CS 252, Lecture 13: Pseudorandomness: Hardness vs Randomness

1 Introduction

So far in the course, we have seen many useful randomized algorithms. All of them need uniformly random bits–how do we obtain these random bits? There are various practical ways to do this, such as using a last digit of a high precision computer clock, using a function of CPU clocks etc. An issue with these methods is that there is no way to certify the random bits that we obtain. They often have correlations among the bits, which can impact the correctness or running time of the randomized algorithms. In this lecture, we look at this issue and study how to obtain good random bits from perfectly random bits.

Generating good random bits is essential to cryptography as well. For example, one time padding requires a random string of length equal to the message length, and this random string cannot be reused. Can we somehow reduce this randomness requirement? In general, can we generate more random bits from a small number of random bits? From an information theory perspective, Shannon proved that such an absolute function $f : \{0,1\}^s \to \{0,1\}^n$ cannot generate uniformly random bits (or even some approximation of it) from a uniformly random s bits. Intuitively, the "entropy" of uniform distribution on s bits is less than that on $n > s$ bits, and no function can increase entropy by itself.

Instead, we study this problem from a computational perspective: what if we want to generate $n > s$ bits that are not necessarily uniformly random, but look random to any adversary with limited computational power? This key concept is called as "pseudorandomness" where in we caliberate a distribution as uniform from an observer's view or put in other words, "randomness in the eye of the beholder".

2 Computational Indistinguishability

Suppose that D_1 and D_2 are probability distributions on $\{0,1\}^n$. We can view them as for every $x \in \{0,1\}^n$, $D_1(x)$ (and respectively $D_2(x)$) denote the probability of x in the distribution D_1 (and respectively D_2). We say that D_1 and D_2 are *statistically ϵ-close* if for every function $T : \{0,1\}^n \to \{0,1\}$,

$$|\Pr_{x\sim D_1}[T(x) = 1] - \Pr_{x\sim D_2}[T(x) = 1]| \leq \epsilon$$

Definition 1. (Total Variation Distance) For a pair of distributions D_1, D_2 on $\{0,1\}^n$, we define the total variation distance $\Delta_{TV}(D_1, D_2)$ as the maximum distinguishability between the distributions over all possible functions $T : \{0,1\}^n \to \{0,1\}$.

$$\Delta_{TV}(D_1, D_2) = \max_T |\Pr_{x\sim D_1}[T(x) = 1] - \Pr_{x\sim D_2}[T(x) = 1]|$$

Exercise. Prove that $\Delta_{TV}(D_1, D_2) = \frac{1}{2} \|D_1 - D_2\|_1 = \frac{1}{2} \sum_{x \in \{0,1\}^n} |D_1(x) - D_2(x)|$.

Note that for any pair of probability distributions D_1 and D_2, $\Delta_{TV}(D_1, D_2) \in [0, 1]$.

Now, we formally define computational indistinguishability.

Definition 2. (Computational Indistinguishability) A pair of distributions D_1 and D_2 on $\{0,1\}^n$ are said to be (C, ϵ)-computationally indistinguishable if for every machine A of complexity $C(n)$ cannot distinguish between D_1 and D_2 better than ϵ i.e.

$$|\Pr_{x \sim D_1}[A(x) = 1] - \Pr_{x \sim D_2}[A(x) = 1]| \leq \epsilon$$

In the definition, we have intentionally used a vague definition for "machine" and "complexity" since it can be instantiated with more concrete settings such as Turing machines with complexity being the running time, Boolean circuits with complexity being the size of the circuits etc.

A distribution D on $\{0,1\}^n$ is said to be (C, ϵ)-pseudorandom if any machine with complexity $C(n)$ is fed with this distribution instead of the uniform distribution, the acceptance probability changes by at most ϵ.

Definition 3. ((C, ϵ)-pseudorandom) A distribution D over $\{0,1\}^n$ is said to be (C, ϵ)-pseudorandom if the uniform distribution $\mathcal{U}_n$ on $\{0,1\}^n$ and D are (C, ϵ)-computationally indistinguishable.

3 Psedudorandom generators

How to produce pseudorandom distributions? The starting point is usually a small number of perfectly random bits, known as *seed*.

