Preserving Randomness for Adaptive Algorithms

William M. Hoza[1]
Department of Computer Science, University of Texas at Austin, Austin, TX, USA
whoza@utexas.edu
https://orcid.org/0000-0001-5162-9181

Adam R. Klivans
Department of Computer Science, University of Texas at Austin, Austin, TX, USA
klivans@cs.utexas.edu

Abstract

Suppose Est is a randomized estimation algorithm that uses n random bits and outputs values in $\mathbb{R}^d$. We show how to execute Est on k adaptively chosen inputs using only $n + O(k \log(d+1))$ random bits instead of the trivial nk (at the cost of mild increases in the error and failure probability). Our algorithm combines a variant of the INW pseudorandom generator [12] with a new scheme for shifting and rounding the outputs of Est. We prove that modifying the outputs of Est is necessary in this setting, and furthermore, our algorithm's randomness complexity is near-optimal in the case $d \leq O(1)$. As an application, we give a randomness-efficient version of the Goldreich-Levin algorithm; our algorithm finds all Fourier coefficients with absolute value at least θ of a function $F : \{0,1\}^n \to \{-1,1\}$ using $O(n \log n) \cdot \mathrm{poly}(1/\theta)$ queries to F and $O(n)$ random bits (independent of θ), improving previous work by Bshouty et al. [3].

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases pseudorandomness, adaptivity, estimation

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.43

Related Version [10], `https://arxiv.org/abs/1611.00783`

Acknowledgements We thank David Zuckerman for observations about block decision trees. We thank an anonymous reviewer for pointing out Impagliazzo and Zuckerman's previous work on this subject [13, 11].

1 Introduction

Let Est be a randomized algorithm that estimates some quantity $\mu(C) \in \mathbb{R}^d$ when given input C. The canonical example is the case when C is a Boolean circuit, $d = 1$, $\mu(C) \overset{\text{def}}{=} \Pr_x[C(x) = 1]$, and Est estimates $\mu(C)$ by evaluating C at several randomly chosen points. Suppose that Est uses n random bits, and $\Pr[\|\mathsf{Est}(C) - \mu(C)\|_\infty > \varepsilon] \leq \delta$.

Furthermore, suppose we want to use Est as a *subroutine*, executing it on inputs $C_1, C_2, \dots, C_k$, where each C_i is chosen adaptively based on the previous outputs of Est. The naïve implementation uses nk random bits and fails with probability at most $k\delta$.

In this work, we show how to generically improve the randomness complexity of any algorithm with this structure, without increasing the number of executions of Est, at the

[1] Supported by the NSF GRFP under Grant DGE-1610403 and by a Harrington Fellowship from the University of Texas at Austin.

expense of mild increases in the error and failure probability. Our algorithm efficiently finds $Y_1, \dots, Y_k \in \mathbb{R}^d$ with $\|Y_i - \mu(C_i)\|_\infty \leq O(\varepsilon d)$ for every i, our algorithm has failure probability $k\delta + \gamma$ for any $\gamma > 0$, and our algorithm uses a total of $n + O(k \log(d+1) + (\log k)\log(1/\gamma))$ random bits.

1.1 The randomness steward model

We model the situation described above by imagining two interacting agents: the *owner* (who plays the role of the outer algorithm) chooses the inputs $C_1, \dots, C_k$, while the *steward* (who replaces the direct execution of Est) provides the output vectors $Y_1, \dots, Y_k \in \mathbb{R}^d$. The reader might find it useful to think of the steward as a trusted "cloud computing" service. To justify the names, imagine that the owner gives the steward "stewardship" over her random bits. The steward's job is to "spend" as little randomness as possible without sacrificing too much accuracy.

To describe the model more rigorously, say that a function $f : \{0,1\}^n \to \mathbb{R}^d$ is (ε, δ)-*concentrated at* $\mu \in \mathbb{R}^d$ if $\Pr_{X \in \{0,1\}^n}[\|f(X) - \mu\|_\infty > \varepsilon] \leq \delta$. In each round i, the chosen input C_i defines a concentrated function $f_i(X) \overset{\text{def}}{=} \mathsf{Est}(C_i, X)$, so it is equivalent to imagine that the owner picks an arbitrary concentrated function. In the following definition, ε' is the error of the steward, and δ' is its failure probability.

