0368-4283: Space-Bounded Computation 15/5/2018 – Lecture 9

PRGs for space-bounded computation: INW, Nisan

Amnon Ta-Shma and Dean Doron

1 PRGs

Definition 1. *Let $\mathcal{C}$ be a collection of functions $C : \Sigma^n \to \{0,1\}$. $G : \Sigma^\ell \to \Sigma^n$ is an ε-PRG against $\mathcal{C}$, if for every $C \in \mathcal{C}$,*

$$\left| \Pr_{x \in U_{\Sigma^n}} (C(x) = 1) - \Pr_{y \in U_{\Sigma^\ell}} (C(G(y)) = 1) \right| \leq \varepsilon.$$

We think of each $C \in \mathcal{C}$ as a test, and G ε–"fools" all tests in the class.

Fix some function $C : \Sigma^n \to \{0,1\}$. It defines a set $A \subseteq \Sigma^n$ of accepting inputs of some density p. If we pick L random inputs from Σ^n, then by Chernoff the probability we fall $p \pm \varepsilon$ times into A is at most $e^{-2L\varepsilon^2}$. If we pick $L \geq \frac{\log \frac{1}{\delta}}{\varepsilon^2}$ this is smaller than δ. If We want to be good against a class $\mathcal{C}$ of $|\mathcal{C}|$ tests, all we need is too take $L \geq \frac{\log \frac{|\mathcal{C}|}{\delta}}{\varepsilon^2}$. Thus:

Lemma 2. *Let $\mathcal{C}$ be a family of functions $C : \Sigma^n \to \{0,1\}$. Let $\ell = \log\log|\mathcal{C}| + 2\log(\frac{1}{\varepsilon}) + \log\log\frac{1}{\delta}$. Then all functions $G : \Sigma^\ell \to \Sigma^n$, except perhaps a δ function, are ε–PRG against $\mathcal{C}$. In particular such an ε–PRG exists. The required seed length is $O(\log(nS))$.*

Example 3. *Say we want to fool all circuits over n binary inputs and size at most S (S counts the number of vertices plus the number of edges). The log of the number of such circuits is at most $O(S \log S)$.*

Example 4. *Now we want to fool all branching programs of length T, width W and alphabet Σ. The log of the number of such branching programs is at most $O(TW|\Sigma|\log(W))$. If $T, W, |\Sigma| = \mathrm{poly}(n)$ the required seed length is $\ell = O(\log n)$.*

In the following we will *fool* bounded-width branching programs, and consequently prove:

Theorem 5 ([2, 1])**.** *Let $\mathcal{C}$ be the class of length n, width $W = 2^s$ branching programs, and let $\varepsilon > 0$. Then, there exists a PRG $G\colon \{0,1\}^\ell \to \{0,1\}^T$ against $\mathcal{C}$ with seed-length $\ell = O(\log T \cdot \log(\frac{TW}{\varepsilon}))$ and error ε. G is computable in $O(\ell)$ space.*

As a consequence, we get:

Corollary 6. *There exists a PRG $G\colon \{0,1\}^\ell \to \{0,1\}^n$ against BPL with seed-length $\ell = O(\log^2 n)$ and error $\varepsilon = \frac{1}{\mathrm{poly}(n)}$. G is computable in $O(\ell)$ space.*

We will start with an INW (Impagliazzo, Nisan and Wigderson) type PRGs and then do Nisan's. Historically, [1] is a modification of the breakthrough result of Nisan [2].

As USTCON $\in \mathsf{BPL}$ we reproved Savitch for undirected graphs. We will show that in fact we get more: $\mathsf{BPL} \subseteq \mathsf{DTISP}(\mathrm{poly}(n), \log^2 n)$ [3]. We remark that very little is known in that direction for the connectivity problem over directed graphs (does it have a substantially sub-linear space algorithm running in polynomial time?).

2 The INW generator

2.1 The length-2 case

We start with the length-2 case and $\Sigma = \{0,1\}^{m_1}$. We choose $x \in \Sigma$ at random and feed it to our branching program M. From M's intial state, x brings us to some $\sigma(x) \in [W]$. How many bits of information (out of m_1) did the transition reveal to M? *At most* w bits.

