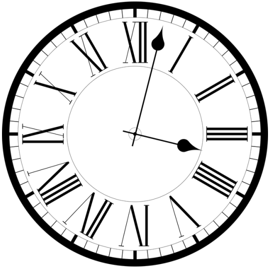
Typically-Correct Derandomization for Small Time and Space

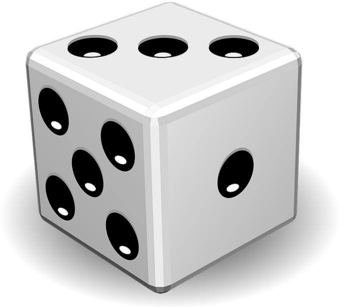
William M. Hoza[1]
The University of Texas at Austin

3/21/18
HUJI CS Theory Seminar

[1] Supported by the NSF GRFP under Grant No. DGE1610403.

Time, space, and randomness





Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$

Derandomization

- Suppose $L \in \textbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$

Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem** [Klivans, van Melkebeek '02]:

Derandomization

- Suppose $L \in \textbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem** [Klivans, van Melkebeek '02]:
  - Assume SAT has exponential circuit complexity

Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem** [Klivans, van Melkebeek '02]:
  - Assume SAT has exponential circuit complexity
  - Then $L \in \mathbf{DTISP}(\text{poly}(T), S)$

Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem** [Klivans, van Melkebeek '02]:
  - Assume SAT has exponential circuit complexity
  - Then $L \in \mathbf{DTISP}(\text{poly}(T), S)$

- **Theorem** [Nisan, Zuckerman '96]:

Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem** [Klivans, van Melkebeek '02]:
  - Assume SAT has exponential circuit complexity
  - Then $L \in \mathbf{DTISP}(\text{poly}(T), S)$

- **Theorem** [Nisan, Zuckerman '96]:
  - Suppose $S \geq T^{\Omega(1)}$

Derandomization

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem** [Klivans, van Melkebeek '02]:
  - Assume SAT has exponential circuit complexity
  - Then $L \in \mathbf{DTISP}(\text{poly}(T), S)$

- **Theorem** [Nisan, Zuckerman '96]:
  - Suppose $S \geq T^{\Omega(1)}$
  - Then $L \in \mathbf{DSPACE}(S)$ (runtime $2^{\Theta(S)}$)

Main result

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

Main result

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem**:

Main result

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem**:
  - Suppose $T \leq n \cdot \text{poly}(S)$

- Think $T = \widetilde{O}(n)$, $S = O(\log n)$

Main result

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem**:
  - Suppose $T \leq n \cdot \text{poly}(S)$
  - Then there is a $\mathbf{DSPACE}(S)$ algorithm for L...

- Think $T = \widetilde{O}(n)$, $S = O(\log n)$

Main result

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem**:
  - Suppose $T \leq n \cdot \text{poly}(S)$
  - Then there is a $\mathbf{DSPACE}(S)$ algorithm for L...
  - ...that succeeds on the vast majority of inputs of each length.

- Think $T = \widetilde{O}(n)$, $S = O(\log n)$

Main result

- Suppose $L \in \mathbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **Theorem**:
  - Suppose $T \leq n \cdot \text{poly}(S)$
  - Then there is a $\mathbf{DSPACE}(S)$ algorithm for L...
  - ...that succeeds on the vast majority of inputs of each length.

- Think $T = \widetilde{O}(n)$, $S = O(\log n)$
  - [Saks, Zhou '95]: Space $\Theta(\log^{1.5} n)$

Typically-correct derandomizations

- Is $110100001101001111001010110111011010100011100 \in L$?

Typically-correct derandomizations

- Is $110100001101001111001010110111011010100011100 \in L$?
  - If only we had some randomness...

Typically-correct derandomizations

- Is $11010000110100111100101011011101101010001110 0 \in L$?
  - If only we had some randomness...

- Let A be a randomized algorithm

Typically-correct derandomizations

- Is $110100001101001111001010110111011010100011100 \in L$?
  - If only we had some randomness...

- Let A be a randomized algorithm
- Naïve derandomization: Run $A(x, x)$

Typically-correct derandomizations

- Is 110100001101001111001010110111011010100011100 $\in L$?
  - If only we had some randomness...

- Let A be a randomized algorithm
- Naïve derandomization: Run $A(x, x)$
- Might fail on all x because of correlations between input, coins

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously
   - [Goldreich, Wigderson '02]: Undirected s-t connectivity

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously
   - [Goldreich, Wigderson '02]: Undirected s-t connectivity
   - [Arvind, Torán '04]: Solvable group isomorphism

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously
    - [Goldreich, Wigderson '02]: Undirected s-t connectivity
    - [Arvind, Torán '04]: Solvable group isomorphism

2. Extract randomness from input using specialized extractor

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously
   - [Goldreich, Wigderson '02]: Undirected s-t connectivity
   - [Arvind, Torán '04]: Solvable group isomorphism

2. Extract randomness from input using specialized extractor
   - [Zimand '08]: Sublinear time algorithms

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously
   - [Goldreich, Wigderson '02]: Undirected s-t connectivity
   - [Arvind, Torán '04]: Solvable group isomorphism
2. Extract randomness from input using specialized extractor
   - [Zimand '08]: Sublinear time algorithms
   - [Shaltiel '11]: Two-party communication protocols, streaming algorithms, $\mathbf{BPAC}^0$

