Recent advances in Complexity CIS 6930/CIS 4930	November 19, 2002
Lecture 25	
Lecturer: Dr. Meera Sitharam	Scribe: Srijit Kamath

1 Construction of Efficient Extractors

Our goal is to contruct effcient extractors that convert weakly random strings to strongly random ones.

It is known that $\forall n, k, \epsilon$, $\exists(k, \epsilon)$ extractor $EXT : \{0,1\}^k \times \{0,1\}^t \longrightarrow \{0,1\}^m$ where $t = O(logn/\epsilon), m = k + t - 2log(1/\epsilon) - O(1)$.

Conjecture: EXT is computable in time polynomial in n.

The best known extractor is due to Trevisan. The parameters are:

- $k = n^{\Omega(1)}, m = k^{1-\alpha}, t = O(logn)$
- any $k, m = k^{1-\alpha}, t = O(log^2 n/logk)$

1.1 Simulation of RP computations

First we explain how to deterministically simulate RP computations using extractors.

Definition 1 *A language $Z \in RP$ if there exists a deterministic polynomial time machine M that takes 2 inputs, z (to be decided in Z or not) and a random string y, $|y| \leq |z|^k$, and*
$z \in Z \Rightarrow$ on atleast 1/2 fraction of strings y, M says "yes".
$z \notin Z \Rightarrow$ on all y, M says "no".

Exercise 1 *Show that 1/2 can be replaced by any constant > 0 to achieve same acceptance probability by running M sufficiently often.*

While designing a pseudorandom generator we need to keep in mind that: (i) It should run in reasonable time and (ii) It's use results in accurate output. Previously we ran the input exhaustively on all random seeds to guarantee accuracy. We will not do that now.

Our simulation for RP computations proceeds as follows. Given X on $\{0,1\}^n$, a weakly random source, get a single element x. From x generate deterministically $y'_1, y'_2, \ldots, y'_{poly(n)}$, strings $\in \{0,1\}^m$, then run M on each y'. We accept

if M accepts on atleast one y_i' (in case of BPP do what M does on majority of y_i's). Clearly the simulation takes polynomial time.

Fact 1 *The above simulation is a $\alpha-$accurate deterministic polynomial time simulation of RP if for every set $W \subseteq \{0,1\}^m$, $|W| \geq 2^m/2$, $P[\exists i : y_i' \in W] \geq 1-\alpha$.*

Note that if $\alpha = 0$ we have a fully accurate simulation.

Proof. If the original RP algorithm M accepted for $\geq 1/2$ of the possible $y \in \{0,1\}^n$, let us call this set of ys as W. By the assumption (in Fact 1) on the extractor, atleast one of the outputs of extractor (with probability $1-\alpha$) $\in W$, and by the simulation rule we accept. If the original RP algorithm M does not accept any one of the ys $\in \{0,1\}^m$ our simulation will not accept either since the y_is $\in \{0,1\}$ as well. ■

Intuitively the extractor outputs a set of strings with the property that if we took any large enough subset we will find one output of extractor in it. The y_i's in Fact 1 are outputs of a 'disperser' which is a weaker form of an extractor.

Definition 2 *A function $DISP : \{0,1\}^n \times \{0,1\}^t \longrightarrow \{0,1\}^m$ is a (k,ϵ) disperser if for every subset $W \subseteq \{0,1\}^m$, $|W| \geq \epsilon 2^m$ and for every random variable X of min entropy $\geq k$, $P[DISP(X,U_t) \in W] > 0$.*

This means that for any enough subset of the output set the disperser has non- zero probability of finding something in that set.

Exercise 2 *Prove that a (k,ϵ) extractor is a (k,ϵ) disperser.*

Fact 2 *For the BPP simulation we actually need the extractor property.*

1.2 Extractors for Derandomization

Recall that in the analysis of NW generator G, we assumed that

$$|P[C(y) = 1] - P[C(G(x)) = 1]| > \frac{1}{n}.$$

We created a circuit D' that computes the function f (from which G was constructed) such that $size(D') \approx size(C)$, provided $f \in EXPTIME$. The analysis continued by showing that it can not be that both f is hard and $size(C)$ is small. Key observations for constructing an extractor EXT from the NW generator are the following.

- The circuit D' takes C as a subroutine inside it. In other words if C is fixed, D' depends only on f (the additional $O(n^2)$ size or time comes from computation of f on strings of $logn$ bits). So G can be thought of as being parameterized by f. No matter what f is, G proceeds in the same manner using the general construction. Call this circuit D_f'. The construction of D' works for any C as a blackbox subroutine. Note that the fact that C belonged to a certain complexity class was eventually used. Here we will not use it as we are considering an arbitrary statistical test instead of C.

- Construction of G works for any f as blackbox. $Size(D')$ beyond $size(C)$ and runtime of G depend on $f \in EXPTIME$.
- The number of fs that have small circuits is small. Of the 2^{2^n} fs only 2^{n^k} have polynomial size circuits.

