Pseudorandom Hashing for Space-bounded Computation with Applications in Streaming

Praneeth Kacham
CMU

Mikkel Thorup
U. Of Copenhagen

Rasmus Pagh
U. Of Copenhagen

David Woodruff
CMU

Main Results

- A generalization of Nisan's Pseudorandom generator for small space algorithms

- Has a **symmetry** property useful for derandomizing some streaming algorithms

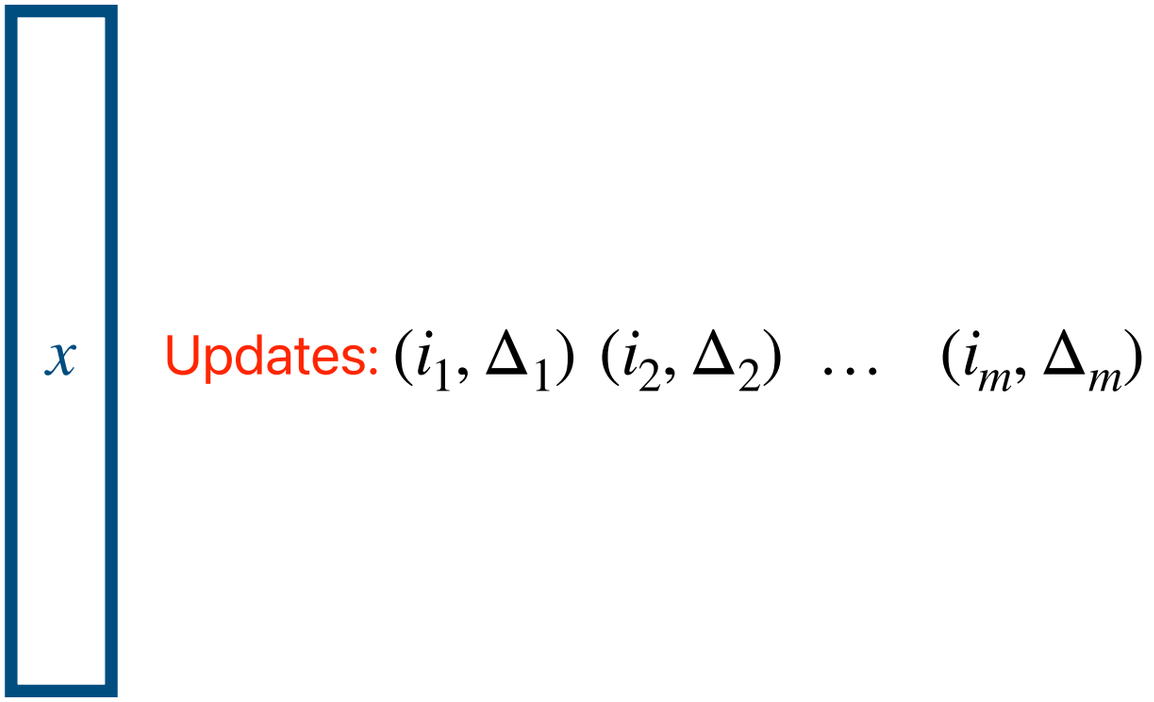
  - Tail bounds for CountSketch estimates, estimating $\|x\|_\infty$

- Provides a space-vs-time tradeoff: Can **speedup** extraction of blocks from the pseudorandom string

  - Improves **update times** of turnstile streaming algorithms

Turnstile Streaming

- Initialize $x \leftarrow 0 \in \mathbb{R}^d$
- On update (i, Δ) :
  - Set $x[i] \leftarrow x[i] + \Delta$
- Answer queries about x at the end of the stream using **small space**
  - What was $\|x\|_\infty$ or $x[i]$ or $\|x\|_2$ …
- **Update time**: Time to process a new update



Our Results

Improving Update Times

- $F_p(x) = \sum_i |x[i]|^p$

- For $p > 2$, can approximate $F_p(x)$ up to $O_p(1)$ using an $\tilde{O}(d^{1-2/p})$ space algorithm with **constant** update time
    - Derandomizing [Andoni17]
    - Optimal space for linear sketches

