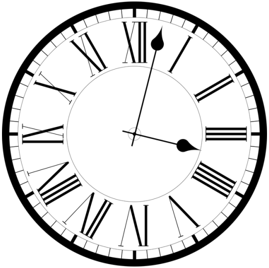
Typically-Correct Derandomization for Small Time and Space

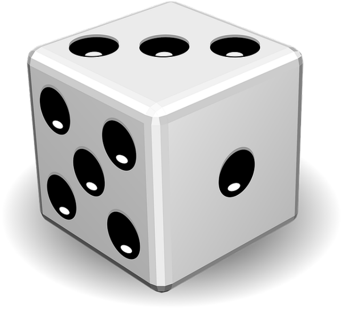
William M. Hoza[1]
University of Texas at Austin

July 18
CCC 2019

[1] Supported by the NSF GRFP under Grant No. DGE1610403 and by a Harrington fellowship from UT Austin

Time, space, and randomness





Derandomization

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

Derandomization

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

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

- **Theorem** [Klivans, van Melkebeek '02]:
  - Assume some language in $\mathbf{DSPACE}(O(n))$ 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 some language in $\mathbf{DSPACE}(O(n))$ 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 some language in $\mathbf{DSPACE}(O(n))$ 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 some language in $\mathbf{DSPACE}(O(n))$ 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 some language in $\mathbf{DSPACE}(O(n))$ 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 $L \in \textbf{BPTISP}(T, S)$
  - $T = T(n) \geq n$
  - $S = S(n) \geq \log n$

- **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 $T \leq n \cdot \mathrm{poly}(S)$
  - 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 $110100001101001111001010110111011010100011100 \in L$?
  - 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, $\mathbf{BPAC}^0$

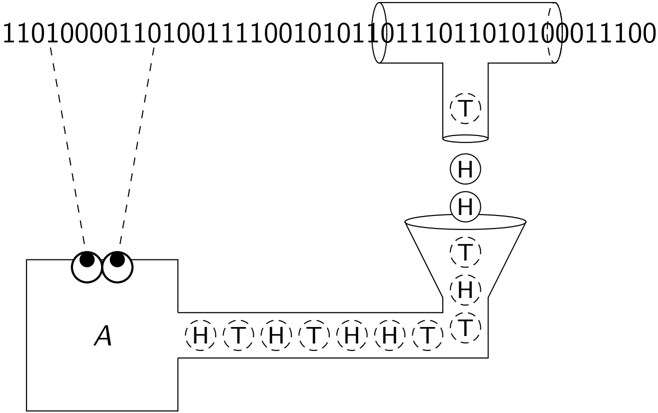
3. Plug input into seed-extending pseudorandom generator
   - [Kinne, van Melkebeek, Shaltiel '12]: Multiparty communication protocols, $\mathbf{BPAC}^0$ with symmetric gates

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

110100001101001111001010110111011010100011100

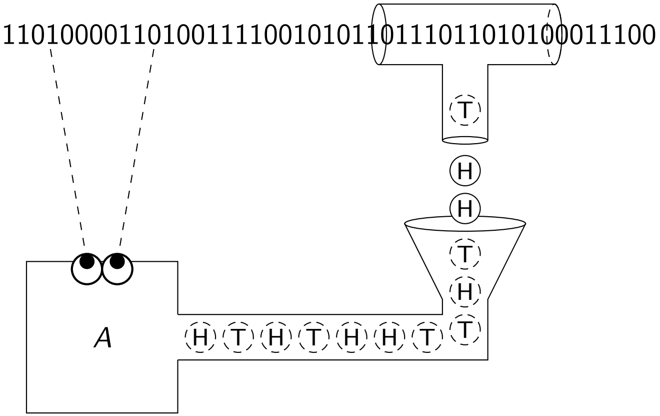
Our technique: “Out of sight, out of mind”

- 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...)

