Preserving Randomness for Adaptive Algorithms

William M. Hoza[1] Adam R. Klivans

The University of Texas at Austin

August 20, 2018
RANDOM

[1]Supported by the NSF GRFP under Grant DGE-1610403 and by a Harrington Fellowship from UT Austin

Randomized estimation algorithms

- Algorithm $\mathrm{Est}(C)$ estimates some value $\mu(C) \in \mathbb{R}^d$

Randomized estimation algorithms

- Algorithm $\mathrm{Est}(C)$ estimates some value $\mu(C) \in \mathbb{R}^d$

$$\Pr[\|\mathrm{Est}(C) - \mu(C)\|_\infty > \varepsilon] \leq \delta$$

Randomized estimation algorithms

- Algorithm Est(C) estimates some value $\mu(C) \in \mathbb{R}^d$

$$\Pr[\|\text{Est}(C) - \mu(C)\|_\infty > \varepsilon] \leq \delta$$

- Canonical example:

Randomized estimation algorithms

- Algorithm $\mathrm{Est}(C)$ estimates some value $\mu(C) \in \mathbb{R}^d$

$$\Pr[\|\mathrm{Est}(C) - \mu(C)\|_\infty > \varepsilon] \leq \delta$$

- Canonical example:
  - C is a Boolean circuit

Randomized estimation algorithms

- Algorithm $\mathsf{Est}(C)$ estimates some value $\mu(C) \in \mathbb{R}^d$

$$\Pr[\|\mathsf{Est}(C) - \mu(C)\|_\infty > \varepsilon] \leq \delta$$

- Canonical example:
  - C is a Boolean circuit
  - $\mu(C) \stackrel{\text{def}}{=} \Pr_x[C(x) = 1] \quad (d = 1)$

Randomized estimation algorithms

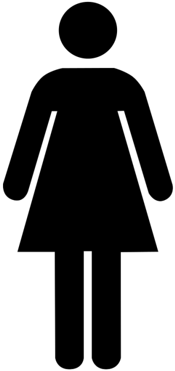
- Algorithm $\mathrm{Est}(C)$ estimates some value $\mu(C) \in \mathbb{R}^d$

$$\Pr[\|\mathrm{Est}(C) - \mu(C)\|_\infty > \varepsilon] \le \delta$$

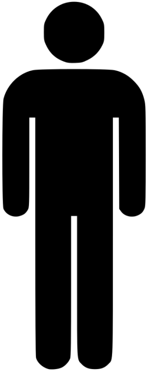
- Canonical example:
  - C is a Boolean circuit
  - $\mu(C) \stackrel{\text{def}}{=} \Pr_x[C(x) = 1] \quad (d = 1)$
  - $\mathrm{Est}(C)$ evaluates C at several randomly chosen points