- For $0 < p < 2$, $1 \pm \varepsilon$ approximation to $F_p(x)$ using $O(\varepsilon^{-2} \log d)$ bits of space and an update time of $O(\log d)$ (requires $\varepsilon < 1/d^c$)
    - Derandomizing [KNW10] which has an update time of $O(\log^2 d \cdot \text{poly}(\log \log d))$ in this regime
    - Optimal space for **any** streaming algorithm

Our Results

Derandomizing CountSketch via Symmetry

- There is a streaming algorithm such that at the end, for any i we can compute $\hat{x}[i]$ such that for $\alpha \leq 1$

$$\Pr[\,|x[i] - \hat{x}[i]| > \alpha \frac{\|x\|_2}{\sqrt{t}}] \leq 2\exp(-\alpha^2 r) + 1/\text{poly}(d)$$

  - Obtained by derandomizing [Minton and Price 14]

- The algorithm uses $O(tr\log(d) + \log^2 d)$ bits of space

- We use symmetry to reorder the blocks of the pseudorandom string in the analysis

Our Results

Estimating $\|x\|_\infty$

- An **algorithm** and a **tight lower bound** to estimate $\|x\|_\infty$ up to an additive $\varepsilon\|x\|_2$ using $O(\varepsilon^{-2}\log d \log 1/\varepsilon)$ bits
  - **Doesn't** require the new PRG
- Using the tight guarantees for CountSketch, an algorithm that uses $O(\varepsilon^{-2}\log d)$ bits if $\|x\|_\infty = \Theta(\|x\|_2)$.
  - Symmetry of **PRG** maybe helpful to get more such results!

