Notes for Lecture 8 & 9

In these notes we introduce Levin's theory of average-case complexity.

This theory is still in its infancy: in these notes 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. Next time we will 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 $\mu = \mu_1, \ldots, \mu_n, \ldots$ is a collection of distributions.*

Intuitively, we think of each μ_n as a distribution over the "inputs of length n," although for some problems the natural way of describing the distributions will be such that the support of D_n contains inputs of length different from n when encoded as bit strings.[1]

(Levin took a slightly different definitional approach, and defined μ to be a single distribution over all possible inputs. This approach simplifies some part of the theory and complicates some other parts. Thinking of μ as a collection of distributions makes the theory closer to the way average-case complexity is treated in cryptography and in the analysis of algorithms. The two approaches are essentially equivalent.)

2 Polynomial-Time on Average

Given a distributional problem $\langle L, \mu \rangle$ and an algorithm A that runs in time $t_A(x)$ on input x, what does it mean to say that A solves $\langle L, \mu \rangle$ in polynomial time on average?

A first attempt would be to require the running time of the algorithm to be polynomial in expectation, that is, that there is a constant c such that

$$\mathop{\mathbb{E}}_{x \sim \mu_n} [t_A(x)] \leq O(n^c)$$

This definition is quite appealing, but is still subject to the fatal flaw 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 the model of computation, and an algorithm that has expected polynomial running time in one model may have exponential expected running time when simulated on a different model, say, with quadratic slowdown.

[1]For example, we may be interested in solving the Max Clique problem in $G_{n,1/2}$ graphs; each graph in the support of $G_{n,1/2}$, which would play the role of the distribution μ_n, has length $\Theta(n^2)$ when represented as an adjacency matrix.

To see why these problems arise, let μ be the uniform distribution, and suppose that, for an input x of length n, we have

$$t_A(x) = 2^n \text{ if } x = \mathbf{0},\ t_A(x) = n^2 \text{ otherwise.} \tag{1}$$

The average running time is about n^2. But suppose now that $t_A(x)$ is replaced by $t_A^2(x)$: then the expected running time becomes exponential.

The following is a more satisfying definition.

Definition 2 (Polynomial on average) *Suppose A is an algorithm for a distributional problem $\langle L, \mu\rangle$ that runs in time $t_A(x)$ on input x. We say that A has polynomial running time on average is there is a constant c such that for every n*

$$\mathop{\mathbb{E}}_{x\sim\mu_n}\left[t_A(x)^{1/c}\right] = O(n)$$

Notice, first, that this definition is satisfied by any algorithm that runs in worst-case polynomial time. If $t_A(x) = O(n^c)$ for every x of length n, then $t_A^{1/c}(x) = O(n)$ and so the expectation of $t_A^{1/c}(x)$ is also $O(n)$.

Furthermore, if the expected running time is $O(n^c)$, then we have

$$\mathop{\mathbb{E}}_{x\sim\mu_n}[t_A^{1/c}(x)] \leq \left(\mathop{\mathbb{E}}_{x\sim\mu_n}[t_A(x)]\right)^{1/c} = O(n)$$

where the first inequality is an application of Hölder's inequality, so the notion of polynomial-on-average is a relaxation of the notion of expected polynomial time.

A third observation is that the notion of polynomial-on-average is "closed under polynomial slowdown," that is, if $t_A(\cdot)$ is the running time of a polynomial on average algorithm with parameter c, and B is an algorithm of running time $t_B(x) \leq (t_A(x))^k$, then B is a polynomial on average algorithm with parameter ck.

A useful fourth observation is that if an algorithm has polynomial on average running time, then the probability of having a large running time is small, and the distribution of running times has a "tail" that is at most inverse polynomial.

Fact 3 *If $t_A(\cdot)$ is a time bound that is polynomial on average with parameter c with respect to a collection of distribution $\{\mu_n\}_{n\geq 1}$, then*

$$\mathop{\mathbb{P}}_{x\sim\mu_n}[t_A(x) \geq T] \leq O(1)\cdot\frac{n}{T^{1/c}}$$

Proof: This is a simple application of Markov's inequality:

$$\mathop{\mathbb{P}}_{x\sim\mu_n}[t_A(x) \geq T] = \mathop{\mathbb{P}}_{x\sim\mu_n}[t_A^{1/c}(x) \geq T^{1/c}] \leq \mathop{\mathbb{E}}_{x\sim\mu_n} t_A^{1/c}(x)\cdot\frac{1}{T^{1/c}} \leq O(n)\cdot\frac{1}{T^{1/c}}$$

□

Interestingly, this property of the tail of the running time distribution provides a *characterization* of polynomial on average running time.

