Notes for Lecture 13

Today we introduce Levin's theory of average-case complexity.

This theory is still in its infancy: in this chapter we will introduce the notion of "distributional problems," discuss various formalizations of the notion of "algorithms that are efficient on average," introduce a reducibility that preserves efficient average-case solvability, and finally prove that there is a problem that is complete for the distributional version of **NP** under such reductions. It is still an open question how to apply this theory to the study of natural distributional problems that are believed to be hard on average.

1 Distributional Problems

Definition 1 (Distributional Problem) *A* distributional problem *is a pair* $\langle L, \mu \rangle$*, where* L *is a decision problem and* μ *is a distribution over the set* $\{0,1\}^*$ *of possible inputs.*

In other settings where average-case hardness is considered (e.g. in the study of one-way functions) we normally describe the distribution of inputs as a collection of distributions $\mu_1, \ldots, \mu_n, \ldots$ where μ_n is a distribution over the set $\{0,1\}^n$ of inputs of a given input length.[1] There are various reasons why this single-distribution approach is convenient for the purposes of this chapter. We will discuss it again later, but for now the basic intuition is that we will discuss reductions where the length of the output is not a function of the length of the input, so that sampling inputs from a fixed-length distribution and passing them to a reduction does not produce a fixed-length distribution (unlike the case of cryptographic reductions).

We will restrict to the study of distributional problems where μ is "polynomial-time computable." What do we mean by that? Fo all $x \in \{0,1\}^*$, let

$$\mu(x) = \sum_{y \leq x} \Pr[y]. \tag{1}$$

where '$\leq$' denotes lexicographic ordering. Then μ must be computable in $\operatorname{poly}(|x|)$ time. Clearly this notion is at least as strong as the requirement that $\Pr[x]$ be computable in polynomial time, because

$$\Pr[x] = \mu'(x) = \mu(x) - \mu(x-1), \tag{2}$$

$x - 1$ being the lexicographic predecessor of x. Indeed one can show that, under reasonable assumptions, there exist distributions that are efficiently computable in the second sense but not polynomial-time computable in our sense.

We can define the "uniform distribution" to be

[1] One can always reduce the approach of distinct distributions to the approach of this chapter by assuming that μ first picks at random a certain input length n, and then it samples from μ_n.

$$\Pr[x] = \frac{1}{|x|(|x|+1)} 2^{-|x|}; \tag{3}$$

that is, first choose an input size at random under some polynomially-decreasing distribution, then choose an input of that size uniformly at random. It is easy to see that the uniform distribution is polynomial-time computable.

2 DistNP

We define the complexity class

$$\mathbf{DistNP} = \{\langle L, \mu \rangle : L \in NP, \mu \text{ polynomial-time computable}\}. \tag{4}$$

There are at least two good reasons for looking only at polynomial-time computable distributions.

1. One can show that there exists a distribution μ such that every problem is as hard on average under μ as it is in the worst case. Therefore, unless we place some computational restriction on μ, the average-case theory is identical to the worst-case one.

2. Someone, somewhere, had to generate the instances we're trying to solve. If we place computational restrictions on ourselves, then it seems reasonable also to place restrictions on whoever generated the instances.

It should be clear that we need a whole *class* of distributions to do reductions; that is, we can't just parameterize a complexity class by a single distribution. This is because a problem can have more than one natural distribution; it's not always obvious what to take as the 'uniform distribution.'

3 Reductions

Definition 2 (Reduction) *We say that a distributional problem $\langle L_1, \mu_1 \rangle$* reduces *to a distributional problem $\langle L_2, \mu_2 \rangle$ (in symbols, $\langle L_1, \mu_1 \rangle \le \langle L_2, \mu_2 \rangle$) if there exists a polynomial-time computable function f such that:*

1. *$x \in L_1$ iff $f(x) \in L_2$.*

2. *There is a $\epsilon > 0$ such that, for every x, $|f(x)| = \Omega(|x|^\epsilon)$.*

3. *For all $y \in \{0,1\}^*$,*
$$\sum_{x: f(x)=y} \mu\prime_1(x) \le \text{poly}(|y|)\, \mu\prime_2(y). \tag{5}$$

The first condition is the standard condition of many-to-one reductions in complexity theory: it ensures that an algorithm that is always correct for L_2 can be converted into an algorithm that is always correct for L_1. The second condition is a technical one, that is necessary to show that the reduction preserves efficient-on-average algorithms. All known reductions satisfies this condition naturally.