Prior techniques for dealing with correlations

1. Find algorithm A where most random strings are good for all inputs simultaneously
   - [Goldreich, Wigderson '02]: Undirected s-t connectivity
   - [Arvind, Torán '04]: Solvable group isomorphism

2. Extract randomness from input using specialized extractor
   - [Zimand '08]: Sublinear time algorithms
   - [Shaltiel '11]: Two-party communication protocols, streaming algorithms, $\mathbf{BPAC}^0$

3. Plug input into seed-extending pseudorandom generator

Prior techniques for dealing with correlations

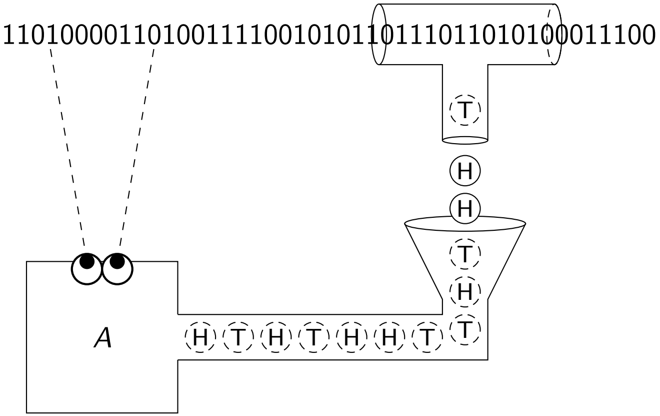
1. Find algorithm A where most random strings are good for all inputs simultaneously
   - [Goldreich, Wigderson '02]: Undirected s-t connectivity
   - [Arvind, Torán '04]: Solvable group isomorphism
2. Extract randomness from input using specialized extractor
   - [Zimand '08]: Sublinear time algorithms
   - [Shaltiel '11]: Two-party communication protocols, streaming algorithms, $\mathbf{BPAC}^0$
3. Plug input into seed-extending pseudorandom generator
   - [Kinne, van Melkebeek, Shaltiel '12]: Multiparty communication protocols, $\mathbf{BPAC}^0$ with symmetric gates

Our technique: "Out of sight, out of mind"

110100001101001111001010110111011010100011100

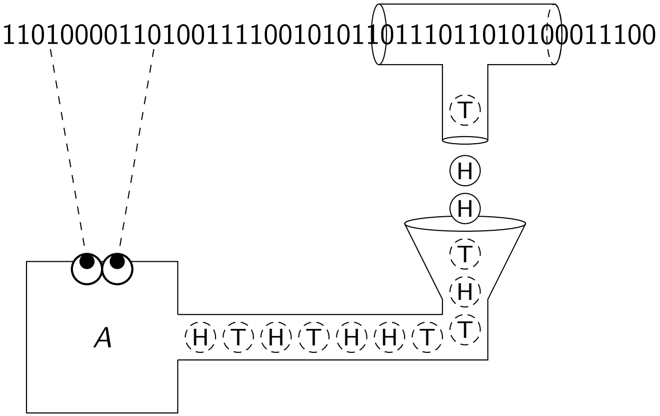
Our technique: “Out of sight, out of mind”

- Use part of the input as a source of randomness while A is processing the rest of the input



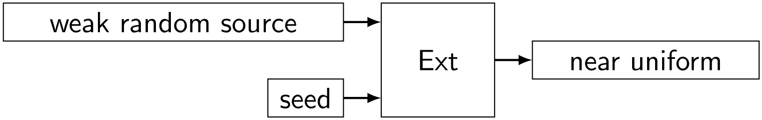
Our technique: “Out of sight, out of mind”

- Use part of the input as a source of randomness while A is processing the rest of the input



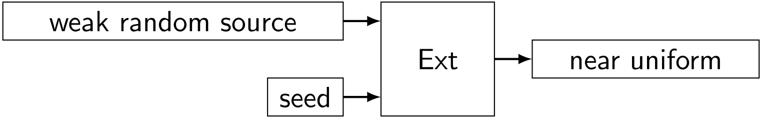
- (Additional ideas needed to make this work...)

Randomness extractors



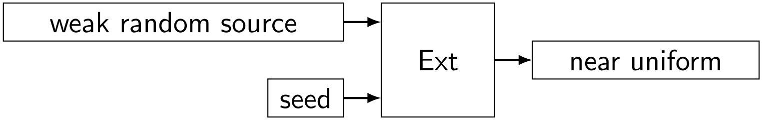
- (k, ε)-extractor $\text{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$

Randomness extractors



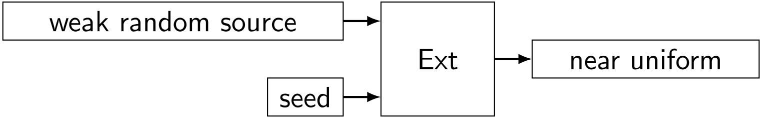
- (k, ε)-extractor $\mathsf{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$
- Assume X has "at least k bits of randomness" (min-entropy)

Randomness extractors



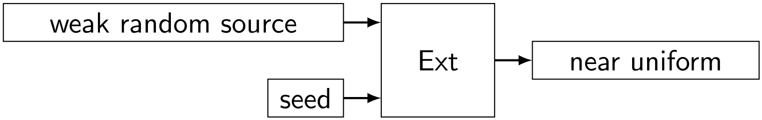
- (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$
- Assume X has "at least k bits of randomness" (min-entropy)
- Assume Y is uniform random, independent of X