▶ **Definition 1.** An (ε', δ')*-steward for* k *adaptively chosen* (ε, δ)*-concentrated functions* $f_1, \dots, f_k : \{0,1\}^n \to \mathbb{R}^d$ is a randomized algorithm S that interacts with an *owner* O according to the following protocol.

1. For $i = 1$ to k:
   a. O chooses $f_i : \{0,1\}^n \to \mathbb{R}^d$ that is (ε, δ)-concentrated at some point $\mu_i \in \mathbb{R}^d$ and gives it to S.
   b. S chooses $Y_i \in \mathbb{R}^d$ and gives it to O.

Write $\mathsf{O} \leftrightarrow \mathsf{S}$ ("the interaction of O with S") to denote the above interaction. The requirement on S is that for all O,

$$\Pr[\max_i \|Y_i - \mu_i\|_\infty > \varepsilon' \text{ in } \mathsf{O} \leftrightarrow \mathsf{S}] \leq \delta'. \tag{1}$$

The probability is taken over the internal randomness of S and O.

From an information-theoretic perspective, stewards as defined above are not particularly interesting, because S could exhaustively examine all outputs of f_i to deterministically compute a point Y_i where f_i is concentrated. But we would like to avoid executing Est more than k times in total, so we will restrict attention to *one-query stewards*:

▶ **Definition 2.** A *one-query steward* is a steward that only accesses each f_i by querying it at a single point $X_i \in \{0,1\}^n$. (The point X_i is not seen by the owner.)

1.2 Our results

1.2.1 Main result: A one-query steward with good parameters

Our main result is the explicit construction of a one-query steward that simultaneously achieves low error, low failure probability, and low randomness complexity:

ε'	δ'	Randomness complexity	Reference
ε	$k\delta$	nk	Naïve
$O(\varepsilon d)$	$k\delta + \gamma$	$n + O(k \log(d+1) + (\log k)\log(1/\gamma))$	Theorem 3 (main)
$O(\varepsilon)$	$k\delta + \gamma$	$n + O(kd + (\log k)\log(1/\gamma))$	Full version [10]
$O(\varepsilon d)$	$k\delta + \gamma$	$n + k\log(d+2) + 2\log(1/\gamma) + O(1)$	Full version [10][a]
$O(\varepsilon d)$	$2^{O(k \log(d+1))} \cdot \delta$	n	Full version [10]
$O(\varepsilon)$	$2^{O(kd)} \cdot \delta$	n	Full version [10]
$O(\varepsilon kd/\gamma)$	$k\delta + \gamma$	$n + O(k\log k + k\log d + k\log(1/\gamma))$	Based on [21]
$O(\varepsilon)$	$k\delta + k \cdot 2^{-n^{\Omega(1)}}$	$O(n^6 + kd)$	Based on [13]
Any	Any ≤ 0.2	$n + \Omega(k) - \log(\delta'/\delta)$ (lower bound)	Full version [10]

[a] Computationally inefficient.

Figure 1 Upper and lower bounds for one-query stewards. Recall that ε, δ are the concentration parameters of $f_1, \ldots, f_k$ (i.e. the error and failure probability of the estimation algorithm Est); ε', δ' are the error and failure probability of the steward S; n is the number of input bits to each f_i (i.e. the number of random coins used by Est); k is the number of rounds of adaptivity; d is the dimension of the output of each f_i (i.e. the dimension of the output of Est). Everywhere it appears, γ denotes an arbitrary positive number.

▶ **Theorem 3.** *For any $n, k, d \in \mathbb{N}$ and any $\varepsilon, \delta, \gamma > 0$, there exists a one-query $(O(\varepsilon d), k\delta + \gamma)$-steward for k adaptively chosen (ε, δ)-concentrated functions $f_1, \ldots, f_k : \{0,1\}^n \to \mathbb{R}^d$ with randomness complexity*

$$n + O(k \log(d+1) + (\log k)\log(1/\gamma)). \tag{2}$$

The total running time of the steward is $\operatorname{poly}(n, k, d, \log(1/\varepsilon), \log(1/\gamma))$.

We also give several variant stewards that achieve tradeoffs in parameters. (See Figure 1.)