More formally, $H_\infty(X) = m_1$, whereas if we denote S to be the random variable, taking values in $[W]$, that represents the state we have reached after reading a uniform random string of length T, then typically expect $H_\infty(X \mid S = \sigma) \geq m_1 - w$. That is, although when reaching the next layer X is no longer uniform, it still has enough min-entropy so we can extract more randomness from it using a seeded extractor.

Let $\mathsf{Ext}\colon \{0,1\}^{m_1} \times \{0,1\}^d \to \{0,1\}^{m_2}$ be a strong (k,ε) seeded extractor for $k = m_1 - w - \log(1/\varepsilon)$. We define $G_1\colon \{0,1\}^{m_1+d} \to \{0,1\}^{m_1+m_2}$ by

$$G_1(x;y) \;=\; x \circ \mathsf{Ext}(x,y).$$

Definition 7. *We say that $\sigma \in [W]$ is* bad *if $H_\infty(X|S=\sigma) < k$.*

Claim 8. $\Pr[S \text{ is bad}] \leq \varepsilon$.

Proof. Fix a bad $\sigma \in [W]$. Then, there exists $x_0 \in \{0,1\}^{m_1}$ such that $\Pr[X = x_0 \mid S = \sigma] > 2^{-k}$. On the one hand, $\Pr[X = x_0] = 2^{-m_1}$. On the other hand,

$$\Pr[X = x_0] \geq \Pr[X = x_0 \mid S = \sigma] \Pr[S = \sigma] > 2^{-k} \Pr[S = \sigma],$$

so $\Pr[S = \sigma] < 2^{-m_1+k} = 2^{-w-\log(1/\varepsilon)}$. Overall, by the union-bound, the probability that S is bad is at most $|W| 2^{-w-\log(1/\varepsilon)} = \varepsilon$. □

We now have:

Claim 9. *Let M be a binary branching program of length m_1+m_2 and width W. Then, G_1 2ε-fools M.*

Proof. With probability at least $1-\varepsilon$ over $\sigma \in S$, $H_\infty(X \mid S = \sigma) \geq k$. Fix such a σ, and let $X_2 = \mathsf{Ext}(X,Y)$ where Y is uniform over $\{0,1\}^d$ and independent of X. By the properties of the extractor,

$$|(Y, X_2 \mid S = \sigma) - (Y, U_{m_2})| \leq \varepsilon,$$

so the overall error is at most 2ε.

We can also view that scenario as a two-player game. Player A holds $x \in \{0,1\}^{m_1}$, walks according to x over M and sends the state $\sigma \in [W]$ she ended up in to Player B. Player B holds $x_2 \in \{0,1\}^{m_2}$ and a public $y \in \{0,1\}^d$. From σ, Player B can continue walking over M either with x_2 or with $\mathsf{Ext}(x_1,y)$. Up to an error of ε, the players cannot distinguish between the case where x_1, x_2 and y are all uniform and independent and the case where x_1 and y are uniform and independent, but x_2 is a *deterministic* function of x_1 and y, namely $x_2 = \mathsf{Ext}(x, y_1)$. □

2.2 The length-n case

The general PRG is constructed as follows.

$$G_0(x) = x$$

For the induction rule we have two variants:

1. Carve the seed from the input itself.
$$G_t(x \circ x'_t) = G_{t-1}(x) \circ G_{t-1}(\mathsf{Ext}(x, x'_t)).$$
2. Use one public (random) seed per level, using the same seed in all applications at the same level.
$$G_t(x; y_1, \ldots, y_t) = G_{t-1}(x; y_2, \ldots, y_t) \circ G_{t-1}(\mathsf{Ext}(x, y_1); y_2, \ldots, y_t).$$

In either variant:

Lemma 10. *$G_t(U)$ $(2^t - 1)\varepsilon$ fools binary branching programs of width w.*

The proof is by a hybrid argument. Think of a tree of level t, with 2^t leaves, where at each non-leaf we double the block.