Randomness extractors



- (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$
- Assume X has "at least k bits of randomness" (min-entropy)
- Assume Y is uniform random, independent of X
- Then $\mathrm{Ext}(X, Y) \sim_\varepsilon U_s$

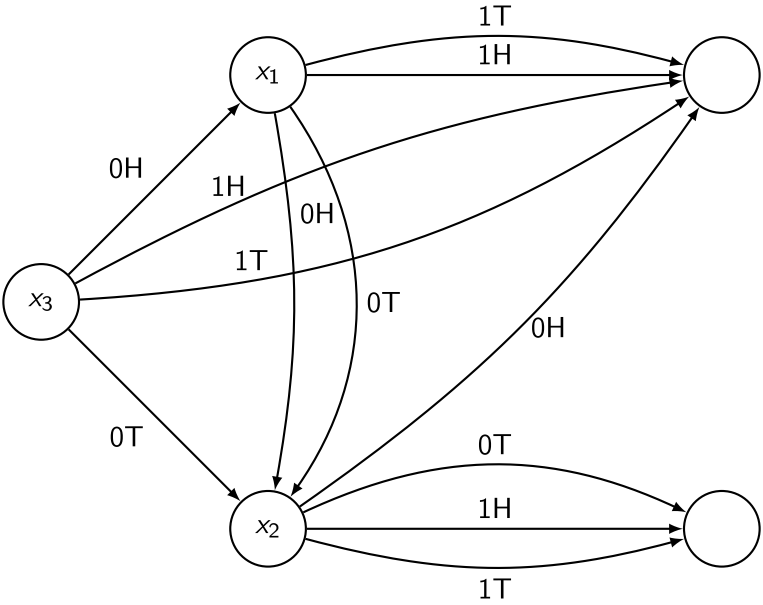
Randomness extractors



- (k, ε)-extractor $\mathrm{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$
- Assume X has "at least k bits of randomness" (min-entropy)
- Assume Y is uniform random, independent of X
- Then $\mathrm{Ext}(X, Y) \sim_\varepsilon U_s$

- Think $s \approx k$ and $d \approx \log(\ell/\varepsilon)$.

Randomized branching programs



Randomized branching programs (2)

- Length $=$ length of longest path

Randomized branching programs (2)

- Length = length of longest path
- Size = # vertices

Randomized branching programs (2)

- Length = length of longest path
- Size = # vertices
- Time $\widetilde{O}(n)$, space $O(\log n)$ $\implies$ length $\widetilde{O}(n)$, size $\mathrm{poly}(n)$

Randomized branching programs (2)

- Length = length of longest path
- Size = # vertices
- Time $\widetilde{O}(n)$, space $O(\log n) \implies$ length $\widetilde{O}(n)$, size poly(n)

- $\mathcal{P}(v; x, y)$ = the terminal vertex reached if you start from vertex v, read input $x \in \{0,1\}^n$, use random bits $y \in \{0,1\}^T$

Nisan's generator

- **Theorem** (Nisan '92): There is a pseudorandom generator

$$\text{NisGen} : \{0,1\}^s \to \{0,1\}^T \tag{1}$$

that fools programs of size poly(n):

$$\mathcal{P}(v; x, U_T) \sim_\varepsilon \mathcal{P}(v; x, \text{NisGen}(U_s)) \tag{2}$$

Nisan's generator

- **Theorem** (Nisan '92): There is a pseudorandom generator

$$\mathrm{NisGen} : \{0,1\}^s \to \{0,1\}^T \tag{1}$$

that fools programs of size poly(n):

$$\mathcal{P}(v; x, U_T) \sim_\varepsilon \mathcal{P}(v; x, \mathrm{NisGen}(U_s)) \tag{2}$$

  - Seed length $s = O(\log^2 n)$

Nisan's generator

- **Theorem** (Nisan '92): There is a pseudorandom generator

$$\text{NisGen} : \{0,1\}^s \to \{0,1\}^T \tag{1}$$

that fools programs of size poly(n):

$$\mathcal{P}(v; x, U_T) \sim_\varepsilon \mathcal{P}(v; x, \text{NisGen}(U_s)) \tag{2}$$

  - Seed length $s = O(\log^2 n)$
  - Runs in space $O(\log n)$ given two-way access to seed

Main technical result

- **Theorem**: Suppose we're given

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0,1\}^n$

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0,1\}^n$
  - Initial vertex v_0.

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0, 1\}^n$
  - Initial vertex v_0.

Can approximately sample from $\mathcal{P}(v_0; x, U_T)$ using

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0,1\}^n$
  - Initial vertex v_0.

  Can approximately sample from $\mathcal{P}(v_0; x, U_T)$ using
  - Space $O(\log n)$

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0,1\}^n$
  - Initial vertex v_0.

  Can approximately sample from $\mathcal{P}(v_0; x, U_T)$ using
  - Space $O(\log n)$
  - Randomness polylog n

Main technical result

- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0, 1\}^n$
  - Initial vertex v_0.