1.2.2 Application: Acceptance probabilities of Boolean circuits

A natural application of Theorem 3 is a time- and randomness-efficient algorithm for estimating the acceptance probabilities of many adaptively chosen Boolean circuits.

▶ **Corollary 4.** *There exists a randomized algorithm with the following properties. Initially, the algorithm is given parameters $n, k \in \mathbb{N}$ and $\varepsilon, \delta > 0$. Then, in round i $(1 \leq i \leq k)$, the algorithm is given a Boolean circuit C_i on n input bits and outputs a number $Y_i \in [0,1]$. Here, C_i may be chosen adversarially based on $Y_1, \ldots, Y_{i-1}$. With probability $1 - \delta$, every Y_i is $\mu(C_i) \pm \varepsilon$, where $\mu(C_i) \overset{\text{def}}{=} \Pr_x[C_i(x) = 1]$. The total running time of the algorithm is*

$$O\left(\frac{\log k + \log(1/\delta)}{\varepsilon^2} \cdot \sum_{i=1}^{k} \operatorname{size}(C_i)\right) + \operatorname{poly}(n, k, 1/\varepsilon, \log(1/\delta)), \tag{3}$$

and the total number of random bits used by the algorithm is $n + O(k + (\log k) \cdot \log(1/\delta))$.

Corollary 4 should be compared to the case when $C_1, \ldots, C_k$ are chosen nonadaptively, for which the randomness complexity can be improved to $n + O(\log k + \log(1/\delta))$ by applying the Goldreich-Wigderson randomness-efficient sampler for Boolean functions [9] and reusing randomness. The proof of Corollary 4 works by combining the GW sampler with our steward. We also give similar algorithms for simulating an oracle for **promise-BPP** or **APP**.

1.2.3 Application: The Goldreich-Levin algorithm

For our main application, we give a randomness-efficient version of the Goldreich-Levin algorithm [8] (otherwise known as the Kushilevitz-Mansour algorithm [14]) for finding noticeably large Fourier coefficients. Given oracle access to $F : \{0,1\}^n \to \{-1,1\}$, for any $\theta > 0$, we show how to efficiently find a list containing all U with $|\widehat{F}(U)| \geq \theta$. (Alternatively, thinking of F as an exponentially long bitstring $F \in \{-1,1\}^{2^n}$, our algorithm finds all Hadamard codewords that agree with F in a $\left(\frac{1}{2}+\theta\right)$-fraction of positions.) Our algorithm makes $O(n\log(n/\delta)) \cdot \mathrm{poly}(1/\theta)$ queries to F, uses $O(n + (\log n)\log(1/\delta))$ random bits, and has failure probability δ. The number of random bits *does not depend on* θ. To achieve such a low randomness complexity, we first improve the randomness efficiency of each estimate in the Goldreich-Levin algorithm using the GW sampler. Then, we reduce the number of rounds of adaptivity by a factor of $\log(1/\theta)$ by making many estimates within each round. Interestingly, we apply our steward with $d = \mathrm{poly}(1/\theta)$, unlike the proof of Corollary 4 where we choose $d = 1$. (Recall that d is the number of real values estimated in each round.)

1.2.4 Randomness complexity lower bound

We prove a randomness complexity lower bound of $n + \Omega(k) - \log(\delta'/\delta)$ for any one-query steward. In the case $d \leq O(1)$, this comes close to matching our upper bounds. For example, to achieve $\delta' \leq O(k\delta)$, this lower bound says that $n + \Omega(k)$ random bits are needed; our main steward (Theorem 3) achieves $\varepsilon' \leq O(\varepsilon), \delta' \leq O(k\delta)$ using $n + O(k + (\log k)\log(1/\delta))$ random bits. At the other extreme, if we want a one-query steward that uses only n random bits, this lower bound says that the failure probability will be $\delta' \geq \exp(\Omega(k)) \cdot \delta$; one of our variant stewards uses n random bits to achieve $\varepsilon' \leq O(\varepsilon)$ and $\delta' \leq \exp(O(k)) \cdot \delta$.

