6.841 Advanced Complexity Theory May 4, 2009

Lecture 23

Lecturer: Madhu Sudan *Scribe: Rishi Gupta*

Project Presentations

Presentations are this Wednesday 9am-12pm and Thursday 9am-1pm. Slots may be moved around to space them out/ensure we don't run up against room reservation deadlines.

The 20 minute limit will be enforced. Powerpoint is highly recommended, since you won't be able to write that much or get through much material in a 20 minute interactive session. If you do write on the board, you should have all your intermediate steps planned out beforehand. Also, know what you're going to do if we do make it interactive and you have a few minutes less than you think you do.

Send in your paper and presentation before you present, preferably at least 24 hours before.

Average-Case Complexity

Today we're going to give an overview of *Average-Case Complexity*. This was originally a three day lecture, and now it's a one day lecture; what's being lost is the proofs. We will give motivation for the field, some definitions, and a flavor of the results.

Motivation

We've already seen that worst-case instances of Permanent reduce to random instances of Permanent (Lipton), implying that for Permanent the average-case complexity equals the worst-case complexity. This is not very encouraging.

However, we hear about "SAT-solvers" and other algorithms for NP-hard problems all the time. Sometimes people are just mistaken (eg. they don't realize their graphs have some hidden constraint), sometimes they are just using an approximation algorithm, but sometimes it legitimately seems like the empirical, real-life problems are simply not the worst-case ones.

The key is, the distribution of real-life problems is not the same as the distribution of random problems (say, of Clique). People are very interested in finding algorithms that work on real-life distributions of NP. An example might be compiler optimization.

Another application of average-case complexity is cryptography. People would like to find functions f such that f is easy to compute and f^{-1} is hard to compute (where easy/hard refer to polynomial-time, as usual). More strongly, people want functions f for which they both know the value of f^{-1} on some inputs x_i and where it's *still* hard to compute f^{-1} on those x_i. This means it's not just sufficient to solve all the average-case or easy instances of a function and declare rest as hard.

Interesting papers on the subject:

- Impagliazzo ('95): A personal view of average-case complexity. Gives five possible universes of the field going from the universe if P=NP to cryptomania where there exist trap door one-way fuctions.
- Goldreich ('08): Computational Complexity: A Conceptual Perspective. The chapter on average-case complexity talks about definitions, which is more interesting than you think. Eg. the first definitions people thought of were correct, but not very intuitive.
- Bogdanov-Trevisan('06): Average-case Complexity. Technical approach.

Algorithmic Developments

There are three main groups of people looking at this. The approaches intersect in a Venn diagram, in the sense that there's some overlap and some independence in each approach.

1. Random Graph Model, $G_{n,p}$. The goal is to find for instance a clique, or hamiltonian cycle, in random graphs over n verticies, where each edge is chosen with probability p.

   We have results like: if $p >> (\log n)/n$, there's an algorithm that runs in expected polynomial time that takes a random graph and terminates with a hamiltonian cycle with probability 1-o(1). Note that o(1) can probably be replaced with an exponential in n.

   We think of it as a probability because not every graph has a hamiltonian cycle, and we allow our algorithm to occasionally not find one even if one is there. Note that we can also just fix the running time to some polynomial instead of worrying about "expected" polynomial time.

2. Random (3)-CNF formulae. Statistical physicists like this one. We select m clauses independantly and at random from the $8 \cdot \binom{n}{3}$ possibilities.

   There're not too many theorems here, but some interesting conjectures. It seems like there exist α and β such that $m/n > \alpha \Rightarrow$ the formula is not satisfiable with high probability, and $m/n < \beta \Rightarrow$ the formula is satisfiable with high probability. There's also a gap from β to some β_{alg} where $m/n < \beta_{alg} \Rightarrow$ there's a known algorithm that finds a satisfying assingment with high probability.

   Physicists think $\beta_{alg} = \beta = \alpha$, from empirical evidence. Note that for each n, some such $\alpha(n) = \beta(n)$ does exist. At the threshhold (at $\alpha = \beta$), physicists think that the problems are "hard to decide" one way or the other, though some work has to be done to find definitions that even make that a reasonable claim. Current experimental data: $\alpha = 3.1, \beta = 4.59$, physicists actually claim $\alpha = 3.9$. Note that resolution proofs are exponentially long at every m/n; for small ratios it's because they don't exist.

3. SAT Solvers. Least formal; very few theorems or claims. They often try to prove both sides; eg. if they don't find you a satisfying assignment they'll look for a resolution. Very successful. They fail on a bigger-than-exponential subset of their input, but claim that the failures are almost always only on unsatisfiable formulae, and that failure means that the resolution proof was probably just too long (ie. they think they generally will find a satisfying assignment if there is one.)

Formal Definition (Complexity-Theoretic Examination)

For SAT do we look at SAT on a random 3-SAT distribution? For Random Graphs do we look at $G_{n,p}$ with $p = 1/2, 1/3$ or something else?

