CS 880: Pseudorandomness and Derandomization 4/8/2013

Lecture 19: Pseudorandomness for Regular Branching Programs

Instructors: Holger Dell and Dieter van Melkebeek Scribe: Nick Pappas

DRAFT

Until now, our exploration of pseudorandomness for ROBP only exploits that "small width" indicates small-entropy loss of the seed conditioned on the middle state. In this section we consider exploiting more structural properties of ROBP. Towards this end, we will prove that INW generator fools poly-logarithmic width *regular* branching programs using seed length $O(\log n \log\log n)$ (the dependence on error ε will be elaborated later).

An ROBP is regular if every vertex (except the starting one) has exactly two incoming edges. In the highest level, we will exploit regularity to improve the ordinary INW hybrid-argument analysis: The key concept we shall exploit is the *weight* of a program.

To get some first intuitions about weight, let's consider, for each vertex v, the quantity $q(v)$ which is the accepting probability of running the program starting at v *over uniform input*. An important observation is that, if for a layer i and any two vertices u and v in that layer, we have $q(u) = q(v)$, then we could forget about all the layers before i. This is simply because the end goal of pseudorandomness is to approximate the accepting probability, and no matter which vertex we start, we will end with the same.

To capture this intuition, for each edge (u, v), we define the *weight of the edge* to be $|q(u)-q(v)|$. Then, in the above case, every edge before layer i has weight zero. This leads us to a natural measure of weight of a program, by summing up the weights of all edges:

Definition 1 (Weight of a Program). *Given an ROBP P, its weight is defined to be the sum of weights of all edges.*

Intuitively, a program with small weight means that most part of the program has little impact on the accepting probability, and thus could be "forgotten". To analyze the effect of weight, we introduce the concept of *valuation function*. Given a program P on m inputs, its valuation function is a function $\mathrm{val}_P : \{0,1\}^m \mapsto \mathtt{R}$, so that on input $x \in \{0,1\}^m$, returns $q(\mathrm{path}(x)_{m+1})$, where $\mathrm{path}(x)$ is the sequence of vertices induced by starting vertex of P and reading input x.

We will use INW generator of the following form

$$G_j : \{0,1\}^{k+kj} \mapsto \{0,1\}^{2^j}$$

where the extractor we use one with the following special property:

Fact 1. *For any $\beta > 0$, there exists $k = \Theta(\log \frac{1}{\beta})$ and an explicit extractor $E_j : \{0,1\}^{kj} \times \{0,1\}^k \mapsto \{0,1\}^{kj}$ such that the following holds: let $z = (z_0, z_1, \ldots, z_{j-1}) \sim \{0,1\}^{kj}$, and $z_i \sim \{0,1\}^k$, then for any event A that only depends on z and $\Pr_z[A] \geqslant \beta$, $\Delta\Big(E_j(z \mid A, z_i), U_{kj}\Big) \leqslant \beta$.*

Proof. Let X be the uniform distribution over $\{0,1\}^{kj}$, let's consider the min-entropy of X conditioned on A, which is defined to be

$$\mathrm{H}_\infty(X|A) = \log \frac{1}{\max_x \Pr\Big[X = x|A\Big]}$$

Note that

$$\Pr\Big[X = x|A\Big] = \frac{\Pr\Big[X = x, A\Big]}{\Pr\Big[A\Big]} \leqslant \frac{\Pr\Big[X = x\Big]}{\Pr\Big[A\Big]} \leqslant \frac{\Pr\Big[X = x\Big]}{\beta}$$

Thus $\mathrm{H}_\infty(X|A) \geqslant \mathrm{H}_\infty(X) - \log(1/\beta)$. It follows that the source has entropy at least $n - \log(1/\beta)$ conditioned on A. Using spectral extractor, we have that by supplying a seed of length $\log(1/\beta) + \log(1/\varepsilon)$ the output of the extractor is ε-close to uniform, the fact follows by setting $\varepsilon = \beta$. □

For a program P reading 2^j input bits, our end goal is to bound

$$\Big|\mathbb{E}_{x\sim U_{kj}, y\sim U_k}\Big[\mathrm{val}_P(G_j(x,y))\Big] - \mathbb{E}_{u_1,u_2\sim U_{2^j-1}}\Big[\mathrm{val}_P(u_1,u_2)\Big]\Big| \tag{1}$$

The bound will depend on "the weight" of P, so we write the bound as $e_j(P)$. By definition, (1) is nothing more than

$$\Big|\mathbb{E}_{x\sim U_{kj}, y\sim U_k}\Big[\mathrm{val}_P\Big(G_{j-1}(x), G_{j-1}(E_j(x,y))\Big)\Big] - \mathbb{E}_{u_1,u_2\sim U_{2^j-1}}\Big[\mathrm{val}_P\Big(u_1,u_2\Big)\Big]\Big|$$