  Can approximately sample from $\mathcal{P}(v_0; x, U_T)$ using
  - Space $O(\log n)$
  - Randomness polylog n **(one-way access!)**

Main technical result

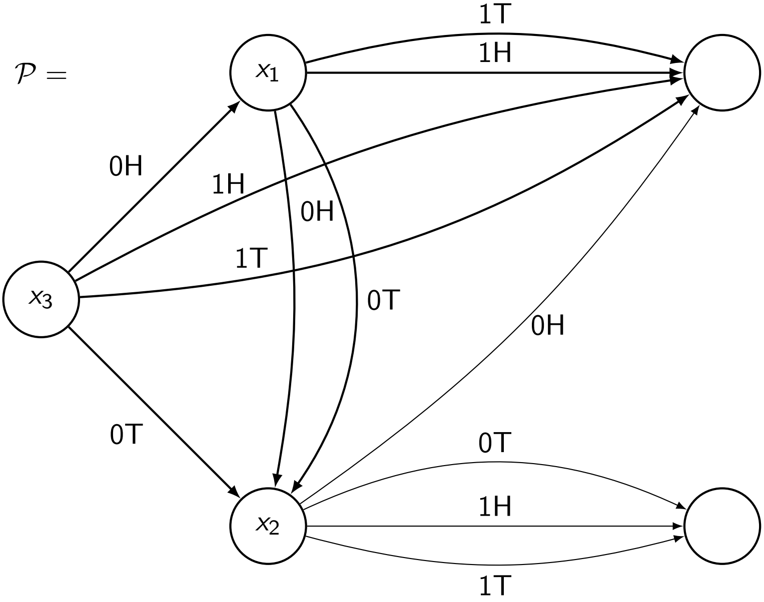
- **Theorem**: Suppose we're given
  - Program $\mathcal{P}$ of size poly(n), length $T = \widetilde{O}(n)$
  - Input $x \in \{0,1\}^n$
  - Initial vertex v_0.

  Can approximately sample from $\mathcal{P}(v_0; x, U_T)$ using
  - Space $O(\log n)$
  - Randomness polylog n **(one-way access!)**

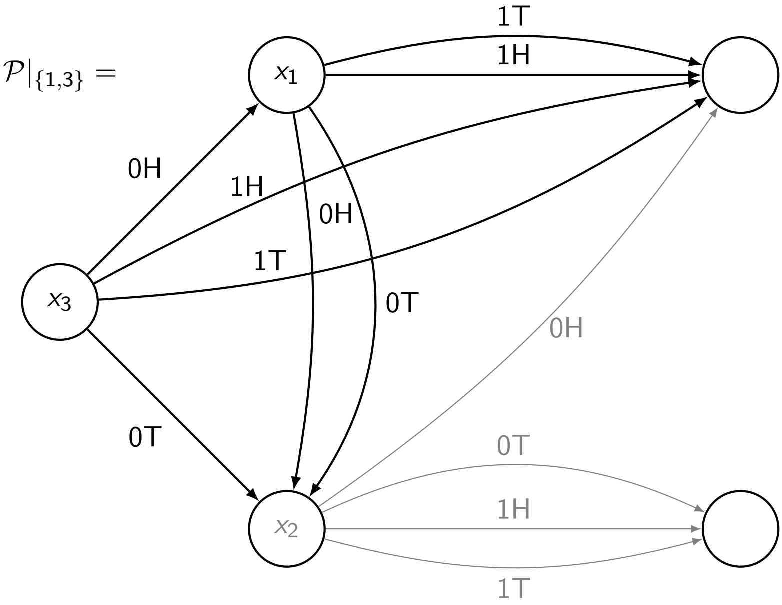
  Caveat: Sampling error is large for tiny fraction of x values

Restriction of a program

$\mathcal{P} =$

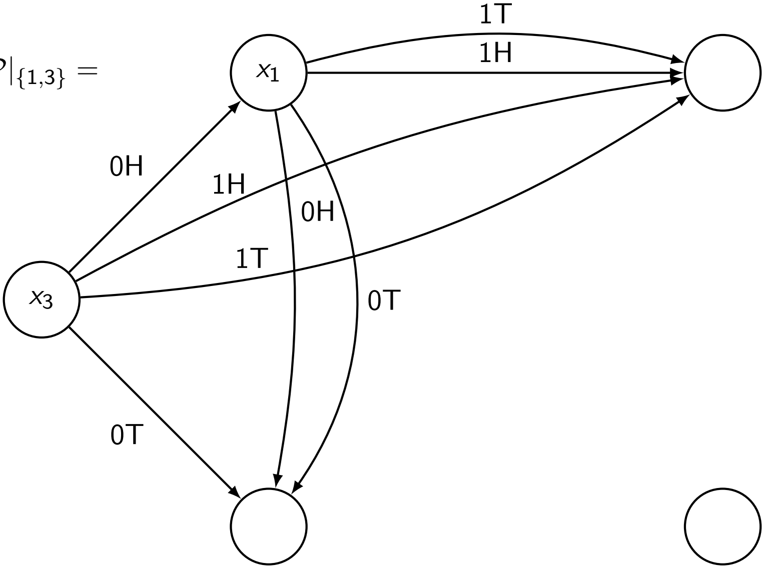


Restriction of a program



Restriction of a program

$\mathcal{P}|_{\{1,3\}} =$



Main algorithm

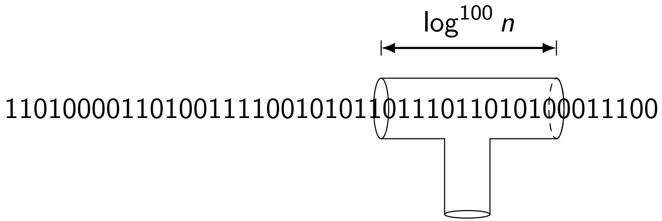
11010000110100111100101011011101101010100011100