1.3 Techniques

1.3.1 Block decision trees

A key component in the proof of our main result (Theorem 3) is a pseudorandom generator (PRG) for a new model that we call the *block decision tree* model. Informally, a block decision tree is a decision tree that reads its input from left to right, n bits at a time:

▶ **Definition 5.** For a finite alphabet Σ, a (k, n, Σ) *block decision tree* is a rooted tree $T = (V, E)$ of height k in which every node v at depth $< k$ has exactly $|\Sigma|$ children (labeled with the symbols in Σ) and has an associated function $v : \{0,1\}^n \to \Sigma$. We identify T with a function $T : (\{0,1\}^n)^{\leq k} \to V$ defined recursively: T(the empty string) = the root node, and if $T(X_1, \ldots, X_{i-1}) = v$, then $T(X_1, \ldots, X_i)$ is the child of v labeled $v(X_i)$.

The standard nonconstructive argument shows that there exists a γ-PRG for block decision trees with seed length $n + k\log|\Sigma| + 2\log(1/\gamma) + O(1)$. (See Appendix A.1 for the definition of a PRG in this setting.) We construct an explicit PRG with a nearly matching seed length:

▶ **Theorem 6.** *For every $n, k \in \mathbb{N}$, every finite alphabet Σ, and every $\gamma > 0$, there exists a γ-PRG* $\mathsf{Gen} : \{0,1\}^s \to \{0,1\}^{nk}$ *for (k, n, Σ) block decision trees with seed length*

$$s \leq n + O(k\log|\Sigma| + (\log k)\log(1/\gamma)). \tag{4}$$

The PRG can be computed in $\mathrm{poly}(n, k, \log|\Sigma|, \log(1/\gamma))$ *time.*

We prove Theorem 6 by modifying the INW generator for space-bounded computation [12].

1.3.2 Shifting and rounding

1.3.2.1 PRGs alone are not enough

Interestingly, to design good stewards, the standard technique of replacing truly random bits with pseudorandom bits is not sufficient. That is, define a *pseudorandom generation steward* to be a steward that simply queries each f_i at a pseudorandomly chosen point X_i and returns $Y_i = f_i(X_i)$. In the full version of this paper [10], we show that any pseudorandom generation steward must use at least $\Omega(nk)$ random bits, assuming $\delta' \leq 1/2$ and $\delta \geq 2^{-n/2+1}$.

1.3.2.2 Deliberately introducing error

To circumvent this $\Omega(nk)$ lower bound, our steward is forced to *modify the outputs* of $f_1, \ldots, f_k$. In each round, our main steward queries f_i at a pseudorandom point X_i, chooses a nearby value $Y_i \approx f_i(X_i)$, and returns this modified value Y_i to the owner O. The motivation for this idea is that by deliberately introducing a small amount of error, we reduce the amount of information about X_i that is leaked by Y_i. This way, we can recycle some of the randomness of X_i for future rounds.

To be more specific, for a steward S, let $\mathsf{S}(X)$ denote S using randomness X. Our main steward is of the form $\mathsf{S}(X) \overset{\text{def}}{=} \mathsf{S}_0(\mathsf{Gen}(X))$. Here, Gen is our PRG for block decision trees, and S_0 is a randomness-inefficient one-query steward. In each round, S_0 queries f_i at a fresh random point $X_i \in \{0,1\}^n$, but S_0 computes the return value Y_i by carefully *shifting and rounding* each coordinate of $f_i(X_i)$. This deterministic shifting and rounding procedure and its analysis are our main technical contributions.

1.3.2.3 Relation to block decision trees

To explain why this works, observe that when any steward and owner interact, it is natural to model the owner's behavior by a decision tree that branches at each node based on the value Y_i provided by the steward. The *branching factor* of this decision tree is a simple measure of the amount of information leaked. If S_0 simply returned $f_i(X_i)$ without any shifting or rounding, the branching factor for $\mathsf{O} \leftrightarrow \mathsf{S}_0$ would be 2^n. Three ideas dramatically reduce this branching factor.

