6.841 Advanced Complexity Theory April 30, 2007

Lecture 21

Lecturer: Madhu Sudan *Scribe: Mayank Varia*

1 Overview

For the next three lectures, we will study average-case complexity. In this lecture, we form precise definitions to answer the following questions.

- What is a distributional problem?
- What does it mean to be easy in the average case?
- What does it mean to be hard in the average case?
- What are reductions between distributional problems?

2 Theory versus practice

The goal of complexity theory is to determine which problems are "easy" and which are "hard," but sometimes complexity theory does not match the empirical evidence found by computer scientists. For instance, complexity theorists tend to call problems in P "easy," but in practice such a problem may be infeasible if the polynomial bound to the running time has large degree (like n^{100}) or a large constant (like 1,000,000). This does not usually happen, however, because in practice once a problem is shown to be in P, eventually a very efficient algorithm is found to solve the problem.

The opposite can happen too. A complexity theorist considers a problem to be "hard" if it is hard in the worst case, and sometimes the worst-case problems are contrived and unlikely to occur in practice. Therefore, a practitioner may look at empirical evidence and say "but on instances that I face in the real world, I am able to solve them nearly optimally 99.99% of the time." We wish to develop a theoretical framework that allows us to make statements of this form.

One way to understand the phrase "nearly optimal" is to use probabilistically checkable proofs and the related theory of approximation, which we studied in the previous three lectures. Our goal today is to develop a theoretical framework to capture the idea of taking "instances that I face in the real world" and solving them "99.99% of the time." Of course, we need to choose an underlying probability distribution in order to make statements like "99.99% of the time," and today's goal is to find a good distribution.

3 Probabilistic analysis of algorithms

The first attempt at developing a framework for average-case analysis came in the 1970s and 1980s. Complexity theorists tried to fix a distribution on the instances of a problem that somehow corresponds to "natural" instances from real life, and then they tried to come up with algorithms that had a fast expected time with respect to the given distribution.

For example, consider the traveling salesman problem, which is NP-complete. Many interesting real-life examples of the traveling salesman problem occur on the plane, and we can form two probability distributions on these instances as follows:

1. Pick n points uniformly at random in the unit square $[0,1]^2 \subseteq \mathbb{R}^2$.
2. Assuming that n is a perfect square, start by placing the n points in a grid pattern and then randomly perturb each point by a small amount.

Interestingly, computer scientists have found extremely efficient (almost always polynomial time) algorithms that solve the traveling salesman problem on the first distribution. This seems like a big success, since we said that TSP problems on the plane were "real-life examples," and we can find efficient algorithms to solve cases from the first distribution. Unfortunately, with high probability it is very difficult to compute the optimal tour for instances generated under the second distribution. Therefore, the expected running time of our algorithm depends not only on our choice of "natural instances" but on the probability distribution we attach to them.

As another example, let's look at the graph Hamiltonicity problem, and suppose that we are interested in directed graphs that have v vertices and $5v$ edges. Here are two distributions to place on the input space:

1. For each vertex, pick 5 outgoing edges.
2. Pick $5v$ edges uniformly at random from the $v(v-1)$ potential directed edges.

If a graph G is picked from the first distribution, then with high probability G does have a Hamiltonian cycle, but if G is picked from the second distribution, then with high probability G does not have a Hamiltonian cycle because with high probability G is not even connected (and checking graph connectivity is fast). Hence, even though both problems have the same input space, the choice of distribution has a large impact on the typical answer.

Although the probabilistic analysis of the 1970s and 1980s discovered many interesting algorithms to problems like the TSP, we notice that this type of analysis is heavily dependent on the underlying distribution chosen. It is not fair to promote one distribution of inputs over another, because the instances that occur "naturally" in one person's research may not come up "naturally" in another person's research. Because the choice of distribution is so important, we want to have a theoretical framework that considers all possible distributions.

4 Distributional problems

Our underlying concern is to study NP search problems and understand their average-case behavior. Motivated by the above discussion, we attach a probability distribution as part of the problem specification.

Definition 1 *A* distributional problem *is a pair* (Π, D)*, where* $\Pi \subseteq \{0,1\}^* \times \{0,1\}^*$ *is an* NP*-relation and* $D = \{D_n\}_{n\in\mathbb{N}}$ *is an ensemble where each* $D_n : \{0,1\}^n \to [0,1]$ *is a probability distribution. In particular, for all* n*, the sum* $\sum_{x\in\{0,1\}^n} D_n(x) = 1$*.*

The goal of a distributional problem is the following: given $x \leftarrow D_n$, find $y \in \{0,1\}^*$ such that $(x,y) \in \Pi$. Note that this is slightly different from a typical NP problem in which the goal is merely to determine if y exists.