Main algorithm

110100001101001111001010110111011010100011100

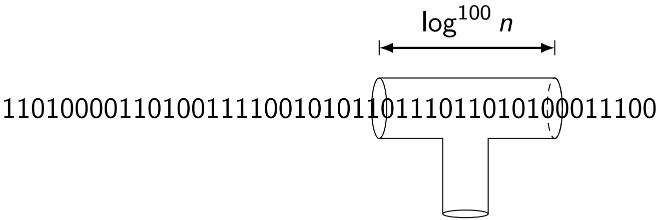
1. Initialize $v := v_0$. Repeat polylog(n) times:

Main algorithm



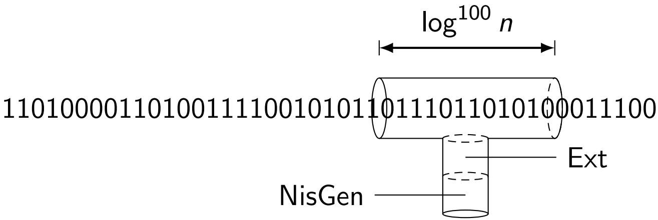
1. Initialize $v := v_0$. Repeat $\text{polylog}(n)$ times:
    1.1 Pick a random contiguous block $I \subseteq [n]$

Main algorithm



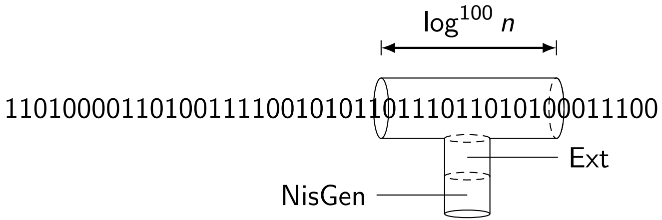
1. Initialize $v := v_0$. Repeat polylog(n) times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$

Main algorithm



1. Initialize $v := v_0$. Repeat polylog(n) times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \text{NisGen}(\text{Ext}(x|_I, y))$

Main algorithm



1. Initialize $v := v_0$. Repeat $\text{polylog}(n)$ times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \text{NisGen}(\text{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

Efficiency analysis

1. Initialize $v := v_0$. Repeat polylog(n) times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

Efficiency analysis

1. Initialize $v := v_0$. Repeat polylog(n) times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

- Runs in $O(\log n)$ space!

Efficiency analysis

1. Initialize $v := v_0$. Repeat polylog(n) times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v

Efficiency analysis

1. Initialize $v := v_0$. Repeat $\text{polylog}(n)$ times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \text{NisGen}(\text{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v
  - $O(\log n)$ bits to run Ext, NisGen

Efficiency analysis

1. Initialize $v := v_0$. Repeat polylog(n) times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v
  - $O(\log n)$ bits to run Ext, NisGen
- We can give NisGen two-way access to its seed, because we have two-way access to x

Efficiency analysis

1. Initialize $v := v_0$. Repeat $\text{polylog}(n)$ times:
   1.1 Pick a random contiguous block $I \subseteq [n]$
   1.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   1.3 Let $z = \text{NisGen}(\text{Ext}(x|_I, y))$
   1.4 Update
   $$v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$$
2. Output v

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v
  - $O(\log n)$ bits to run Ext, NisGen
- We can give NisGen two-way access to its seed, because we have two-way access to x
- Randomness $\text{polylog}\, n$ (one-way access!)

Extractors are good samplers

- **Lemma** [Zuckerman '97]:

Extractors are good samplers

- **Lemma** [Zuckerman '97]:
  - Suppose $\mathrm{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$ is a (k, ε)-extractor

Extractors are good samplers

- **Lemma** [Zuckerman '97]:
  - Suppose $\mathrm{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$ is a (k, ε)-extractor
  - For any $F : \{0,1\}^s \to V$, for most x,

$$F(U_s) \sim_{\varepsilon \cdot |V|/2} F(\mathrm{Ext}(x, U_d)).$$

Extractors are good samplers

- **Lemma** [Zuckerman '97]:
  - Suppose $\mathrm{Ext} : \{0,1\}^\ell \times \{0,1\}^d \to \{0,1\}^s$ is a (k, ε)-extractor
  - For any $F : \{0,1\}^s \to V$, for most x,

$$F(U_s) \sim_{\varepsilon \cdot |V|/2} F(\mathrm{Ext}(x, U_d)).$$

  - $(\#\text{ bad } x \leq 2^{k+1}|V|)$

Correctness proof sketch

True algorithm

1. Pick random $y \in \{0,1\}^{O(\log n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$
3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$ 2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$ 3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$ 2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$ 3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$ 2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$ 3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \mathrm{NisGen}(y')$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$
- \# bad x bounded by