As the first step to derive the bound, we insert a hybrid $\mathrm{val}_P\Big(G_{j-1}(x), u_2\Big)$. Thus it suffices to bound

$$\Big|\mathbb{E}_{x\sim U_{kj}, y\sim U_k}\Big[\mathrm{val}_P\Big(G_{j-1}(x), G_{j-1}(E_j(x,y))\Big)\Big] - \mathbb{E}_{x\sim U_{kj}, u_2\sim U_{2^j-1}}\Big[\mathrm{val}_P\Big(G_{j-1}(x), u_2\Big)\Big]\Big| \tag{2}$$

$$\Big|\mathbb{E}_{x\sim U_{kj}, u_2\sim U_{2^j-1}}\Big[\mathrm{val}_P\Big(G_{j-1}(x), u_2\Big)\Big] - \mathbb{E}_{u_1,u_2\sim U_{2^j-1}}\Big[\mathrm{val}_P\Big(u_1,u_2\Big)\Big]\Big| \tag{3}$$

The following fact indicates that (3) can be bounded using e_{j-1}.

Fact 2. $\mathbb{E}_{u\sim U_r}[\mathrm{val}_P(x,u)] = \mathrm{val}_P(x)$.

Let's be very careful, we have made it clear that the bound e_j shall depend on the structure of a program. In this case, e_{j-1} depends on the first half of P. Thus we decompose P into two halves Q and R, so (3) is bounded by $e_{j-1}(Q)$.

To bound (2), we write it according to x,

$$\Big|\mathbb{E}_x\Big[\mathbb{E}_y\Big[\mathrm{val}_P\Big(G_{j-1}(x), G_{j-1}(E_j(x,y))\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_P\Big(G_{j-1}(x), u_2\Big)\Big]\Big]\Big|$$

We rewrite this expression according to the state we reach after reading $G_{j-1}(x)$. Formally, for $b \in [w]$, let A_b be the event that we reach vertex b in layer $2^{j-1}+1$ after reading $G_{j-1}(x)$, and R_b be the program starting at b, thus the above can be written as

$$\begin{aligned}
&\left|\sum_b \Pr_x[A_b]\Big(\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(E_j(x|A_b,y))\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_{R_b}\Big(u_2\Big)\Big]\Big)\right| \\
\leqslant &\sum_b \Pr_x[A_b]\left|\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(E_j(x|A_b,y))\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_{R_b}\Big(u_2\Big)\Big]\right| \qquad (4)
\end{aligned}$$

We need the following simple fact:

Fact 3. *For any two distributions X, Y*

$$\Big|\mathbb{E}[\mathrm{val}_P(X)] - \mathbb{E}[\mathrm{val}_P(Y)]\Big| \leqslant \mathrm{weight}(P)\cdot\Delta(X,Y)$$

There are two cases

(I). $\Pr_x[A_b] < \beta$,

$$\left|\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(E_j(x|E_b,y))\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_{R_b}\Big(u_2\Big)\Big]\right| \leqslant 1$$

This is because both expectations are convex combinations of values in $[0,1]$.

(II). $\Pr_x[A_b] \geqslant \beta$, then Fact 1 tells that $G_{j-1}(E_j(x \mid A_b, y))$ is β-close to $G_{j-1}(U_{kj})$, thus

$$\begin{aligned}
&\left|\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(E_j(x|A_b,y))\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_{R_b}\Big(u_2\Big)\Big]\right| \\
\leqslant &\left|\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(E_j(x|A_b,y))\Big)\Big] - \mathbb{E}\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(U_{kj})\Big)\Big]\right| + \left|\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(U_{kj})\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_{R_b}\Big(u_2\Big)\Big]\right| \\
\leqslant &\ \mathrm{weight}(R_b)\cdot\beta + e_{j-1}(R_b)
\end{aligned}$$

Let $B = \{b \ : \ \Pr[A_b] < \beta\}$, and put $e_j(P) := a_j \mathrm{weight}(P)$, thus (4) is bounded as follows:

$$\begin{aligned}
&\sum_b \Pr_x[A_b]\left|\mathbb{E}_y\Big[\mathrm{val}_{R_b}\Big(G_{j-1}(E_j(x|A_b,y))\Big)\Big] - \mathbb{E}_{u_2}\Big[\mathrm{val}_{R_b}\Big(u_2\Big)\Big]\right| \\
= &\sum_{b\in B}\Pr_x[A_b]\cdot 1 + \sum_{b\notin B}\Pr_x[A_b]\cdot\Big(\mathrm{weight}(R_b)\cdot\beta + e_{j-1}(R_b)\Big) \\
\leqslant &\sum_{b\in B}\Pr_x[A_b]\cdot 1 + \sum_{b\notin B}\Pr_x[A_b]\cdot\Big(\mathrm{weight}(R)\cdot\beta + e_{j-1}(R)\Big) \\
\leqslant &\sum_{b\in B}\beta + \sum_{b\notin B}\Pr_x[A_b]\cdot\Big(\mathrm{weight}(R)\cdot\beta + e_{j-1}(R)\Big) \\
\leqslant &\ w\beta + \Big(\mathrm{weight}(R)\cdot\beta + e_{j-1}(R)\Big) \\
= &\Big(w + \mathrm{weight}(R)\Big)\cdot\beta + e_{j-1}(R)
\end{aligned}$$