Using Est as a subroutine

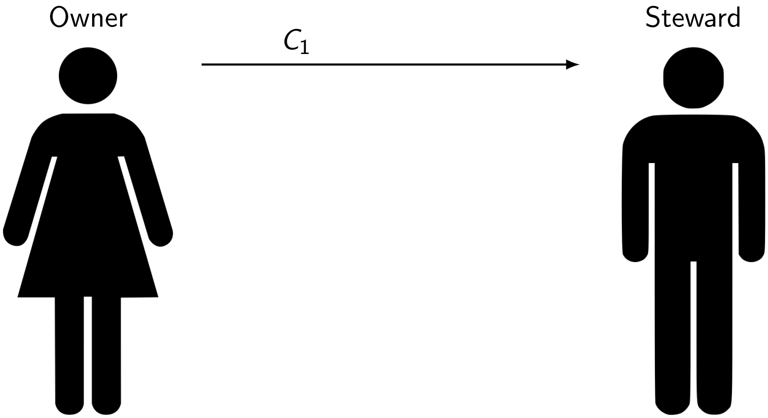
Owner



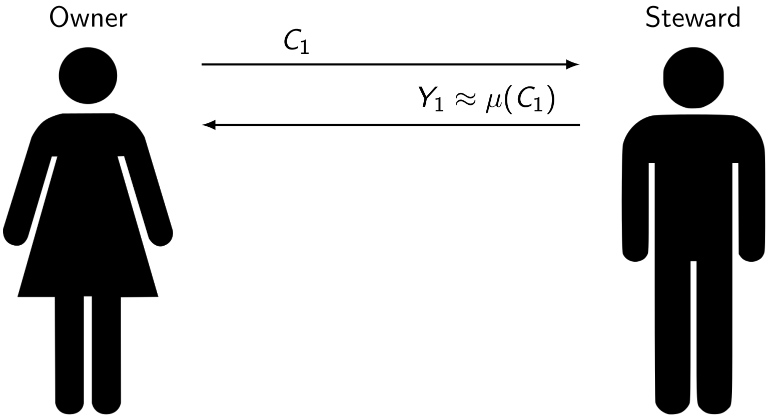
Steward



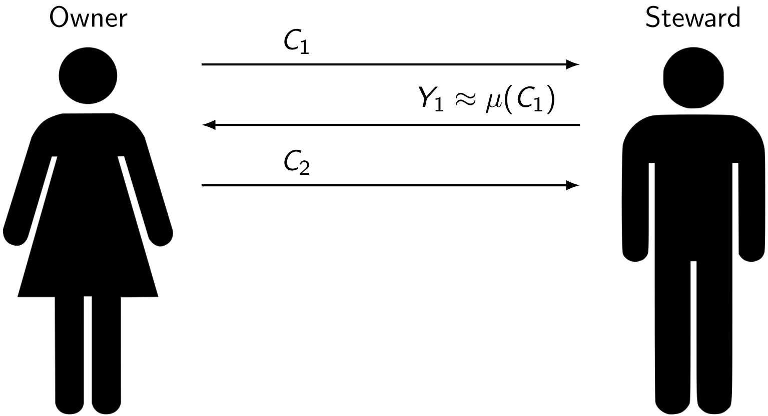
Using Est as a subroutine



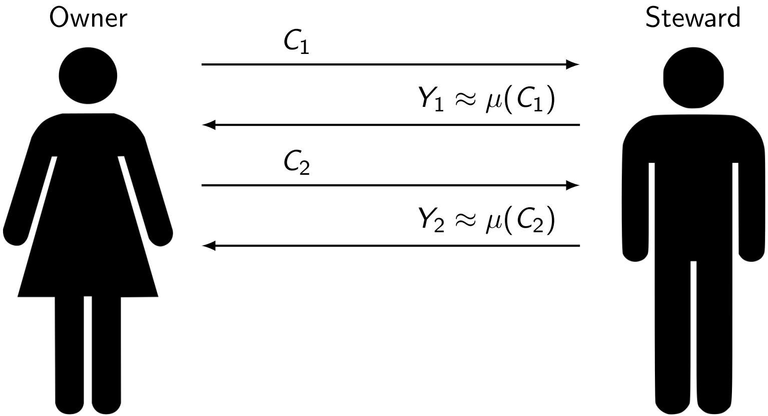
Using Est as a subroutine



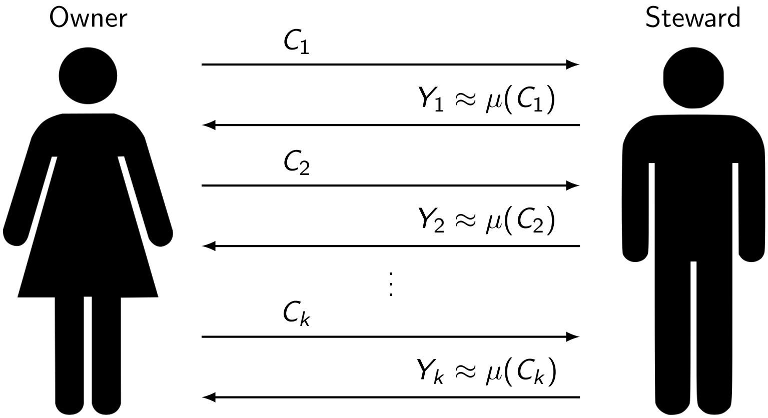
Using Est as a subroutine



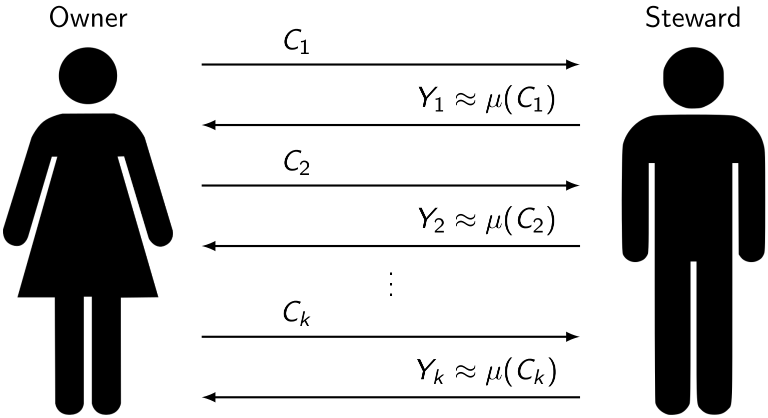
Using Est as a subroutine



Using Est as a subroutine

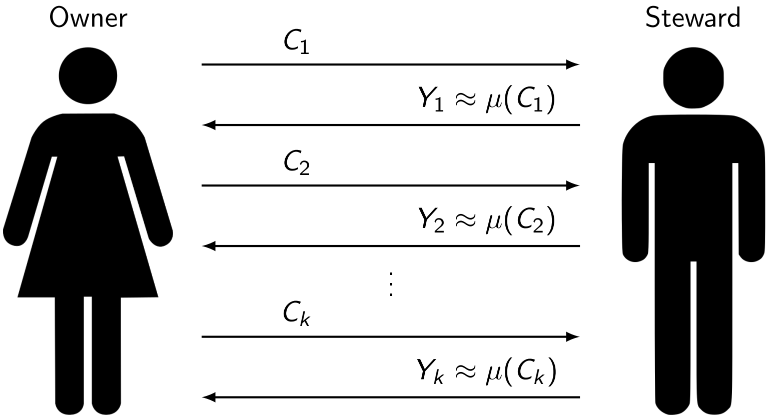


Using Est as a subroutine



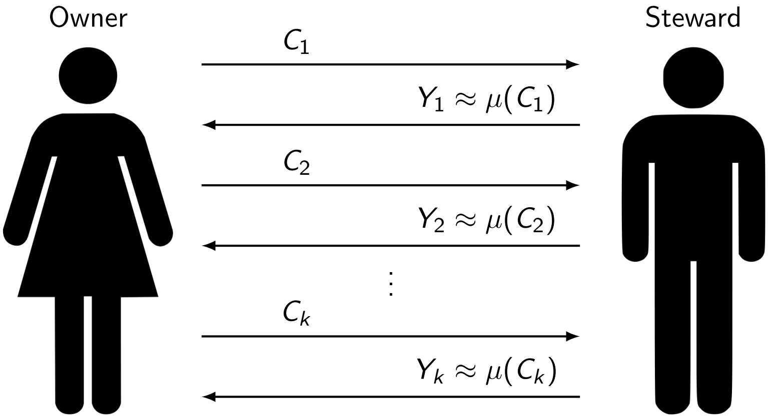
- Suppose Est uses n random bits

Using Est as a subroutine



- Suppose Est uses n random bits
- Naïvely, total number of random bits $= nk$

Using Est as a subroutine



- Suppose Est uses n random bits
- Naïvely, total number of random bits $= nk$
- **Can we do better?**

Main result

- **Theorem** (informal): There is a steward that uses just

$$n + O(k \log(d+1))$$

random bits!

Main result

- **Theorem** (informal): There is a steward that uses just

$$n + O(k \log(d + 1))$$

  random bits!
- Mild increases in both error and failure probability

Main result

- **Theorem** (informal): There is a steward that uses just

$$n + O(k \log(d + 1))$$

  random bits!
- Mild increases in both error and failure probability
- **Prior work**:

Main result

- **Theorem** (informal): There is a steward that uses just