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$
- # bad x bounded by

$$\underbrace{(2^{O(\log^2 n)} \cdot \mathrm{poly}(n))}_{\#\text{ bad } x|_I} \cdot$$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$
- # bad x bounded by

$$\underbrace{(2^{O(\log^2 n)} \cdot \mathrm{poly}(n))}_{\#\text{ bad } x|_I} \cdot \underbrace{(2^{n-\log^{100} n})}_{\#\, x|_{[n]\setminus I}}.$$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$
- $\#$ bad x bounded by

$$\underbrace{\left(2^{O(\log^2 n)} \cdot \mathrm{poly}(n)\right)}_{\#\text{ bad } x|_I} \cdot \underbrace{\left(2^{n-\log^{100} n}\right)}_{\#\ x|_{[n]\setminus I}} \cdot \underbrace{(n)}_{\#\ I} \cdot$$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$
- \# bad x bounded by

$$\underbrace{\left(2^{O(\log^2 n)} \cdot \mathrm{poly}(n)\right)}_{\#\text{ bad } x|_I} \cdot \underbrace{\left(2^{n-\log^{100} n}\right)}_{\#\, x|_{[n]\setminus I}} \cdot \underbrace{(n)}_{\#\, I} \cdot \underbrace{(\mathrm{poly}(n))}_{\#\, v}$$

Correctness proof sketch

True algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{Ext}(x|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$	3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$

- Consider $F(y') = \mathcal{P}|_{[n]\setminus I}(v; x, \mathrm{NisGen}(y'))$
- # bad x bounded by

$$\underbrace{(2^{O(\log^2 n)} \cdot \mathrm{poly}(n))}_{\#\text{ bad } x|_I} \cdot \underbrace{(2^{n-\log^{100} n})}_{\#\, x|_{[n]\setminus I}} \cdot \underbrace{(n)}_{\#\, I} \cdot \underbrace{(\mathrm{poly}(n))}_{\#\, v} \ll 2^n$$

Correctness proof sketch (2)

First hybrid distribution

1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \mathrm{NisGen}(y')$
3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$

Correctness proof sketch (2)

First hybrid distribution	Second hybrid distribution
1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \mathrm{NisGen}(y')$ 3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$	

Correctness proof sketch (2)

First hybrid distribution	Second hybrid distribution
1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \mathrm{NisGen}(y')$ 3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$	1. Pick random $z \in \{0,1\}^T$

Correctness proof sketch (2)

First hybrid distribution	Second hybrid distribution
1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \mathrm{NisGen}(y')$ 3. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$	1. Pick random $z \in \{0,1\}^T$ 2. Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$

Correctness proof sketch (3)

Second hybrid distribution

1. Initialize $v := v_0$
2. Repeat polylog(n) times:
   2.1 Pick random $I \subseteq [n]$
   2.2 Pick random $z \in \{0,1\}^T$
   2.3 Update $v := \mathcal{P}|_{[n]\setminus I}(v; x, z)$
3. Output v

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Initialize $v := v_0$ 2. Repeat polylog(n) times: 2.1 Pick random $I \subseteq [n]$ 2.2 Pick random $z \in \{0,1\}^T$ 2.3 Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$ 3. Output v	

Correctness proof sketch (3)

Second hybrid distribution	Target distribution

1. Initialize $v := v_0$
2. Repeat polylog(n) times:
  2.1 Pick random $I \subseteq [n]$
  2.2 Pick random $z \in \{0,1\}^T$
  2.3 Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$
3. Output v
	1. Pick random $z \in \{0,1\}^T$

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Initialize $v := v_0$ 2. Repeat polylog(n) times: 2.1 Pick random $I \subseteq [n]$ 2.2 Pick random $z \in \{0,1\}^T$ 2.3 Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$ 3. Output v	1. Pick random $z \in \{0,1\}^T$ 2. Output $\mathcal{P}(v_0; x, z)$

Correctness proof sketch (3)

Second hybrid distribution	Target distribution

1. Initialize $v := v_0$
2. Repeat polylog(n) times:
  2.1 Pick random $I \subseteq [n]$
  2.2 Pick random $z \in \{0,1\}^T$
  2.3 Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$
3. Output v
	1. Pick random $z \in \{0,1\}^T$
	2. Output $\mathcal{P}(v_0; x, z)$

- In each phase, make $n / \log^{100} n$ steps through program w.h.p.

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Initialize $v := v_0$ 2. Repeat polylog(n) times: 2.1 Pick random $I \subseteq [n]$ 2.2 Pick random $z \in \{0,1\}^T$ 2.3 Update $v := \mathcal{P}\|_{[n]\setminus I}(v; x, z)$ 3. Output v	1. Pick random $z \in \{0,1\}^T$ 2. Output $\mathcal{P}(v_0; x, z)$

- In each phase, make $n/\log^{100} n$ steps through program w.h.p.
- After polylog(n) phases, reach terminal vertex w.h.p.

Uniform consequence

- **Corollary**:

Uniform consequence

- **Corollary**:
  - Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$

Uniform consequence

- **Corollary**:
  - Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
  - There is a $\mathbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just polylog n random bits...

Uniform consequence

- **Corollary**:
  - Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
  - There is a $\mathbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just polylog n random bits...
  - ...that succeeds on the vast majority of inputs of each length.

