of the rare inputs on which H is wrong. Instead, use the identity

$$\langle x, r + e_i\rangle = \langle x, r\rangle \oplus \langle x, e_i\rangle = \langle x, r\rangle \oplus x_i.$$

(Here $+$ and $\oplus$ are bitwise XOR.)

Given a random r, query $H(r)$ and $H(r + e_i)$ and compute

$$\widetilde{x}_i(r) = H(r + e_i) \oplus H(r).$$

If both $H(r)$ and $H(r + e_i)$ are correct then $\widetilde{x}_i(r) = x_i$.

By a union bound,

$$\Pr_r[\text{both correct}] \geq 1 - \Pr[H(r) \text{ wrong}] - \Pr[H(r + e_i) \text{ wrong}] \geq 1 - \frac{1}{16} - \frac{1}{16} = \frac{7}{8}.$$

Thus a single trial gives the right value of x_i with probability at least $7/8$.

Repeating this $O(\log n)$ times independently and taking the majority gives x_i correctly with probability $1 - 1/\text{poly}(n)$ by Chernoff bounds. Applying a union bound over $i = 1, \ldots, n$, we get all bits of x correctly with high probability using a polynomial number of queries to H.

The Barrier: Only a Small Advantage

In the real theorem we only know that H predicts $\langle x, r\rangle$ with advantage ϵ over $1/2$, i.e.

$$\Pr_r[H(r) = \langle x, r\rangle] = \frac{1}{2} + \epsilon.$$

Trying the same trick as above, we see that

$$\Pr_r[\widetilde{x}_i(r) = x_i] = \Pr[\text{both correct}] + \Pr[\text{both wrong}].$$

We can bound

$$\Pr[\text{both correct}] \geq 1 - 2 \cdot \Pr[H(r) \text{ wrong}] = 1 - 2\left(\frac{1}{2} - \epsilon\right) = 2\epsilon,$$

but now ϵ might be tiny, and Pr[both wrong] could be of the same order. In fact, one can check that this strategy yields at best an advantage $O(\epsilon)$, which is too small to give a good probability of recovery even after polynomially many repetitions.

We need a new idea.

12.6.1 *A List-Decoding Viewpoint*

The key is to step back and think of the following more general "learning" problem:

Lemma 12.10 (Goldreich–Levin List-Decoding Lemma)**.** *Suppose $H : \{0,1\}^n \to \{0,1\}$ is a function such that there exists some (unknown) $x \in \{0,1\}^n$ with*

$$\Pr_r[H(r) = \langle x, r\rangle] \;\geq\; \frac{1}{2} + \epsilon.$$

Then there is a probabilistic algorithm L^H running in time $\text{poly}(n, 1/\epsilon)$ that outputs a list L of at most $O(1/\epsilon^2)$ strings in $\{0,1\}^n$ such that $x \in L$ with probability at least, say, $2/3$.

We do not assume uniqueness: there might be many different x's that correlate with H; the algorithm outputs a short list containing all of them (up to small probability of failure).

Once we have this lemma, we can use it to invert f as follows. Given $y = f(x)$:

1. Define $H_y(r) = A(y, r)$. By assumption, for a non-negligible fraction of x's (the "good" ones) we have $\Pr_r[H_{f(x)}(r) = \langle x, r\rangle] \geq 1/2 + \epsilon$.
2. Run the list-decoding algorithm with oracle H_y to obtain a list L_y of candidates.
3. Output any $x' \in L_y$ such that $f(x') = y$.

Whenever x is good and the list-decoding succeeds, we recover x. A careful analysis shows that this inverts f with probability $\text{poly}(\epsilon)$, contradicting one-wayness. So it remains to prove the lemma.

12.6.2 *Attempt 1: Guessing Inner Products on Random Samples*

Pick k random strings $r_1, \ldots, r_k \in \{0,1\}^n$ independently and uniformly. For the moment, imagine that we magically know the k bits

$$b_j = \langle x, r_j\rangle \qquad (j = 1, \ldots, k).$$

There are 2^k possibilities for $(b_1, \ldots, b_k)$; one of them is correct.

Step 1: Boosting a Weak Predictor. Using these k samples and their (assumed) correct labels b_j, we can boost the success probability of H from $1/2 + \epsilon$ up to, say, $31/32$.

For each j define the shifted function

$$G_j(r) = H(r + r_j) \oplus b_j.$$

If $H(r + r_j)$ happens to be equal to $\langle x, r + r_j \rangle$, then

$$G_j(r) = \langle x, r + r_j \rangle \oplus b_j = \langle x, r \rangle \oplus \langle x, r_j \rangle \oplus b_j = \langle x, r \rangle,$$

since $b_j = \langle x, r_j \rangle$. Thus each G_j is also a predictor for $\langle x, r \rangle$ with the same advantage ϵ:

$$\Pr_r[G_j(r) = \langle x, r \rangle] = \frac{1}{2} + \epsilon.$$

Now consider

$$G(r) = \text{majority}\{G_j(r) \mid j = 1, \ldots, k\}.$$

Claim 12.11. If the r_j's are independent and $k = \Theta(1/\epsilon^2)$, then

$$\Pr_r[G(r) = \langle x, r \rangle] \;\geq\; \frac{31}{32}.$$

Proof sketch. Let $Z_j(r)$ be the indicator of the event $G_j(r) = \langle x, r \rangle$. Then $\mathbb{E}[Z_j] = 1/2 + \epsilon$ and the Z_j's are independent over j (for fixed r). Let $S(r) = \sum_j Z_j(r)$. Then $\mathbb{E}[S] = k(1/2 + \epsilon)$ and $G(r)$ is correct iff $S(r) > k/2$.