The third condition is called *domination*. To motivate this condition, consider that we want reductions to preserve the existence of algorithms that are efficient on average. Suppose that we have an algorithm A_2 for problem L_2 such that, when we pick y according to distribution μ_2, $A(y)$ is efficient on average; if we want to solve L_1 under distribution μ_1, then, starting from an input x distributed according to $\mu_1(x)$, we compute $f(x)$ and then apply algorithm A_2 to $f(x)$. This will certainly be correct, but what about the running time? Intuitively, it could be the case that A_2 is very slow on some inputs, but such inputs are unlikely to be sampled according to distribution μ_2; the domination condition ensures us that such inputs are also unlikely to be sampled when we sample x according to μ_1 and then consider $f(x)$.

4 Polynomial-Time on Average

Given a problem $\langle L, \mu \rangle$ and an algorithm A that runs in time $t(x)$ on input x, what does it mean to say that A solves $\langle L, \mu \rangle$ in polynomial time on average? We will consider some flawed definitions before settling on the best one and on an alternate one.

A first difficulty comes with the fact that we are dealing with a single distribution on all possible inputs. The most intuitive choice of saying that A is efficient if

$$\mathbf{E}[t(x)] \text{ is small}$$

is problematic because the expectation could be infinite even if A runs in worst-case polynomial time.

One can work around this difficulty by defining A to be polynomial provided that for some constant c and for every sufficiently large n,

$$\mathbf{E}[t(x)|\ |x| = n] \leq n^c$$

However we chose to define distributional problems and reducibility without separating problems by input size, and we would run into several difficulties in separating them now. Besides, it seems reasonable that there could be input lengths on which A takes a long time, but that are generated with very low probability under μ; in such cases A may still be regarded as efficient, but this is not taken into account in the above definition.

Our next attempt folds the polynomial running time into the single distribution by defining A to be polynomial on average if there is a polynomial p such that

$$\mathbf{E}\left[\frac{t(x)}{p(|x|)}\right] = O(1)$$

This definition is quite appealing, but is still subject to the fatal of not being *robust*, in that: (1) reductions do not preserve this definition of polynomial solvability on average and (2) the definition is sensitive to trivial representation changes such as replacing a matrix representation of a graph by an adjacency list.

To see why these problems arise, let μ be the uniform distribution, and let

$$t(x) = 2^n \text{ if } x = \overrightarrow{0},\ t(x) = n^2 \text{ otherwise.} \tag{6}$$

The average running time is about n^2. But suppose now that n is replaced by $2n$ (because of a change in representation, or because of the application of a reduction), then

$$t(x) = 2^{2n} \text{ if } x = \overrightarrow{0},\ t(x) = 4 \cdot n^2 \text{ otherwise.} \tag{7}$$

Similarly, if $t(x)$ is replaced by $t^2(x)$, the average running time becomes exponential.

We now come to a satisfying definition.

Definition 3 (Polynomial on average) *Suppose A is an algorithm for a distributional problem $\langle L, \mu \rangle$ that runs in time $t(x)$ on input x. We say that A has polynomial running time on average is there is a constant c such that*

$$\mathbf{E}\left[\frac{t(x)^{1/c}}{|x|}\right] = O(1)$$

Notice, first, that this definition is satisfied by any algorithm that runs in worst-case polynomial time. If $t(x) = O(|x|^c)$, then $t(x)^{1/c} = O(|x|)$ and the sum converges. More interestingly, suppose $t()$ is a time bound for which the above definition is satisfied; then an algorithm whose running time is $t'(x) = t(x)^2$ also satisfies the definition, unlike the case of the previous definition. In fact we have the following result, whose non-trivial proof we omit.

Theorem 1 *If $\langle L_1, \mu_1 \rangle \leq \langle L_2, \mu_2 \rangle$ and $\langle L_2, \mu_2 \rangle$ admits an algorithm that is polynomial on average, then $\langle L_1, \mu_1 \rangle$ also admits an algorithm that is polynomial on average.*

There is an additional interesting property of the definition of polynomial of average: there is a high probability that the algorithm runs in polynomial time.

Suppose that

$$\mathbf{E}\left[\frac{t(x)^{1/c}}{|x|}\right] = O(1). \tag{8}$$

and that we wish to compute $\Pr[t(x) \geq k \cdot |x|^c]$. Such a probability is clearly the same as

$$\Pr[t(x)^{1/c} \geq k^{1/c}|x|]$$

and by Markov's inequality this is at most $O(1/k^{1/c})$, which can be made fairly small by picking k large enough. Since the algorithm runs in time at most kn^c for a subset of inputs having probability $1 - k^{-1/c}$, we see that our definition gives a smooth quantitative tradeoff for how much time we need to solve an increasing fraction of inputs.

In the setting of one-way functions and in the study of the average-case complexity of the permanent and of problems in EXP (with applications to pseudorandomness), we normally interpret "average case hardness" in the following way: that an algorithm of limited running time will fail to solve the problem on a noticeable fraction of the input. Conversely, we would interpret average-case tractability as the existence of an algorithm that solves the problem in polynomial time, except on a negligible fraction of inputs. This leads to the following formal definition.

