Instructor: Andrej Bogdanov

Notes by: Andrej Bogdanov

1 Samplable distributions

In the last lecture we saw that there is an ensemble of computable distributions ν such that the distributional version (BH, ν) of the bounded halting problem is complete for the distributional class $(\mathrm{NP}, \mathrm{PComp})$ with respect to average polynomial-time reductions.

The reason we decided to work with polynomial-time computable ensembles is because this was a rich enough class that includes several "interesting" ensembles, in particular the uniform one, but also the ensemble ν that defines our complete distributional problem. However the class PComp happens to leave out (under reasonable complexity assumptions) an important class of ensembles that arise out naturally in the study of efficient computation and cryptography.

To explain what these ensembles look like let us go back to our original motivation for the study of average-case complexity: To determine the complexity of instances of problems that occur "in practice" as opposed to ones that might be artificially contrived. What is a reasonable model for instances "in practice"? From the standpoint of computational complexity, if these instances are to be found anywhere they must have been generated by some computationally efficient entity (be it a machine, nature, or a person), possibly with a source of randomness at its disposal.

A randomized algorithm that generates candidate instances for our average-case algorithm is called a sampler. Distributions that describe the output of a sampler are called samplable.

Definition 1. *An ensemble of distributions μ is* polynomial-time samplable *if there is an expected polynomial-time randomized algorithm S that, on inputs of the form 1^n, outputs each x in $\{0,1\}^n$ with probability exactly $\mu_n(x)$.*

We denote the class of polynomial-time samplable ensembles by PSamp, and the class of corresponding distributional problems in NP by $(\mathrm{NP}, \mathrm{PSamp})$.

Every polynomial-time computable distribution is also polynomial-time samplable. However, the converse is not true unless $\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}$, and in this lecture we will see an important example of ensembles that are polynomial-time samplable but unlikely to be polynomial-time computable.

Does the distributional class $(\mathrm{NP}, \mathrm{PSamp})$ contain harder problems than $(\mathrm{NP}, \mathrm{PComp})$? Not so if we allow the use of randomized algorithms or circuits. We won't give formal definitions of these computational models in the average case. Impagliazzo and Levin show that if (BH, ν) has randomized heuristics, then every problem in $(\mathrm{NP}, \mathrm{PSamp})$ also has randomized heuristics. They

also show that if (BH, ν) has randomized average polynomial-time algorithms, then every problem in $(\mathrm{NP}, \mathrm{PSAMP})$ also has randomized average polynomial-time algorithms.

2 One-way functions

So far we have only looked at decision problems in distributional NP. Just as in worst-case complexity, we can also consider search problems. An $(\mathrm{NP}, \mathrm{PSAMP})$ search problem is specified by an NP-relation R and a polynomial-time samplable ensemble μ. For "worst-case" NP, we saw that if P = NP then we can also solve all NP-search problems. An analogous statement holds in the average-case setting: For every $(\mathrm{NP}, \mathrm{PSAMP})$ search problem (R, μ) there is a decision problem $(L, \mu') \in (\mathrm{NP}, \mathrm{PSAMP})$ so that if (L, μ') has efficient randomized heuristics, so does (R, μ). The same holds for randomized average polynomial-time algorithms.

There is a special kind of search problem in distributional NP that plays a central role in the theory of cryptography. This is the problem of inverting a candidate "one-way" function.

Definition 2. *A family of functions $f_n : \{0,1\}^n \to \{0,1\}^n$ is* one-way *if the following two conditions hold:*

- *There is a polynomial-time algorithm that on input x of length n outputs $f_n(x)$.*
- *For every polynomial-size circuit family $C_1, C_2, \ldots$, every polynomial p, and large enough n,*

$$\Pr[f_n(C_n(f_n(X))) = f_n(X)] < 1/p(n),$$

where the probability is over X chosen uniformly from the set $\{0,1\}^n$.