$$n + O(k \log(d+1))$$

  random bits!
- Mild increases in both error and failure probability

- **Prior work**:
  - [Saks, Zhou '99], [Impagliazzo, Zuckerman '89] both imply stewards

Main result

- **Theorem** (informal): There is a steward that uses just

$$n + O(k \log(d+1))$$

random bits!
- Mild increases in both error and failure probability

- **Prior work**:
  - [Saks, Zhou '99], [Impagliazzo, Zuckerman '89] both imply stewards
  - Our steward has better parameters

Outline of our steward

1. Compute pseudorandom bits $X_i \in \{0,1\}^n$

Outline of our steward

1. Compute pseudorandom bits $X_i \in \{0,1\}^n$
2. Compute $W_i := \mathrm{Est}(C_i, X_i)$

Outline of our steward

1. Compute pseudorandom bits $X_i \in \{0, 1\}^n$
2. Compute $W_i := \mathrm{Est}(C_i, X_i)$
3. Compute Y_i by carefully modifying W_i

Pseudorandom bits

- $\text{Gen} : \{0,1\}^s \to \{0,1\}^{nk}$: Variant of INW pseudorandom generator

Pseudorandom bits

- $\mathsf{Gen} : \{0,1\}^s \to \{0,1\}^{nk}$: Variant of INW pseudorandom generator
- Before first round, steward computes

$$(X_1, X_2, \ldots, X_k) = \mathsf{Gen}(U_s)$$

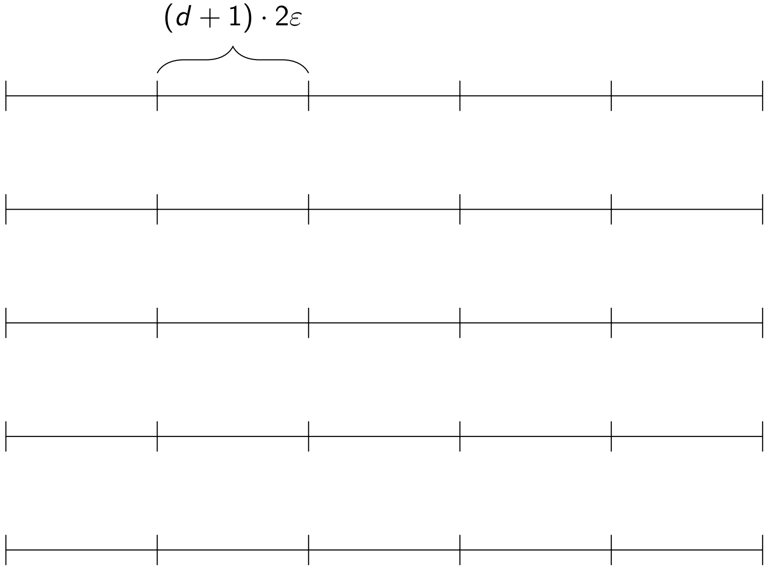
Pseudorandom bits

- $\mathsf{Gen} : \{0,1\}^s \to \{0,1\}^{nk}$: Variant of INW pseudorandom generator
- Before first round, steward computes

