80240233: Computational Complexity
ITCS, Tsinghua Univesity, Fall 2007

Lecture 18
7 December 2007

Instructor: Andrej Bogdanov

Notes by: Andrej Bogdanov

1 Average-case complexity

In our talk about efficient computation so far, we looked at several types of algorithms – deterministic, randomized, and circuits – and looked at the existence of *hard problems*: Is it true that no matter which algorithm we try there are instances of the problem on which the algorithm will fail?

However there are settings in which this view of hardness is not completely satisfactory. For instance, the NP-hardness of SAT does not prevent people from implementing SAT solvers of increasing sophistication and using them in various applications. The successes enjoyed by this approach may be justified by the view that even though in general, SAT is hard do solve, the hard instances of SAT rarely (or never) come up "in practice".

Average-case complexity provides the formal framework for investigating this intuition. Its starting point is a relaxed notion of solvability. So far we have implicitly required that algorithms, besides being efficient, solve the problem in question correctly on all instances. In average-case complexity, we will only require that the algorithm provides the correct answer on "most" instances; namely the algorithm is allowed to fail on rarely occurring instances. To formalize the notions of "most instances" and "rarely occurring instances" we will view the input as coming from some probability distribution.

Besides capturing a natural relaxation of solvability, this average-case view of hardness is fundamentally connected to many phenomena in the computational theory of randomness and in cryptography.

2 Distributional problems

In average-case complexity, in order to specify a problem, we must give beside the property we are interested in (e.g. is this graph 3-colorable?) a way of choosing instances of the problem (what a random graph?)

To do this, we specify, for every input length n, a distribution μ_n on the set $\{0,1\}^n$ which tell us how to choose a random instance of length n. For instance, if we represent a random graph on k vertices as a list of $n_k = k(k-1)/2$ bits specifying the presence or absence of each possible edge in the graph, and take μ_{n_k} to be the uniform distribution on $\{0,1\}^{n_k}$, then a random graph from $\{0,1\}^{n_k}$ is a graph obtained by choosing each of its possible n_k edges independently at random

with probability 1/2.[1]

We call the collection of distributions $\{\mu_n\}$ an *ensemble of distributions* and denote it by μ.

We now define distributional problems. Here we will talk about decision problems, but one can of course also consider search or counting problems.

Definition 1. *A* distributional problem *is a pair (L, μ), where L is a decision problem and μ is an ensemble of distributions.*

3 Average-case algorithms

We now want to say what it means for an algorithm that works well on average for the distributional problem (L, μ). Naturally, we want such an algorithm to be efficient. There are two reasonable possibilities to consider:

1. For every x, the algorithm $A(x)$ correctly determines whether $x \in L$. Moreover, A runs in time poly($|x|$) for most x, but may take more time for a few x.
2. For every x, the algorithm $A(x)$ runs in time poly($|x|$). Moreover, $A(x)$ correctly determines whether $x \in L$ for most x, but may fail to give the correct answer for a few x.

Here we will focus on deterministic algorithms.

3.1 Algorithms that are efficient on average

Let us begin by looking at the first possibility. How should we go about formalizing the requirement that A runs in polynomial time for most inputs? One alternative is to require that the expected running time of A is polynomial on inputs from μ_n. In other words, there exists a polynomial p such that for fixed n,

$$\mathrm{E}_{x \sim \mu_n}[t(x)] = p(n)$$

where $t(x)$ indicates the running time of A on input x.

However, it turns out that this definition does not capture the intuitive notion of average-case efficiency. Suppose that an algorithm time n on all inputs of length n, except for the input 1^n, in which case its running time is 2^{2n}. Then the expected running time of A for inputs chosen from the uniform distribution is exponential. But when we choose an input uniformly at random, the chance of hitting the instance 1^n is extremely small, so it seems unreasonable to say that A runs in exponential time.

There is another more important reason why this definition is inadequate. One property we want efficient algorithms to have is that they be composable: if algorithm A is efficient and some algorithm B works by running A as a subroutine, then B should also be efficient. This is exactly what

[1] Note that we have defined μ_n only for those n that are of the form $k(k-1)/2$; we will view these input lengths as "permissible" and not worry about the other ones.

makes reductions useful. However the definition fails this reasonable requirement even for some very simple distributions on inputs.