Next, we need to consider which algorithms are "good" at solving a given distributional problem. Let's consider two algorithms A and B that solve Π, meaning that for all x, $(x, A(x)) \in \Pi$ and the same is true for B. Let's suppose their running times are as follows, for x chosen according to distribution D_n:

1. With probability $1 - 2^{-\sqrt{n}}$, A runs in time n^2, and with probability $2^{-\sqrt{n}}$, A runs in time 2^n.
2. For all $c \in \mathbb{N}$, algorithm B runs in time $\geq n^c$ with probability $\leq \frac{1}{n^{(c^2)}}$.

Note that the expected running time of A is exponential. However, intuitively it seems like a useful algorithm because it usually runs quickly. Therefore, expected running time is not a useful metric for capturing our intuitive notion of "good." Another way of thinking about algorithm A is to consider a polynomially-bounded observer that samples $x \leftarrow D_n$. The observer cannot hope to find an input x for which algorithm A takes a long time, and so from the point of view of the observer, A really does run in time n^2.

On the other hand, the expected running time of B is a polynomial, but using the previous reasoning we categorize algorithm B as "bad." Consider an observer that runs in time n^d and chooses instances $x \leftarrow D_n$. This observer can find an instance x for which algorithm B takes time $n^{\sqrt{d}}$ to compute $B(x)$. Allowing d to become large, we notice that by creating observers with larger and larger polynomial bounds, we can

determine instances of the problem on which B performs arbitrarily bad. Therefore, we consider B to be a poor algorithm for the distributional problem (Π, D).

Since expected running time does not reflect our intuitive notions of "good" and "bad" algorithms, we must come up with a different definition.

Definition 2 *Algorithm A runs in "average time" $T(n)$ solving (Π, D) if for all c and for sufficiently large n,*

$$\Pr_{x \leftarrow D_n}[A \text{ takes time } \geq T(n) \text{ to compute } A(x), \text{ or } A(x) \text{ is incorrect}] \leq \frac{1}{n^c}.$$

Additionally, we can allow A to be randomized and consider the probability also over the random coin tosses of A.

Motivated by this definition, we form the class of "easy" distributional problems in the obvious way. Define

$$\text{Avg-BPP} = \{(\Pi, D) : \exists \text{ algorithm } A \text{ and poly } p(n) \text{ s.t. } A \text{ runs in time } p(n) \text{ solving } (\Pi, D)\}.$$

5 Hard problems

Next, we wish to define a class of distributional problems that is analogous to NP for decision problems. One obvious choice is to form the class

$$\text{DNP}_1 = \{(\Pi, D) : \Pi \in \text{NP and } D \text{ is a distribution}\},$$

where the D stands for distributional. Unfortunately, this definition is not very interesting by the following theorem.

Theorem 3 *If* NP $\not\subseteq$ prBPP*, then* $\text{DNP}_1 \not\subseteq$ Avg-BPP*.*

Proof If NP $\not\subseteq$ prBPP, then we can say without loss of generality that $SAT \notin$ prBPP. Suppose that algorithm A runs in probabilistic polynomial time and attempts to solve satisfiability. Since $SAT \notin$ prBPP, it must fail some of the time, and let $D^{\text{adv}}_{A,n}$ be the uniform distribution on the set

$$\{x \mid A(x) \text{ incorrect for satisfiability}\}.$$

Then, A never solves the distributional problem (SAT, D^{adv}_A), where $D^{\text{adv}}_A = \{D^{\text{adv}}_{A,n}\}_{n \in \mathbb{N}}$ is the ensemble of all of the probability distributions created above. Unfortunately, this is not sufficient because we want one distribution that fails for all algorithms.

Let $A_1, A_2, A_3, \ldots$ be an enumeration of all probabilistic polynomial time Turing machines, and form the distribution

$$D^{\text{adv}}_n = \sum_i \frac{1}{i^2} D^{\text{adv}}_{A_i,n},$$

and then form the corresponding ensemble $D^{\text{adv}} = \{D^{\text{adv}}_n\}_{n \in \mathbb{N}}$. For all i, algorithm A_i fails on (SAT, D^{adv}) with probability at least $\frac{1}{i^2}$. As a result, $(SAT, D^{\text{adv}}) \in \text{DNP}_1$ but $(SAT, D^{\text{adv}}) \notin$ Avg-BPP, so it follows that $\text{DNP}_1 \not\subseteq$ Avg-BPP as desired. ■

If we regard DNP_1 as our average-case variant of NP, then this theorem basically says that problems that are hard in the worst-case are also hard in the average-case. However, the theorem is not very interesting because it constructs a distribution D^{adv} that is very contrived and unlikely to come up in practice. Essentially, we are being too harsh by presuming that nature can come up with such a contrived distribution.

Instead, we think of nature as having a source of pure randomness, and then being able to apply any easy (polynomial time) algorithm to its source of randomness. Motivated by this view, we make the following definition.