Example 11. *For the second variant, and 4 output blocks, we have the 4 distributions:*

- *D_0 is the distribution $(X_1, X_2, X_3, X_4) = U_{m_1} \times U_{m_2} \times U_{m_2} \times U_{m_3}$.*
- *D_1 is the distribution $(X_1, X_2, X_3, \mathsf{Ext}_1(X_3, Y_2))$.*
- *D_2 is the distribution $(X_1, \mathsf{Ext}_1(X_1, Y_1), X_3, \mathsf{Ext}_1(X_3, Y_2))$.*
- *D_3 is the distribution $G(U_{m_1}; U_{d_1+d_2}) = (X_1, \mathsf{Ext}_1(X_1, Y_1), \mathsf{Ext}_1(X_1, Y_2), \mathsf{Ext}_2(\mathsf{Ext}_1(X_1, Y_1), Y_2))$.*

In the first variant we just need extractors, in the second variant we need strong extractors

2.3 Setting parameters

For the first variant: The error accumulates. We apply the extractor with seed length $O(\log(s+\frac{T}{\varepsilon}))$. Assuming $T > s$, this is $O(\log \frac{T}{\varepsilon})$. The entropy loss at each level come because of two reasons:

1. The $s = \log(|W|)$ bits that the machine knows,
2. The extractor's entropy loss, which is at least $O(\log \frac{T}{\varepsilon})$.

We have $\log(T)$ levels. Altogether, we need initial seed length $O(\log T(s + \log(\frac{T}{\varepsilon}))$.

Notice that while the loss at (1) is indeed s, we treat the losses as if they accumulate whereas truly the machine has only s memory, so if we lose s new bits at some time, we must have forgotten information we have learnt before.

The analysis of the second variant is the same, except that the loss at each stage is the seed y_i itself and the blocks keep the same length.

2.4 A comparison of Rozenman-Vadhan and INW

Let's consider the action of the first variant on two levels. We need some notaion. If x is a string, and we partition it to two, then x' is the prefix and x'' the suffix, for the appropriate lengths (we omit the lengths to reduce the clatter). Let's also denote $x[y] = E(x,y)$ for the extractor with appropriate lengths. In this notation:

$$\begin{aligned} G_1(x) &= x' \circ x'[x''] \\ G_{k+1}(x) &= G_k(x') \circ G_k(x'[x'']). \end{aligned}$$

Let's think of Rozenman Vadahan for locally invertible graphs. In this case we generate a sequence (independent of the vertex of the graph in which we start). We used the sequence to generate UTSs for locally invertible graphs and UESs for geberal graphs. Now let us assume the local inversion is the identity function (as we did for the unitary operator for directed, consistently labeled graphs). What is this sequence?

If $x \in [D]$ and the y_is are from $[d_2]$, then it was:

$$\begin{aligned} RV_1(x, y_1) &= x \circ x[y_1] \\ RV_{k+1}(x, y_1, \ldots, y_{k+1}) &= RV_k(x, y_1, \ldots, y_k) \circ RV_k((x, y_1, \ldots, y_k)[y_{k+1}]). \end{aligned}$$

We conclude that the two constructions are the same. Yet, the analysis is very different. This is because:

- The INW analysis uses full fledged extractors with sufficiently low error $\frac{\varepsilon}{T}$, so that even when the errors accumulate they are less than ε. This gives a PRG. The cost is the long seed length (basically, for polynomial width, length and error $O(\log^2 n)$.
- In RV the seed length is constant. The number of applications is super-constant. Thus, we cannot accumulate errors, and we need to argue differently. The alternative argument is that the gap slowly improves. This only solves connectivity for undirected graphs. We also got a UTS for locally invertible graphs, but even at best a UTS is not a PRG. Yet, the construction is much cheaper and takes $O(\log n)$ space.

2.5 Explicitness

Ahmmm... It can be easily done in space which is order the seed length and simultaneously polynomial time.

Can we hope to anything better given that the seed length is $O(\log^2 n)$?

3 Nisan's generator

3.1 The length-2 case

The generator: We take a family of hash functions $\mathcal{H} = \{h : \Sigma \to \Sigma\}$ such that for every $A, B \subseteq \Sigma$,

$$\Pr_{h\in\mathcal{H}}\left[\left|\Pr_{x\in\Sigma}[x \in A \wedge h(x) \in B] - \rho(A)\rho(B)\right| \geq \alpha\right] \leq \frac{1}{\alpha^2|\Sigma|} \quad = \delta \tag{1}$$

for $\alpha = \sqrt{\frac{1}{\delta|\Sigma|}}$, and $\mathcal{H}$ is explicit, it has, say, cardinality $|\Sigma|^2$, and an efficient indexing from $h \in [|\Sigma|^2]$ to the function $h : x \to h(x)$.