What does this mean? Let us think of the family $\{f_n\}$ as a single function f from $\{0,1\}^*$ to $\{0,1\}^*$. The first condition says that f can be computed in polynomial time. To understand the second condition, let us imagine the following scenario: Suppose I hold some "secret" value x, I give you the value $y = f(x)$, and ask you to guess what my secret is. This might be difficult for you to do for the simple reason that there are many possible secrets x' that map to the same y. So instead of requiring you to guess my secret x, you win if you can come up with any x' such that $f(x') = y$. The second condition says that if my secret x was random, and if your secret-guessing procedure is efficient (in this definition, efficient means it can be implemented as a polynomial-size circuit family), then you cannot succeed in guessing x' with *any* inverse-polynomial probability. So the function f hides the secret x in a very strong sense.

The second condition in the definition of "one-way" in particular says that the pair (R, μ), where R is the NP-search problem "Given y, find x such that $f_n(x) = y$" and μ_n is the distribution $f(X), X \sim \{0,1\}^n$ does not have heuristic search algorithms. So to have one-way functions, it must at least be the case that $(\mathrm{NP}, \mathrm{PSAMP})$ has problems that are intractable for heuristic search.

There are several examples of function families that are believed to be one way. One example is *integer multiplication*:

$$f_n(x_1, y_1, \ldots, x_m, y_m) = (x_1 y_1, \ldots, x_m y_m)$$

where $x_1, y_1, \ldots, x_m, y_m$ are integers represented by strings of $n^{1/3}/2$ bits each, $m = n^{2/3}$, and $x_i y_i$ is the product of x_i and y_i.

3 Pseudo-random generators

Suppose you hold a box that produces random samples of length n from either the distribution μ_n, or the uniform distribution. Can you tell which is the case?

Of course you can never know for sure: Maybe the box produces a sample that is uniformly random most of the time, except that with some very small probability it outputs the string 1^n. In this case, the distribution μ_n is very "close" to the uniform distribution; the interesting question to ask is what happens when μ_n is sufficiently "far" from uniform.

To make this precise we need to define a notion of "distance" between distributions.

Definition 3. *The statistical distance between two probability distributions μ_n and ν_n on $\{0,1\}^n$ is the quantity.*

$$\frac{1}{2} \cdot \sum_{x \in \{0,1\}^n} |\mu_n(x) - \nu_n(x)|.$$

This is always a number in the range $[0, 1]$. It equals zero when the two distributions are the same: Given a sample x, there is no way of telling which distribution it came from. It equals one when μ_n and ν_n have disjoint support: Given any sample x, there is only one distribution that it could have come from.

These two extremes illustrate a universal principle: The statistical distance between μ_n and ν_n is ϵ if and only if there is a "test" that can tell whether a sample came from μ_n or from ν_n with confidence at least ϵ.

Lemma 4. *The statistical distance between μ_n and ν_n equals ϵ if and only if there exists a function $T : \{0,1\}^n \to \{0,1\}$ such that*

$$\left|\Pr_{X \sim \mu_n}[T(X) = 1] - \Pr_{X \sim \nu_n}[T(X) = 1]\right| = \epsilon.$$

The function T here plays of the role of a "statistical test": It takes a sample x and classifies it as originating either from μ_n ($T(x) = 1$) or from ν_n ($T(x) = 0$). The lemma implies that if μ_n and ν_n are far in statistical distance, then T has a good chance of telling them apart.

Does this intuition carry over to the computationally efficient setting? More precisely, suppose the above box again generates samples either from μ_n or from the uniform distribution, but now μ_n is efficiently samplable. Is there an *efficient* test that can tell which of these is the case?

It turns out that if one-way functions exist, then the answer is "no", even if μ_n is very far from uniform in statistical distance. To prove this we introduce an important object, the pseudorandom generator.

Definition 5. *A family of functions $G_n : \{0,1\}^n \to \{0,1\}^{m(n)}$, where $m(n) > n$, is a pseudorandom generator (against polynomial-size circuits) if the following two conditions hold:*

- *There is a polynomial-time algorithm that, on input x of length n, outputs $G_n(x)$.*
- *(Computational indistinguishability) For every polynomial-size circuit family $C_1, \ldots,$ every polynomial p and sufficiently large n,*

$$\left|\Pr_{X\sim\{0,1\}^n}[C_{m(n)}(G_n(X)) = 1] - \Pr_{Y\sim\{0,1\}^{m(n)}}[C_{m(n)}(Y) = 1]\right| < 1/p(n).$$

The larger the output length $m(n)$ is, the harder it is to construct a pseudorandom generator, but even the case $m(n) = n + 1$ is very interesting.