Fact 4 *If $t_A(\cdot)$ is a time bound such that, for some constants k, ϵ, we have*

$$\mathop{\mathbb{P}}_{x \sim \mu_n}[t_A(x) \geq T] \leq O(1) \cdot \frac{n^k}{T^\epsilon}$$

then $t_A(\cdot)$ is a polynomial on average time bound with parameter $2k/\epsilon$.

Proof: We have

$$\begin{aligned}\mathbb{E}\, t_A^{\epsilon/2}(x) &= \int_0^\infty \mathbb{P}[t_A^{\epsilon/2}(x) \geq T]\ dT \\ &= \int_0^\infty \mathbb{P}[t_A(x) \geq T^{2/\epsilon}]\ dT \\ &\leq O(1) \cdot \int_0^\infty \frac{n^k}{T^2}\ dT \\ &O(n^k)\end{aligned}$$

And so

$$\mathbb{E}\, t_A^{\epsilon/2k}(x) \leq \left(\mathbb{E}\, t_A^{\epsilon/2}(x)\right)^{1/k} = O(n)$$

□

Theorem 5 *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.*

Proof: If B is an algorithm for L_2, then consider the algorithm A that, on input x, computes $f(x)$ and then applies B to $f(x)$; then A is an algorithm for L_1. We want to argue that if B is polynomial-on-average for $\langle L_2, \mu_2 \rangle$, then A is polynomial-on-average for $\langle L_1, \mu_1 \rangle$.

Let $t_B(y)$ be the running time of algorithm B on input y. If B is polynomial-on-average for $\langle L_2, \mu_2 \rangle$ then, by our alternative characterization, we have that for every input length m:

$$\mathop{\mathbb{P}}_{y \sim \mu_m}[t_B(y) \geq T] \leq m^{O(1)} \cdot \frac{1}{T^{\Omega(1)}}$$

Now we bound the probability that A has a large running time by noting that the inputs x on which $B(f(x))$ has large running time cannot have a high probability according to μ_1, because their images $f(x)$ have low probability according to μ_2 and because of the properties of the reduction. Fix an input length n, and let $N \leq n^{O(1)}$ be an upper bound to the length of $f(x)$.

$$\begin{aligned}&\mathop{\mathbb{P}}_{x \sim \mu_n}[t_A(x) \geq T] \\ &= \mathop{\mathbb{P}}_{x \sim \mu_n}[t_B(f(x)) \geq T] \\ &= \sum_{y : t_B(y) \geq T} \mathop{\mathbb{P}}_{x \sim \mu_n}[f(x) = y]\end{aligned}$$

$$\begin{aligned}
&\le \sum_{y: t_B(y) \ge T} \sum_{m=1}^{N} m^{O(1)} \mu_m(y) \\
&= \sum_{m=1}^{N} m^{O(1)} \mathop{\mathbb{P}}_{y \sim \mu_m} [t_B(y) \ge T] \\
&\le \sum_{m=1}^{N} m^{O(1)} \cdot m^{O(1)} \cdot \frac{1}{T^{\Omega(1)}} \\
&\le n^{O(1)} \cdot \frac{1}{T^{\Omega(1)}}
\end{aligned}$$

□

3 Heuristic Polynomial Time

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.

Impagliazzo gave the following related definition.

Definition 6 (Heuristic polynomial time) *We say that an algorithm A is a heuristic polynomial time algorithm for a distributional problem $\langle L, \mu \rangle$ if $A(x, \delta)$ runs in time polynomial in $|x|$ and $1/\delta$, and for every n and δ we have*

$$\mathop{\mathbb{P}}_{x \sim \mu_n} [A(x) \neq \mathbb{1}_L(x)] \le \delta$$

where $\mathbb{1}_L$ is the indicator function of the language L.

Every polynomial-on-average algorithm can easily be transformed into a heuristic polynomial time algorithm. If A is an algorithm for L that is always correct and such that

$$\mathop{\mathbb{E}}_{x \in \mu_n} t^{1/c} t_A(x) \le a \cdot n$$

then we can design an algorithm B that, given x of length n, runs $A(x)$ for $(an/\delta)^c$ steps and provides the answer found by A, or rejects if the $A(x)$ does not halt within $(an/\delta)^c$ steps. Then we have

$$\begin{aligned}
\mathop{\mathbb{P}}_{x \sim \mu_n} [B(x) \neq \mathbb{1}_L(x)] &\le P_{x \sim \mu_n} [t_A(x) \ge (an/\delta)^c] \\
&= \mathop{\mathbb{P}}_{x \sim \mu_n} [t_A^{1/c}(x) \ge an/\delta] \le \delta
\end{aligned}$$