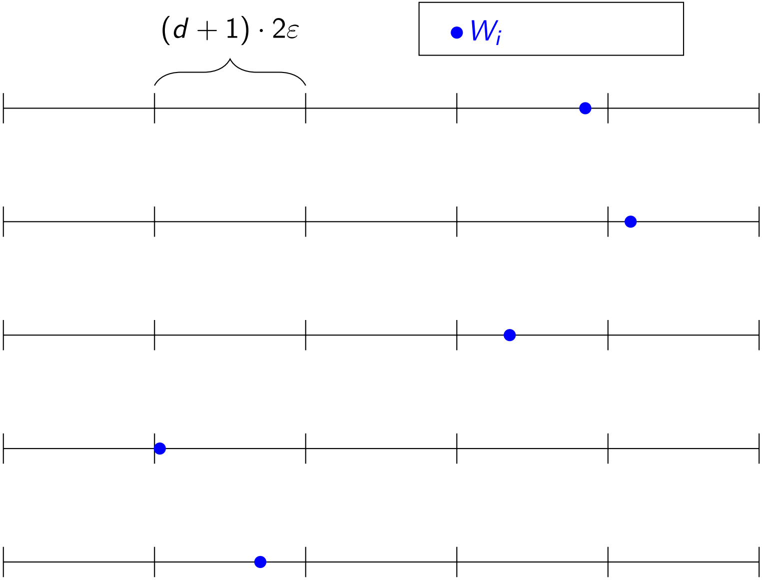
$$(X_1, X_2, \ldots, X_k) = \mathsf{Gen}(U_s)$$

