Pseudorandom generators against space-bounded computation

Topics in Complexity Theory and Pseudorandomness (Spring 2013)
Rutgers University
Swastik Kopparty
Scribes: Justin Gilmer, Ben Lund

1 PRGs against space-bounded computation

In this class we will construct pseudorandom generators which fool low space machines. Before going into the construction, we first define what we mean by a low space algorithm.

Definition 1. *Algorithm A is a randomized space S and time T algorithm computing a function $f : \{0,1\}^n \to \{0,1\}$ if the following hold:*

1. *A takes 2 inputs $x \in \{0,1\}^n, r \in \{0,1\}^R$ and outputs a single bit $A(x,r)$, where x refers to the input (read only) tape and r refers to the random bits used by A (viewed as read only tape).*
2. *For all inputs x, $P[A(x,r) \neq f(x)] < 1/3$.*
3. *A uses space S (i.e., the work tape for A has length S), and has running time T.*
4. *A has only 1-way access to the input r. That is, if is currently reading bit k in the input r, it may no longer go back and read bit $k-1$.*

For our purposes, we will view $S = O(log(n)), T = poly(n)$, and $R \leq T$. Requiring the algorithm to have running time T is necessary to limit the power of low space machines. Without such an assumption on the running time, such machines would be able to solve SAT! For example, if $R = 2^n n$, then the algorithm A can simply try all 2^n inputs given by the random bits, if it finds a satisfying assignment, it outputs 1, otherwise it will output 0. If f is not satisfiable, then the algorithm will get the correct answer with probability 1. If f is satisfiable, then each random input it tries has probability $\geq 2^{-n}$ of being a satisfying assignment. Thus the probability of failure would be $\leq (1-2^{-n})^{2^n} < 1/3$.

Assumption 4 is necessary to help with the analysis of our PRG constructions, it is an interesting (and open) question if removing the assumption increases the power of low space machines.

For now, our target theorem is that a randomized space S and time T algorithm implies a deterministic space $O(S)$ and time $O(T)$ algorithm. Such a theorem would be a best possible derandomization result, which, for example, may be applied to the randomized log-space algorithm for undirected connectivity which we saw in a previous lecture.

1.1 Read-once branching programs

A natural way to model low space algorithms is with *read-once branching programs* . A read-once branching program with length R, denoted as $B : \{0,1\}^R \to \{0,1\}$, is a special case of a circuit

defined as follows. There are R layers of nodes $, L_1, L_2, \cdots, L_R, L_{R+1}$. The layer L_1 consists of a single node which is the starting node for the computation, the layer L_{R+1} consists of 2 nodes (a 0 node and a 1 node) which is the output of the computation. Each node in layer L_k reads the same input bit r_k and has two directed edges e_0, e_1, towards nodes in layer L_{k+1}, if $r_k = 0$ then the circuit moves along edge e_0 otherwise the computation moves along edge e_1.

We say a read-once branching program uses space S if $|L_k| = 2^S$ for all k. These programs model low space algorithms in the following way: The nodes in layer L_k correspond to all the possible global states of a randomized algorithm (i.e., what is written in memory, the position of the work head, and the finite state of the machine). Moving from layer L_k to L_{k+1} corresponds to an algorithm reading the random bit r_k, performing some deterministic computation based on this bit, and then ending in some global state in L_{k+1}. This is precisely why we need the assumption that algorithms only have one-way access to the random bits. Any algorithm which uses space S has 2^S possible strings written in memory, S possible positions of the work head, and some constant number of finite states of the machine. Thus the number of global states of such a machine is $O(S2^S)$. This shows that a space S with length R branching programs models space $O(S/log!(S))$ machines which use R random bits.

We now state a theorem which reduces the problem of derandomizing space $O(S/log(S))$ algorithms to providing a PRG fooling read-once branching programs which use space S.