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

Theorem 7 *If $\langle L_1, \mu_1 \rangle \le \langle L_2, \mu_2 \rangle$ 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.*

4 Samplable and Computable Distributions

We will prove a completeness results for distributional problems where where μ is "polynomial-time computable." What do we mean by that? For a given input length n and string x, we define the cumulative probability

$$F_{\mu_n}(x) = \sum_{y \leq x} \Pr[y]. \tag{2}$$

where '$\leq$' denotes lexicographic ordering. We say that μ is "polynomial time computable" if, given n, x, and t, we can compute $F_{\mu_n}(x)$ to t-digit precision in time polynomial in n and t.

This notion is at least as strong as the requirement that $\mu_n(x)$ be computable in polynomial time, because

$$\mu_n(x) = F_{\mu_n}(x) - F_{\mu_n}(x-1) \tag{3}$$

Where we use $x-1$ to denote 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.

Informally, we say that a distribution μ is polynomial time samplable if there is a randomized algorithm A (the "sampler") that, on input n, runs in time $n^{O(1)}$ and produces an output distributed according to μ_n. To make the definition precise, we need to pay attention to what we mean when we say that the sampler runs in polynomial time. The sampler is a randomized algorithm, and so it has access to a stream of random bits; if we require that with probability 1 the sampler halts within time $p(n)$, for some polynomial p, then all possible outputs of the sampler are produced with a probability that is an integer multiple of $1/2^{p(n)}$, which means that even very simple distributions would not be samplable according to this definition. (For example, consider the distribution μ in which 0 has probability 1/3 and 1 has probability 2/3.)

One possibility is to allow the sampler to fail with bounded probability, and to condition on the sampler not failing.

Definition 8 (Strict Polynomial Time Samplability) *A distribution μ is strict polynomial time samplable if there is a polynomial p and an algorithm A that, on input n, runs with probability 1 in time $\leq p(n)$ and such that*

1. $\mathbb{P}[A(n) = \perp] \leq \frac{1}{2}$
2. *For every x,* $\mathbb{P}[A(n) = x | A(n) \neq \perp] = \mu_n(x)$

Where $\perp$ is a special "failure" symbol.

The upper bound of 1/2 on the failure probability could be equivalently replaced with $1/2^n$, since we can restart the sampling algorithm after a failure, and keep running it several times until it produces a valid output. If we run it for up to n times, we maintain the strict polynomial running time property, and we exponentially reduce the failure probability. If we repeat the sampling procedure until it doesn't fail, we output a sample with the correct distribution, and the expected running time is at most $2p(n)$. This motives the following definition.

Definition 9 (Expected Polynomial Time Samplability) *A distribution μ is expected polynomial time samplable if there is a polynomial p and an algorithm A such that*

1. *The expected running time of $A(n)$ is at most $p(n)$*
2. *For every x, $\mathbb{P}[A(n) = x] = \mu_n(x)$*

Every distribution that is strict polynomial time samplable is also expected polynomial time samplable, but under reasonable assumptions the converse is not true. This distinction comes up in a few applications of the notions of samplability, for example in the definition of Perfect Zero Knowledge protocol.

It is easy to see that every polynomial time computable distribution is also expected polynomial time samplable but it is not clear (to Luca as he is writing these notes) whether one can also show that every polynomial time computable distribution is also strict polynomial time samplable.

5 DistNP

We define the complexity class

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

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.

6 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 $\langle M, x, 1^t \rangle$, where M is an encoding of a non-deterministic Turing machine and 1^t is a sequence of t ones. Then we define the following non-deterministic bounded halting problem NBH:

- Decide whether there is an accepting computation of M on input x that takes at most t steps.

That NBH is **NP**-complete follows easily from the definition. From the standard characterization of **NP** we have that $L \in \mathbf{NP}$ if there exists a nondeterministic Turing machine M_L and a polynomial $p(\cdot)$ such that $x \in L$ if and only if $M(x)$ has an accepting computation that takes $\leq p(|x|)$ steps. Thus, to reduce L to NBH we need only map x onto $(M_L, x, 1^{p(|x|)})$.