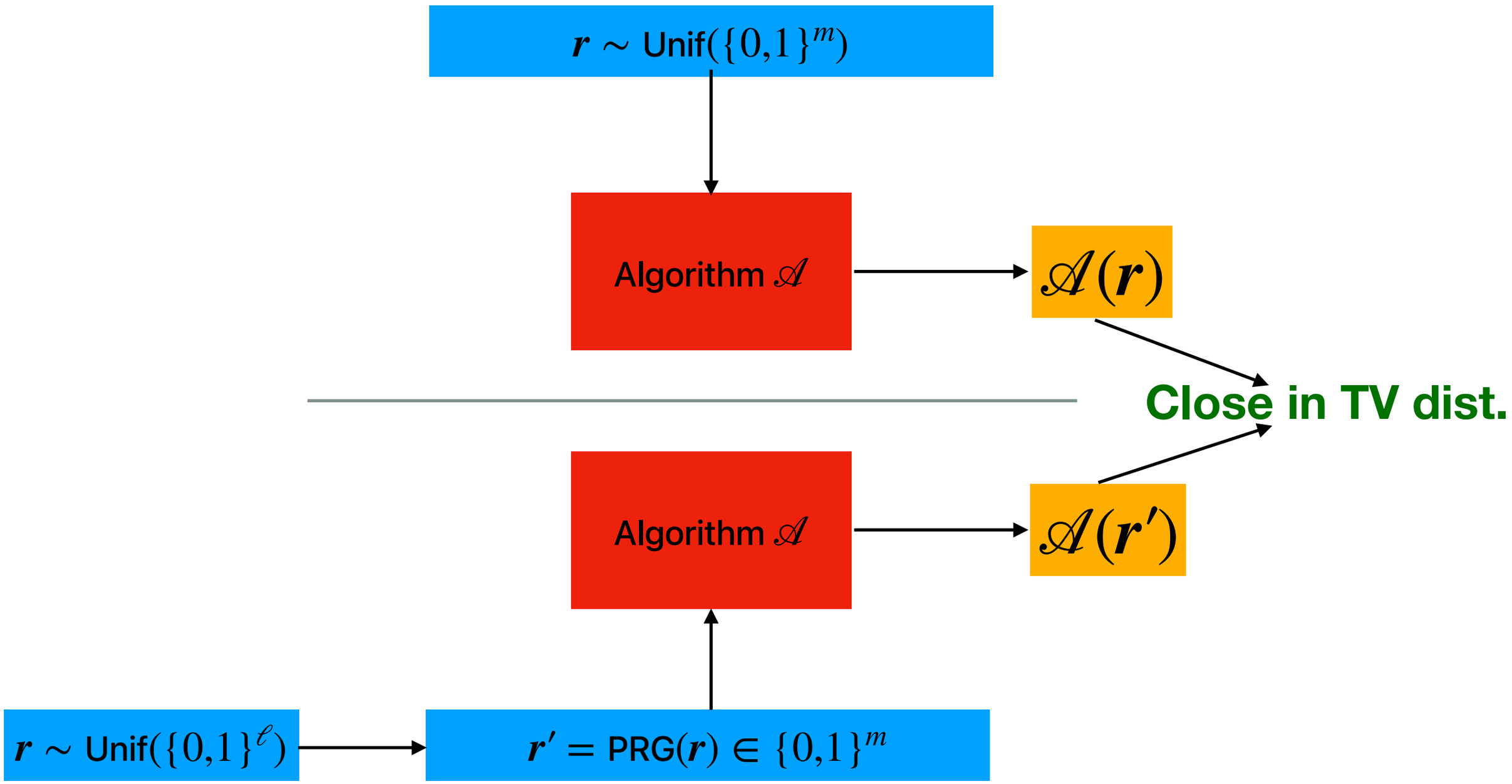
Pseudorandom Generators

- Pseudorandom generators (PRGs) take a uniform random string $\boldsymbol{r}$ (“seed”) of length ℓ and use it to create a string $\boldsymbol{r'}$ of length $m > \ell$

- **Goal:** Make the string $\boldsymbol{r'}$ look like a length m uniform random string

- **In this talk:** PRGs that “fool” space-bounded algorithms

A Picture


$r \sim \text{Unif}(\{0,1\}^m)$
Algorithm $\mathscr{A}$
$\mathscr{A}(r)$
Close in TV dist.
Algorithm $\mathscr{A}$
$\mathscr{A}(r')$
$r \sim \text{Unif}(\{0,1\}^\ell)$
$r' = \text{PRG}(r) \in \{0,1\}^m$

Space-Bounded Algorithms

- Uses w bits to store its state
- Makes a single pass (from left to right) over the random bits
- Updates its state while streaming through the bits
- The state updates can be arbitrarily complicated

An Example

- Consider t-bit integers $a_1, \ldots, a_m$
- Algorithm initializes state $s \leftarrow 0$
- Streaming through the random bits, it updates