- The first idea is to round. Suppose that S_0 rounded each coordinate of $f_i(X_i)$ to the nearest multiple of 2ε (with no shifting). Then the branching factor would be reduced to $2^d + \delta 2^n$. The $\delta 2^n$ term corresponds to the outputs of f_i that are far from its concentration point μ_i. The 2^d term corresponds to outputs $f_i(X_i)$ that are close to μ_i; if each coordinate of μ_i is approximately equidistant from two multiples of 2ε, then the corresponding coordinate of $f_i(X_i)$ could be rounded to either of those two values.
- The second idea is to *shift* each coordinate of $f_i(X_i)$ before rounding. In particular, S_0 finds a single value Δ_i such that after adding $\Delta_i \cdot 2\varepsilon$ to each coordinate of $f_i(X_i)$, every coordinate is ε-far from every rounding boundary. Then, S_0 rounds the shifted coordinates to obtain Y_i. This procedure reduces the branching factor down to $d+1+\delta 2^n$. To understand why, think of Δ_i as a *compressed* representation of Y_i. Assuming $f_i(X_i)$ is close to μ_i, given unlimited computation time, O could recover Y_i from Δ_i by computing the *true* vector μ_i, shifting *it* according to Δ_i, and rounding. Hence, each node of the tree just needs to have one child for each possible Δ_i value (along with the $\delta 2^n$ children for the case that $f_i(X_i)$ is far from μ_i).
- The third idea is to *relax* the requirement that the tree perfectly computes $\mathsf{O} \leftrightarrow \mathsf{S}_0$. In particular, for every owner O, we construct a block decision tree T_{O} that merely *certifies*

correctness of $\mathsf{O} \leftrightarrow \mathsf{S}_0$. That is, for any $X_1, \dots, X_k$, if the node $T_{\mathsf{O}}(X_1, \dots, X_k)$ indicates "success", then the error $\max_i \|Y_i - \mu_i\|_\infty$ in $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \dots, X_k)$ is small. On the other hand, if $T_{\mathsf{O}}(X_1, \dots, X_k)$ does not indicate success, then "all bets are off": the error $\max_i \|Y_i - \mu_i\|_\infty$ in $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \dots, X_k)$ may be small or large. Our certification tree has the additional property that

$$\Pr_{X_1,\dots,X_k}[T_{\mathsf{O}}(X_1, \dots, X_k) \text{ indicates success}] \geq 1 - k\delta. \tag{5}$$

This relaxation allows us to reduce the branching factor down to just $d + 2$, because for each node, the $\delta 2^n$ children corresponding to outputs of f_i that are far from μ_i can all be merged into a single "failure" node.

Putting everything together, to save random bits, we don't need to try to fool $\mathsf{O} \leftrightarrow \mathsf{S}_0$. Instead, it suffices for Gen to fool the certification tree T_{O}. The small branching factor of T_{O} allows Gen to have a correspondingly small seed length.

1.4 Why can't we just reuse the random bits?

Notwithstanding our lower bounds, the reader might be tempted to think that randomness stewards are trivial: why not just pick $X \in \{0,1\}^n$ uniformly at random *once* and reuse it in every round? For the purpose of discussion, let us generalize, and suppose we are trying to execute an n-coin algorithm A (not necessarily an estimation algorithm) on k inputs $C_1, \dots, C_k$. If $C_1, \dots, C_k$ are chosen *non-adaptively* (i.e. all in advance), then we really can use the same X for each execution. By the union bound, the probability that $\mathsf{A}(C_i, X)$ fails for any i is at most $k\delta$.

That argument breaks down in the adaptive case, because C_2 is chosen based on $\mathsf{A}(C_1, X)$, and hence C_2 may be *stochastically dependent* on X, so $\mathsf{A}(C_2, X)$ is not guaranteed to have a low failure probability. For example, if X is encoded in the output $\mathsf{A}(C_1, X)$, then an adversarially chosen C_2 could *guarantee* that $\mathsf{A}(C_2, X)$ fails.

Even if $C_1, \dots, C_k$ are chosen adaptively, randomness can be safely reused in an important special case: Suppose A is a **BPP** algorithm. Then we can let $\widehat{C}_1, \widehat{C}_2, \dots, \widehat{C}_k$ be the inputs that would be chosen if A never failed. Then each $\widehat{C}_i$ really is independent of X, so by the union bound, with probability $1 - k\delta$, $\mathsf{A}(\widehat{C}_i, X)$ does not fail for any i. But if $\mathsf{A}(\widehat{C}_i, X)$ does not fail for any i, then by induction, $C_i = \widehat{C}_i$ for every i. So the overall failure probability is once again at most $k\delta$.