By Chebyshev's inequality,

$$\Pr\big[S \leq k/2\big] \;\leq\; \frac{\mathrm{Var}(S)}{(\mathbb{E}S - k/2)^2} \;\leq\; \frac{k}{(k\epsilon)^2} = \frac{1}{k\epsilon^2}.$$

Taking $k \geq 32/\epsilon^2$ makes this probability at most $1/32$, so G is correct with probability at least $31/32$. □

Thus, given the correct labels b_j, we can build a very strong predictor G for $\langle x, r \rangle$ and then revert to the "high probability" warm-up (Version 1) to recover x with high probability.

Step 2: Guessing the Labels. Of course we do not know the b_j's. But we can simply try all possibilities:

- For each k-bit string $(\hat{b}_1, \ldots, \hat{b}_k) \in \{0,1\}^k$, pretend that these are the correct labels, construct a corresponding predictor $\hat{G}$, and run the Version 1 inversion procedure to obtain a candidate vector $\hat{x}$.
- Output the list of all candidates obtained.

One of the 2^k guesses for $(\hat{b}_1, \ldots, \hat{b}_k)$ is correct, and the corresponding candidate $\hat{x}$ equals the true x with high probability. Hence x appears in the list.

The problem is that to get the Chebyshev bound we needed $k = \Theta(1/\epsilon^2)$, so the list size $2^k = 2^{\Theta(1/\epsilon^2)}$ is enormous (and the running time is the same). We need to keep k small enough that 2^k is polynomial in $1/\epsilon$.

12.6.3 Attempt 2: Using Pairwise Independence

The fix is to notice that in the claim above we only used pairwise independence of the Z_j's (to bound $\mathrm{Var}(S)$), not full independence. This means it suffices that the underlying r_j's are pairwise independent.

So instead of taking k independent samples, we will generate k pairwise independent ones from a much smaller random seed.

Step 1: A Small Basis. Pick a small number $k' = \Theta(\log(1/\epsilon))$ of independent uniform vectors $s_1, \ldots, s_{k'} \in \{0,1\}^n$. For every subset $S \subseteq \{1, \ldots, k'\}$ define

$$r_S = \bigoplus_{j \in S} s_j.$$

This gives $k = 2^{k'}$ vectors r_S. It can be checked that the family $\{r_S\}$ is pairwise independent (and each r_S is uniform). Thus the k corresponding indicators $Z_S(r)$ in the Chebyshev argument are pairwise independent, and the same concentration bound applies with $k = 2^{k'} = \Theta(1/\epsilon^2)$.

Step 2: Guessing Only the Basis Labels. Instead of guessing all $b_S = \langle x, r_S \rangle$ independently, we only guess the labels on the basis vectors:

$$b_j = \langle x, s_j \rangle, \qquad j = 1, \ldots, k'.$$

There are now only $2^{k'} = \mathrm{poly}(1/\epsilon)$ possibilities. For any particular guess $(\hat{b}_1, \ldots, \hat{b}_{k'})$, we can compute for every subset S

$$\hat{b}_S = \bigoplus_{j \in S} \hat{b}_j,$$

which is our guess for $\langle x, r_S \rangle$ (by linearity of the inner product). If the basis labels are guessed correctly, then all $\hat{b}_S$ are correct, and we are exactly in the setting of Attempt 1 with the r_S's and their true labels.

We now repeat the same construction:

- For each guess of the k' basis labels, construct the boosted predictor $\hat{G}$ using the pairwise independent family $\{r_S\}$.

- Use the Version 1 inversion procedure on $\hat{G}$ to obtain a candidate $\hat{x}$.

The number of guesses is $2^{k'} = \text{poly}(1/\epsilon)$, and by the Chebyshev argument the boosted predictor is correct with probability at least $31/32$ when the basis labels are correct. Hence with probability at least $2/3$ we obtain a list of size $\text{poly}(1/\epsilon)$ that contains x. This proves the list-decoding lemma and completes the proof of the Goldreich–Levin theorem.

12.7 Summary

- **Pseudorandom Generators (PRGs):** Convert short random seeds into long strings that look random to any algorithm in a restricted class (e.g. polynomial-time algorithms). If we had PRGs with logarithmic seed length for all polytime algorithms, we would have $\mathbf{BPP} = \mathbf{P}$.
- **Pairwise Independence:** We can explicitly construct pairwise independent distributions over $\{0,1\}^n$ using finite fields and a $2\log n$-bit seed. If the analysis of a randomized algorithm only uses pairwise independence, we can derandomize it by enumerating all seeds.
- **Method of Conditional Expectations:** Starting from a randomized algorithm with a good expected value, we can deterministically fix the random choices one by one by always choosing the value that keeps the conditional expectation high, obtaining a deterministic algorithm with performance at least as good as the expectation.
- **Randomness from Hardness:** Assuming one-way permutations (or more generally one-way functions) exist, we can construct hard-core bits and then PRGs whose outputs are indistinguishable from random by any efficient adversary.
- **Goldreich–Levin Theorem:** The inner product $\langle x, r\rangle$ modulo 2 is a hard-core bit for any one-way permutation f: if an algorithm can predict it from $(f(x), r)$ with advantage ϵ, then we can invert f with probability $\text{poly}(\epsilon)$. The proof combines boosting, list decoding, and pairwise independence.