Definition 4 *A distribution D is "sampleable" if there exists a deterministic polynomial time Turing machine M that maps inputs of length n to outputs of length n such that for all $x \in \{0,1\}^n$,*

$$\Pr_{s \in_U \{0,1\}^n}[G(s) = x] = D(x).$$

Now we define $\text{DNP} = \{(\Pi, D) : \Pi \in \text{NP}$ *and D is a sampleable distribution*$\}$.

Under this definition of DNP, we believe that $\text{DNP} \not\subseteq$ Avg-BPP but we do not know how to prove it, even under a worst-case assumption like $\text{NP} \not\subseteq$ prBPP. Note that the opposite inclusion does not hold because there are (uninteresting) problems in Avg-P (the deterministic variant to Avg-BPP) that are not in DNP: one example is to take $\Pi = \{0,1\}^* \times \{0,1\}^*$ so that every y is a witness for every x, but to make the distribution D such that it is not sampleable.

In order to compare DNP and Avg-BPP, we wish to develop the same tools that we use to compare NP and P, which yields the following questions:

1. What is the concept of a reduction between two distributional problems?
2. Under this reduction, is there a notion of a "complete" problem in DNP?
3. Are there natural examples of DNP-complete problems?

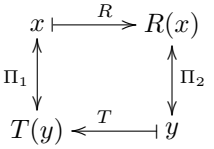
In this lecture, we'll answer the first question and propose a candidate solution to the second question. We'll continue to study the idea of DNP-completeness in the next lecture.

6 Reductions and complete problems

The notion of deterministic reductions is pretty straightforward: a well-defined function is used to meet simple requirements. Probabilistic reductions, like the Valiant-Vazirani reduction from SAT to unique-SAT, are already more interesting to define because the reduction is randomized and does not always succeed. In the context of distributional problems, the reduction can be randomized *and* the problem inputs must fit a given distribution, so making a rigorous definition is even more difficult.

Intuitively, we want a reduction $(\Pi_1, D_1) \leq (\Pi_2, D_2)$ of distributional problems to consist of a mapping $x \mapsto R(x)$ with the following properties:

1. If x is distributed according to D_1, then $R(x)$ is distributed according to D_2.
2. If a witness y is found to $R(x)$, then it should be easy to map y back into a witness for x. Formally, there exists a function T such that $(R(x), y) \in \Pi_2$ implies that $(x, T(y)) \in \Pi_1$.



It turns out that the first condition is a bit too restrictive. The intuitive idea is that $R(x)$ does not have to be distributed just like D_2, but it should have the same basic structure so it should not be too concentrated at points that D_2 is not interested in. We formalize this idea with the following definition.

Definition 5 *Given two distributions D_2 and D_2' and a function $\alpha : \mathbb{N} \to \mathbb{R}^+$, we say that D_2 "α-dominates" D_2' if for all n and for all $x \in \{0,1\}^n$, $D_2'(x) \leq \alpha(n) \cdot D_2(x)$.*

Now we replace condition 1 with the following criterion: if x is distributed according to D_1, then the distribution of $R(x)$, which we call D_2', has the property that there exists a polynomial α such that D_2 α-dominates D_2'.

Defining a concept of reduction becomes more difficult when R is allowed to be a randomized reduction. However, if we restrict ourselves to looking at deterministic reductions, then it is possible to prove that no reductions exist where D_2 is a certain type of uniform distribution. This is undesirable, so we do allow R to

be randomized, and we also soften condition 2 slightly by saying that it only has to hold when the random coins of R are "good," and the reduction can do anything when the coins are "bad." Writing out the formal definition of the reduction is complicated, so we avoid doing so here.

Now that we have a notion of reductions, we look at the question of completeness. One NP-complete problem is the following language:

$$L = \{(M, x) : M \text{ is a nondeterministic TM that runs in time } n^2, \text{ and } x \in L(M)\}.$$

The relation that decides L is the relation

$$\Pi((M, x), y) = \begin{cases} 1 & \text{if } M \text{ runs in time } n^2 \text{ and } y \text{ is a witness to the fact that } x \in L(M), \\ 0 & \text{otherwise.} \end{cases}$$

We can form a distributional problem using Π by creating a uniform distribution D as follows:

- Select the length of M by choosing M to have length l with probability 2^{-l}. Then, choose $M \in_U \{0,1\}^l$.
- Select the length of x by choosing x to have length n with probability $\frac{1}{n^2}$. Then, choose $x \in_U \{0,1\}^n$.

In the next lecture, we will prove that the distributional problem (Π, D) is DNP-complete.

7 Reference

This lecture is based on Section 10.2.1 of Oded Goldreich's book *Computational Complexity: A Conceptual Perspective*. Goldreich's book explains all of the definitions that we covered in class and also provides a precise definition of reductions between distributional problems. Goldreich's book is available online at the website `http://www.wisdom.weizmann.ac.il/~oded/cc-drafts.html` (there is a link to this page from the 6.841 course website).