Computational Complexity: A Modern Approach

Draft of a book: Dated January 2007
Comments welcome!

Sanjeev Arora and Boaz Barak
Princeton University
complexitybook@gmail.com

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to
complexitybook@gmail.com — **Thank You!!**

Chapter 15

Average Case Complexity: Levin's Theory

[1]

NEEDS MORE WORK

Our study of complexity —- **NP**-completeness, #**P**-completeness etc.— thus far only concerned worst-case complexity. However, algorithms designers have tried to design efficient algorithms for NP-hard problems that work for "many" or "most" instances. This motivates a study of the difficulty of the "average" instance. Let us first examine the issues at an intuitive level, so we may be better prepared for the elements of the theory we will develop.

Many average case algorithms are targeted at graph problems in *random graphs*. One can define random graphs in many ways: the simplest one generates a graph on n vertices randomly by picking each potential edge with probability 1/2. (This method ends up assigning equal probability to every n-vertex graph.) On such rand om graphs, many **NP**-complete problems are easy. 3-COLOR can be solved in linear time with high probability (exercise). CLIQUE and INDEPENDENT SET can be solved in $n^{2\log n}$ time (exercise) which is only a little more than polynomial and much less than $2^{\epsilon n}$, the running time of the best algorithms on worst-case instances.

However, other NP-complete problems appear to require exponential time even on average. One example is SUBSET SUM: we pick n integers $a_1, a_2, \ldots, a_n$ randomly from $[1, 2^n]$, pick a random subset S of $\{1, \ldots, n\}$, and produce $b = \sum_{i \in S} a_i$. We do not know of any efficient average-case algorithm that, given the a_i's and b, finds S. Surprisingly, efficient algorithms do exist if the a_i's are picked randomly from the slightly larger interval $[1, 2^{n \log^2 n}]$. This illustrates an important point, namely, that average-case complexity is sensitive to the choice of the input distribution.

The above discussion suggests that even though **NP**-complete problems are essentially equivalent with respect to worst case complexity, they may differ vastly in their average case complexity. Can we nevertheless identify some problems that remain "complete" even for the average case; in other words, are at least as hard as every other average-case **NP** problem?

This chapter covers Levin's theory of average-case complexity. We will formalize the notion of "distributional problems," introduce a working definition of "algorithms that are efficient on

[1] This chapter written with Luca Trevisan

NOTE 15.1 (IMPAGLIAZZO'S POSSIBLE WORLDS)
At the moment we don't know if the best algorithm for 3SAT runs in time $O(n)$ or $2^{\Omega(n)}$ but there are also many other *qualitative* open questions about the hardness of problems in **NP**. Russell Impagliazzo characterized a central goal of complexity theory as the question of finding out which of the following possible worlds is the world we live in:

Algorithmica.

Heuristica.

Pessiland.

Minicrypt.

average," and define a reduction that preserves efficient average-case solvability. We will also exhibit an **NP**-complete problem that is complete with respect to such reductions. However, we cannot yet prove the completeness of natural distributional problems such as SUBSET SUM or one of the number theoretic problems described in the chapter on cryptography.

15.1 Distributional Problems

In our intuitive discussion of average case problems, we first fixed an input size n and then considered the average running time of the algorithm when inputs of size n are chosen from a distribution. At the back of our mind, we knew that complexity has to be measured asymptotically as a function of n. To formalize this intuitive discussion, we will define distributions on all (infinitely many) inputs.

DEFINITION 15.2 (DISTRIBUTIONAL PROBLEM)
A *distributional problem is a pair* $\langle L, \mathcal{D} \rangle$*, where* L *is a decision problem and* $\mathcal{D}$ *is a distribution over the set* $\{0,1\}^*$ *of possible inputs.*

EXAMPLE 15.3
We can define the "uniform distribution" to be one that assigns an input $x \in \{0,1\}^*$ the probability

$$\Pr[x] = \frac{1}{|x|(1+|x|)} 2^{-|x|}. \quad (1)$$

We call this "uniform" because it assigns equal probabilities to all strings with the same length. It is a valid distribution because the probabilities sum to 1:

rect??

$$\sum_{x\in\{0,1\}^*} \frac{1}{|x|\,(1+|x|)} 2^{-|x|} = \sum_{n\geq 0} 2^n \frac{2^{-n}}{n(n+1)} = 1. \tag{2}$$

Here is another distribution; the probabilities sum to 1 since $\sum_{n\geq 1} \frac{1}{n^2} = \pi^2/6$.

$$\Pr[x] = \frac{6}{\pi^2}\frac{2^{-|x|}}{|x|^2} \qquad \text{if } |x| \geq 1 \tag{3}$$

To pick a string from these distributions, we can first an input length n with the appropriate probability (for the distribution in (2), we pick n with probability $6/\pi^2 n^2$) and then pick x uniformly from inputs of length n. This uniform distribution corresponds to the intuitive approach to average case complexity discussed in the introduction. However, the full generality of Definition 15.2 will be useful later when we study nonuniform input distributions.

