Derandomizing Computation on Time/Space

How many random bits are needed for randomized algorithms?

Algorithm $A(x,r)$ - input $x$, random $r$.

Derandomization: estimate $\text{Prob} \ A(x,r)$

Generic derandomization: find a list $\{i_1, \ldots, i_k\}$ try all $A(x, i_j)$

Given truly random $(i_1, \ldots, i_k \in \{0,1\}^b, \ b \text{-bit } "\text{seed}"angle$

$G(i_1, \ldots, i_k) = r$ is a pseudorandom generator.

Define $G: \{0,1\}^b \rightarrow \{0,1\}^b$ is an $\varepsilon$-prog, for algorithms in class $C$

if $\forall \ x, \forall A \in C,$

\[
\left| \text{Prob} \left[ A(x, r) \right] - \text{Prob} \left[ A(x, G(i)) \right] \right| \leq \varepsilon
\]

Class $C$ for time/space tradeoff

Read-once, oblivious branching program, also OBDD.

For each fixed input $x$: $A(x, \_)$ define an OBDD:

Space $S$ branching program has $\leq 2^S$ many nodes (configurations)

Time $T$ has $T$ layers, last layer has

two nodes: "accept" or "reject".

Space $= \log \left( \text{width}(L) \right)$

Time $= \# \text{ of random bits}$

Now work with class $\mathcal{C} = \text{OBDD}$, with $\text{Time} T \times \text{Space} S$

An $\varepsilon$-prog for $\mathcal{C}$ gives an $\varepsilon$-prog for $\text{TISP}(T,S)$. 
Goal: Construct ε-PRG with small \( b \), for \((S, T)\)-OBDDs.

Want \( b \) small, since derandomized algorithm uses the \( T^* = 2^b \).

Also want \( b \) to be compatible in small space \( S \).

Nisan-Zuckerman — this talk.

Ajtai-Komlos-Szemeredi, Babai-Nisan-Szegedy — earlier work.

In \((S, T)\)-OBDD: Let \( N \) = \# of random bits so far (\( S \) layer).

When \( N > S \), each state (at layer \( N \)) is reached in about \( 2^n / 2^S \) ways (\( \frac{2^n}{2^S} \approx 2^N \) if \( N > S \)).

Given \( N \) but story, can count state entropy of state is \( S \).

Entropy of random bits in \( N \).

Intuition: Use the unused entropy is \( \approx NS \).

Alternate intuition: Use "poor quality" randomness to create truly (enough) random bits.

Extractor: will use a small amount of true randomness.

\[ E : \{0, 1\}^N \times \{0, 1\}^S \rightarrow \{0, 1\}^M \]

\( N \)-"flawed" randomness.

\( S \)-true randomness.

\( M \)-extracted randomness.

Defn \( E(x, s) \leftarrow U_M \)

\( \forall x \in \{0, 1\}^N, \ |x| \geq 2^k, \) the statistical distance is less than \( \varepsilon \):

\[ E(x, s) \approx U_M \]

\( x \in X \)

\( S \in \{0, 1\}^S \)

where statistical distance equals \( L_1 \)-distance or total variation.

(\( \approx \frac{1}{2} \text{ tosses } x? \))
Claim: \( \exists \) construction \((k, \varepsilon, N, \lambda, M) - \) extractors

provided \( M = 5\mu(k) \), \( \lambda = \Omega(\log N + \log \log \varepsilon) \)

**Strong extractor:** \( (E(x; s), s) \sim \mathcal{U}_n \times \mathcal{U}_n \)

Claim: Some claim holds \( \exists \) strong extractor.

Let \( N = O(S) \) \((N = O(S), \text{ constant})\)

\[
\begin{align*}
    k &= N - 2S \\
    \varepsilon &= \frac{1}{poly(T)} \\
    M &= S(5) \\
    \lambda &= O(\log S + \log T)
\end{align*}
\]

We'll first \( N + \frac{T}{N} \varepsilon \) bits to simulate \( T \) random bits

and apply the construction recursively.