- In round i, steward runs $\mathsf{Est}(C_i, X_i)$

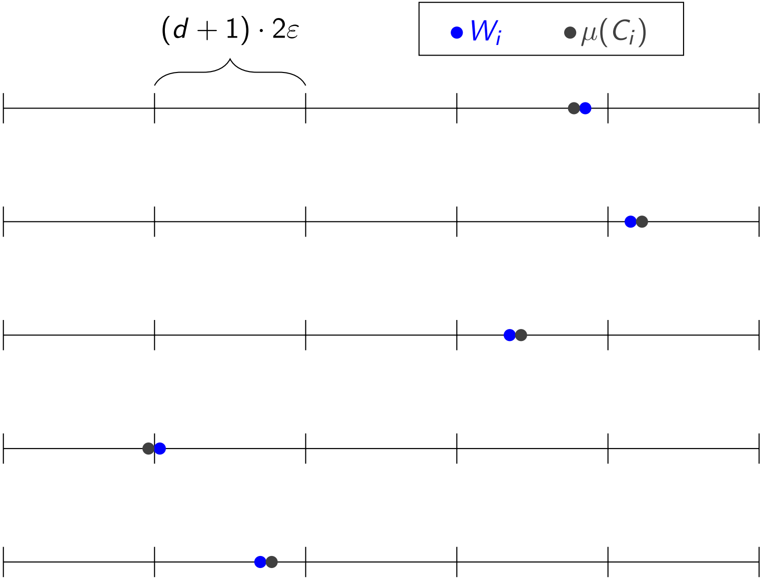
Shifting and rounding



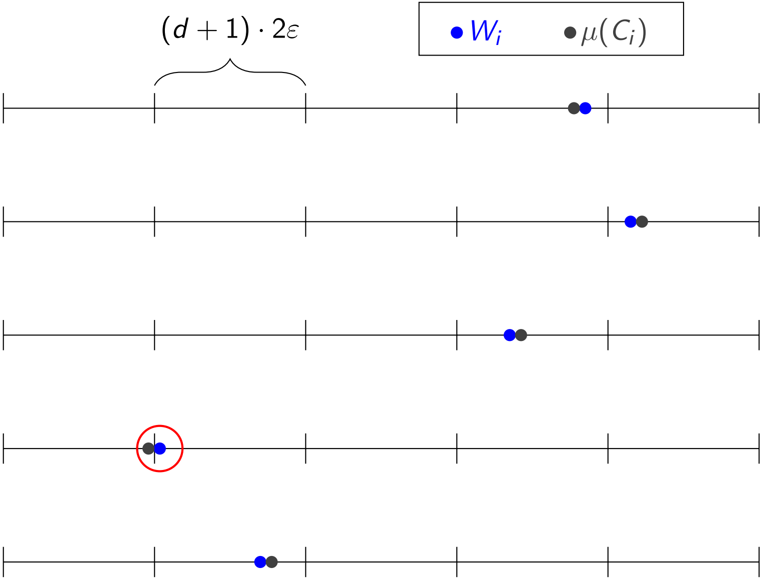
Shifting and rounding



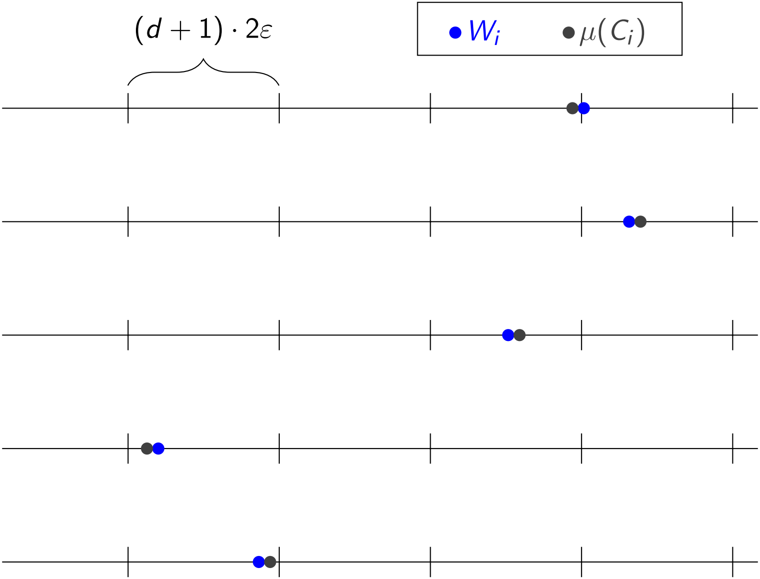
Shifting and rounding



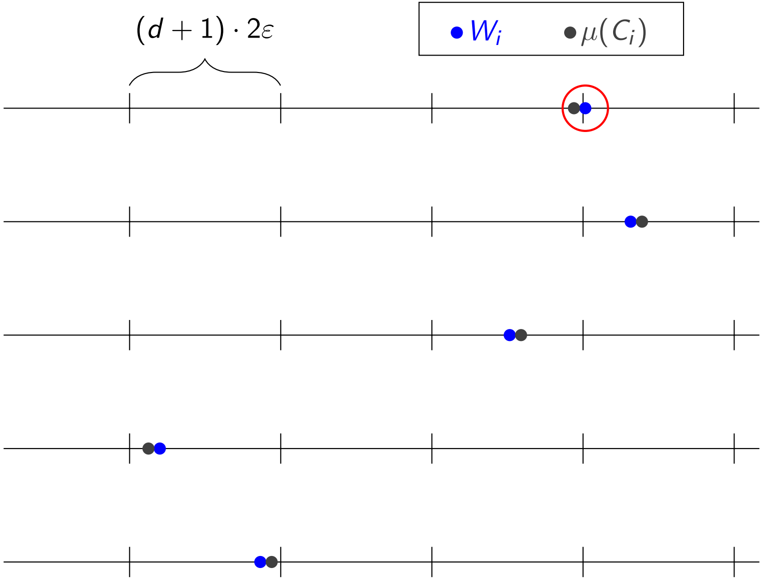
Shifting and rounding



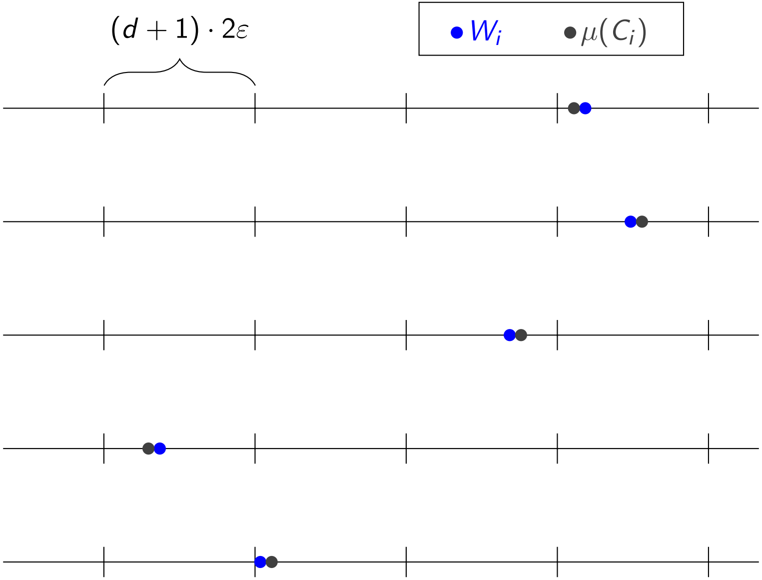
Shifting and rounding



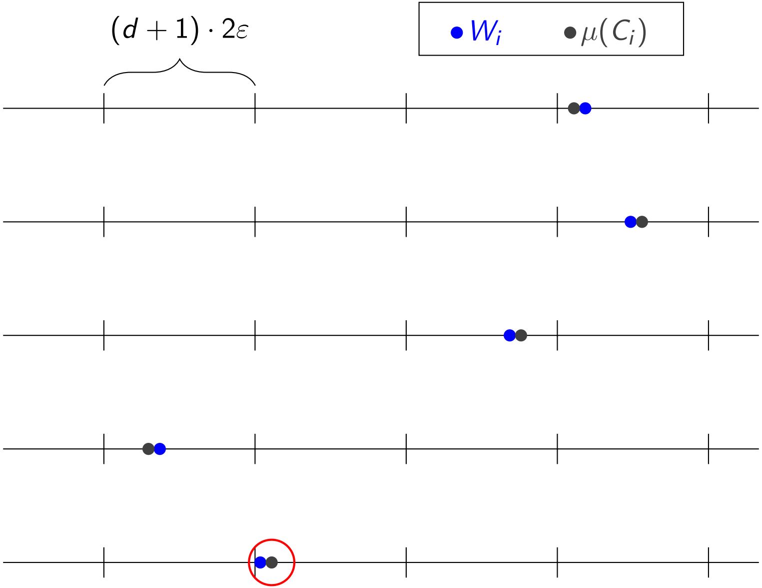
Shifting and rounding



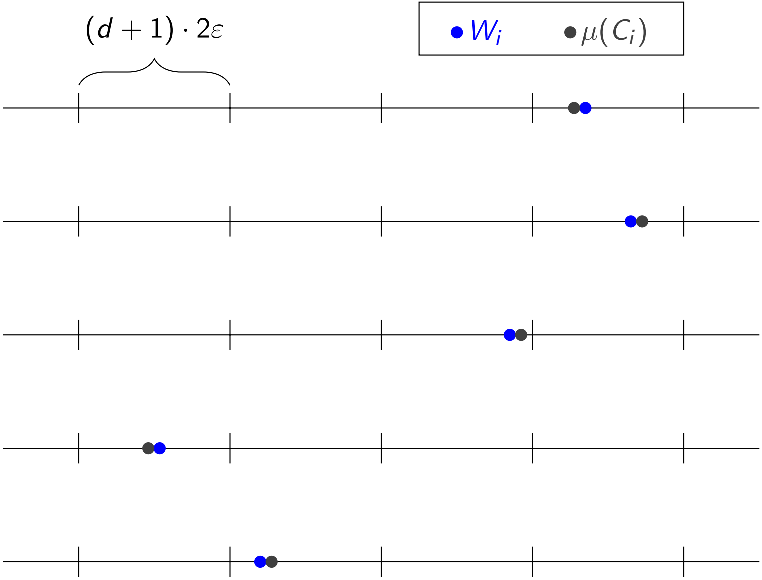
Shifting and rounding



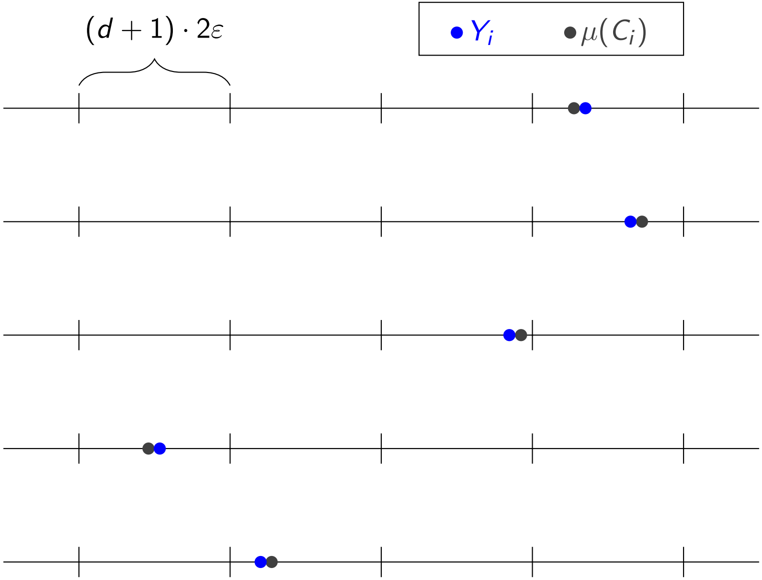
Shifting and rounding



Shifting and rounding



Shifting and rounding



Analysis

- **Theorem** (informal): With high probability, for every i,

$$\|Y_i - \mu(C_i)\|_\infty \leq O(\varepsilon d).$$

Analysis

- **Theorem** (informal): With high probability, for every i,

$$\|Y_i - \mu(C_i)\|_\infty \leq O(\varepsilon d).$$

- **Notation**: For $W \in \mathbb{R}^d, \Delta \in [d+1]$, define $\lfloor W \rceil_\Delta \in \mathbb{R}^d$ by rounding each coordinate to nearest value y such that

$$y \equiv 2\varepsilon\Delta \mod (d+1) \cdot 2\varepsilon$$

Analysis

- **Theorem** (informal): With high probability, for every i,

$$\|Y_i - \mu(C_i)\|_\infty \leq O(\varepsilon d).$$

- **Notation**: For $W \in \mathbb{R}^d, \Delta \in [d+1]$, define $\lfloor W \rceil_\Delta \in \mathbb{R}^d$ by rounding each coordinate to nearest value y such that

$$y \equiv 2\varepsilon\Delta \mod (d+1) \cdot 2\varepsilon$$

- In this notation,

$$Y_i = \lfloor W_i \rceil_\Delta$$

  for a suitable $\Delta \in [d+1]$

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If X_i = fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If $X_i =$ fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

- Consider $d + 2$ cases:

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If $X_i =$ fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

- Consider $d + 2$ cases:
  - $Y_i = \lfloor \mu(C_i) \rceil_1$, or

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If $X_i =$ fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

- Consider $d + 2$ cases:
  - $Y_i = \lfloor \mu(C_i) \rceil_1$, or
  - $Y_i = \lfloor \mu(C_i) \rceil_2$, or

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If $X_i =$ fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