Restriction of algorithm

110100001101001111001010110111011010100011100

- Let $I \subseteq [n]$

Restriction of algorithm

110100001101001111001010110111011010100011100

- Let $I \subseteq [n]$
- Algorithm $A|_{[n]\setminus I}$:

Restriction of algorithm

110100001101001111001010110111011010100011100

- Let $I \subseteq [n]$
- Algorithm $A|_{[n]\setminus I}$:
  1. Run A like normal...

Restriction of algorithm

110100001101001111001010110111011010100011100

- Let $I \subseteq [n]$
- Algorithm $A|_{[n]\setminus I}$:
  1. Run A like normal...
  2. ...except, if A is about to query x_i for some $i \in I$, halt immediately.

Restriction of algorithm

11010000110100111100101010  00011100

- Let $I \subseteq [n]$
- Algorithm $A|_{[n]\setminus I}$:
  1. Run A like normal...
  2. ...except, if A is about to query x_i for some $i \in I$, halt immediately.

Main Lemma: Reducing randomness to polylog n

- **Main Lemma**:

Main Lemma: Reducing randomness to polylog n

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

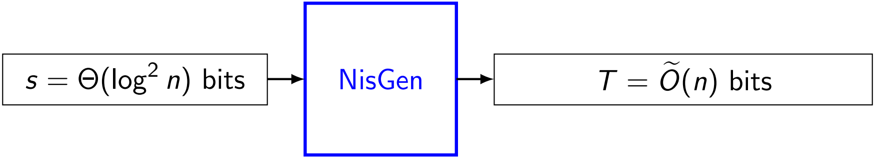
Main Lemma: Reducing randomness to polylog n

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

Main Lemma: Reducing randomness to polylog n

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

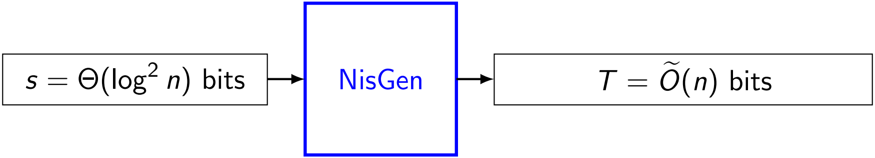
Tool 1: Nisan's Pseudorandom Generator



- For any $O(\log n)$-space A, input x,

$$A(x, \mathrm{NisGen}(U_s)) \approx A(x, U_T)$$

Tool 1: Nisan's Pseudorandom Generator

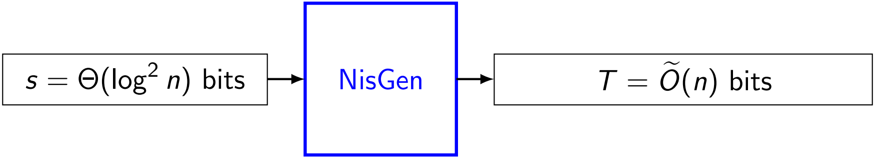


- For any $O(\log n)$-space A, input x,

$$A(x, \mathrm{NisGen}(U_s)) \approx A(x, U_T)$$

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

Tool 1: Nisan's Pseudorandom Generator

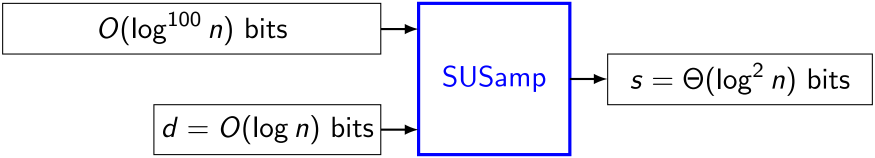


- For any $O(\log n)$-space A, input x,

$$A(x, \mathrm{NisGen}(U_s)) \approx A(x, U_T)$$

- Runs in space $O(\log n)$...
- ...given two-way access to seed

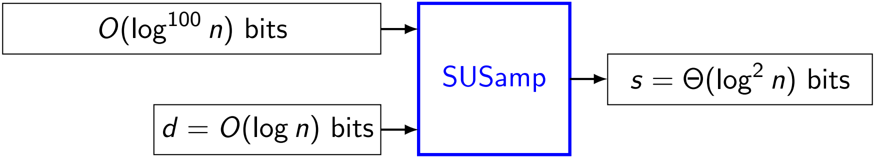
Tool 2: Shaltiel-Umans Averaging Sampler