We again use

$$G(x; h) \;=\; (x, h(x)).$$

Remark 12. *We have several options for this $\mathcal{H}$: An extractor (the family is indexed by the seed) with output length equals input length, an expander (which is just such an extractor), or a 2-universal family of hash functions (as we saw in the HW), which, we note, is again an extractor of the same kind.*

We want to fool a branching program M of width W, length 2 and alphabet Σ. We can view the action of M as matrix multiplication: Denote

$$A_{i,k} = \{\sigma \in \Sigma \mid M \text{ moves from } i \text{ to } k \text{ with } \sigma\}$$

as the set of all $a \in \Sigma$ and similarly $B_{k,j}$ for moving from k to j in the second level. Then the probability M mover from i to j is

$$\sum_k \rho(A_{i,k})\rho(B_{k,j}),$$

where ρ is the density function. If the two layers are the same, $A = B$ and the two steps compute matrix squaring. We want to show the PRG approximates the product (or squaring). Define a matrix M_h where $M_h[i, j]$ is the probability M goes from i to j when we walk according to $x \circ h(x) \in \Sigma^2$.

With these notations, we can write

$$M[i, j] = \sum_{k=1}^{W} \Pr_{x_1,x_2}[x_1 \in A_{i,k} \wedge x_2 \in B_{k,j}]$$

$$M_h[i, j] = \sum_{k=1}^{W} \Pr_{x}[x \in A_{i,k} \wedge h(x) \in B_{k,j}].$$

Definition 13. *We say that h is α-good for M if for every $i, j, k \in [W]$ it holds that*

$$\left|\Pr_{x\in\Sigma}[x \in A_{i,k} \wedge h(x) \in B_{k,j}] - \rho(A_{i,k})\rho(B_{k,h})\right| < \alpha.$$

The next claim simply follows from the union-bound:

Claim 14. *For every $A \subseteq \Sigma$, $\Pr_{h\in\mathcal{H}}[h$ is not α-good for $A] \leq \delta W^3$.*

Next, we wish to claim that we indeed approximate the transition probabilities. Throughout, we use the induced infinity norm, $\|A\| = \max_i \sum_j |A[i,j]|$. This norm is sub-additive (as is every norm) and sub-multiplicative. Also, if A and B are both stochastic matrices and p is some integer, then $\|A^p - B^p\| \leq p\,\|A - B\|$ (prove it).

Claim 15. *Let h be α-good for A. Then, $\|M - M_h\| \leq W^2\alpha$.*

Proof. Fix some $i, j \in [W]$. Then,

$$\begin{aligned}
|M[i,j] - M_h[i,j]| &= \left|\sum_{k=1}^{W} \Pr_{x_1,x_2}[x_1 \in A_{i,k} \wedge x_2 \in B_{k,j}] - \sum_{k=1}^{W} \Pr_{x}[x \in A_{i,k} \wedge h(x) \in B_{k,j}]\right| \\
&= \left|\sum_{k=1}^{W} \rho(A_{i,k})\rho(B_{k,j}) - \sum_{k=1}^{W} \Pr_{x}[x \in A_{i,k} \wedge h(x) \in B_{k,j}]\right| \\
&\leq \sum_{k=1}^{W} \left|\rho(A_{i,k})\rho(B_{k,j}) - \Pr_{x}[x \in A_{i,k} \wedge h(x) \in B_{k,j}]\right| \leq W\alpha.
\end{aligned}$$

□

To conclude the length-2 case: We saw that h is α-good for A with probability at least $1 - W^2\alpha$, and if indeed so, then $\|M - M_h\| \leq W^2\alpha$.

3.2 The length-n case

The natural idea is to apply the above technique recursively. We will not get greedy, and we will choose a "fresh" hash function for every level of the recursion. Formally, we construct $G_t \colon \Sigma \times \mathcal{H}^t \to \Sigma^{(2^t)}$ as follows:

$$\begin{aligned}
G_0(x) &= x \\
G_t(x; h_1, \ldots, h_t) &= G_{t-1}(x; h_2, \ldots, h_t) \circ G_{t-1}(h_1(x); h_2, \ldots, h_t).
\end{aligned}$$

Remark 16. *This looks similar to the second variant we presented for INW as the hash functions h_i are public, random strings. There is a difference, though. In the second variant before we used:*

1. strong *extrators*
2. *with* shrinking block length.