Fortunately, a small modification gets rid of these undesirable properties:

Definition 2. *We say that algorithm A is an average polynomial-time algorithm for (L, μ) if there is a constant c such that for every n,*

$$\mathrm{E}_{x\sim\mu_n}[t(x)^{1/c}] = O(n),$$

where $t(x)$ is the running time of A on input x.

Namely, instead of saying that the running time is polynomial in expectation, we require that some root of the running time of the algorithm is linear in expectation. (There is nothing special about linear here, and any other polynomial in n will give an equivalent definition.)

What does this mean? Suppose that A satisfies this definition. Let's take a time bound $T(n)$ and ask on what fraction of inputs A takes time more than T. We have that

$$\Pr[t(x) \geq T(n)] = \Pr[t(x)^{1/c} \geq T(n)^{1/c}] = O(n/T(n)^{1/c})$$

Thus if $T(n)$ is a sufficiently large polynomial in n, the fraction of inputs on which A takes time more than $T(n)$ is inverse polynomial in n. Now suppose we want to solve L on, say, a $1 - 1/n^2$ fraction of inputs of length n. To do this, we can choose $T(n) = n^{3c}$, run $A(x)$ for $T(|x|)$ steps and output an answer if $A(x)$ happens to answer within this time bound. Of course there is nothing special about $1/n^2$, and in fact we can solve L on more and more inputs by simulating A for more and more steps.

Efficiency on average is preserved under composition, once we properly define a way of composing average-case algorithms. We will see this shortly when we describe average-case reductions.

3.2 Heuristic algorithms

Heuristics are algorithms that are mostly correct but may give an incorrect answer on some small fraction of inputs. But what is a small fraction of inputs? One possibility is to fix the fraction $\epsilon(n)$ of inputs of length n on which the algorithm fails, and consider algorithms that are correct on all but an $\epsilon(n)$ fraction of inputs. For instance, this is the notion we considered when we showed that if permanent can be computed on a $7/8 + o(1)$ fraction of inputs, then it can be computed on all inputs.

Instead, it will be more convenient to have a definition that allows a smooth tradeoff between the running time and the fraction of instances on which the algorithm fails: Given more time, we can solve more instances.

Definition 3. *A polynomial-time heuristic for (L, μ) is an algorithm A that, given an instance $x \in \{0,1\}^*$ and an error parameter $\epsilon > 0$, runs in time polynomial in $|x|$ and $1/\epsilon$ and for every n and ϵ, satisfies the following property:*

$$\Pr_{x\sim\mu_n}[A(x, \epsilon) \neq L(x)] < \epsilon.$$

Suppose that we have a polynomial-time heuristic A, and we want to solve L on $1 - 1/n^2$ fraction of inputs of length n. Then on input x of length n, we run $A(x, 1/n^2)$. Similarly we can solve L on any inverse-polynomial fraction of instances by setting ϵ appropriately.

In fact, if (L, μ) has an average polynomial-time algorithm, it also has a polynomial-time heuristic: On input (x, ϵ), simulate the average polynomial-time algorithm for $(|x|/\epsilon)^c$ steps, output an answer if produced by the algorithm, and something arbitrary otherwise.

4 Computable ensembles and distributional NP

The next step is to give an average-case analog of the class NP. We could try to look at the class of all distributional problems (L, μ), where $L \in$ NP and μ is an arbitrary ensemble. It turns out that this class does not capture the intuitive notion of "average-case NP". Roughly, the reason is that for every language L we could choose a distribution μ that places all its weight on the "hard instances" for L,[2] so we are back to studying worst-case hardness. At the other end of the spectrum, we could restrict μ to be uniform on every input length. This option is a bit too restrictive; for some problems there is no natural way (or sometimes there are many good ways) to define a "uniformly random instance".

Indeed, we want to consider a class of ensembles that is neither too restrictive, nor too extensive. Again, there are several reasonable ways of doing this. To begin with, we will look at a class that is rich enough to include many natural ensembles, yet structured enough to avoid pathological examples.

Definition 4. *An ensemble of distributions μ is* polynomial-time computable *if there is an efficient algorithm that on input x of length n computes the cumulative distribution function*

$$\overline{\mu}_n(x) = \Pr_{X \sim \mu_n}[X \leq x].$$

Here $\leq$ is the lexicographic ordering over $\{0,1\}^n$. What this means is that we can efficiently compute, for every string x, the *exact* probability that a random string from μ_n is lexicographically at most x. We can then also compute the probability

$$\mu_n(x) = \Pr_{X \sim \mu_n}[X \leq x] = \overline{\mu}_n(x) - \overline{\mu}_n(x-1).$$

where $x - 1$ is the lexicographic predecessor of x.

