Recent advances in Complexity CIS 6930/CIS 4930 November 21, 2002

Lecture 26

Lecturer: Dr. Meera Sitharam Scribe: Srijit Kamath

1 Derandomizing Specific Classes of Algorithms

Lots of randomized algorithms can be seen as "algorithmizing" a probabilistic existence proof, i.e., one gets a randomized algorithm that with high probability constructs the combinatorial object whose existence the probabilistic proof establishes. In other words the proof serves as "proof of correctness" of the algorithm (although, usually the proof precedes the algorithm). For specific types of such randomized algorithms (i.e., randomized algorithms in whose analysis a particular type of probabilistic existence proof appears), we give a general method of derandomizing them.

1.1 General Overview

Suppose that it is desired to compute $f(x)$ given x ($f(x)$ could be Boolean-valued for example when we are trying to recognize whether x belongs in a set). A randomized algorithm M takes as inputs x and a additionally a string y of m random bits. It outputs $f(x)$ exactly (or approximately) with high probability. The proof of correctness of M will at some point make use of the fact that it gets truly random bits. If we plug truly random bits into a statistical test, it behaves in a certain way and that behavior establishes that M works (with high probability).

Idea: Extract a statistical test out of this segment of the proof. Then establish complexity of this statistical test. Let us say it is C. We can claim that any psuedorandom generator G that fools computations in class C will derandomize M. Note that C has nothing to do with anything in the algorithm. It only has to do with the portion of the proof of correctness of the algorithm that says the use of truly random bits results in a certain output with high probability.

The class of randomized algorithms we will look at are those whose proofs of correctness embed a statistical test that can be computed in $LOGSPACE$ since there is a very good psuedorandom generator for $LOGSPACE$. Recall that $LOGSPACE \subseteq P$. Also many of the common problems known to be in P also lie in $LOGSPACE$. Another interesting thing to note is that $LOGSPACE$ can be thought of as the set of very efficiently parallelizable computations in P (with good speedup), since parallel time (circuit depth) is generally equated to sequential space.

1.2 Proofs with $LOGSPACE$ Statistical Tests

Sivakumar (2001) observed that many existing algorithms randomized algorithms employ $LOGSPACE$ statistical tests. From the ideas presented above and this observation it follows that a pseudorandom generator that fools computations in $LOGSPACE$ will suffice to derandomize many existing algorithms. He classified these randomized algorithms into 3 classes as follows.
Algorithms whose correctness is based on

1. Chernoff-Hoeffding Lemma/Bound.
2. Semidefinite relaxation bound.
3. Johnson-Lindenstrass Lemma.

An example of the problem of the first type is the set descrepancy problem also known as the lattice approximation problem. The two versions of the problem follow.

Version 1: Given m sets $A_1, A_2, \ldots, A_m \subseteq \{1, \ldots, n\}$, the goal is to compute a set Q such that $max_i \Delta_i = ||A_i \cap Q| - |A_i\ Q||$ is minimized.

Version 2: Given m vectors $a_1, \ldots, a_m \in R^n$ with $||a_i||_\infty \leq 1$ for $i = 1, \ldots m$ and a vector p, the goal is to find a vector $q \in \{0,1\}^m$ such that $max_i | < a_i, p > - < a_i, q > |$ is minimized.

Exercise 1 *How are these two versions of the problem related? Hint: Show that version 1 is a special case of version 2. I.e, reduce version 1 to version 2.*

A related problem is the shortest vector problem for lattices. Given an arbitrary basis for R^m, $u_1, u_2, \ldots, u_m$. What is the length of the shortest vector in the lattice that is an integer linear combination of the basis vectors? In other words minimize $\Sigma \lambda_i u_i$. This is the first problem known to have the following property: if its worst case is hard then its average case is hard. Such a result is important to applications like cryptography, where it is not sufficent that the worst cases are hard for the adversary to break, but rather that the average case is hard for the adversary to break.