- For any $F\colon \{0,1\}^s \to V$, for most x,

$$F(\mathrm{SUSamp}(x, U_d)) \approx F(U_s)$$

Tool 2: Shaltiel-Umans Averaging Sampler

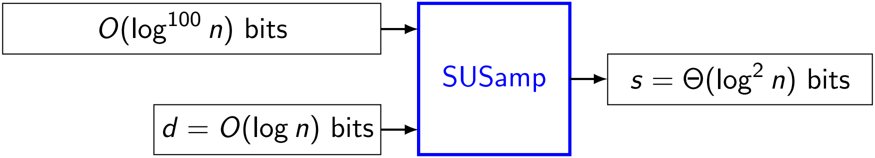


- For any $F \colon \{0,1\}^s \to V$, for most x,

$$F(\text{SUSamp}(x, U_d)) \approx F(U_s)$$

- $\#$ bad x: at most $2^{O(\log^6 n)} \cdot |V|$

Tool 2: Shaltiel-Umans Averaging Sampler



- For any $F\colon \{0,1\}^s \to V$, for most x,

$$F(\text{SUSamp}(x, U_d)) \approx F(U_s)$$

- \# bad x: at most $2^{O(\log^6 n)} \cdot |V|$
- Runs in space $O(\log n)$ given two-way access to x

Algorithm of Main Lemma

11010000110100111100101011011101101010100011100

Algorithm of Main Lemma

110100001101001111001010110111011010100011100

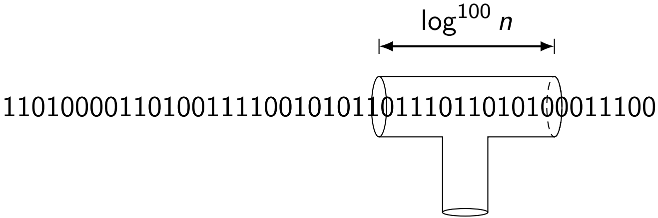
1. Let v be the initial configuration of A

Algorithm of Main Lemma

11010000110100111100101011011101101010 0011100

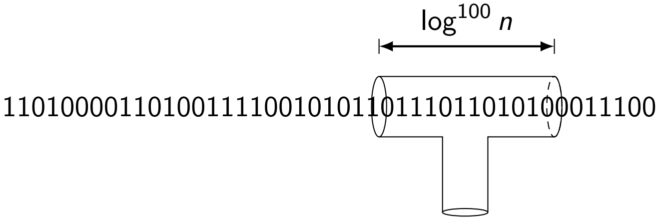
1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:

Algorithm of Main Lemma



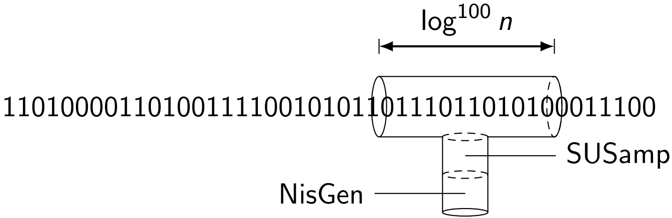
1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:
   2.1 Pick a random contiguous block $I \subseteq [n]$