$$s \leftarrow s + (-1)^{r'_i} a_i$$

- This is a space $t + \log_2 m$ algorithm

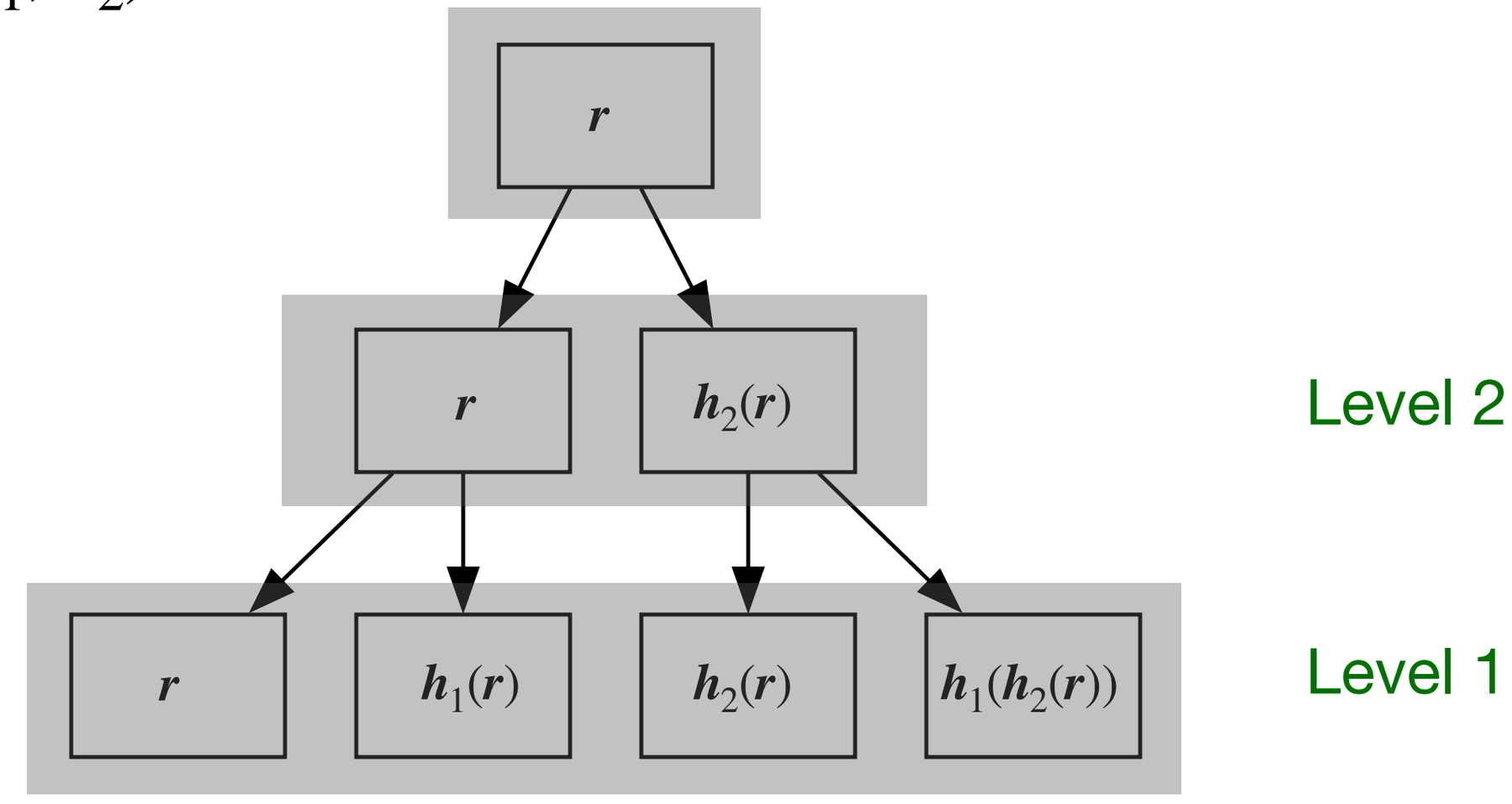
Nisan's PRG

- Nisan, in 1990, gave a PRG for space-bounded algorithms using pairwise independent hash functions
- There is $\mathscr{H} = \{h : \{0,1\}^{\ell} \to \{0,1\}^{\ell}\}$ such that each $h \in \mathscr{H}$ can be identified using $O(\ell)$ bits
- Can sample a random hash function from $\mathscr{H}$ using $O(\ell)$ random bits

Construction of Nisan's PRG

- Let $\boldsymbol{r} \sim \{0,1\}^{\ell}$ and $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_t \sim \mathscr{H} \Rightarrow$ seed length of $O(t \cdot \ell)$
- PRG$(\boldsymbol{r}, \boldsymbol{h}_1, \boldsymbol{h}_2)$ is defined as follows