15.1.1 Formalizations of "real-life distributions."

Real-life problem instances arise out of the world around us (images that have to be understood, a building that has to be navigated by a robot, etc.), and the world does not spend a lot of time tailoring instances to be hard for our algorithm —arguably, the world is indifferent to our algorithm. One may formalize this indifference in terms of computational effort, by hypothesizing that the instances are produced by an efficient algorithm. We can formalize this in two ways.

Polynomial time computable distributions. Such distributions have an associated deterministic polynomial time machine that, given input x, can compute the *cumulative probability* $\mu_{\mathcal{D}}(x)$, where

$$\mu_{\mathcal{D}}(x) = \sum_{y\leq x} \Pr_{\mathcal{D}}[y] \tag{4}$$

Here $\Pr_{\mathcal{D}}[y]$ denotes the probability assigned to string y and $y \leq x$ means y either precedes x in lexicographic order or is equal to x. Denoting the lexicographic predecessor of x by $x-1$, we have

$$\Pr_{\mathcal{D}}[x] = \mu_{\mathcal{D}}(x) - \mu_{\mathcal{D}}(x-1), \tag{5}$$

which shows that if $\mu_{\mathcal{D}}$ is computable in polynomial time, then so is $\Pr_{\mathcal{D}}[x]$. The uniform distributions in (1) and (1) are polynomial time computable, as are many other distributions that are defined using explicit formulae.

Polynomial time samplable distributions. These distributions have an associated probabilistic polynomial time machine that can produce samples from the distribution. In other words, it outputs x with probability $\Pr_{\mathcal{D}}[x]$. The expected running time is polynomial in the length of the output $|x|$.

Many such samplable distributions are now known, and the sampling algorithm often uses Monte Carlo Markov Chain (MCMC) techniques.

If a distribution is polynomial time computable then we can efficiently produce samples from it. (Exercise.) However, if $\mathbf{P} \neq \mathbf{P}^{\#\mathbf{P}}$ there are polynomial time samplable distributions (including some very interesting ones) that are not polynomial time computable. (See exercises.)

In this lecture, we will restrict attention to distributional problems involving a polynomial time computable distribution. This may appear to a serious limitation, but with some work the results of this chapter can be generalized to samplable distributions.

15.2 DistNP and its complete problems

The following complexity class is at the heart of our study of average case complexity.

$$\text{dist } \mathbf{NP} = \{\langle L, \mathcal{D}\rangle : L \in NP, \mathcal{D} \text{ polynomial-time computable}\}. \tag{6}$$

Since the same NP language may have different complexity behavior with respect to two different input distributions (SUBSET SUM was cited earlier as an example), the definition wisely treats the two as distinct computational problems. Note that every problem mentioned in the introduction to the chapter is in dist **NP**.

Now we need to define the average-case analogue of **P**.

15.2.1 Polynomial-Time on Average

Now we define what it means for a deterministic algorithm A to solve a distributional problem $\langle L, \mathcal{D}\rangle$ in polynomial time on average. The definition should be *robust* to simple changes in model of computation or representation. If we migrate the algorithm to a slower machine that has a quadratic slowdown (so t steps now take t^2 time), then polynomial-time algorithms should not suddenly turn into exponential-time algorithms. (This migration to a slower machine is not merely hypothetical, but also one way to look at a reduction.) As we will see, some intuitively appealing definitions do not have this robustness property.

Denote by $t(x)$ the running time of A on input x. First, note that $\mathcal{D}$ is a 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.

Next, we could try to define A to be polynomial provided that for some constant c and for every sufficiently large n,

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

This has two problems. First, it ignores the possibility that there could be input lengths on which A takes a long time, but that are generated with very low probability under $\mathcal{D}$. In such cases A may still be regarded as efficient, but the definition ignores this possibility. Second, and

more seriously, the definition is not robust to changes in computational model. To give an example, suppose $\mathcal{D}$ is the uniform distribution and $t(x_0) = 2^n$ for just one input x_0 of size n For every other input of size n, $t(x) = n$. Then $E[t(x) \mid |x| = n] \leq n + 1$. However, changing to a model with a quadratic slowdown will square all running times, and $E[(t(x))^2 \mid |x| = n] > 2^n$.