Algorithm of Main Lemma



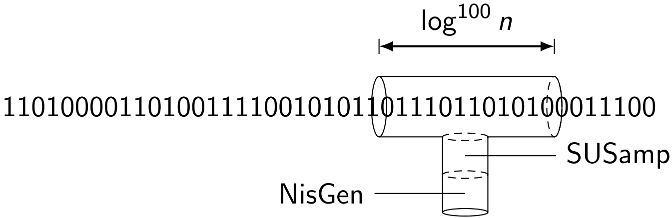
1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:
   2.1 Pick a random contiguous block $I \subseteq [n]$
   2.2 Pick a random $y \in \{0,1\}^{O(\log n)}$

Algorithm of Main Lemma



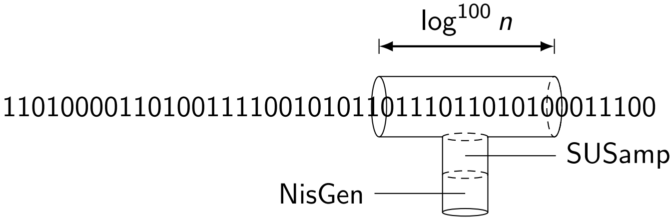
1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:
   - 2.1 Pick a random contiguous block $I \subseteq [n]$
   - 2.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   - 2.3 Let $z = \text{NisGen}(\text{SUSamp}(x|_I, y))$

Algorithm of Main Lemma



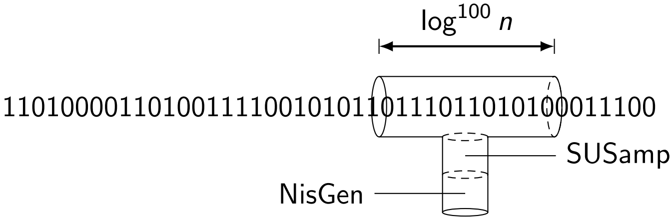
1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:
   2.1 Pick a random contiguous block $I \subseteq [n]$
   2.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   2.3 Let $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   2.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts

Algorithm of Main Lemma



1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:
   2.1 Pick a random contiguous block $I \subseteq [n]$
   2.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   2.3 Let $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   2.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   2.5 Update v to be the final configuration

Algorithm of Main Lemma



1. Let v be the initial configuration of A
2. Loop for polylog(n) iterations:
   2.1 Pick a random contiguous block $I \subseteq [n]$
   2.2 Pick a random $y \in \{0,1\}^{O(\log n)}$
   2.3 Let $z = \text{NisGen}(\text{SUSamp}(x|_I, y))$
   2.4 Run $A|_{[n]\setminus I}(x,z)$ from configuration v until it halts
   2.5 Update v to be the final configuration
3. Accept if v is an accepting configuration, else reject

Efficiency analysis

1. Loop for polylog(n) iterations:
   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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   1.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   1.5 Update v to be the final configuration
2. Accept iff v accepts

Efficiency analysis

1. Loop for polylog(n) iterations:
   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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   1.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   1.5 Update v to be the final configuration
2. Accept iff v accepts

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

Efficiency analysis

1. Loop for polylog(n) iterations:
   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 $z = \text{NisGen}(\text{SUSamp}(x|_I, y))$
   1.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   1.5 Update v to be the final configuration
2. Accept iff v accepts

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

Efficiency analysis

1. Loop for polylog(n) iterations:
   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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   1.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   1.5 Update v to be the final configuration
2. Accept iff v accepts

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v
  - $O(\log n)$ bits to run SUSamp, NisGen, $A|_{[n]\setminus I}$

Efficiency analysis

1. Loop for polylog(n) iterations:
   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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   1.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   1.5 Update v to be the final configuration
2. Accept iff v accepts

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v
  - $O(\log n)$ bits to run SUSamp, NisGen, $A|_{[n]\setminus I}$
- We can give SUSamp, NisGen two-way access to their inputs, because we have two-way access to x

Efficiency analysis

1. Loop for polylog(n) iterations:
   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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
   1.4 Run $A|_{[n]\setminus I}(x, z)$ from configuration v until it halts
   1.5 Update v to be the final configuration
2. Accept iff v accepts

- Runs in $O(\log n)$ space!
  - $O(\log n)$ bits to store I, y, v
  - $O(\log n)$ bits to run SUSamp, NisGen, $A|_{[n]\setminus I}$
- We can give SUSamp, NisGen two-way access to their inputs, because we have two-way access to x
- Randomness polylog n (one-way access!)