Result: \( \rho_T \) has seed length \( \lambda' = O(S, \log T) \)

Top level of recursion:

\[
G_1(R, s_1, \ldots, s_m)
\]

\( N \)-bits \( \frac{2}{\lambda} \)-bit each

Random bits: \( R \circ E(R, s_1) \circ E(R, s_2) \ldots \circ E(s_{\mu}, T_1) \)

\( m \)-bits \( m \)-bits \( m \)-bits

(to make the analysis easier, don't use \( R \) as random)

Intuitively results of pseudo-random bits are close indistinguishable to truly random \( T \)-bits.
Resume application

\[ \frac{s_0}{s_0} \cdots \frac{s_m}{s_m} \]

Use the \( \frac{F}{2^m} \) bit of randomness

Handle this recursively.

Split into \( N/2 \) steps

Recursively \( \ell = \frac{\log T}{\log (M^2)} \) times: Random bits are \( R_1, \ldots, R_\ell \)

Last level uses \( R_1, \ldots, R_\ell, \hat{s}_1^{(\ell)}, \ldots, \hat{s}_M^{(\ell)} \)

Transverse recursion tree to extract bits at level \( \ell \),
Space to compute: \( \text{height of tree} ) \cdot M \)

Total random bits = \( O(NM) = O(\frac{s \log T}{\ell \cdot M}) \).
Extractor

Get randomness out of a partially random input as an approximately
uniformly random string.

\[ E(x, s) : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^m \]

If \( x \) has \( k \) bits of randomness and \( s \in \{0, 1\}^m \),
then \( E(x, s) \sim u \{0, 1\}^m \)

\[ \text{mbits} \rightarrow \text{mbits} \]

Start

\[ x, n \text{ bits} \]

\[ y_1, y_2 \]

\[ z_1 = E(x, s_1) \]

\[ z_2 = E(x, s_2) \]

\[ y_1, y_2 \ldots \text{ are random bits that would be used in the actually (uniformly} \]

\[ \text{probabilistic) output.} \]

\[ z_1, z_2, \ldots \text{ replace the } x \text{'s} \]

\[ s_1, s_2, \ldots \text{ randomly chosen seeds, each } r \text{ bits.} \]

New randomness = \( n + (\frac{1}{m})n \)

New space = \( S \) (\( S \) is also the original space).

Apply recursively:

\[ \text{mbits} \rightarrow \text{mbits} \]

\[ \text{Acc} \]

\[ \text{Ref} \]

Redo block + apply recursively, in = "tree" of randomness.
Lemma: If $E(x, s)$ is an $(n, r, m, k, e)$-extractor, then

\[
\left| \text{Prob}[\text{original computer accept}] - \text{Prob}[\text{extractors based algorithm accept}] \right| \\
\leq O\left( eT + \delta T \cdot 2^{-O\left( \left( \frac{n-k}{2} \right)^{2} \right)} \right) \\
= O\left( (e + 2^{-\left( \frac{n-k}{2} \right)}) T \right) \\
= O\left( (e + 2^{-\left( n-5-k \right)}) T \right).
\]

Intuition: We need $k$ bits of "randomness" at each stage. Start off with $n$ bits of (true) randomness. But after the $1^\text{st}$ level, only have $n-5$ bits of randomness. So need $k < n-5$.

Proof: Hybrid argument.

$D_0 = y_1 \cdots y_{Tm}$

$D_{Tm} = xz_1 \cdots z_{Tm}$

$D_i = xz_1 \cdots z_i \cdot y_i \cdots y_{Tm}$ $P_i$

Need to show $|P_i - P_{i+1}| \leq O\left( (e + 2^{-\left( n-5-k \right)}) \right)$.

$D_{Tn} = x z_1 \cdots z_i \cdot y_i \cdots y_{Tm}$ - changes $y_i$ are placed from $D_i$.

Can fix $z_1 \cdots z_i$, and $y_i \cdots y_{Tm}$ to maximize the difference $|P_i - P_{i+1}|$.