We could try to define A to be polynomial if there is a $c > 0$ such that

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

but this is also not robust. (Verify this!)

We now come to a satisfying definition.

Definition 15.4 (Polynomial on average and dist **P**)
A problem $\langle L, \mathcal{D}\rangle \in$ dist **NP** is said to be in dist **P** if there is an algorithm A for L that satisfies for some constants c, c_1

$$\mathbf{E}[\ \frac{t(x)^{1/c}}{|x|}\ ] = c_1, \tag{7}$$

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

Notice that **P** $\subseteq$ dist **P**: if a language can be decided deterministically in time $t(x) = O(|x|^c)$, then $t(x)^{1/c} = O(|x|)$ and the expectation in (7) converges regardless of the distribution. Second, the definition is robust to changes in computational models: if the running times get squared, we just multiply c by 2 and the expectation in (7) again converges.

We also point out an additional interesting property of the definition: there is a high probability that the algorithm runs in polynomial time. For, if

$$\mathbf{E}[\frac{t\,(x)^{1/c}}{|x|}] = c_1, \tag{8}$$

then we have

$$\Pr[t(x) \geq k \cdot |x|^c] \quad = \Pr[\frac{t(x)^{1/c}}{|x|} \geq k^{1/c}] \quad \leq \frac{c_1}{k^{1/c}} \tag{9}$$

where the last claim follows by Markov's inequality. Thus by increasing k we may reduce this probability as much as required.

15.2.2 Reductions

Now we define reductions. Realize that we think of instances as being generated according to a distribution. Defining a mapping on strings (e.g., a reduction) gives rise to a new distribution on strings. The next definition formalizes this observation.

Definition 15.5
If f is a function mapping strings to strings and $\mathcal{D}$ is a distribution then the distribution $f \circ \mathcal{D}$ is one that assigns to string y the probability $\sum_{x:f(x)=y} \Pr_{\mathcal{D}}[x]$

Definition 15.6 (Reduction)
A distributional problem $\langle L_1, \mathcal{D}_1 \rangle$ *reduces* to a distributional problem $\langle L_2, \mathcal{D}_2 \rangle$ (denoted $\langle L_1, \mathcal{D}_1 \rangle \leq \langle L_2, \mathcal{D}_2 \rangle$) if there is a polynomial-time computable function f and an $\epsilon > 0$ such that:

1. $x \in L_1$ iff $f(x) \in L_2$.
2. For every x, $|f(x)| = \Omega(|x|^{\epsilon})$.
3. There are constants c, c_1 such that for every string y,

$$\Pr_{f \circ \mathcal{D}_1}(y) \leq c_1 |y|^c \Pr_{\mathcal{D}_2}(y). \qquad \textbf{(Domination)}$$

The first condition is standard for many-to-one reductions, ensuring that a decision algorithm for L_2 easily converts into a decision algorithm for L_1. The second condition is a technical one, needed later. All interesting reductions we know of satisfy this condition. Next, we motivate the third condition, which says that $\mathcal{D}_2$ "dominates" (up to a polynomial factor) the distribution $f \circ \mathcal{D}_1$ obtained by applying f on $\mathcal{D}_1$.

Realize that the goal of the definition is to ensure that "if $(L_1, \mathcal{D}_1)$ is hard, then so is $(L_2, \mathcal{D}_2)$" (or equivalently, the contrapositive "if $(L_2, \mathcal{D}_2)$ is easy, then so is $(L_1, \mathcal{D}_1)$.") Thus if an algorithm A_2 is efficient for problem $(L_2, \mathcal{D}_2)$, then the following algorithm ought to be efficient for problem $(L_1, \mathcal{D}_1)$: on input x obtained from distribution $\mathcal{D}_1$, compute $f(x)$ and then run algorithm A_2 on $f(x)$. *A priori*, one cannot rule out the possibility that that A_2 is very slow on some inputs, which are unlikely to be sampled according to distribution $\mathcal{D}_2$ but which show up with high probability when we sample x according to $\mathcal{D}_1$ and then consider $f(x)$. The domination condition helps rule out this possibility.

is proof. In fact we have the following result, whose non-trivial proof we omit.

Theorem 15.7
If $\langle L_1, \mathcal{D}_1 \rangle \leq \langle L_2, \mathcal{D}_2 \rangle$ and $\langle L_2, \mathcal{D}_2 \rangle$ has an algorithm that is polynomial on average, then $\langle L_1, \mathcal{D}_1 \rangle$ also has an algorithm that is polynomial on average.