More generally, randomness can be safely reused if A is *pseudodeterministic*, i.e. for each input, there is a unique correct output that A gives with probability $1 - \delta$.[2] (Pseudodeterministic algorithms were introduced by Gat and Goldwasser [7].) A **BPP** algorithm is trivially pseudodeterministic.

In the standard Goldreich-Levin algorithm, randomness is used to estimate $\sum_{U \in \mathcal{U}} \widehat{F}(U)^2$ for certain collections of subsets $\mathcal{U}$. The algorithm's behavior depends on how the estimate compares to $\theta^2/2$. This process is not pseudodeterministic, because if the true value $\sum_{U \in \mathcal{U}} \widehat{F}(U)^2$ is very close to $\theta^2/2$, the estimate falls on each side of $\theta^2/2$ with noticeable probability.

[2] These two conditions (inputs are chosen non-adaptively, A is pseudodeterministic) are both special cases of the following condition under which the randomness X may be safely reused: for every $1 \leq i \leq k$, C_i is a pseudodeterministic function of $(C_0, C_1, \dots, C_{i-1})$, where C_0 is a random variable that is independent of X.

1.5 Related work

1.5.1 Adaptive data analysis

The notion of a randomness steward is inspired by the closely related *adaptive data analysis* problem, introduced by Dwork et al. [6]. In the simplest version of this problem, there is an unknown distribution $\mathcal{D}$ over $\{0,1\}^n$ and a *data analyst* who wishes to estimate the mean values (with respect to $\mathcal{D}$) of k adaptively chosen functions $f_1, \dots, f_k : \{0,1\}^n \to [0,1]$ using as few samples from $\mathcal{D}$ as possible. In this setting, these samples are held by a *mechanism* and are not directly accessible by the data analyst. In round i, the data analyst gives f_i to the mechanism, and the mechanism responds with an estimate of $\mathrm{E}_{x\sim\mathcal{D}}[f_i(x)]$. The mechanism constructs the estimate so as to leak as little information as possible about the sample, so that the same sample points can be safely reused for future estimates.

The data analyst and mechanism in the adaptive data analysis setting are analogous to the owner O and steward S in our setting, respectively. In each case, the idea is that the mechanism or steward can intentionally introduce a small amount of error into each estimate to hide information and thereby facilitate future estimates. Note, however, that in the adaptive data analysis problem, there is just one unknown distribution $\mathcal{D}$ and we are concerned with sample complexity, whereas in the randomness stewardship problem, we can think of each concentrated function f_i as defining a new distribution over $\mathbb{R}^d$ and we are concerned with randomness complexity.

1.5.2 The Saks-Zhou algorithm

Another highly relevant construction is the algorithm of Saks and Zhou [21] for simulating randomized log-space algorithms in deterministic space $O(\log^{3/2} n)$. The key component in this algorithm can be reinterpreted as a one-query randomness steward. This "Saks-Zhou steward" works by *randomly perturbing and rounding* the output of each f_i, and then reusing the same random query point X in each round. The perturbation and rounding are somewhat similar to our construction, but note that we shift the outputs of each f_i deterministically, whereas the Saks-Zhou steward uses random perturbations. The analysis of the Saks-Zhou steward is substantially different than the analysis of our steward. (See the full version of this paper [10] for the description and analysis of the Saks-Zhou steward.)

Our steward achieves better parameters than the Saks-Zhou steward (see Figure 1). In particular, to achieve failure probability $k\delta + \gamma$, the error ε' of the Saks-Zhou steward is $O(\varepsilon k d/\gamma)$ – the error grows linearly with k, the number of rounds of adaptivity, as well as with $1/\gamma$. This implies, for example, that if we tried to use the Saks-Zhou steward to estimate the acceptance probabilities of k adaptively chosen Boolean circuits to within $\pm\varepsilon$ with failure probability δ in a randomness-efficient way, we would need to evaluate each circuit on $\Theta(k^2\delta^{-2}\varepsilon^{-2}\log(k/\delta))$ inputs. In contrast, because of our steward's low error, the algorithm of Corollary 4 evaluates each circuit on just $O(\varepsilon^{-2}\log(k/\delta))$ inputs – an exponential improvement in both k and $1/\delta$. Furthermore, our steward has better randomness complexity than the Saks-Zhou steward.