Thus the total error is

$$e_j(P) \leqslant e_{j-1}(Q) + \Big(w + \text{weight}(R)\Big) \cdot \beta + e_{j-1}(R)$$

Let's determine a_j now, expanding and use the fact that $\text{weight}(R) \leqslant \text{weight}(P)$, we have

$$a_j \leqslant a_{j-1} + \Big(w + \text{weight}(P)\Big) \cdot \beta$$

Given $a_0 = 0$, thus it suffices to set $a_j = j \cdot \Big(w + \text{weight}(P)\Big) \cdot \beta$. This gives our main theorem:

Theorem 4. *G_j fools an ROBP P on 2^j inputs with error $j \cdot \Big(w + \text{weight}(P)\Big) \cdot \beta$.*

1 Weight of Regular Programs

For regular programs, indeed we can show that the weight is small.

Theorem 5. *Consider regular programs with only one accept state, then the weight of a regular program is bounded by $2(w-1)$.*

Given width w, we use $(i, j) \in [n] \times [w]$ to denote the j-th vertex at layer i. Let $q((n, j)) = 1$ if it's an accept state and 0 otherwise, then Theorem 5 is a corollary of the following lemma.

Lemma 6. *The weight of a regular program is bounded by $2\sum_{i \neq j \in [w]} |q((n, i)) - q((n, j))|$.*

Because in Theorem 5 we have only one accept state, it follows that this quantity is $2(w-1)$, as desired. Note in general we could have multiple accept states for a regular branching programs, and the maximum bound achieves when half of the states are accept states, the weight is roughly $2 \cdot (w/2)^2 = w^2/2$.

Proof. The lemma is proved by considering following game. We have $2w$ pebbles on real line $q_1, q_2, \ldots, q_{2w-1}, q_{2w}$. We can do the following move, for two numbers q_i, q_j of distance at least 2δ, one can move them towards each other by a distance of δ, and the gain is 2δ. The goal of this game is to maximize the gain.

What is the connection of this game to the weight? Consider two vertices $j, k \in [w]$ at layer i, and ℓ at layer $i-1$ has two outgoing edges to j, k. Suppose we have two numbers $q((i, j))$ and $q((i, k))$ in the game. Then moving them to the middle position, $\big(q(i, j) + q(i, k)\big)/2$, gives gain $|q((i, j)) - q((i, k))|$. This is exactly *the sum of the weights of edges $\ell \to j$ and $\ell \to k$*. Note that, once $q(i, j)$ and $q(i, k)$ are moved to their median, we have two pebbles of value $q(i-1, \ell)$.

Let's start with

$$q_1 = q((n, 1)), q_2 = q((n, 1)), \ldots, q_{2w-1} = q((n, w)), q_{2w} = q((n, w))$$

The crucial thing is that *if we play the game according to the transition function at the current layer (starting with n), we will get into the same game with these numbers set as probability mass at layer $i-1$*: This is because the program is regular, it follows that each vertex contributes *exactly twice* to construct the probability mass in the previous layer. That is, if we play the game according to the transitions at layer i, not only we get exactly two pebbles for each vertex in layer $i-1$, but also it suffices we put two pebble for each vertex at layer i.

It follows that if we move backward layer by layer, the weight of the program is bounded by the maximum gain of the game, which we claim to be

$$\sum_{i\neq j\in[2w]} |q_i - q_j| = 2 \sum_{i\neq j\in[w]} |q((n,i)) - q((n,j))|$$

Two facts establishes the claim: First, we only need to consider moving *neighboring* pebbles (i.e. no pebble between them), second, if we start with $L = \sum_{i\neq j\in[2w]} |q_i - q_j|$, and move two neighboring numbers a, b close to each other by a distance of 2δ, then for any other pebble c, the sum of distance between c and a, and between c and b remains unchanged, and so L only decreases by 2δ. Because L has to be non-negative, it follows that the gain is upper bound by that. Note we can be arbitrarily close to L, so the bound is tight. □

As a result, we have that G_j fools a width-w regular program P on 2^j-inputs with error $O\Big(j \cdot w \cdot \beta\Big)$, using seed length $(k+1)j$ where $k = \Theta(\log \frac{1}{\beta})$. Put $j = \log n$, with total error ε, we have $\beta = O(\frac{\varepsilon}{w \log n})$, and so the seed length is

$$O(jk) = O\bigg(\log n \cdot \bigg(\log w + \log\log n + \log\frac{1}{\varepsilon}\bigg)\bigg)$$

For $w = \text{poly-log}(n)$ and $\varepsilon = 1/\text{poly-log}(n)$, the seed length is $O\Big(\log n \log\log n\Big)$.