This condition probably appears fairly mysterious at the moment, and it is indeed a bit technical. It will play a crucial role when we talk about average-case completeness. Moreover, it is a stepping stone for understanding the more general and much more natural class of *polynomial-time samplable ensembles* that we will see next time.

The uniform ensemble that gives each string x of length n the probability 2^{-n} is polynomial-time computable.

We can now define an average-case analog of the class NP:

[2] This is not precise, because instances that are hard for one algorithm may be easy for another, but can be made to work via a diagonalization type argument.

Definition 5. *The distributional class* $(\mathrm{NP}, \mathrm{PComp})$ *consists of all distributional problems* (L, μ)*, where* $L \in \mathrm{NP}$ *and* μ *is a polynomial-time computable ensemble.*

5 Reductions between distributional problems

Suppose that we want to reduce distributional problem (L, μ) to (L', μ'). What this means is that we want to use a potential average-case algorithm for (L', μ') to help us solve (L, μ). Naturally, we will want the standard property that the reduction is polynomial-time computable, and that membership in L' determines membership in L. However in the average-case setting this is not sufficient. The reason is that an average-case algorithm for L' works well only if the inputs it receives come from the distribution μ'. To use this algorithm we will have to ensure that the instances generated by the reduction are also distributed according to something like μ'.

Along these lines, we can ask for the following: When x comes from the distribution μ_n, the output of the reduction $R(x)$ follows the distribution $\mu'_{n'}$ for some n'. However this requirement is much too strict to be of interest, because it is difficult to design reductions that exactly follow a given distribution. How can we relax this? Intuitively, the instances that are hard for (L', μ') are those that are rare according to μ', so it seems reasonable to require instead that rare instances of μ' are not generated too often by R. The formal way to say this is that $\mu'_{n'}(y)$ does not exceed the probability of all instances x of μ_n that are mapped to y by more than a polynomial factor, or $\mu'(y)$ "polynomially dominates" $\mu(x)$.

Definition 6. *Distributional problem* (L, μ) *reduces to* (L', μ') *if there is a polynomial-time algorithm* R *such that*

1. *(Correctness) For every* x*,* $x \in L$ *iff* $x' \in L'$*.*
2. *(Polynomial domination) There is a polynomial* p *so that for every* n*, there exists an* n' *with*

$$\sum_{x : y = R(x)} \mu_n(x) \leq p(n) \cdot \mu'_{n'}(y)$$

for all y *of length* n'*.*

If the second condition is satisfied, then for every event E over $\{0,1\}^{n'}$, we have that

$$\Pr_{x \sim \mu_n}[R(x) \in E] \leq p(n) \cdot \Pr_{y \sim \mu'_{n'}}[y \in E].$$

We now show that reductions preserve efficiency for heuristics. They also preserve efficiency for average polynomial-time reductions but we won't show that here.

Theorem 7. *If* (L, μ) *reduces to* (L', μ') *and* (L', μ') *has a polynomial-time heuristic, then* (L, μ) *has a polynomial-time heuristic.*

Proof. Let A' be a polynomial-time heuristic for (L', μ'), R the reduction from (L, μ) to (L', μ'), and $A(x, \epsilon) = A'(R(x), \epsilon/p(|x|))$. Then

$$\begin{aligned}\Pr_{x\sim\mu_n}[A(x,\epsilon) \neq L(x)] &= \Pr_{x\sim\mu_n}[A'(R(x), \epsilon/p(n)) \neq L'(R(x))] \\ &= p(n) \cdot \Pr_{y\sim\mu'_{n'}}[A'(y, \epsilon/p(n)) \neq L'(y)] \\ &\leq p(n) \cdot (\epsilon/p(n)) = \epsilon.\end{aligned}$$

□

6 A complete problem

We now show that there exists a problem in (NP, PComp) that is complete for average polynomial-time reductions. Our candidate problem will be the bounded halting problem BH from the first lecture. Recall that BH consists of the instances

> $(M, x, 1^t)$: There exists a y of length at most t such that M accepts (x, y) in t or fewer steps.

Recall that BH is NP-complete. To prove that it is complete for (NP, PComp) we have to specify a distribution on inputs.

