Lecture 15

This lecture introduces Levin's theory of average-case complexity. The theory is difficult to use and is still useless, in that it doesn't yet allow us to prove anything interesting about important problems. But eventually complexity theory ought to address how hard problems are to solve on average, and Levin's theory is significant as a step in this direction. Levin presented the theory in a one-page paper in FOCS'84 and in a two-pages SICOMP paper [Lev86].

We first define the model—itself a nontrivial issue—and then we prove the existence of a problem that is complete for the average-case class **NP**.

1 Distributional Problems

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. Importantly, μ is *not* a collection of separate distributions, one for each input size. This is because reductions do not in general preserve input sizes.

We want μ to be polynomial-time computable. What does this mean? For 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 $\mathrm{poly}(|x|)$ time. Clearly this notion is at least as strong as the requirement that $\Pr[x]$ be computable in polynomial time, for

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

$x - 1$ being the lexicographic predecessor of x. Indeed one can show that if $\mathbf{P} \neq \mathbf{NP}$, then 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

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

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 two justifications 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 Polynomial-Time Samplability

Say μ is *polynomial-time samplable* if there exists a probabilistic algorithm A, taking no input, that outputs x with probability $\mu'(x)$ and runs in $\operatorname{poly}(|x|)$ time. Any polynomial-time computable distribution is also polynomial-time samplable, provided that for all x,

$$\mu'(x) \geq 2^{-\operatorname{poly}(|x|)} \text{ or } \mu'(x) = 0. \tag{5}$$

To see this: have A first choose a real number r uniformly at random from $[0, 1]$, to $\operatorname{poly}(|x|)$ bits of precision, then use binary search to find the first x such that $\mu(x) \geq r$. However, μ efficiently samplable does *not* imply μ efficiently computable, under weaker assumptions than the existence of pseudorandom generators. So in addition to **DistNP**, we can look at the class

$$\langle \mathbf{NP}, \mathbf{P}\text{-samplable}\rangle = \{\langle L, \mu\rangle : L \in \mathbf{NP}, \mu \text{ polynomial-time samplable}\}. \tag{6}$$

4 Reductions

We say that $\langle L_1, \mu_1\rangle \leq \langle L_2, \mu_2\rangle$ if there exists a polynomial-time computable function f such that:

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

2. For all $y \in \{0,1\}^*$,

$$\sum_{x:f(x)=y} \mu\prime_1(x) \le \operatorname{poly}(|y|)\, \mu\prime_2(y)\,. \tag{7}$$

The second condition is called *domination.*

5 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 three bad definitions before settling on the best one.

Try 1 $\mathrm{EX}[t(x)]$ should be small. But this expectation can be infinite, even for algorithms that run in worst-case polynomial time.

Try 2 For each input size n, $\mathrm{EX}[t(x) \mid |x| = n]$ should be $O(n^c)$ for some constant c. But having defined distributional problems and reducibility without separating problems by input size, it seems inelegant to separate them now.

Try 3 $\mathrm{EX}\left[\frac{t(x)}{|x|^c}\right]$ should be $O(1)$ for some constant c. But this definition is not *robust*, meaning that (1) reductions aren't transitive and (2) the definition is sensitive to trivial changes such as replacing a matrix representation of a graph by an adjacency list. To see this, let μ be the uniform distribution, and let

$$t(x) = 2^n \text{ if } x = \overrightarrow{0},\ t(x) = n^2 \text{ otherwise.} \tag{8}$$

The average running time is about n^2. But now consider squaring the running times:

$$t(x) = 2^{2n} \text{ if } x = \overrightarrow{0},\ t(x) = n^4 \text{ otherwise.} \tag{9}$$

Now the average running time is about 2^n.

Try 4 $\mathrm{EX}\left[\frac{t(x)^{1/c}}{|x|}\right]$ should be $O(1)$ for some constant c. This definition seems to work; it is achieved in particular by any algorithm that runs in worst-case polynomial time. It is the one we'll use[1].

Suppose that

$$\mathrm{EX}\left[\frac{t(x)^{1/c}}{|x|}\right] = O(1)\,. \tag{10}$$

[1]Indeed we might even go a step farther, and say that A is average-case polynomial if there exists an input set S, having probability $1 - o(n^c)$, such that A runs in polynomial time on S. But we probably can't pull off reductions with this last definition.

Then what is

$$\Pr\left[t(x) \geq k\,|x|^c\right] \quad ? \tag{11}$$

Clearly it's the same as

$$\Pr\left[t(x)^{1/c} \geq k^{1/c}\,|x|\right]. \tag{12}$$

By Markov's inequality, this probability is $O\left(k^{-1/c}\right)$. In other words, the algorithm runs in time at most kn^c for a subset of inputs having probability $1 - k^{-1/c}$. Thus we see that our definition gives a quantitative tradeoff for how much time we need to solve an increasing fraction of inputs.

6 Some Results

A difficult result of Levin states that if $\langle L_1, \mu_1\rangle \leq \langle L_2, \mu_2\rangle$ and $\langle L_2, \mu_2\rangle$ is polynomial-time on average (under the right definition), then $\langle L_1, \mu_1\rangle$ is also polynomial-time on average.

Another 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 $\langle$**NP**, **P**-samplable$\rangle$. This result is useful because, for natural problems, it's often easier to prove **DistNP**-completeness than to prove $\langle$**NP**, **P**-samplable$\rangle$-completeness.

7 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.

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

That (*) is **NP**-complete follows directly from the definition. Recall the criterion for $L \in$ **NP**: 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 (*) we need only map x onto $R(x) = \left\langle M, x, 1^t, 1^l\right\rangle$.

But in the average-case setting things are harder. 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:

$$\Pr\left[\left\langle M, x, 1^t, 1^l\right\rangle\right] = \frac{1}{|M|\,(|M|+1)\,2^{|M|}} \cdot \frac{1}{|x|\,(|x|+1)\,2^{|x|}} \cdot \frac{1}{(t+l)\,(t+l+1)}. \tag{13}$$

The trouble is that, because of the domination condition, we can't map x onto $R(x)$ if $\mu'(x) > \text{poly}(|x|)\,2^{-|x|}$. So what we'll do is *compress* x to a shorter string if $\mu'(x)$ 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 1
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: Use Huffman encoding. 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 $\overline{t} = O(\text{poly}(t))$,

$$\mu\prime_2(x) = \frac{1}{O\left(\left|\overline{M}\right|^2\right) 2^{\left|\overline{M}\right|}} \cdot \frac{1}{O\left(|C(x)|^2\right) 2^{|C(x)|}} \cdot \frac{1}{O(t + l + |x|)^2}$$

$$\geq \text{poly}(|x|) \max\left(2^{-|x|}, \mu\prime(x)\right)$$

$$\geq \text{poly}(|x|)\, \mu\prime(x)$$

and we're done.

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

References

[Lev86] Leonid A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, February 1986.