Theorem 2. *Suppose $G : \{0,1\}^d \to \{0,1\}^R$ is such that $G(x)$, for x chosen uniformly from $\{0,1\}^d$, ϵ-fools all space S branching programs. Then, for all randomized space $O(S/\log(S))$ algorithms using R random bits which compute a function f, we have for all x*

$$P_{y \in \{0,1\}^d}[A(x, G(y)) \neq f(x)] < 1/3 + \epsilon.$$

Proof. This follows because we may model $A(x, \cdot) : \{0,1\}^R \to \{0,1\}$ as a read-once branching program which uses space S and length R. □

1.2 The Nisan-Zuckerman generator

Theorem 3. *For all $C \in \mathbb{N}$, there exists a generator with seed length $O(S)$, computable in space S and time poly(S) which ϵ-fools read-once branching programs with space S and length $R = S^C$.*

The construction is as follows. Set $n := 100S$ and let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^S$ be a $(50S, \epsilon)$-extractor. The generator $G : \{0,1\}^n \times \{0,1\}^{ld} \to \{0,1\}^{lS}$ is defined as

$$G(x, y_1, y_2, \cdots, y_l) := (E(x, y_1), E(x, y_2), \cdots, E(x, y_l)) \qquad x \in \{0,1\}^n, y_i \in \{0,1\}^d.$$

We'll let $l = R/S$ and $d = O(log(S) + log(1/\epsilon)$ and prove the following

Claim 4. *The generator G ($l(\epsilon + 2^{-S})$-fools space S branching programs.*

Proof. Let B be a space S branching program which takes as input a string $r = (r_1, r_2, \cdots, r_R)$. Let $L_1, \cdots, L_R, L_{R+1}$ be the layers of nodes for B. We make a slight abuse of notation and use $B(r_1, \cdots, r_i)$ to denote the node in layer L_{i+1} which B arrives at after reading the first i inputs.

Pick $x \in \{0,1\}^n$ uniformly at random and let X be the corresponding random variable. Also pick $y_i \in \{0,1\}^d$ uniformly at random. Let w_i be the vector $(E(x, y_1), E(x, y_2), \cdots, E(x, y_i))$. By

induction on i we prove that the distribution of $B(w_i)$, which is a distribution over nodes in layer L_{i+1}, is $i(\epsilon+2^{-S})$-close to the distribution $B(Z_i)$, where Z_i is uniformly distributed in $\{0,1\}^{Si}$. The base case follows because E is a $(50S, \epsilon)$-extractor. We assume for induction that the hypothesis is true for case i and prove it for case $i+1$. For $v \in L_{Si}$, define p_v to be the probability that $B(w_i) = v$. Denote by X_v the random variable for X conditioned on the event $B(w_i) = v$.

Claim 5. $H_\infty(X_v) \geq n - log(1/p_v)$.

Proof. For $a \in \{0,1\}^n$ we have

$$\mathbb{P}[x = a \mid B(w_i) = v] = \frac{\mathbb{P}[x = a]}{\mathbb{P}[B(w_i) = v]} \leq \frac{2^{-n}}{\mathbb{P}[B(w_i) = v]}.$$

□

Call v bad if $p_v \leq 2^{-2S}$. Then

$$\mathbb{P}[B(w_i) \text{ is bad}] \leq \sum_{v \text{ bad}} p_v \leq 2^S 2^{-2S} \leq 2^{-S}. \tag{1}$$

Our goal is to show that the distribution of $B(w_{i+1})$ is $(i+1)(\epsilon + 2^{-s})$-close to $B(Z_{i+1})$. We chose an element v' in $L_{s(i+1)}$ according to the distribution $B(w_{i+1})$ in the following way: First choose $v \in L_{si}$ according to the distribution $B(w_i)$. Then choose $x \in \{0,1\}^n$ according to the distribution X_v, and choose $y_{i+1} \in \{0,1\}^d$ uniformly at random. Finally choose $v' \in L_{s(i+1)}$ by evaluating the branching program starting at v with input $E(x, y_{i+1})$.

By the induction hypothesis, the distribution of v is $i(\epsilon + 2^{-s})$-close to uniform. If v is not bad, then by Claim 5, $H_\infty(X_v) \geq 100s - 2s = 98s$. In this event, we have that $E(x, y_{i+1})$ is ϵ-close to uniform. Furthermore, the probability that v is bad is $\leq 2^{-s}$. Putting this all together, we have that the statistical distance of the distributions $B(w_{i+1})$ and $B(Z_{i+1})$ is

$$\leq i(\epsilon + 2^{-s}) + 2^{-s} + \epsilon = (i+1)\epsilon + 2^{-s}.$$

□

2 PRGs producing $poly(n)$ bits fooling logspace

The following construction is due to Impagliazzo, Nisan, and Wigderson.