The distribution ν_N over instances of BH of length N will be described by a random process that generates such an instance. We first uniformly generate three random numbers m, n, t such that $m+n+t = n$. We then choose M and x uniformly from the sets $\{0,1\}^m$ and $\{0,1\}^n$ respectively and generate the instance $(M, x, 1^t)$. It is not difficult to check that this ensemble is polynomial-time computable.

Now suppose we are given a distributional problem (L, μ) in (NP, PComp). Then L is described by a verifier M and a polynomial running time bound p. Recall that when we showed BH is NP-complete, we mapped instance x of L into the instance $(M, x, 1^{p(|x|)})$ of BH. The problem here is that $\nu_N(M, x, 1^{p(|x|)})$ might not polynomially dominate $\mu(x)$. Note that for a fixed x, the probability $\nu_N(M, x, 1^t)$ is inverse exponential in the length of x. So if $\mu_n(x)$ is large, say $\mu_n(x) = 1/2$, then $\nu_N(M, x, 1^{p(|x|)})$ will be substantially smaller than $\mu_n(x)$.

To avoid this issue, what we will need to do is map probable instances of L into short instances of BH. Another way of saying this is we want to compress x: Likely instances should have short compressions. This can be done efficiently for computable ensembles.

Lemma 8. *Let μ be a polynomial-time computable ensemble. There is a polynomial-time algorithm C that maps inputs of length n to outputs of length at most $n + 1$ such that*

1. *C is injective: For every n and x, x' of length n, if $x \neq x'$ then $C(x) \neq C(x')$.*
2. *$|C(x)| < \log_2 1/\mu_n(x) + 2$.*

Proof. If $\mu_n(x) \leq 2^{-n}$, let $C(x) =$ "$1x$". Otherwise, write out the binary expansion $0.x_1x_2x_3\ldots$ of the number $\overline{\mu}_n(x)$. Let $C(x) =$ "$0x_1x_2\ldots x_k$", where k is smallest index such for which $0.x_1x_2\ldots x_k$

falls inside the interval $(\overline{\mu}_n(x-1), \overline{\mu}_n(x)]$. Since $0.x_1x_2\ldots x_{k-1}$ is outside this interval, we must have

$$2^{-(k-1)} \leq \overline{\mu}_n(x) - 0.x_1x_2\ldots x_{k-1} < \overline{\mu}_n(x) - \overline{\mu}_n(x-1) = \mu_n(x)$$

so that $k < \log_2 1/\mu_n(x) + 1$. This proves property 2. For property 1, if $C(x) = C(x')$, then $C(x)$ and $C(x')$ represent a number in the same interval $(\overline{\mu}_n(z-1), \overline{\mu}_n(z)]$, so it must be that $x = x' = z$. □

We now consider the reduction

$$R(x) = (M', C(x), 1^t)$$

Where M' is the machine that, on input $(z, (x, y))$ checks that $C(x) = z$ and $M(x, y)$ accepts, and the output length of R is $|M'| + q(|x|)$, where q is a polynomial that is sufficiently large so that t exceeds the running time of $M'(x)$.

The reduction is obviously correct, since M' accepts $C(x)$ if and only if M accepts x. (This uses the injectivity of C.) We prove polynomial domination. Consider an instance $(M', z, 1^t)$ of length N of BH. There is at most one possible string x of length n such that $C(x) = z$. If there is no x there is nothing to prove, so let us assume that there is such an x. The probability of generating instance $(M', C(x), 1^t)$ in ν_N is

$$\Omega(1/N^2) \cdot 2^{-|M'|} \cdot 2^{-|C(x)|} \geq \Omega(1/N^2) \cdot 2^{-|M'|} \cdot 2^{\log_2 \mu_n(x) - 2} = 2^{-|M'|}/\mathrm{poly}(n) \cdot \mu_n(x).$$

Since M' is a fixed machine that does not vary with n, it follows that $\nu_N(M', C(x), 1^t)$ polynomially dominates $\mu_n(x)$.

We just proved the following theorem.

Theorem 9. *The distributional problem* (BH, ν) *is complete for* $(\mathrm{NP}, \mathrm{PComp})$ *under average-case reductions.*

How about average-case versions of other NP-complete problems? In fact, many of them, including satisifability, clique, graph coloring, etc., have average-case complete versions with respect to some polynomial-time computable distribution. However the distributions for which these problems are known to be average-case complete are not natural. The average-case complexity of many interesting distributional problems (natural problems with respect to natural distributions) is still unsettled.