The Nisan-Zuckerman generator

- **Theorem** (Nisan, Zuckerman '96): For every constant c, there is a pseudorandom generator

$$\mathsf{NZGen} : \{0,1\}^d \to \{0,1\}^{\log^c n}$$

that fools programs of size $\mathrm{poly}(n)$:

$$\mathcal{P}(v; x, U_{\log^c n}) \sim_\varepsilon \mathcal{P}(v; x, \mathsf{NZGen}(U_d)).$$

The Nisan-Zuckerman generator

- **Theorem** (Nisan, Zuckerman '96): For every constant c, there is a pseudorandom generator

$$\mathsf{NZGen} : \{0,1\}^d \to \{0,1\}^{\log^c n}$$

that fools programs of size $\mathrm{poly}(n)$:

$$\mathcal{P}(v; x, U_{\log^c n}) \sim_\varepsilon \mathcal{P}(v; x, \mathsf{NZGen}(U_d)).$$

  - Seed length $d \leq O(\log n)$

The Nisan-Zuckerman generator

- **Theorem** (Nisan, Zuckerman '96): For every constant c, there is a pseudorandom generator

$$\mathsf{NZGen} : \{0,1\}^d \to \{0,1\}^{\log^c n}$$

that fools programs of size $\mathrm{poly}(n)$:

$$\mathcal{P}(v; x, U_{\log^c n}) \sim_\varepsilon \mathcal{P}(v; x, \mathsf{NZGen}(U_d)).$$

  - Seed length $d \leq O(\log n)$
  - Runs in space $O(\log n)$

Eliminating randomness

- Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$

Eliminating randomness

- Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:

Eliminating randomness

- Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:
  - There is a $\mathbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just $O(\log n)$ random bits...

Eliminating randomness

- Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:
  - There is a $\mathbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just $O(\log n)$ random bits...
  - ...that succeeds on the vast majority of inputs of each length.

Eliminating randomness

- Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:
  - There is a $\mathbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just $O(\log n)$ random bits...
  - ...that succeeds on the vast majority of inputs of each length.
- **Corollary**:

Eliminating randomness

- Suppose $L \in \textbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:
  - There is a $\textbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just $O(\log n)$ random bits...
  - ...that succeeds on the vast majority of inputs of each length.
- **Corollary**:
  - There is a $\textbf{DSPACE}(\log n)$ algorithm for L...

Eliminating randomness

- Suppose $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:
  - There is a $\mathbf{BPTISP}(\widetilde{O}(n), \log n)$ algorithm for L that uses just $O(\log n)$ random bits...
  - ...that succeeds on the vast majority of inputs of each length.
- **Corollary**:
  - There is a $\mathbf{DSPACE}(\log n)$ algorithm for L...
  - ...that succeeds on the vast majority of inputs of each length.

Derandomization with advice

- Adleman's argument shows $\mathbf{BPL} \subseteq \mathbf{L}/\operatorname{poly}$

Derandomization with advice

- Adleman's argument shows $\mathbf{BPL} \subseteq \mathbf{L}/\operatorname{poly}$
- Fortnow, Klivans '06: $\mathbf{BPL} \subseteq \mathbf{L}/O(n)$

Derandomization with advice

- Adleman's argument shows $\mathbf{BPL} \subseteq \mathbf{L}/\operatorname{poly}$
- Fortnow, Klivans '06: $\mathbf{BPL} \subseteq \mathbf{L}/O(n)$
- Goldreich, Wigderson '02: Critical threshold at n bits of advice

Derandomization with advice

- Adleman's argument shows $\mathbf{BPL} \subseteq \mathbf{L}/\operatorname{poly}$
- Fortnow, Klivans '06: $\mathbf{BPL} \subseteq \mathbf{L}/O(n)$
- Goldreich, Wigderson '02: Critical threshold at n bits of advice
  - Roughly: Derandomization with $< n$ bits of advice $\implies$ typically-correct derandomization with no advice

Contribution 1: Derandomizing **BPL** with less advice

- **Theorem**: $\mathbf{BPL} \subseteq \mathbf{L}/(n + O(\log^2 n))$.

Contribution 1: Derandomizing **BPL** with less advice

- **Theorem**: $\mathbf{BPL} \subseteq \mathbf{L}/(n + O(\log^2 n))$.
- **The algorithm**: Given $x \in \{0,1\}^n$, $a \in \{0,1\}^{n+O(\log^2 n)}$:

Contribution 1: Derandomizing **BPL** with less advice

- **Theorem**: $\mathbf{BPL} \subseteq \mathbf{L}/(n + O(\log^2 n))$.
- **The algorithm**: Given $x \in \{0,1\}^n$, $a \in \{0,1\}^{n+O(\log^2 n)}$:
  - For every $y \in \{0,1\}^{O(\log n)}$, run **BPL** algorithm with randomness $\mathsf{NisGen}(\mathsf{Ext}(a, y))$

Contribution 1: Derandomizing **BPL** with less advice

- **Theorem**: $\mathbf{BPL} \subseteq \mathbf{L}/(n + O(\log^2 n))$.
- **The algorithm**: Given $x \in \{0,1\}^n$, $a \in \{0,1\}^{n+O(\log^2 n)}$:
  - For every $y \in \{0,1\}^{O(\log n)}$, run **BPL** algorithm with randomness $\mathsf{NisGen}(\mathsf{Ext}(a, y))$
  - Output majority answer