$\boldsymbol{r}' :=$



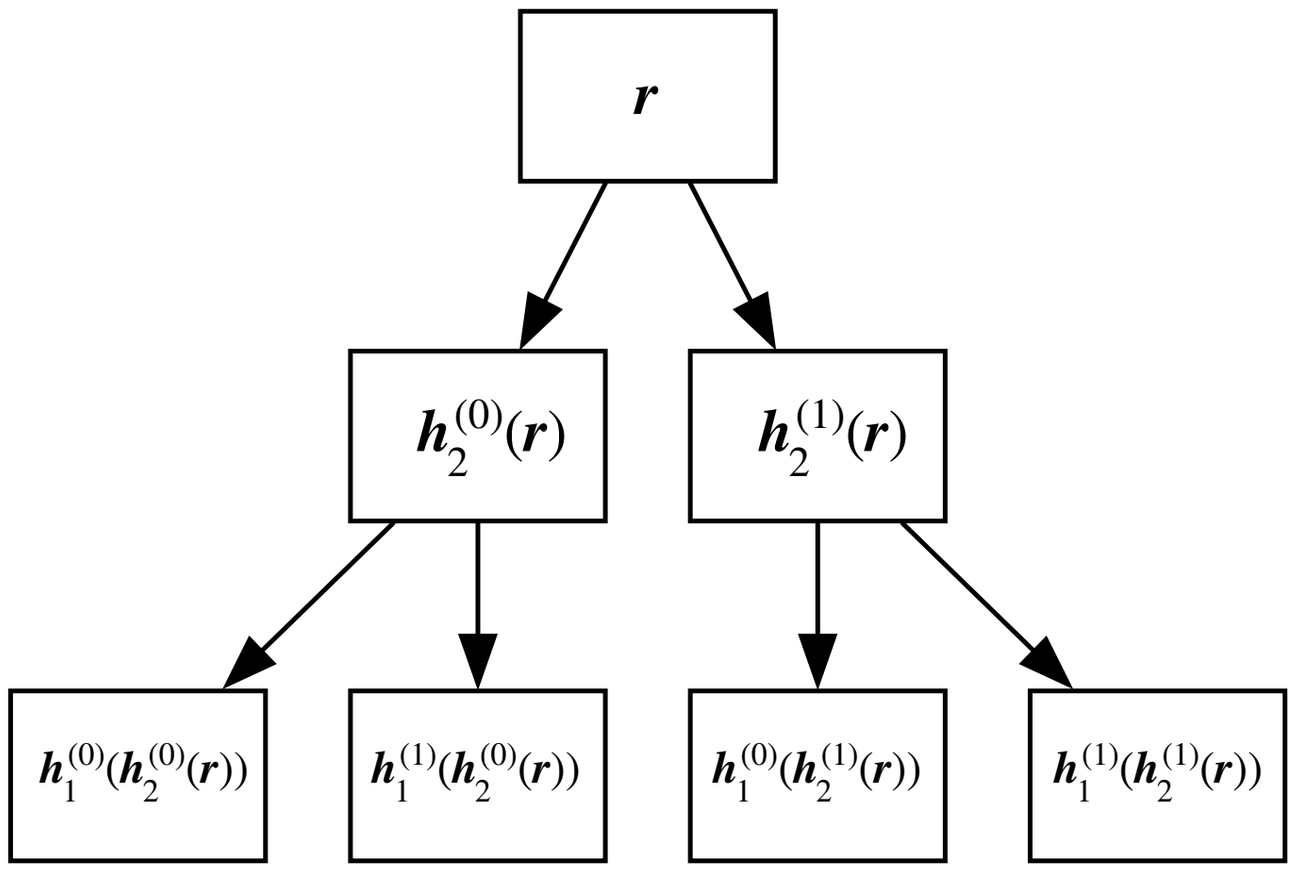
- $\boldsymbol{r}' =$ PRG$(\boldsymbol{r}, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_t)$ is defined similarly with t levels

Guarantees of Nisan's PRG

- $\boldsymbol{r}'$ has a length $m = 2^t \cdot \ell$
- Nisan shows that if $t, w \leq c \cdot \ell$ for a constant c, then the PRG fools a w space algorithm
- If $w = \Omega(\log d)$ and the algorithm reads poly(d) random bits $\Rightarrow$ seed length of $O(w \log d)$ bits
- Can compute any block in time required to apply t hash functions from $\{0,1\}^\ell \to \{0,1\}^\ell$
    - Fool larger space $\Rightarrow$ ℓ needs to be large and the evaluation is slow

Symmetrizing the distribution

- We use **two** hash functions $\boldsymbol{h}_i^{(0)}, \boldsymbol{h}_i^{(1)}$ for each level