Idea: Define our extractor $EXT_{NW}(X, U_t) = G_f(x)$, where X is defined over the truth tables of boolean functions f of $logn$ variables; Note that the size of the table is n which is what we want. Think of U_t as a uniform distribution over seeds of t bits, i.e, the seeds x that are input into the pseudorandom generator G_f. .

To show that this extractor is a (k, ϵ) extractor we begin by assuming that it is not one. That means $\exists$ a statistical test T such that $|P[T(EXT_{NW}(X, U_t)) = 1] - P[T(U_m) = 1]| \geq \epsilon$. The first probability is over X and U_t, while the second is over U_m. Now look at how many strings x (values of random variable X) are "bad". Using the first probability over U_t alone, the same inequality holds for such x i.e., $|P_{U_t}[T(EXT_{NW}(x, U_t)) = 1] - P[T(U_m) = 1]| \geq \epsilon$.

Every such bad x can be interpreted as the truth table of a function f that can be computed by a circuit of size $O(m^2)$ with access to an oracle (subroutine) for T. For this fixed T, there are only $O(2^{m^2})$circuits of that type. Therefore the number of bad xs (for T) is bounded by $O(2^{m^2})$. From this we can contradict the original assumption that T distinguished the output of EXT from U_m within ϵ. However there is a catch. Now the NW analysis only gets a D' approximating f (this analysis is sufficient to contradict NW result's strong assumption about f being non approximable by small circuits). This only means that bad xs are within a certain distance of truth tables f computable by small circuits. There could be lots of such bad xs. So our proof will not go through as is. (Note however, that a stronger pseudorandom generator by Impagliazzo and Wigderson which starts from a somewhat weaker assumption that f is just not computable by smaller circuits. Their analysis therefore has to be stronger, and therefore constructs a circuit that exactly computes f – thus if we use their pseudorandom generator, the above proof idea does go through; the problem is that the stronger pseudorandom generator of IW is not as clean in structure, nor as efficient to compute as the weaker NW pseudorandom generator, so we would like to stick with the NW generator if we can, in constructing our extractor.).

The solution is to modify EXT_{NW} as follows on input x (value of random variable X). EXT_{NW} first encodes x into a codeword x' from a code (minimum distance code) that has a good "spread". By this we mean that for any Hamming ball of a certain radius (r) there are at most $k(r)$ codewords inside it. So the number of bad xs that get associated with any one of the functions that gets computed quickly by the circuit (constructed in the NW analysis) is still small and the argument goes through.

1.3 Uses of Extractors

1. Extractors are used in BPP/RP simulations using weak random sources, as we have seen above.

2. Extractors can be viewed immediately as Expander graphs (and superconcentrator graphs). Expander graphs can in turn be used as follows: First note that the probabilistic method for proving existence of certain combinatorial objects often translates to a randomized algorithm for finding this object (or computing its properties)(Chapter 5/6 of Raghavan and Motwani's book).

   (i) Deterministic (probability/accuracy) amplification of *general* probabilistic algorithms (Chapter 5/6 of Raghavan and Motwani's book) can be achieved by a combination of: (a) the probabilistic method for establishing existence of expanders (with certain easily computable/constructible local properties), and (b) showing rapidly mixing properties of random walks on expanders.

   (ii) Using this idea of amplification, expanders give so called "oblivious sample sets" i.e, a set of sample points from which one can obtain various properties of an arbitrary real-valued function (from some class) over a finite domain, in particular the function's expected value. These can be used for derandomizing randomized approximation/learning algorithms.(In this context, note that 3 papers by Sitharam in the reading list on sampling and derandomized learning all rely on construction of oblivious sample sets, not using extractors, but using hardness-based pseudorandom generators).

   (iii) As another application of deterministic amplification, existence and explicit/efficient constructions of expanders can be used show nonapproximability of $NP-hard$ problems (unless $P = NP$) either by proving "gap preserving" reductions to known nonapproximable problems (see Papdimitriou's complexity book), or through direct deterministic amplifications of probabilistically checkable proof verifiers for NP (Result by Zuckermann 95).

   (iv) Explicit/efficient constructions of expander graphs can be used to constructivize probabilistic proofs of existence and derandomize the corresponding algorithms that rely on properties of random graphs (expanders embody their "randomness" properties).

3. Extractors are used to show an unexpected connection between the power of randomness and the seemingly unrelated question of Time vs. Space. An extractor construction from 95 (Saks, Srinivasan, Zhou) proved the following conjecture of Sipser from 88. "If $P \neq RP$, then there is some $\alpha > 0$ such that for any function t, all problems in $DTIME(t)$ have algorithms that require only space of $O(t^{1-\alpha})$ for infinitely many inputs."

4. Recall we used the NW pseudorandom generator to build an extractor (Trevisan's result presented in class). In fact, it is even easier to build pseudorandom generators that fool certain classes of algorithms, starting from extractors. (I.e, get rid of the weak random source, retain only the random seed as input, and on the other hand, only require the output to pass statistical tests in a particular complexity class).