The second condition says this: Take a random n-bit string, apply G_n to it, and call the resulting distribution μ_m. Let ν_m be the uniform distribution on m bits. An efficient algorithm cannot tell whether a sample came from μ_m or ν_m. However, at least half the samples y of ν_m are not even in the range of G_n, so the statistical distance between μ_m and ν_m is at least $1/2$.

This type of pseudorandom generator is sometimes called a Blum-Micali-Yao (BMY) generator. There are two important differences between this kind of pseudorandom generator and the Nisan-Wigderson (NW) generator we saw in lecture 9.

To explain the first difference, let us recall how we used the NW generator. The application of the NW generator we looked at can be roughly described as follows: Someone gives you a polynomial-size circuit that takes randomness, and the NW generator provides a list of random-looking strings for this circuit that can be enumerated in polynomial time. One subtle point is that *the running time of the NW generator can (and will) be larger than the size of the circuit it is targeting*; e.g. to fool circuits of size n^2, the running time of the NW generator may be n^5. In contrast, the BMY generator is a single polynomial-time computable function that must fool *all* polynomial-size circuits, so it will need to be effective even against circuits whose size far exceed its running time.

The second difference is closely related to the first: In order to look pseudorandom against all polynomial-size circuits, the BMY generator must have subexponential output length $m(n)$. A circuit that stores a table of all possible 2^n inputs of the generator will distinguish the output of a generator from a random string. This circuit has size $O(m(n) \cdot 2^n)$, so if the generator is pseudorandom against all circuits of size $\text{poly}(m)$ it must be that $m(n) = 2^{o(n)}$. In contrast, for the NW generator we looked at a setting of parameters where $m(n) = 2^{\Omega(n)}$.

For this reason, the BMY generator is not as effective in derandomization as the NW generator. However it has many other applications in complexity theory and cryptography, some of which we will see in the next few lectures.

Lemma 6. *Every pseudorandom generator is a one-way function.*

Proof. Suppose we have a family of circuits $I_1, \ldots$ that inverts G_n as a family of one-way functions, namely

$$\Pr_{X\sim\{0,1\}^n}[G_n(I_m(G_n(X))) = G_n(X)] \geq 1/p(n)$$

consider the following "distinguisher" D_m: On input y, output 1 when $G_n(I_m(y)) = y$. Then

$$\Pr_{X\sim\{0,1\}^n}[D_m(G_n(X))) = 1] \geq 1/p(n). \tag{1}$$

Now consider a y such that $D_m(y) = 1$. This y is a possible output of G_n, so

$$\Pr_{X\sim\{0,1\}^n}[G_n(X) = y] \geq 2^{-n} = 2^{m-n} \Pr_{Y\sim\{0,1\}^m}[Y = y]$$

Summing over all such y, we have

$$\Pr_{X\sim\{0,1\}^n}[D_m(G_n(X)) = 1] \geq 2^{m-n} \Pr_{Y\sim\{0,1\}^m}[I_m(Y) = 1] \tag{2}$$

From (1) and (2) we have

$$\Pr_{X\sim\{0,1\}^n}[D_m(G_n(X)) = 1] - \Pr_{Y\sim\{0,1\}^m}[D_m(Y) = 1] \geq (1 - 2^{-m+n})/p(n). \qquad \square$$

The opposite is not true in general, but

Theorem 7. *If one-way functions exist, then for every polynomial $m(n) > n$, there exist pseudorandom generators with output length $m(n)$.*

Next time we will prove a weaker version of this.

The distribution μ_m obtained by evaluating a pseudorandom generator G_n on a uniformly random input x is clearly polynomial-time samplable, but it it cannot be polynomial-time computable.