Definition 4 (Heuristic polynomial time) *We say that an algorithm A is a heuristic polynomial time algorithm for a distributional problem $\langle L, \mu \rangle$ if A always runs in polynomial time and for every polynomial p*

$$\sum_{x:A(x)\neq\chi_L(x)} \mu'(x)p(|x|) = O(1)$$

In other words, a polynomial time algorithm for a distributional problem is a heuristic if the algorithm fails on a negligible fraction of inputs, that is, a subset of inputs whose probability mass is bounded even if multiplied by a polynomial in the input length. It might also make sense to consider a definition in which A is always correct, although it does not necessarily work in polynomial time, and that A is heuristic polynomial time if there is a polynomial q such that for every polynomial p, $\sum_{x\in S_q} \mu'(x)p(|x|) = O(1)$, where S_q is the set of inputs x such that $A(x)$ takes more than $q(|x|)$ time. Our definition is only more general, because from an algorithm A as before one can obtain an algorithm A satisfying Definition 4 by adding a clock that stops the computation after $q(|x|)$ steps.

The definition of heuristic polynomial time is *incomparable* with the definition of average polynomial time. For example, an algorithm could take time 2^n on a fraction $1/n^{\log n}$ of the inputs of length n, and time n^2 on the remaining inputs, and thus be a heuristic polynomial time algorithm with respect to the uniform distribution, while not beign average polynomial time with respect to the uniform distribution. On the other hand, consider an algorithm such that for every input length n, and for $1 \leq k \leq 2^{n/2}$, there is a fraction about $1/k^2$ of the inputs of length n on which the algorithm takes time $\Theta(kn)$. Then this algorithm satisfies the definition of average polynomial time under the uniform distribution, but if we impose a polynomial clock there will be an inverse polynomial fraction of inputs of each length on which the algorithm fails, and so the definition of heuristic polynomial time cannot be met.

It is easy to see that heuristic polynomial time is preserved under reductions.

Theorem 2 *If $\langle L_1, \mu_1 \rangle \leq \langle L_2, \mu_2 \rangle$ and $\langle L_2, \mu_2 \rangle$ admits a heuristic polynomial time algorithm, then $\langle L_1, \mu_1 \rangle$ also admits a heuristic polynomial time algorithm.*

PROOF: Let A_2 be the algorithm for $\langle L_2, \mu_2 \rangle$, let f be the function realizing the reduction, and let p be the polynomial witnessing the domination property of the reduction. Let c and ϵ be such that for every x we have $|x| \leq c|f(x)|^{1/\epsilon}$.

Then we define the algorithm A_1 than on input x outputs $A_2(f(x))$. Clearly this is a polynomial time algorithm, and whenever A_2 is correct on $f(x)$, then A_1 is correct on x. We need to show that for every polynomial q

$$\sum_{x:A_2(f(x))\neq\chi_{L_2}(f(x))} \mu_1'(x)q(|x|) = O(1)$$

and the left-hand side can be rewritten as

$$\begin{aligned}
& \sum_{y:A_2(y)\neq\chi_{L_2}(y)} \sum_{x:f(x)=y} \mu_1'(x)q(|x|) \\
\leq\ & \sum_{y:A_2(y)\neq\chi_{L_2}(y)} \sum_{x:f(x)=y} \mu_1'(x)q(c\cdot|y|^{1/\epsilon})) \\
\leq\ & \sum_{y:A_2(y)\neq\chi_{L_2}(y)} \mu_2'(y)p(|y|)q'(|y|) \\
=\ & O(1)
\end{aligned}$$

where the last step uses the fact that A_2 is a polynomial heuristic for $\langle L_2, \mu_2\rangle$ and in the second-to-last step we introduce the polynomial $q'(n)$ defined as $q(c\cdot n^{1/\epsilon})$

□

5 Existence of Complete Problems

We now show that there exists a problem (albeit an artificial one) complete for **DistNP**. Let the inputs have the form $\left\langle M, x, 1^t, 1^l\right\rangle$, where M is an encoding of a Turing machine and 1^t is a sequence of t ones. Then we define the following "universal" problem U.

- Decide whether there exists a string y such that $|y| \leq l$ and $M(x,y)$ accepts in at most t steps.

That U is **NP**-complete follows directly from the definition. Recall the definition of **NP**: we say that $L \in \mathbf{NP}$ if there exists a machine M running in $t = \text{poly}(|x|)$ steps such that $x \in L$ iff there exists a y with $y = \text{poly}(|x|)$ such that $M(x,y)$ accepts. Thus, to reduce L to U we need only map x onto $R(x) = \left\langle M, x, 1^t, 1^l\right\rangle$ where t and l are sufficiently large bounds.