Theorem 6. *For any $\epsilon > 0$, there is a pseudorandom generator taking a uniformly random $O_\epsilon(\log^2 n)$ seed and producing n bits that ϵ-fool any $s = O_\epsilon(\log n)$ space read-once branching program.*

Proof. The construction is as follows. Let t, d be constants to be fixed later. Let $G_0 : \{0,1\}^t \to \{0,1\}$ be a function that returns the first bit of the input; i.e.

$$G_0(x) = x_1.$$

Let $G_i : \{0,1\}^{t+id} \to \{0,1\}^{2^i}$ be a function such that

$$G_i(x,y) = G_{i-1}(x)G_{i-1}(E_i(x,y)),$$

where x is the first $t + (i-1)d$ bits of input, and $E_i : \{0,1\}^{t+(i-1)d} \times \{0,1\}^d \to \{0,1\}^{t+(i-1)d}$ is a $(t + (i-1)d - 2s, \epsilon')$-extractor, with ϵ' to be chosen later.

We can take E_i to be the adjacency map of a good absolute eigenvalue expander. Let $k = t + (i-1)d - 2s$, and let $n' = t + (i-1)d$. If $E_i : \{0,1\}^{n'} \times \{0,1\}^d \to \{0,1\}^{n'}$, then we get $d = O(n' - k + \log(1/\epsilon')) = O(2s - \log(1/\epsilon'))$.

To fool a read-once branching program $B : \{0,1\}^n \to \{0,1\}$, we'll need to use $G_{\log n}$, which will use $t + d \log n$ random bits. It will turn out that we can take $\epsilon' = 1/poly(n)$, so $d = O(\log n)$. So, we'll need an $O(\log^2 n)$ length random seed.

Claim: G_i $(\epsilon' + 2^{-s})(2^{i+1} - 1)$-fools read-once branching programs of space s.

We'll prove the claim by induction on i.

For v in layer 2^{i-1} of the branching program, let $w = G_{i-1}(x)$, and let $p_v = Pr[B(w) = v]$. Let $X_v = X|_{B(w)=v}$.

Claim: $H_\infty(x_v) \geq t + (i-1)d - \log(1/p_v)$.

Call v bad if $p_v \leq 2^{-2s}$; then $\sum_{v \text{ bad}} p_v \leq 2^s 2^{-2s} \leq 2^{-s}$. For good v, $H_\infty(X_v) \geq t + (i-1)d - 2s$.

Pick v according to p_v. By induction, this is $(\epsilon' + 2^{-s})(2^{i+1} - 1)$-close to the v chosen from $B(z)$, for z uniformly random. If v is good, then $E(x,v)$ is ϵ'-close to $U_{t+(i-1)d}$, so $G_{i-1}(E(x_v, y))$ is ϵ'-close to $G_{i-1}(U_{t+(i-1)d})$.

Let B_v be the length 2^{i-1} branching program starting at v. We know $B_v(G_{i-1}(U_{t+(i-1)d}))$ is ϵ'-close to $B_v(G_{i-1}(U_{2^{i-1}}))$. By induction, $B_v(G_{i-1}(U_{t+(i-1)d}))$ is $(\epsilon' + 2^{-s})(2^i - 1)$-close to $B_v(U_{2^{i-1}})$.

So, for v chosen from the distribution of $G_{i-1}(U_{t+(i-1)d})$ and v' chosen from the distribution of $B(U_{2^{i-1}})$, we have that $(v, B_v(G_{i-1}(E(x,y))$ is $(\epsilon' + 2^{-s})(2^i - 1) + 2^{-s} + \epsilon' + (\epsilon' + 2^{-s})(2^i - 1)$-close to $(v', B_v(U_{2^{i-1}}))$.

Since $i \leq \log n$, provided that s is a sufficiently large multiple of $\log n$ and $\epsilon' 2^{\log n}$ is sufficiently small relative to ϵ, it follows that $G_{\log n}$ ϵ-fools any branching program of size s, using $O(\log^2 n)$ random bits. $\square$