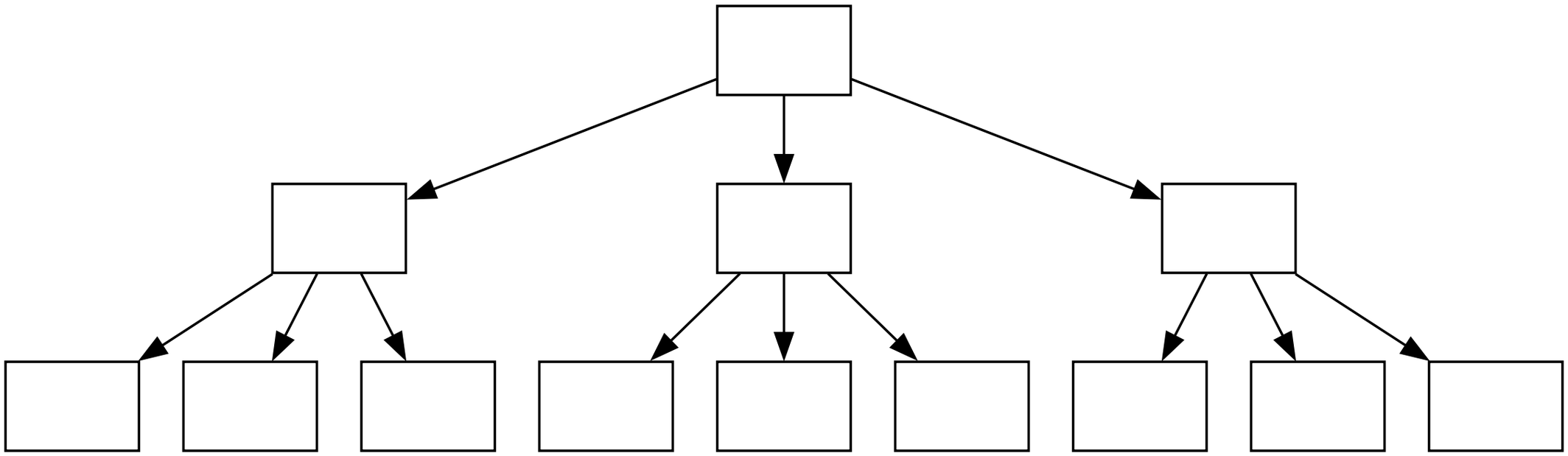


- Switching $\boldsymbol{h}_i^{(0)}$ and $\boldsymbol{h}_i^{(1)}$ amounts to reordering blocks

- Pseudorandom string can be reordered in specific ways without changing the distribution!

Increasing the Branching Factor

- Have b hash functions for each level instead of just 2



- Low depth:
  - Faster to compute any particular block ✅
  - Needs more space to store more hash functions ❌

Important Special Case

- In all our applications, we fool $O(\log d)$ space algorithms that use poly(d) random bits
- Branching factor $b = d^{1/u}$:
  - **Time**: $O(u)$ hash function evaluations per block
  - **Space**: $O(d^{O(1/u)} \log^2 d)$ bits if using Simple Tabulation hashing
- Hash function evaluation in $O(u)$ time in WordRAM model using simple tabulation hashing
- $u = O(\log d)$ recovers Nisan's parameters

Summary of PRGs

- Proof is even simpler than Nisan's original proof!
- Slight but important modification to make sure that the large branching factor doesn't mess up the analysis
- Symmetry allows the analysis to work with a different order of blocks

Recipe for Constructing Streaming Algorithms

- Use linear sketches!
- Design an $s \times d$ sketching matrix $\boldsymbol{S}$ with each of its columns sampled independently from some distribution
  - $s \ll d$
- Show that there is some function f such that $f(\boldsymbol{S}x)$ recovers the statistic we care about
  - F_p moments ($0 < p \leq 2$): $\boldsymbol{S}$ is made of p-stable random vars.

Streaming via Linear Sketch

- Initialize $y \leftarrow 0$
- On update (i, Δ) to x
  - Retrieve $\boldsymbol{S}_{*i}$ - the i-th column of $\boldsymbol{S}$
  - Update $y \leftarrow y + \Delta \cdot \boldsymbol{S}_{*i}$
- $y = \boldsymbol{S}x$ at all points!
- How would we store the **huge** random matrix $\boldsymbol{S}$ though?

Black Box Reduction using PRGs

- PRGs fooling Small Space ⇒ Derandomize Turnstile Streaming Algorithms
    - Indyk, in 2000, gave a simple proof of this
        - Construct an “analysis algo” that generates $\boldsymbol{S}_{*i}$ using i-th block of bits and computes vector y on the fly
        - y’s distribution is preserved!
- Space complexity: space for y + “space for seed”
    - $O(\log d)$ factor space blow up in general
- Symmetry allows to “look ahead” in the reductions

Reducing space blowup

- Do we need to preserve the full distribution of $\boldsymbol{y}$?
- We only care about preserving $f(\boldsymbol{y})$
- If we can compute $f(\boldsymbol{y})$ on the fly, we have to fool a "smaller" space algorithm
- For a lot of algorithms, we can do this and hence only need to fool $O(\log d)$ space algorithms even when $\boldsymbol{y}$ is large
  - No factor $O(\log d)$ space blow up to store the seed!
  - Bulk of the work in our proofs is showing that PRG needs to fool an $O(\log d)$ space algo

Thank you!