To give a reduction that satisfies the domination condition is indeed harded. Let $\langle L, \mu\rangle \in$ **DistNP**. Define a uniform distribution over the $\left\langle M, x, 1^t, 1^l\right\rangle$ as follows:

$$\mu'\left(\left\langle M, x, 1^t, 1^l\right\rangle\right) = \frac{1}{|M|\left(|M|+1\right)2^{|M|}} \cdot \frac{1}{|x|\left(|x|+1\right)2^{|x|}} \cdot \frac{1}{(t+l)\left(t+l+1\right)}. \tag{9}$$

The trouble is that, because of the domination condition, we can't map x onto $R(x)$ if $\mu\prime(x) > \text{poly}(|x|)\, 2^{-|x|}$. We work around this problem by *compressing* x to a shorter string if $\mu\prime(x)$ is large. Intuitively, by mapping high-probability strings onto shorter lengths, we make their high probability less conspicuous. The following lemma shows how to do this.

Lemma 3 *Suppose μ is a polynomial-time computable distribution over x. Then there exists a polynomial-time algorithm C such that*

1. *C is injective: $C(x) \neq C(y)$ iff $x \neq y$.*
2. *$|C(x)| \leq 1 + \min\left\{|x|, \log\frac{1}{\mu\prime(x)}\right\}$.*

Proof: If $\mu\prime(x) \leq 2^{-|x|}$ then simply let $C(x) = 0x$, that is, 0 concatenated with x. If, on the other hand, $\mu\prime(x) > 2^{-|x|}$, then let $C(x) = 1z$. Here z is the longest common prefix of $\mu(x)$ and $\mu(x-1)$ when both are written out in binary. Since μ is computable in polynomial time, so is z. C is injective because only two binary strings s_1 and s_2 can have the longest common prefix z; a third string s_3 sharing z as a prefix must have a longer prefix with either s_1 or s_2. Finally, since $\mu\prime(x) \leq 2^{-|z|}$, $|C(x)| \leq 1 + \log \frac{1}{\mu\prime(x)}$. □

Now the reduction is to map x onto $R_2(x) = \left\langle \overline{M}, C(x), 1^{\overline{t}}, 1^{l+|x|} \right\rangle$. Here $\overline{M}$ is a machine that on input z, x, y checks that $C(x) = z$ (i.e., that x is a valid decoding of z) and that $M(x, y)$ accepts. The running time of $\overline{M}$ is $\overline{t}$. Clearly $x \in L$ iff $\overline{M}$ accepts. To show that domination holds, let $\mu_2\prime(x) = \Pr[R_2(x)]$. Then, since the map is one-to-one, we need only show that $\mu\prime(x) \leq \text{poly}(|x|)\, \mu\prime_2(x)$. Since $\overline{t} = O(\text{poly}(t))$,

$$\begin{aligned}
\mu\prime_2(x) &= \frac{1}{O\left(\left|\overline{M}\right|^2\right) 2^{\left|\overline{M}\right|}} \cdot \frac{1}{O\left(|C(x)|^2\right) 2^{|C(x)|}} \cdot \frac{1}{O(t + l + |x|)^2} \\
&\geq \text{poly}(|x|) \max\left(2^{-|x|}, \mu\prime(x)\right) \\
&\geq \text{poly}(|x|)\, \mu\prime(x)
\end{aligned}$$

and we are done.

Note that, since we mapped longer inputs to shorter ones, we could not have done this reduction input-length-wise.

6 Polynomial-Time Samplability

Definition 5 (Samplable distributions) *We say that a distribution μ is* polynomial-time samplable if there exists a probabilistic algorithm A, taking no input, that outputs x with probability $\mu\prime(x)$ and runs in $\text{poly}(|x|)$ time.

Any polynomial-time computable distribution is also polynomial-time samplable, provided that for all x,

$$\mu\prime(x) \geq 2^{-\text{poly}(|x|)} \text{ or } \mu\prime(x) = 0. \tag{10}$$

For a polynomial-time computable μ satisfying the above property, we can indeed construct a sampler A that first chooses a real number r uniformly at random from $[0, 1]$, to $\text{poly}(|x|)$ bits of precision, and then uses binary search to find the first x such that $\mu(x) \geq r$.

On the other hand, under reasonable assumptions, there are efficiently samplable distributios μ that are not efficiently computable.

In addition to **DistNP**, we can look at the class

$$\langle \mathbf{NP}, \mathbf{P}\text{-samplable} \rangle = \{\langle L, \mu \rangle : L \in \mathbf{NP}, \mu \text{ polynomial-time samplable}\}. \tag{11}$$

A result due to Impagliazzo and Levin states that if $\langle L, \mu \rangle$ is **DistNP**-complete, then $\langle L, \mu \rangle$ is also complete for the class $\langle \mathbf{NP}, \mathbf{P}\text{-samplable} \rangle$.

This means that the completeness result established in the previous section extends to the class of NP problems with samplable distributions.