Correctness proof sketch

Our algorithm

1. Pick random $y \in \{0,1\}^{O(\log n)}$
2. Let $z = \text{NisGen}(\text{SUSamp}(x|_I, y))$
3. Run $A|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration

Correctness proof sketch

Our algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$ 2. Let $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$ 3. Run $A|_{[n]\setminus I}(x, z)$ from v 4. Update $v :=$ final configuration	

Correctness proof sketch

Our algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$ 2. Let $z = \mathrm{NisGen}(\mathrm{SUSamp}(x\|_I, y))$ 3. Run $A\|_{[n]\setminus I}(x, z)$ from v 4. Update $v :=$ final configuration	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$

Correctness proof sketch

Our algorithm	First hybrid distribution
1. Pick random $y \in \{0,1\}^{O(\log n)}$ 2. Let $z = \mathrm{NisGen}(\mathrm{SUSamp}(x\|_I, y))$ 3. Run $A\|_{[n]\setminus I}(x, z)$ from v 4. Update $v :=$ final configuration	1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \mathrm{NisGen}(y')$

Correctness proof sketch

Our 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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x, z)$ from v	3. Run $A\|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

Correctness proof sketch

Our 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 $z = \text{NisGen}(\text{SUSamp}(x\|_I, y))$	2. Let $z = \text{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x, z)$ from v	3. Run $A\|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

- Let $F(y')$ = final configuration when running $A|_{[n]\setminus I}(x, \text{NisGen}(y'))$ from v

Correctness proof sketch

Our 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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x, z)$ from v	3. Run $A\|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

- Let $F(y') =$ final configuration when running $A|_{[n]\setminus I}(x, \mathrm{NisGen}(y'))$ from v
- $\#$ bad x bounded by

Correctness proof sketch

Our 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 $z = \text{NisGen}(\text{SUSamp}(x\|_I, y))$	2. Let $z = \text{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x, z)$ from v	3. Run $A\|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

- Let $F(y') =$ final configuration when running $A|_{[n]\setminus I}(x, \text{NisGen}(y'))$ from v
- $\#$ bad x bounded by

$$\underbrace{(2^{O(\log^6 n)} \cdot \text{poly}(n))}_{\#\text{ bad } x|_I} \cdot$$

Correctness proof sketch

Our 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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x, z)$ from v	3. Run $A\|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

- Let $F(y') =$ final configuration when running $A|_{[n]\setminus I}(x, \mathrm{NisGen}(y'))$ from v
- $\#$ bad x bounded by

$$\underbrace{(2^{O(\log^6 n)} \cdot \mathrm{poly}(n))}_{\#\text{ bad } x|_I} \cdot \underbrace{(2^{n - \log^{100} n})}_{\# x|_{[n]\setminus I}}.$$

Correctness proof sketch

Our algorithm

1. Pick random $y \in \{0,1\}^{O(\log n)}$
2. Let $z = \mathrm{NisGen}(\mathrm{SUSamp}(x|_I, y))$
3. Run $A|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration

First hybrid distribution

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

- Let $F(y') =$ final configuration when running $A|_{[n]\setminus I}(x, \mathrm{NisGen}(y'))$ from v
- $\#$ bad x bounded by

$$\underbrace{\left(2^{O(\log^6 n)} \cdot \mathrm{poly}(n)\right)}_{\#\text{ bad } x|_I} \cdot \underbrace{\left(2^{n - \log^{100} n}\right)}_{\#\ x|_{[n]\setminus I}} \cdot \underbrace{(n)}_{\#\ I} \cdot$$