Of course, Theorem 15.7 is useful only if we can find reductions between interesting problems. Now we show that this is the case: we exhibit a problem (albeit an artificial one) that is complete for dist **NP**. Let the inputs have the form $\langle M, x, 1^t, 1^l \rangle$, 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.

Since part of the input is in unary, we need to modify our definition of a "uniform" distribution to the following.

$$\Pr_{\mathcal{D}}\left(\left\langle M, x, 1^t, 1^l \right\rangle\right) = \frac{1}{|M|(|M|+1)2^{|M|}} \cdot \frac{1}{|x|(|x|+1)2^{|x|}} \cdot \frac{1}{(t+l)(t+l+1)}. \qquad (10)$$

This distribution is polynomial-time computable (exercise).

THEOREM 15.8 (LEVIN)
$\langle U, \mathcal{D}\rangle$ is complete for dist **NP**, *where $\mathcal{D}$ is the uniform ditribution.*

The proof requires the following lemma, which shows that for polynomial-time computable distributions, we can apply a simple transformation on the inputs such that the resulting distribution has no "peaks" (i.e., no input has too high a probability).

LEMMA 15.9 (PEAK ELIMINATION)
Suppose $\mathcal{D}$ is a polynomial-time computable distribution over x. Then there is a polynomial-time computable function g such that

1. *g is injective: $g(x) = g(z)$ iff $x = z$.*
2. *$|g(x)| \leq |x| + 1$.*
3. *For every string y, $\Pr_{g\circ\mathcal{D}}(y) \leq 2^{-|y|+1}$.*

PROOF: For any string x such that $\Pr_{\mathcal{D}}(x) > 2^{-|x|}$, define $h(x)$ to be the largest common prefix of binary representations of $\mu_{\mathcal{D}}(x)$, $\mu_{\mathcal{D}}(x-1)$. Then h is polynomial-time computatable since $\mu_{\mathcal{D}}(x) - \mu_{\mathcal{D}}(x-1) = \Pr_{\mathcal{D}}(x) > 2^{-|x|}$, which implies that $\mu_{\mathcal{D}}(x)$ and $\mu_{\mathcal{D}}(x-1)$ must differ in the somewhere in the first $|x|$ bits. Thus $|h(x)| \leq \log 1/\Pr_{\mathcal{D}}(x) \leq |x|$. Furthermore, h 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.

Now define

$$g(x) = \begin{cases} 0x & \text{if } \Pr_{\mathcal{D}}(x) \leq 2^{-|x|} \\ 1h(x) & \text{otherwise} \end{cases} \tag{11}$$

Clearly, g is injective and satisfies $|g(x)| \leq |x| + 1$. We now show that $g \circ \mathcal{D}$ does not give probability more than $2^{-|y|+1}$ to any string y. If y is not $g(x)$ for any x, this is trivially true since $\Pr_{g\circ\mathcal{D}}(y) = 0$.

If $y = 0x$, where $\Pr_{\mathcal{D}}(x) \leq 2^{-|x|}$, then $\Pr_{g\circ\mathcal{D}}(y) \leq 2^{-|y|+1}$ and we also have nothing to prove.

Finally, if $y = g(x) = 1h(x)$ where $\Pr_{\mathcal{D}}(x) > 2^{-|x|}$, then as already noted, $|h(x)| \leq \log 1/\Pr_{\mathcal{D}}(x)$ and so $\Pr_{g\circ\mathcal{D}}(y) = \Pr_{\mathcal{D}}(x) \leq 2^{-|y|+1}$.

Thus the Lemma has been proved. ■

Now we are ready to prove Theorem 15.8.

PROOF: (Theorem 15.8) At first sight the proof may seem trivial since U is just the "universal" decision problem for nondeterministic machines, and every **NP** language trivially reduces to it. However, we also need to worry about the input distributions and enforce the domination condition as required by Definition 15.6.

Let $\langle L, \mathcal{D}_1\rangle \in$ dist **NP**. Let M be a proof-checker for language L that runs in time n^c; in other words, $x \in L$ iff there is a witness y of length $|y| = |x|^c$ such that $M(x, y) =$ Accept. (For notational ease we drop the big-O notation in this proof.) In order to define a reduction from L to

U, the first idea would be to map input x for L to $\left\langle M, x, 1^{|x|^c}, 1^{|x|^c} \right\rangle$. However, this may violate the domination condition because the uniform distribution assigns a probability $2^{-|x|}/\mathrm{poly}(|x|)$ to $\langle M, x, 1^{|x|^c} \rangle$ whereas x may have much higher probability under $\mathcal{D}_1$. Clearly, this difficulty arises only if the distribution $\mathcal{D}_1$ has a "peak" at x, so we see an opportunity to use Lemma 15.9, which gives us an injective mapping g such that $g \circ \mathcal{D}_1$ has no "peaks" and g is computable say in n^d time for some fixed constant d.