The extractor we use here cannot be strong because there is an unavoidable entropy loss. Instead, the hash functions can be fixed such that property in Definition 13 holds. The test in Definition 13 is weaker (it's a small subset of the tests of statistical closeness) and, in a sense, we get strongness (we can almost always fix the seed and make it public, and the output is good for it) without the unavoidable entropy loss extractors have. Therefore, in particular, we retain the block length, and this allows x to be $O(s)$ long rather than $O(s \log \frac{T}{\varepsilon})$ in INW.

For simplicity let us assume (w.l.o.g.) that the transition matrix of every layer is the same, let us call it A. We therefore try to approximate A^T. For $h_1, \ldots, h_t \in \mathcal{H}$ define $A_{h_1,\ldots,h_t}[i,k]$ to be the probability over a random $x \in \Sigma$ that i is moved to k via $G_t(x, h_1, \ldots, h_t)$. $A_\emptyset = A$.

Claim 17. *Let $\{h_1, \ldots, h_t\} \subseteq \mathcal{H}$ such that for every $i \leq t$, h_i is α-good for $A_{h_{i+1},\ldots,h_t}$. Then, $\left\|A^{(2^t)} - A_{h_1,\ldots,h_t}\right\| \leq (2^t - 1)W^2\alpha$.*

Proof. By induction on t. For $t = 1$ we already proved it. Next, write

$$\left\|A^{(2^t)} - A_{h_1,\ldots,h_t}\right\| \leq \left\|A^{(2^t)} - A^2_{h_2,\ldots,h_t}\right\| + \left\|A^2_{h_2,\ldots,h_t} - A_{h_1,\ldots,h_t}\right\|.$$

The first term is at most

$$2\left\|A^{(2^{t-1})} - A_{h_2,\ldots,h_t}\right\| \leq 2(2^{t-1} - 1)W^2\alpha = (2^t - 2)W^2\alpha$$

by the induction's hypothesis. The second term is at most $W^2\alpha$ due to the $t = 1$ case. Overall,

$$\left\|A^{(2^t)} - A_{h_1,\ldots,h_t}\right\| \leq (2^t - 2)W^2\alpha + W^2\alpha = (2^t - 1)W^2\alpha,$$

as desired. □

3.3 Setting parameters

Given A, We want to ε-fool A^T for $T = 2^t$. First, for good $h_1, \ldots, h_t$ we want that $\left\|A^{(2^t)} - A_{h_1,\ldots,h_t}\right\| \leq \frac{\varepsilon}{2}$, so we need $T \cdot W^2\alpha \leq \varepsilon$, which implies that we can take $\alpha = \frac{\varepsilon}{T \cdot W^2}$.

The probability over the choices of $\{h_1, \ldots, h_t\}$ that h_t is α-good for A and every h_i for $i < t$ is α-good for $A_{h_{i+1},\ldots,h_t}$ is at least $1 - t\delta W^3$. To get $t\delta W^3 \leq \frac{\varepsilon}{2}$, we need $\delta \leq \frac{1}{\alpha^2|\Sigma|} = \frac{\varepsilon}{2tW^3}$ so we can set

$$|\Sigma| = \frac{2tW^3}{\alpha^2\varepsilon} = \frac{2tT^2W^7}{\varepsilon^3}.$$

The seed length of G_t is $\log|\Sigma| + t \cdot 2\log|\Sigma|$. As $\log|\Sigma| = O(\log(\frac{TW}{\varepsilon}))$, we get a seed length of $\ell = O(\log T \cdot \log(\frac{TW}{\varepsilon}))$, as required by Theorem 5, precisely as in INW.

What is left is the issue of explicitness.

Claim 18. *Every bit of G_t is computable in space $O(\ell)$.*

Proof. For every $j \in \{0,1\}^t$, the j-th element of $G_t(x; h_1, \ldots, h_t)$ is given by

$$h_t^{j_t}(h_{t-1}^{j_{t-1}}(\cdots h_1^{j_1}(x)))$$

where h_i^1 means applying h_i and h_i^0 means taking the identity. As x and $h_1, \ldots, h_t$ are written on the input-tape, the explicitness follows from the fact that we can evaluate every h using $O(\log|\Sigma|)$ space. This is similar to the second variant of INW we have seen.