It will be more complicated to provide an average-case reduction. We define a "uniform distribution" U over instances of NBH as follows: to sample from U_n, we first pick at random three positive integers a, b, c such that $a + b + c = n$, then we sample a random $M \in \{0,1\}^a$ and $x \in \{0,1\}^b$, and we output $(M, x, 1^c)$. So we have

$$U_n(M, x, 1^t) = \Theta\left(\frac{1}{n^2}\right) \cdot \frac{1}{2^{|M|}} \cdot \frac{1}{2^{|x|}}$$

Now let μ be a polynomial time computable distribution and let L, as before, be a language in **NP**. We want to give an average-case reduction from $\langle L, \mu\rangle$ to $\langle NBH, U\rangle$. It is instructive to first check whether the worst-case reduction $x \to (M_L, x, 1^{p(|x|)})$ also works as an average-case reduction. To be an average-case reduction, we need the condition that for every possible output $(M_L, x, 1^{p(|x|)})$ of the reduction, the probability of generating such an output by applying the reduction to an element sampled from μ_n is at most $n^{O(1)}$ times the probability of sampling $(M_L, x, 1^{p(|x|)})$ from U. That is, we need the condition that for every n and every z

$$\begin{aligned}
&\mathop{\mathbb{P}}_{x \sim \mu_n}[(M_L, x, 1^{p(|x|)}) = (M_L, z, 1^{p(|z|)})] \\
&\leq n^{O(1)} \cdot U_{|M_L|+n+p(n)}(M_L, z, 1^{p(|z|)}) \\
&= n^{O(1)} \cdot \Theta(1) \cdot \frac{1}{p^2(n)} \cdot \frac{1}{2^{|M_L|}} \cdot \frac{1}{2^n}
\end{aligned}$$

and, since the length of M_L is a constant and the first expression equals $\mu_n(z)$, we need, for every n and z,

$$\mu_n(z) \leq O\left(\frac{n^{O(1)}}{2^n}\right)$$

which is false for many distributions.

The point of this calculation is that if the reduction outputs an instance $(M, x, 1^t)$, then it must do so with probability at most $n^{O(1)}/2^{|x|}$, so if the reduction is given a string x such that $\mu_n(x)$ is much bigger than $1/2^{|x|}$, the reduction cannot simply copy x verbatim into its output. The reduction, however, is allowed to create an output $(M, x', 1^t)$ provided that $\mu_n(x) \leq n^{O(1)}/2^{|x'|}$. The idea of the reduction is to use a polynomial time computable and invertible compression algorithm C, and, given x, output $(M'_L, C(x), 1^t)$, where M'_L is a machine that, given a string c, first decompresses it, that is, first finds the string x such that $C(x) = c$,and simulates M_L on input x. The parameter t is chosen large enough so that M'_L has time to simulate M on input x for at least $p(|x|)$ steps. We will show that, provided that μ is polynomial time computable, there is a compression algorithm C such that $\mu_n(x) \leq n^{O(1)}/2^{|C(x)|}$. Indeed, a stronger result is known.

Lemma 10 (Optimal Compression of Polynomial Time Computable Distributions) *Suppose μ is a polynomial-time computable distribution. Then there exists a polynomial-time algorithm C such that for every n*

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

Proof: If $\mu_n\left(x\right) \leq 2^{-|x|}$ then simply let $C\left(x\right) = 1x$, that is, 1 concatenated with x. If, on the other hand, $\mu_n\left(x\right) > 2^{-|x|}$, then we let $C\left(x\right) = 0z$. Here z is the longest common prefix of $F_{\mu_n}\left(x\right)$ and $F_{\mu_n}\left(x-1\right)$ 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 $\mu_n\left(x\right) \leq 2^{-|x|}$, $\left|C\left(n,x\right)\right| \leq 1 + \log\frac{1}{\mu_n(x)}$. □

Now the reduction is to map x onto

$$f\left(x\right) := \left\langle \overline{M}, n, C\left(x\right), 1^{\bar{t}} \right\rangle$$

Here, $\overline{M}$ is a machine that, given n, c, finds the x such that $C(n,x) = c$ and then simulates M_L on input x. The running time $\bar{t}$ is a time sufficient for $\overline{M}$ to find x and then complete $p(|x|)$ steps of simulation of $M_L(x)$. Clearly $x \in L$ iff $f(x) \in$NBH.

To verify the other condition, let us consider a possible output $y = f(x) = \left\langle \overline{M}, n, C\left(x\right), 1^{\bar{t}} \right\rangle$ of the reduction, and let $m = n^{O(1)}$ be its length. Then

$$\mathop{\mathbb{P}}_{z \sim \mu_n}\left[f(z) = y\right] = \mu_n(x)$$

and

$$U_m(y) = \frac{1}{n^{O(1)}} \cdot \frac{1}{2^{|C(x)|}} \geq \frac{1}{n^{O(1)}} \cdot \mu_n(x)$$