1.5.3 Pseudorandom generators for adaptive algorithms

Impagliazzo and Zuckerman [13, 11] were the first to consider the problem of saving random bits when executing a randomized algorithm A on many adaptively chosen inputs. Instead of assuming that A is an estimation algorithm, Impagliazzo and Zuckerman's result assumes a known bound on the Shannon entropy of the output distribution of A (e.g., the number of bits output by A). They constructed a pseudorandom generator for this setting; for $k \gg n^6$, the seed length is approximately the sum of the entropy bounds for all the executions of A.

In contrast, we make no assumptions about the entropy of $\mathsf{Est}(C)$. Since $\mathsf{Est}(C)$ is a vector of arbitrary-precision real numbers, the entropy could be as large as n, the number of random bits used by Est. And indeed, the lower bound described in Section 1.3.2.1 implies that the approach of Impagliazzo and Zuckerman fails in our setting.

One might protest that the entropy of $\mathsf{Est}(C)$ can be reduced by simple rounding. In the full version of this paper [10], we construct and analyze a steward that straightforwardly rounds each output and then uses the Impagliazzo-Zuckerman generator in a black-box way. Our main steward achieves much better randomness complexity and failure probability than this "Impagliazzo-Zuckerman steward" (see Figure 1). The improvements come from our more powerful PRG and the fact that we shift before rounding.

1.5.4 Decision trees and branching programs

In the most common decision tree model, the branching factor $|\Sigma|$ is just 2, and each node reads an arbitrary bit of the input. In the more general *parity decision tree* model, each node computes the parity of some subset of the input bits. Kushilevitz and Mansour showed [14] that the Fourier ℓ_1 norm of any Boolean function computed by a parity decision tree is at most 2^k, the number of leaves in the tree. It follows immediately that any small-bias generator [17] fools parity decision trees with asymptotically optimal seed length.

Decision trees in which each node computes a more complicated function have also been studied previously. Bellare [2] introduced the *universal decision tree* model, in which each node computes an arbitrary Boolean function of the input bits. He gave a bound on the ℓ_1 norm of any Boolean function computed by a universal decision tree in terms of the ℓ_1 norms of the functions at each node. Unfortunately, his bound does not immediately imply any nontrivial PRGs for block decision trees.

A block decision tree can be thought of as a kind of space-bounded computation. Indeed, a block decision tree is a specific kind of *ordered branching program* of width $|\Sigma|^k$ and length k that reads n bits at a time. Hence, we could directly apply a PRG for ordered branching programs, such as the INW generator [12]. For these parameters, the INW generator has seed length of $n + O(k \log k \log |\Sigma| + \log k \log(1/\gamma))$. This seed length can be slightly improved by instead using Armoni's generator [1], but even that slightly improved seed length is larger than the seed length of the generator we construct.

1.5.5 Finding noticeably large Fourier coefficients

Our randomness-efficient version of the Goldreich-Levin algorithm should be compared to the results of Bshouty et al. [3], who gave several algorithms for finding noticeably large Fourier coefficients, all closely related to one another and based on an algorithm of Levin [15].

- Bshouty et al. gave one algorithm [3, Figure 4] that makes $O(\frac{n}{\theta^2}\log(\frac{n}{\delta\theta}))$ queries and uses $O(n\log(\frac{n}{\theta})\log(\frac{1}{\delta\theta}))$ random bits. Our algorithm has better randomness complexity, but worse query complexity.
- Bshouty et al. gave another algorithm [3, Figure 5] that makes only $O(n/\theta^2)$ queries and uses just $O(\log(n/\theta)\cdot\log(1/\theta))$ random bits, but it merely outputs a list such that with probability 1/2, some U in the list satisfies $|\widehat{F}(U)| \geq \theta$, assuming such a U exists.