- Consider $d + 2$ cases:
  - $Y_i = \lfloor \mu(C_i) \rceil_1$, or
  - $Y_i = \lfloor \mu(C_i) \rceil_2$, or

  $\vdots$

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If $X_i =$ fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

- Consider $d + 2$ cases:
  - $Y_i = \lfloor \mu(C_i) \rceil_1$, or
  - $Y_i = \lfloor \mu(C_i) \rceil_2$, or

    $\vdots$

  - $Y_i = \lfloor \mu(C_i) \rceil_{d+1}$, or

Analysis (continued)

- $Y_i = \lfloor W_i \rceil_\Delta$
- If X_i = fresh randomness, then w.h.p.,

$$\lfloor W_i \rceil_\Delta = \lfloor \mu(C_i) \rceil_\Delta$$

- Consider $d+2$ cases:
  - $Y_i = \lfloor \mu(C_i) \rceil_1$, or
  - $Y_i = \lfloor \mu(C_i) \rceil_2$, or

    $\vdots$
  - $Y_i = \lfloor \mu(C_i) \rceil_{d+1}$, or
  - Y_i = something else.

Block decision tree

C_1

Block decision tree

$$\begin{tikzcd}
 & & & C_1 \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & & \\
C_2^{(1)} & & C_2^{(2)} & & C_2^{(3)} & & \perp
\end{tikzcd}$$

Block decision tree

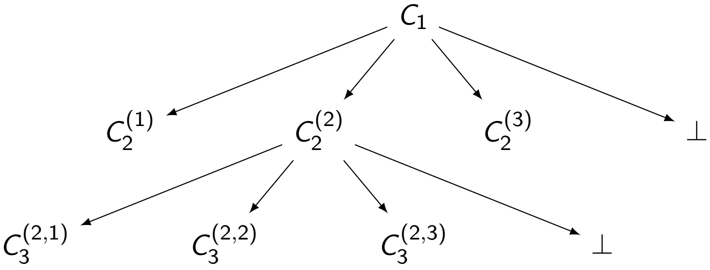
$$
\begin{tikzcd}
 & & & & C_1 \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & & \\
 & C_2^{(1)} & & C_2^{(2)} \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & C_2^{(3)} & & \perp \\
C_3^{(2,1)} & & C_3^{(2,2)} & & C_3^{(2,3)} & & \perp &
\end{tikzcd}
$$

Block decision tree

$$
\begin{tikzcd}
 & & & & C_1 \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & & \\
 & C_2^{(1)} & & C_2^{(2)} \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & C_2^{(3)} & & \perp \\
C_3^{(2,1)} & & C_3^{(2,2)} & & C_3^{(2,3)} & & \perp &
\end{tikzcd}
$$

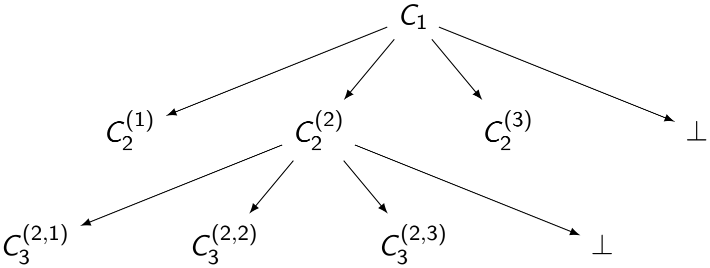
- A sequence $(X_1, \ldots, X_k)$ determines:

Block decision tree



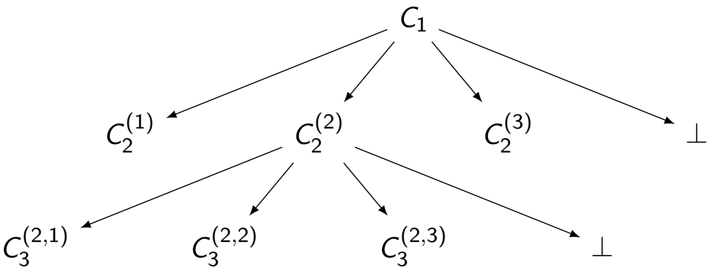
- A sequence $(X_1, \ldots, X_k)$ determines:
  - A transcript $(C_1, Y_1, C_2, Y_2, \ldots, C_k, Y_k)$

Block decision tree



- A sequence $(X_1, \ldots, X_k)$ determines:
  - A transcript $(C_1, Y_1, C_2, Y_2, \ldots, C_k, Y_k)$
  - A path P through tree

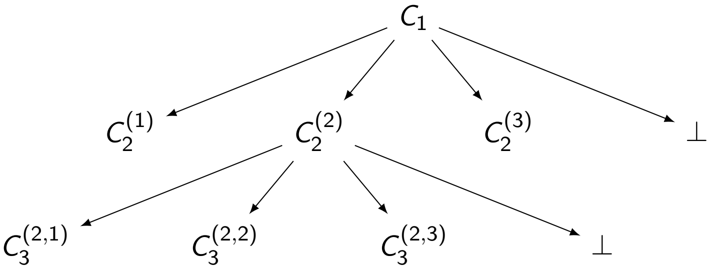
Block decision tree



- A sequence $(X_1, \ldots, X_k)$ determines:
  - A transcript $(C_1, Y_1, C_2, Y_2, \ldots, C_k, Y_k)$
  - A path P through tree
- If we pick $X_1, \ldots, X_k$ independently and u.a.r.,