Let $F(x)$ be the state reached after $1^\text{st}$ stage (using the fixed $z_1 \cdots z_i$)

$F(x): z_0, y_1 \rightarrow z_0, y_1$

Let $g$ be any state where $|F^{-1}(g)| \leq 2^k$

(This is the "bad" case, since $x \not\in F^{-1}(g)$ does not have $k$ bits of randomness.)
For all "bad" $g$,
\[ \operatorname{Prob}_x \left[ F(x) = g \right] < 2^{-k/2n} \]
\[ \operatorname{Prob} \exists \delta \left( \{F(\delta) \leq 2^k \text{ and } F(x) = g \} \right) < 2^k \cdot 2^{5/2n} = \frac{2^k}{2^n} \cdot 2^5 = 2^{-(n-5k)} \]

Let $S_i = |P_i - P_i+1|$,

So $\exists$ fixed $g$ such $\{F(g) \leq 2^k \}$ and
\[ |P_i g - P_{i+1} g| > S_i - 2^{-(n-5k)} \]

"Distinguishing probability" conditioned on reaching configuration $g$.

$T(y)_i = \text{"it start at } g \text{ after } i$, do we accept using random bit } \hat{y}_{i+1} = y$. 

For $P_i$, pick $y$ at random.

For $P_{i+1}$, choose $x \in E_d F(g)$, and choose $z = E(x, \delta_{i+1})$.

Then
\[ |\operatorname{Prob}_x \left[ T(y) \right] - \operatorname{Prob}[T(z)]| \leq \varepsilon \]
and
\[ S_i - 2^{-(n-5k)} \leq |\operatorname{Prob}_x \left[ T(y) \right] - \operatorname{Prob}[T(z)]| \]

\text{ged Lemma.}

\text{RH Big } \varepsilon \text{ not needed.}
A pretty good extractor with a simple construction.

\[ f : \{0,1\}^n \rightarrow \{0,1\}^m \]

\[ g : \{0,1\}^n \rightarrow \{0,1\}^{2n} \]

\[ B \] is random.

If \( T \) is a test that distinguishes pseudo-random strings
from truly random ones,
then \( \exists \text{ circuit } CT \text{ that computes } f, \quad |CT| \leq K \),

Then are \( 2^K \) circuits of size \( k \).

If \( \sigma \) comes from a distribution \( \sigma \) with \( k \) bits of entropy,
then for every test \( T \), most \( \sigma \)'s are not equal to \( CT \), for each \( T \).

If not, \( T(f(\sigma)) \approx T(\sigma) \).

Parameters with earlier argument:
\[ N = N \]
\[ K \approx \text{ circuit size} \]
\[ E = E \]
\[ m \approx \text{ number of bits per} \]

Hardness to randomness construction

If \( f \) is least-case hard, \( g \) is hard to predict w/ \( > \frac{1}{2} + \epsilon \) advantage

[Basic idea: use even correcting code \( \mathcal{C} \)]

Using list-decoding codes from noise \( \frac{1}{2} - \delta \).

Now use Nisan-Wigderson construction.

Design \( S, S_n, S_n \), see \( 15.1 = n \).

List-decoding code's parameters

\[ L = \mathcal{O} \left( \frac{1}{\sqrt{\delta}} \right) \]
\[ |ECC(f)| = \mathcal{O} \left( \log^4(N) \right) \]
\[ R = \mathcal{O}(n) \]
\[ r = n \cdot -n - \text{constant} \]
\[ m = 2 \cdot \mathcal{O}(n) \]
\[ m = \mathcal{n} \cdot \mathcal{O}(1) \]
\[ L = \mathcal{O} \left( \log(N) \right) \]

\[ k = 2^n \cdot n + \mathcal{O}(\log(1/\delta)) = N^{\mathcal{O}(1)} + \mathcal{O}(\log(1/\delta)) \]
\[ M = k \]

\[ R = \mathcal{O}(\log N), \quad E = N - k \]
\[ E = \mathcal{O}(1) \]

\[ n = \mathcal{O}(\max(\log T, S)) = \mathcal{O}(S) \]
\[ m = n^3 \]

\[ T \rightarrow T' = T \cdot \frac{\log S}{S^2} \]

# of recursive iterations \( \mathcal{O} \left( \log S \right) \)

Total amount of randomness \( \approx \mathcal{O}(S \cdot \frac{\log T}{\log S}) \).