Definition 4. (Pseudorandom generators) A function $G : \{0,1\}^{s(n)} \to \{0,1\}^n$ is said to be (C, ϵ)-pseudorandom generator (PRG) with seed length $s(n)$ if $G(\mathcal{U}_k)$ is (C, ϵ)-pseudorandom.

Informally, a PRG stretches a perfectly random string into a longer "random looking" string. As we have mentioned earlier, obtaining a PRG that fools all possible functions without any limitations on the complexity $C(n)$ is statistically impossible, even with $s = n - 1$. Computationally however, it is possible, for eg, with respect to Boolean circuits, polynomial time algorithm etc. We generally seek $s(n) << n$, such as $s(n) = n^\epsilon$ or even $O(\log n)$.

The key motivation for defining PRGs is to use them to derandomize randomized algorithms. We first start with a very small seed, generate a longer random string that we require and use it instead of uniform bits for the algorithm. Since the algorithm cannot distinguish this random distribution from the uniform distribution, the algorithm still works fine. Since we are using much smaller number of random bits, we can just iterate over all possible choices for the seed, and pick the majority result. This gives us a deterministic algorithm that in fact is as good as the original randomized algorithm. We formalize this in the exercise below:

Exercise. (Derandomizing from PRG) If G is a $(C(n), \frac{1}{10})$-PRG with seed length s and can be computed in $2^{O(s)}$ time, then a randomized algorithm with time complexity $C(n)$ and success probability $\frac{2}{3}$ can be derandomized to run in time $2^{O(s)}C(n)$.

4 Hardness vs Randomness

Suppose that we want to construct a pseudorandom generator that expands stretch by just a bit i.e. $G : \{0,1\}^s \to \{0,1\}^{s+1}$. We can achieve this as follows: given an input $x \in \{0,1\}^s$, we output $s+1$ bits $(x, g(x))$ where $g : \{0,1\}^s \to \{0,1\}$ is a "hard" function. Here, hardness of g is in terms of *average case* hardness. Being average case hard for a computational class of complexity $C(n)$ and a small constant ϵ means that for every machine A in this computational class, A cannot compute g even slightly better than random guessing i.e. $\Pr_{x \in \{0,1\}^s}(A(x) = g(x)) \leq \frac{1}{2} + \epsilon$. We contrast this average case hardness with worst case hardness: in the traditional worst case hardness, we say g is hard if every A in the computational class cannot compute g on some input, but in the average case hardness, such an A cannot compute g even on average i.e. just doing better than random guessing over all the input strings is hard.

Now, given such a hard function g, we claim that the output of $G : x \in \{0,1\}^s \to (x, g(x)) \in \{0,1\}^{s+1}$ is (C, ϵ)-pseudorandom. Intuitively, the reasoning is as follows: the first s bits are anyway uniformly random, and for the class C, the last bit appears to be uniform given the first s bits as well. This is discussed more formally in the homework.

The above argument is a key link between "hardness" and "randomness": we have shown how to use an average case hard function to obtain pseudorandom distributions. But there is a catch: if g is supposed to be very hard, how can G itself compute g in the first place? There are two possible avenues to fix this issue:

1. Allow G more time than the tests it fools. This is okay for derandomization, but is not useful for cryptographic purposes.

2. Modify the above PRG construction to create asymmetry that gives G an edge compared to the tests it fools.

We now look at the first avenue in more detail.

4.1 Nisan Wigderson PRG Construction

Even though allowing more time for the PRG is a good idea for derandomization purposes, there is still another issue with the above construction: we are increasing the stretch by only a single bit. One could modify the above construction to output two extra bits by simply partitioning the input into two parts, and applying an average case function g on the first $\frac{s}{2}$ bits, and then similarly applying g on the last $\frac{s}{2}$ bits, and concatenating the resulting two bits with the original s bits. We can extend this to output k bits by partitioning $\{1, 2, \ldots, s\}$ into k parts and applying an average case hard function to each part. However, this method cannot get our output beyond $2s$ bits. In order to do so, we need to use the average case hard function more cleverly, and the way we do this is by applying it to a carefully choosen subsets of $\{1, 2, \ldots, s\}$.