Correctness proof sketch

Our 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 $z = \text{NisGen}(\text{SUSamp}(x\|_I, y))$	2. Let $z = \text{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x,z)$ from v	3. Run $A\|_{[n]\setminus I}(x,z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

- Let $F(y')$ = final configuration when running $A|_{[n]\setminus I}(x, \text{NisGen}(y'))$ from v
- \# bad x bounded by

$$\underbrace{(2^{O(\log^6 n)} \cdot \text{poly}(n))}_{\#\text{ bad } x|_I} \cdot \underbrace{(2^{n-\log^{100} n})}_{\#\, x|_{[n]\setminus I}} \cdot \underbrace{(n)}_{\#\, I} \cdot \underbrace{(\text{poly}(n))}_{\#\, v}$$

Correctness proof sketch

Our 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 $z = \mathrm{NisGen}(\mathrm{SUSamp}(x\|_I, y))$	2. Let $z = \mathrm{NisGen}(y')$
3. Run $A\|_{[n]\setminus I}(x, z)$ from v	3. Run $A\|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration	4. Update $v :=$ final configuration

- Let $F(y') =$ final configuration when running $A|_{[n]\setminus I}(x, \mathrm{NisGen}(y'))$ from v
- $\#$ bad x bounded by

$$\underbrace{(2^{O(\log^6 n)} \cdot \mathrm{poly}(n))}_{\#\text{ bad } x|_I} \cdot \underbrace{(2^{n-\log^{100} n})}_{\#\, x|_{[n]\setminus I}} \cdot \underbrace{(n)}_{\#\, I} \cdot \underbrace{(\mathrm{poly}(n))}_{\#\, v} \ll 2^n$$

Correctness proof sketch (2)

First hybrid distribution

1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$
2. Let $z = \text{NisGen}(y')$
3. Run $A|_{[n]\setminus I}(x, z)$ from v
4. Update $v :=$ final configuration

Correctness proof sketch (2)

First hybrid distribution	Second hybrid distribution
1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \mathsf{NisGen}(y')$ 3. Run $A|_{[n]\setminus I}(x, z)$ from v 4. Update $v :=$ final configuration	

Correctness proof sketch (2)

First hybrid distribution	Second hybrid distribution
1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$ 2. Let $z = \text{NisGen}(y')$ 3. Run $A|_{[n]\setminus I}(x, z)$ from v 4. Update $v :=$ final configuration	1. Pick random $z \in \{0,1\}^T$

Correctness proof sketch (2)

First hybrid distribution	Second hybrid distribution
1. Pick random $y' \in \{0,1\}^{O(\log^2 n)}$	1. Pick random $z \in \{0,1\}^{T}$
2. Let $z = \text{NisGen}(y')$	2. Run $A|_{[n]\setminus I}(x,z)$ from v
3. Run $A|_{[n]\setminus I}(x,z)$ from v	3. Update $v :=$ final configuration
4. Update $v :=$ final configuration	

Correctness proof sketch (3)

Second hybrid distribution

1. Repeat polylog(n) times:
   1.1 Pick random $I \subseteq [n]$
   1.2 Pick random $z \in \{0,1\}^T$
   1.3 Run $A|_{[n]\setminus I}(x, z)$ from v
   1.4 Update $v :=$ final conf.
2. Accept iff v accepts

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Repeat polylog(n) times: 1.1 Pick random $I \subseteq [n]$ 1.2 Pick random $z \in \{0,1\}^T$ 1.3 Run $A\|_{[n]\setminus I}(x,z)$ from v 1.4 Update $v :=$ final conf. 2. Accept iff v accepts	

Correctness proof sketch (3)

Second hybrid distribution

1. Repeat polylog(n) times:
   1.1 Pick random $I \subseteq [n]$
   1.2 Pick random $z \in \{0,1\}^T$
   1.3 Run $A|_{[n]\setminus I}(x, z)$ from v
   1.4 Update $v :=$ final conf.
2. Accept iff v accepts