We have precisely the same seed length $(x, j_1, \ldots, h_t)$ as in INW second variant (up to constant factors) and precisely the same nice feature that any leaf in the tree can be very efficiently calculated once we have the h_i's. So what have we gained?

1. We can fix h_i and *check* they are good.
2. Since x retains its length it can be $O(\log n)$ long rather than $O(\log^2 n)$.

We will use these advantages in the next section (and lectures). □

4 BPL ⊆ SC

A deterministic algorithm that runs in space $O(\log^2 n)$ may potentially run for $2^{O(\log^2 n)}$, which is indeed the case if we want to simulate a probabilistic space-bounded algorithm over all the seed's of, say, Nisan's generator. However, we can do much better.

The key observation is that instead of choosing $h_1, \ldots, h_t$ completely at random, and arguing that with high probability they are all good, we will deterministically *search* for good hash functions. Indeed:

Lemma 19. *Given a matrix A and $h_2, \ldots, h_i \in \mathcal{H}$ for $i \leq t = \log T$, there exists an algorithm that finds $h_1 \in \mathcal{H}$ which is α-good for $A_{h_2,\ldots,h_i}$. The algorithm uses $O(\log(\frac{TW}{\varepsilon}))$ space and $\mathrm{poly}(\frac{T \cdot W}{\varepsilon})$ time.*

Proof. The algorithm goes as follows. For every $h \in \mathcal{H}$:

- For every $s, t \in [W]$:
  - Compute $p_1 = A_{h_1,\ldots,h_i}[s,t]$.
  - Compute $p_2 = \sum_{\ell \in [W]} A_{h_2,\ldots,h_i}[s,\ell] A_{h_2,\ldots,h_i}[\ell,t]$.
  - If $|p_1 - p_2| > \alpha$, proceed to the next h.
- Return $h_1 = h$.

For each h, s and t, The $2W + 1$ computations of A are performed by computing the average over $x \in \Sigma$ of the corresponding G_i with the fixed hash functions. This takes $\mathrm{poly}(|\Sigma|, T)$ time, as each computation of G_i takes time $\mathrm{poly}(\log |\Sigma|, 2^i)$. Overall, the procedure runs in time $\mathrm{poly}(T, W, |\Sigma|) = \mathrm{poly}(T, W, 1/\varepsilon)$. We already proved there *exists* such h_1 (also with high probability), and the h that is returned satisfies $\left\| A_{h_1,\ldots,h_i} - A^2_{h_2,\ldots,h_i} \right\| \leq W^2 \alpha$. Thus, going over all hash functions, we will eventually succeed.

Notice that we crucially used the fact that given a fixed sequence $h_1, \ldots, h_t$:

1. Any output block of the generator can be computed in logarithmic space, and,
2. We can check if the next hash function is good (for the prefix) in logarithmic space, because the number of tests is small.

Notice that we can do neither in the two variants of INW we considered. □

Thus, given A, we can find a good sequence $h_1, \ldots, h_t$ in space $O(t \log |\Sigma|) = O(\log T \cdot \log(\frac{TW}{\varepsilon}))$ and polynomial time (the space is mainly needed for *storing* the sequence). Given $h_1, \ldots, h_t$ so that h_t is $\alpha = \frac{\varepsilon}{nW^2}$-good for A and every h_i for $i < t$ is α-good for $A_{h_{i+1},\ldots,h_t}$, we saw that $\left\| A^{(2^t)} - A_{h_1,\ldots,h_t} \right\| \leq \varepsilon$. Computing $A_{h_1,\ldots,h_t}[s,t]$ for every $s,t \in [W]$ amounts to averaging $G_t(x, h_1, \ldots, h_t)$ over all $x \in \Sigma$, which we already saw that it takes $\mathrm{poly}(\frac{TW}{\varepsilon})$ time.

Plugging-in $T = n$, $W = 2^{O(\log n)}$ and a constant ε, we get:

Theorem 20. $\mathsf{BPL} \subseteq \mathsf{DTISP}(\mathrm{poly}(n), O(\log^2 n)) \subseteq \mathsf{SC}$.

References

[1] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 356–364. ACM, 1994.

[2] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[3] Noam Nisan. RL⊆SC. *Computational Complexity*, 4(1):1–11, 1994.