Contribution 1: Derandomizing **BPL** with less advice

- **Theorem**: $\mathbf{BPL} \subseteq \mathbf{L}/(n + O(\log^2 n))$.
- **The algorithm**: Given $x \in \{0,1\}^n$, $a \in \{0,1\}^{n+O(\log^2 n)}$:
  - For every $y \in \{0,1\}^{O(\log n)}$, run **BPL** algorithm with randomness $\mathsf{NisGen}(\mathsf{Ext}(a, y))$
  - Output majority answer
- **Proof of correctness**: # bad a bounded by

$$\underbrace{\left(2^{O(\log^2 n)}\right)}_{\#\text{ bad } a \text{ for each } x} \cdot \underbrace{(2^n)}_{\#\, x} < 2^{|a|}$$

Contribution 2: Converse to GW '02

- **Theorem**: If $L \in \mathbf{BPL}$ admits a $\mathbf{DSPACE}(\log n)$ algorithm A that fails on ε-fraction of inputs, then

$$L \in \mathbf{L}/(n - \log_2(1/\varepsilon) + O(\log^2 n)).$$

Contribution 2: Converse to GW '02

- **Theorem**: If $L \in \mathbf{BPL}$ admits a $\mathbf{DSPACE}(\log n)$ algorithm A that fails on ε-fraction of inputs, then

$$L \in \mathbf{L}/(n - \log_2(1/\varepsilon) + O(\log^2 n)).$$

- Idea: Run A *and* algorithm with advice

Contribution 2: Converse to GW '02

- **Theorem**: If $L \in \mathbf{BPL}$ admits a $\mathbf{DSPACE}(\log n)$ algorithm A that fails on ε-fraction of inputs, then

$$L \in \mathbf{L}/(n - \log_2(1/\varepsilon) + O(\log^2 n)).$$

- Idea: Run A *and* algorithm with advice
- Advice only needs to be good for atypical x

Contribution 2: Converse to GW '02

- **Theorem**: If $L \in \mathbf{BPL}$ admits a $\mathbf{DSPACE}(\log n)$ algorithm A that fails on ε-fraction of inputs, then

$$L \in \mathbf{L}/(n - \log_2(1/\varepsilon) + O(\log^2 n)).$$

- Idea: Run A *and* algorithm with advice
- Advice only needs to be good for atypical x
- (Detail: Make advice algorithm "zero-error" using $\mathbf{RL} \subseteq \mathbf{SC}$ trick)

Derandomizing quasilinear-time, log-space with advice

- **Corollary**: For every constant c,

$$\mathbf{BPTISP}(\widetilde{O}(n), \log n) \subseteq \mathbf{L}/(n - \log^c n).$$

Sublinear advice

- $\mathbf{BPTISP}_{\mathrm{TM}}(T, S)$: Time-$T$ space-S multitape Turing machines
- **Theorem**: For every constant c,

$$\mathbf{BPTISP}_{\mathrm{TM}}(\widetilde{O}(n), \log n) \subseteq \mathbf{L} / \left(\frac{n}{\log^c n}\right).$$

Beyond quasilinear time

- **Theorem**:

$$\mathbf{BPTISP}_{\mathsf{TM}}(n^{1.99}, \log n) \subseteq \mathbf{L}/(n - n^{\Omega(1)}).$$

Disambiguating nondeterministic algorithms

- Allender et al. '99: Circuit lower bounds $\implies$ $\mathbf{NL} = \mathbf{UL}$

Disambiguating nondeterministic algorithms

- Allender et al. '99: Circuit lower bounds $\implies$ $\mathbf{NL} = \mathbf{UL}$
- Reinhard, Allender '00: $\mathbf{NL} \subseteq \mathbf{UL}/\,\mathrm{poly}$

Disambiguating nondeterministic algorithms

- Allender et al. '99: Circuit lower bounds $\implies$ $\mathbf{NL} = \mathbf{UL}$
- Reinhard, Allender '00: $\mathbf{NL} \subseteq \mathbf{UL}/\operatorname{poly}$
- van Melkebeek, Prakriya '17: $\mathbf{NL} \subseteq \mathbf{USPACE}(\log^{3/2} n)$

Disambiguating nondeterministic algorithms

- Allender et al. '99: Circuit lower bounds $\implies$ $\mathbf{NL} = \mathbf{UL}$
- Reinhard, Allender '00: $\mathbf{NL} \subseteq \mathbf{UL}/\operatorname{poly}$
- van Melkebeek, Prakriya '17: $\mathbf{NL} \subseteq \mathbf{USPACE}(\log^{3/2} n)$

- **Theorem:** $\mathbf{NL} \subseteq \mathbf{UL}/(n + \log^2 n)$

Disambiguating nondeterministic algorithms

- Allender et al. '99: Circuit lower bounds $\implies$ $\mathbf{NL} = \mathbf{UL}$
- Reinhard, Allender '00: $\mathbf{NL} \subseteq \mathbf{UL}/\operatorname{poly}$
- van Melkebeek, Prakriya '17: $\mathbf{NL} \subseteq \mathbf{USPACE}(\log^{3/2} n)$

- **Theorem:** $\mathbf{NL} \subseteq \mathbf{UL}/(n + \log^2 n)$
- **Theorem:** For every constant c,

$$\mathbf{NTISP}(\widetilde{O}(n), \log n) \subseteq \mathbf{USPACE}(\widetilde{O}(\log n))/(n - \log^c n).$$

Main open problem

- Typically-correct derandomization of **BPL**?

Main open problem

- Typically-correct derandomization of **BPL**?
- Thanks! Questions?