Target distribution

1. Pick random $z \in \{0,1\}^T$

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Repeat polylog(n) times: 1.1 Pick random $I \subseteq [n]$ 1.2 Pick random $z \in \{0,1\}^T$ 1.3 Run $A\|_{[n]\setminus I}(x,z)$ from v 1.4 Update $v :=$ final conf. 2. Accept iff v accepts	1. Pick random $z \in \{0,1\}^T$ 2. Accept iff $A(x,z)$ accepts

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Repeat polylog(n) times: 1.1 Pick random $I \subseteq [n]$ 1.2 Pick random $z \in \{0,1\}^T$ 1.3 Run $A\|_{[n]\setminus I}(x,z)$ from v 1.4 Update $v :=$ final conf. 2. Accept iff v accepts	1. Pick random $z \in \{0,1\}^T$ 2. Accept iff $A(x,z)$ accepts

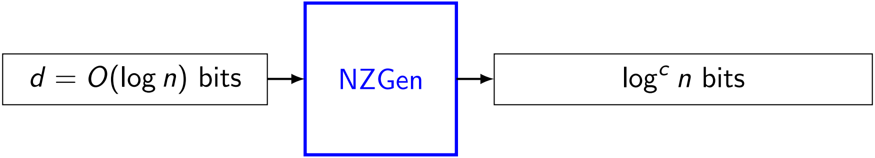
- In each phase, simulate $\Omega(n/\log^{100} n)$ steps of A w.h.p.

Correctness proof sketch (3)

Second hybrid distribution	Target distribution
1. Repeat polylog(n) times: 1.1 Pick random $I \subseteq [n]$ 1.2 Pick random $z \in \{0,1\}^T$ 1.3 Run $A|_{[n]\setminus I}(x,z)$ from v 1.4 Update $v :=$ final conf. 2. Accept iff v accepts	1. Pick random $z \in \{0,1\}^T$ 2. Accept iff $A(x,z)$ accepts

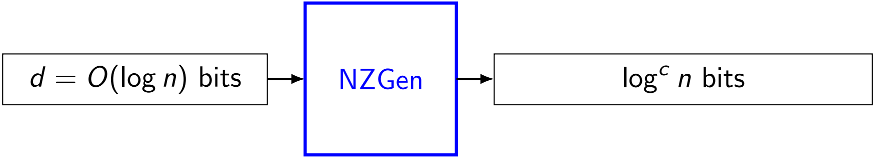
- In each phase, simulate $\Omega(n/\log^{100} n)$ steps of A w.h.p.
- After polylog(n) phases, reach halting configuration w.h.p. $\square$

Tool 3: The Nisan-Zuckerman PRG



- $c =$ arbitrarily large constant

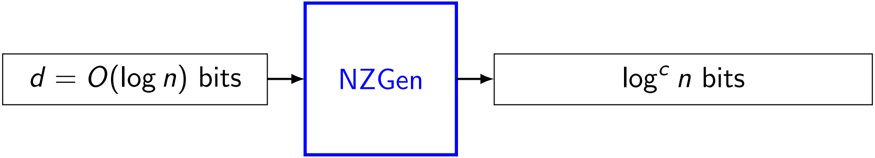
Tool 3: The Nisan-Zuckerman PRG



- $c =$ arbitrarily large constant
- For any $O(\log n)$-space A, input x,

$$A(x, \mathrm{NZGen}(U_d)) \approx A(x, U_{\log^c n})$$

Tool 3: The Nisan-Zuckerman PRG



- $c =$ arbitrarily large constant
- For any $O(\log n)$-space A, input x,

$$A(x, \mathrm{NZGen}(U_d)) \approx A(x, U_{\log^c n})$$

- 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 **BPL** 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 **BPL** 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 **BPL** 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 $L \in \mathbf{BPTISP}(\widetilde{O}(n), \log n)$
- **Corollary**:
  - There is a **BPL** 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...

Eliminating randomness

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

Main open problem

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

Main open problem

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