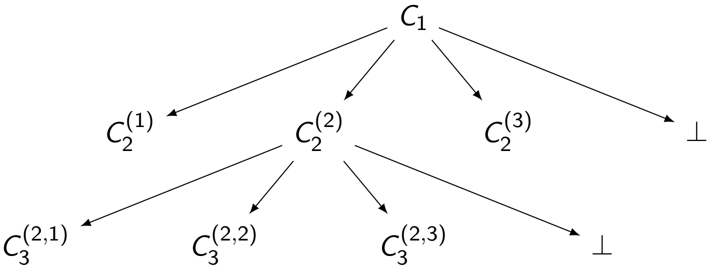
$$\Pr_{(X_1, \ldots, X_k)}[P \text{ has a } \perp \text{ node}] \leq k\delta$$

Fooling the tree



- Tree has low memory

Fooling the tree



- Tree has low memory
- So when $X_1, \ldots, X_k$ are pseudorandom,

$$\Pr_{(X_1,\ldots,X_k)}[P \text{ has a } \perp \text{ node}] \leq k\delta + \gamma$$

The tree certifies correctness

$$
\begin{tikzcd}
 & & & & C_1 \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & & \\
 & C_2^{(1)} & & C_2^{(2)} \arrow[dlll] \arrow[dl] \arrow[dr] \arrow[drrr] & & C_2^{(3)} & & \perp \\
C_3^{(2,1)} & & C_3^{(2,2)} & & C_3^{(2,3)} & & \perp &
\end{tikzcd}
$$

- (Certification) No $\perp$ nodes in $P \implies$ every Y_i has error $O(\varepsilon d)$

Application: Randomness-efficient Goldreich-Levin

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2} + \theta)$-fraction of positions

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2} + \theta)$-fraction of positions
  - Runtime $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$ $\quad$ (δ = failure prob)

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2}+\theta)$-fraction of positions
  - Runtime $\text{poly}(n, 1/\theta, \log(1/\delta))$ $\quad$ ($\delta =$ failure prob)
  - $O(n + \log n \log(1/\delta))$ random bits (independent of θ!)

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2} + \theta)$-fraction of positions
  - Runtime $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$ $\quad$ ($\delta =$ failure prob)
  - $O(n + \log n \log(1/\delta))$ random bits (independent of θ!)
- Previous best: $O(n \log(n/\theta) \log(1/(\delta\theta)))$ random bits (Bshouty et al. '04)

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2} + \theta)$-fraction of positions
  - Runtime $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$ $(\delta = \text{failure prob})$
  - $O(n + \log n \log(1/\delta))$ random bits (independent of θ!)
- Previous best: $O(n \log(n/\theta) \log(1/(\delta\theta)))$ random bits (Bshouty et al. '04)
- Proof ingredients:

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2}+\theta)$-fraction of positions
  - Runtime $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$ $\quad$ (δ = failure prob)
  - $O(n + \log n \log(1/\delta))$ random bits (independent of θ!)
- Previous best: $O(n \log(n/\theta) \log(1/(\delta\theta)))$ random bits (Bshouty et al. '04)
- Proof ingredients:
  - Standard Goldreich-Levin algorithm

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2} + \theta)$-fraction of positions
  - Runtime $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$ $\quad$ (δ = failure prob)
  - $O(n + \log n \log(1/\delta))$ random bits (independent of θ!)
- Previous best: $O(n \log(n/\theta) \log(1/(\delta\theta)))$ random bits (Bshouty et al. '04)
- Proof ingredients:
  - Standard Goldreich-Levin algorithm
  - Our steward with $d = \mathrm{poly}(1/\theta)$

Application: Randomness-efficient Goldreich-Levin

- Oracle access to $x \in \{0,1\}^{2^n}$
- **Theorem**: Can find all Hadamard codewords that agree with x in $(\frac{1}{2} + \theta)$-fraction of positions
  - Runtime $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$ $\quad(\delta = \text{failure prob})$
  - $O(n + \log n \log(1/\delta))$ random bits (independent of θ!)
- Previous best: $O(n \log(n/\theta) \log(1/(\delta\theta)))$ random bits (Bshouty et al. '04)
- Proof ingredients:
  - Standard Goldreich-Levin algorithm
  - Our steward with $d = \mathrm{poly}(1/\theta)$
  - Goldreich-Wigderson sampler

Open questions

- Optimal randomness complexity when d is large?

Open questions

- Optimal randomness complexity when d is large?
- Avoid error blowup $\varepsilon \to O(\varepsilon d)$?

Open questions

- Optimal randomness complexity when d is large?
- Avoid error blowup $\varepsilon \to O(\varepsilon d)$?
- More applications?

Open questions

- Optimal randomness complexity when d is large?
- Avoid error blowup $\varepsilon \to O(\varepsilon d)$?
- More applications?

- Thanks! Questions?