In the Nisan-Wigderson PRG construction, we pick subsets $S_1, S_2, \ldots, S_n \in 2^{[s]}$[1] of cardinality m each. Note that we do not want these subsets to intersect in many places since the bits will no longer be

[1]The set $\{1, 2, \ldots, s\}$ is often represented as $[s]$.

pseudorandom. We achieve this by picking subsets such that for every $i, j \in [n], |S_i \cap S_j| \leq l \leq \frac{m}{100}$. Such a set system of subsets from $[s]$, where all the sets have cardinality m, and any pair of subsets intersect in at most l elements is known as a (s, m, l)-*combinatorial design*. They are well studied objects, and for the above setting of parameters, we have set families with $n = 2^{\Omega(s)}$ exponential number of subsets.

In fact, we can relate these combinatorial designs with the error correcting codes that we have studied in the previous lecture: first represent every subset $S_i \subseteq [s]$ by $\{0,1\}^s$ with 1s in locations $j : j \in S_i$ and 0s elsewhere. Now, we are looking at picking a large number of vectors in $\{0,1\}^s$ such that each of them have exactly m 1s, and every pair of them differ in at least $2(m-l)$ bits. This is precisely a question of picking codes with large distance, and large rate which we can achieve, both using randomized constructions and also efficient explicit concatenation techniques.

After picking these subsets, the fact that any pair of subsets intersect in small number of elements suffices to argue about the pseudorandomness of the output. Note that the output of this approach is exponentially larger than the input size s. Furthermore, there are "worst case to average case" hardness amplification results where in we prove the existence of average case hard function using the existence of a worst case hard function. Combining this with the Nisan Wigderson construction, we can actually prove that if there are problems that can be solved in exponential time $2^{O(n)}$ but also require circuits of size $2^{\Omega(n)}$ (which is widely believed to be true) then every randomized algorithm running in polynomial time can be derandomized to run in **P**. This is a strong evidence to the fact that in fact, randomness might not make a difference for polynomial time algorithms.

Now, we take a look at a different approach to obtaining PRGs.

4.2 Using one way permutations

Previously, we had $G(x) = (x, g(x))$ where g is a hard function to compute. In stead, we can keep an easy function g but modify the input x so that there is no way to recover it from the scrambled output. To be precise, we set $G(x) = (\pi(x), h(x))$ where $\pi : \{0,1\}^s \to \{0,1\}^s$ permutes the input bits. (We interchangably use π as a permutation $\pi : [s] \to [s]$ and also as a function on s Boolean inputs that outputs s bits.) The property that we require of π is that it should be a *one-way* permutation: it is easy to apply, and hard to invert i.e. given $y \in \{0,1\}^s$, it is hard to find $x \in \{0,1\}^s$ such that $\pi(x) = y$. In fact, we also need that it is hard to predict $h(x)$ given $y = \pi(x) \in \{0,1\}^s$. Such a Boolean function $h : \{0,1\}^s \to \{0,1\}$ is called as a *hardcore predicate* for the one way permutation π. Note that $h(x)$ being hard to predict given y implies that $(y, h(x))$ is a pseudorandom distribution.

We now discuss some examples of hardcore predicates:

1. The least significant bit (lsb) function is a hardcore predicate for the RSA function $\pi(x) = x^e \mod N$.
2. The function $\text{half}_{p-1}(x) = \begin{cases} 1, & \text{if} x \in \{0, 1, \ldots, \frac{p-1}{2}\} \\ 0 & \text{otherwise.} \end{cases}$ is a hardcore predicate for the modular exponentiation function $\pi(x) = g^x \mod p$ for some prime p. For this example, lsb function is not a hardcore predicate.

Going from one extra bit to polynomially many extra bits can be done in a straightforward manner: we first start with $x \in \{0,1\}^s$, compute $x_1 = \pi(x), b_1 = h(x)$, and then $x_2 = \pi(x_1), b_2 = h(x_1)$ and so on. Finally, we can append x with all the b_is to obtain a pseudorandom distribution. This helps in expanding a stretch of n^ϵ bits to obtain n pseudorandom bits.