All of these are reasonable, and so we just include the distribution in the problem specification. An average-case complexity problem is some (Π, D), where Π is some NP relation and D is a distribution over $\{0,1\}^n$. Some D will be easy, some will be hard; the D is important! Should we allow D to be different for each n, or does it have to be some "easy" function of n? We'll come back to this in the next section.

What does it mean to have an algorithm for such a problem? Our first (incorrect) attempt might be: A solves Π if the expected running time of A on Π over D is poly(n), which we denote $\mathrm{E}_D[\mathrm{T}_A(\Pi)] \in$ poly(n). However, expected running time does not behave nicely under polytime: $\mathrm{E}[\mathrm{T}_A] \in$ poly(n) does not mean $\mathrm{E}[(\mathrm{T}_A)^2] \in$ poly(n). For instance, if D is the uniform distribution, A takes 2^n steps on 1 input, and 1 step everywhere else, $\mathrm{E}_D[\mathrm{T}_A] \sim 2$ but $\mathrm{E}_D[(\mathrm{T}_A)^2] \sim 2^n$.

The correct formulation is: (Π, D) is in Ave-P if there exists an algorithm A and constant c such that $\mathrm{E}_D[\mathrm{T}_A(x)^{1/c}] \in$ poly(n). Not particularly intuitive, as you can see.

Impagliazzo: The definition above is mostly equivalent to: (Π, D) is in Ave-P if there exists A and a polynomial p such that for every polynomial q, $\Pr_D[\mathrm{T}_A(x) > p(n)] < 1/q(n)$, where $n = |x|$ as usual.

Samplable Distributions

It's a priori conceivable that NP != P, but that for all (Π, D) with $\Pi \in$ NP, $(\Pi, D) \in$ Ave-P. A diagonalization argument shows this not to be the case, though, so P != NP implies the existence of hard instances of NP.

Nevertheless, it takes a lot of time (PSPACE) to find such a hard instance. A reasonable distribution should be P-computable; namely, I should be able to tell you the probability of an element x in polytime. In actuality, we require something slightly stronger: we say D is (P-)computable if $\sum_{y<_{lex}x} D(y)$ can be computed in polytime.

We end up wanting something still stronger, though. We say D is *samplable* if there is a function $S \in$ P such that given access to a (long) random string R, $S(n) = x$ with probability $D_n(x)$. For instance, the uniform distribution is samplable since we can throw away x and return the first n bits of R.

This is nice because every real-life distribution is samplable under our model of the universe, namely that there's some inherent randomness on which computation is done to produce the distributions we see. Samplable distributions also play nicely under composition and reductions.

For these reasons, samplable distributions are what people tend to work with. We call the class of NP problems over samplable distributions DNP. So, is DNP $\subset$ Ave-P?

To start with, what problems are DNP-complete? We first need to say what we mean by reductions over distributions. Ideally, we might want to say R reduces (Π_1, D_1) to (Π_2, D_2) iff $R(x) \in \Pi_2 \Leftrightarrow x \in \Pi_1$ and $R(x) \sim D_2$ if $x \sim D_1$. But it seems annoying to have to work out both Π and D at the same time, so maybe we should find a way to allow more tolerance.

In particular, if we have an algorithm for (Π, D_2), when does this imply an algorithm for (Π, D_1) that's only polynomially worse? One case is if $\max_x D_1(x) \leq p(n)/2^n$ for some polynomial p.

We say D_2 ϵ-*dominates* D_1 if $\Pr_D[D_2(x) < D_1(x)/p(n)] \leq \epsilon(n)$, where $n = |x|$ as usual. If ϵ is sub-polynomial, and D_2 ϵ-dominates D_1, then the identity map reduces (Π, D_1) to (Π, D_2) for any Π. This in general also allows us to change ugly distributions into pretty ones.

Completeness results

Some known results:

- Levin ('86): Showed DNP-completeness of an artificial problem. It was artificial because it was some subset of DNP.
- Impagliazzo-Levin: Showed DNP-completness for problems with uniform distribution. Still vaguely artificial.

  Showing something about, say, Random-CNF would count as not-artificial. But these are much harder, and people don't know how to do things with them yet. Perhaps a proof of P != NP will provide some natural distributions that show Random-CNF is hard.
- Worst-case to Average-case reductions: We have reductions $\Pi' \rightarrow (\Pi, D)$, where Π and D are natural, and Π is natural on D. But then we don't have proof that Π' is hard.
- Other results: Ajtai ('96), Regev, Micciancio, Peikert.
- Regev ('00? '02?): Lattice problem: Given a matrix $A \in \mathbb{Z}_{2^n}^{n \times n}$ and some vector b. Find x in $\mathbb{Z}_{2^n}^n$ that minimizes $\|Ax - b\|$ (euclid norm). I assume the 2^n subscript denotes the fact that the computations are done over n-bit integers?

  This is NP-hard in the worst case. The best known poly-time approximation algorithms are within 2^n of optimal. There are interactive proofs for poly(n) approximations (ie. within a poly factor of optimal), which suggests that poly-approximation is not NP-hard.

Real success would be showing that, say, Random-CNF is hard. There have been few such successes so far, but not for lack of trying.