The reduction is as follows: map x to $\langle M', g(x), 1^{|x|^c+|x|}, 1^{|x|^c+|x|^d} \rangle$. Here M' is a modification of M that expects as input a string z and a witness (x, y) of length $|x| + |x|^c$. Given (z, x, y) where $y = |x|^c$, M' checks in $|x|^d$ time if $g(x) = z$. If so, it simulates M on (x, y) and outputs its answer. If $g(x) \neq z$ then M' rejects.

To check the domination condition, note that $y = \langle M', g(x), 1^{|x|^c+|x|}, 1^{|x|^c+|x|^d} \rangle$ has probability

$$\begin{aligned} \Pr_{\mathcal{D}}(y) &= \frac{2^{-|M'|}}{|M'|\,(|M'|+1)} \cdot \frac{2^{-|g(x)|}}{|g(x)|\,(|g(x)|+1)} \cdot \frac{1}{(|x| + 2\,|x|^c + |x|^d)(|x| + 2\,|x|^c + |x|^d + 1)} \\ &\leq \frac{2^{-|M'|}}{|M'|\,(|M'|+1)} \frac{1}{|x|^{2(c+d+1)}} \cdot 2^{-g(x)} \end{aligned} \tag{12}$$

under the uniform distribution whereas

$$\Pr_{\mathcal{D}_1}(x) \leq 2^{-g(x)+1} \leq G\,|x|^{2(c+d+1)} \Pr_{\mathcal{D}}(y)\ ,$$

if we allow the constant G to absorb the term $2^{|M'|}\,|M'|\,(|M'|+1)$. Thus the domination condition is satisfied.

Notice, we rely crucially on the fact that $2^{|M'|}\,|M'|\,(|M'|+1)$ is a constant once we fix the language L; of course, this constant will usually be quite large for typical **NP** languages, and this would be a consideration in practice. ■

15.2.3 Proofs using the simpler definitions

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 15.10 (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 15.10 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 15.11
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 $q(c \cdot n^{1/\epsilon})$

■

15.3 Existence of Complete Problems

We now show that there exists a problem (albeit an artificial one) complete for dist **NP**. Let the inputs have the form $\langle M, x, 1^t, 1^l \rangle$, 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 $R(x) = \langle M, x, 1^t, 1^l \rangle$ where t and l are sufficiently large bounds.

15.4 Polynomial-Time Samplability

Definition 15.12 (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,

$$\mu\prime(x) \geq 2^{-\text{poly}(|x|)} \text{ or } \mu\prime(x) = 0. \tag{13}$$

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 dist **NP**, we can look at the class

$$\langle \mathbf{NP}, \mathbf{P}\text{-samplable} \rangle = \{\langle L, \mu \rangle : L \in \mathbf{NP}, \mu \text{ polynomial-time samplable}\}. \tag{14}$$

A result due to Impagliazzo and Levin states that if $\langle L, \mu \rangle$ is dist **NP**-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.

Exercises

§1 Describe an algorithm that decides 3-colorability on almost all graphs in linear expected time.

Hint: A 3-colorable graph better not contain a complete graph on 4 vertices.

§2 Describe an algorithm that decides CLIQUE on almost all graphs in $n^{2\log n}$ time.

Hint: The chance that a random graph has a clique of size more than k is at most $\binom{n}{k}2^{-k^2/2}$.

§3 Show that if a distribution is polynomial-time computable, then it is polynomial-time sampleable.

Hint: Binary search.

§4 Show that if $\mathbf{P}^{\#\mathbf{P}} \neq \mathbf{P}$ then there is a polynomial time samplable distribution that is not polynomial time computable.

§5 Show that the function g defined in Lemma 15.9 (Peak Elimination) is efficiently invertible in the following sense: if $y = g(x)$, then given y we can reconstruct x in $|x|^{O(1)}$ time.

§6 Show that if one-way functions exist, then dist **NP** $\not\subseteq$ dist **P**.

Chapter notes and history

Suppose $\mathbf{P} \neq \mathbf{NP}$ and yet dist **NP** $\subseteq$ dist **P**. This would mean that generating hard instances of **NP** problems requires superpolynomial computations. Cryptography is thus impractical. Also, it seems to imply that everyday instances of **NP**-complete problems would also be easily solvable. Such instances arise from the world around us —we want to understand an image, or removing the obstacles in the path of a robot— and it is hard to imagine how the inanimate world would do the huge amounts of computation necessary to generate a hard instance.