We also remark that there is a *deterministic* version of the Goldreich-Levin algorithm for functions with bounded ℓ_1 norm; this follows easily from the work of Kushilevitz and Mansour [14] (see also [19, Section 6.4]). In contrast, our algorithm works for all functions $F : \{0,1\}^n \to \{-1,1\}$.

```
1. For $i = 1$ to $k$:
   a. O chooses $f_i : \{0,1\}^n \to \mathbb{R}^d$ and gives it to $\mathsf{S}_0$.
   b. $\mathsf{S}_0$ picks *fresh randomness* $X_i \in \{0,1\}^n$ and queries to obtain $W_i \stackrel{\text{def}}{=} f_i(X_i)$.
   c. $\mathsf{S}_0$ computes $Y_i$ by shifting and rounding $W_i$ according to the algorithm in Section 2.1.
   d. $\mathsf{S}_0$ gives $Y_i$ to O.
```

Figure 2 Outline of $\mathsf{O} \leftrightarrow \mathsf{S}_0$.

1.6 Outline of this paper

In Section 2, we describe the shifting and rounding steward S_0 and prove that it admits certification trees with a small branching factor. In Section 3, we explain how to combine S_0 with our PRG to prove our main result (Theorem 3). In Section 4, we explain our randomness-efficient Goldreich-Levin algorithm. In Appendix A, we construct and analyze our PRG for block decision trees. In Appendix B, we give the proof of Corollary 4. In Appendix C, we explain our simulation of an oracle for **promise-BPP**, and in Appendix D, we suggest directions for further research. All other details can be found in the full version of this paper [10].

2 The shifting and rounding steward S_0

As a building block for our main steward constructions, we first construct our randomness-inefficient one-query steward S_0. Recall that any one-query steward makes two choices in each round: the input X_i to f_i and the estimate $Y_i \in \mathbb{R}^d$. The steward S_0 focuses on the second choice: each X_i is chosen uniformly at random, but S_0 carefully shifts and rounds the output $f_i(X_i)$. (See Figure 2.)

2.1 The shifting and rounding algorithm

We now describe the algorithm by which S_0 computes $Y_i \in \mathbb{R}^d$ from $W_i \stackrel{\text{def}}{=} f_i(X_i)$. Fix $n, k, d \in \mathbb{N}$ and $\varepsilon, \delta > 0$. Let $[d]$ denote the set $\{1, 2, \ldots, d\}$. Partition $\mathbb{R}$ into half-open intervals of length $(d+1) \cdot 2\varepsilon$. Let $\mathcal{I}$ denote the set of these intervals. For $w \in \mathbb{R}$, let $\mathsf{Round}(w)$ denote the midpoint of the interval in $\mathcal{I}$ containing w. Given $W_i \in \mathbb{R}^d$:

1. Find $\Delta_i \in [d+1]$ such that for every $j \in [d]$, there is some $I \in \mathcal{I}$ such that
$$[W_{ij} + (2\Delta_i - 1)\varepsilon, W_{ij} + (2\Delta_i + 1)\varepsilon] \subseteq I. \tag{6}$$
(We will show that such a Δ_i exists.)
2. For every $j \in [d]$, set $Y_{ij} = \mathsf{Round}(W_{ij} + 2\Delta_i\varepsilon)$.

We must show that this algorithm is well-defined:

▶ **Lemma 7.** *For any $W \in \mathbb{R}^d$, there exists $\Delta \in [d+1]$ such that for every $j \in [d]$, there is a single interval in $\mathcal{I}$ that entirely contains $[W_j + (2\Delta - 1)\varepsilon, W_j + (2\Delta + 1)\varepsilon]$.*

Proof. Consider picking $\Delta \in [d+1]$ uniformly at random. For each j, the probability that two distinct intervals in $\mathcal{I}$ intersect $[W_j + (2\Delta - 1)\varepsilon, W_j + (2\Delta + 1)\varepsilon]$ is precisely $1/(d+1)$ by our choice of the length of the intervals. The union bound over d different j values completes the proof. ◀

2.2 Analysis: Certification trees

As outlined in Section 1.3.2, the key lemma says that for any owner O, there exists a block decision tree T_{O} with a small branching factor that certifies correctness of $\mathsf{O} \leftrightarrow \mathsf{S}_0$:

▶ **Lemma 8.** *Assume $\delta < 1/2$. Let $\Sigma = [d+1] \cup \{\perp\}$. For any deterministic owner O, there exists a (k, n, Σ) block decision tree T_{O} with the following properties.*

1. *For any internal node v, $\Pr_{X \in \{0,1\}^n}[v(X) = \perp] \leq \delta$.*
2. *Fix $X_1, \ldots, X_k \in \{0,1\}^n$, and suppose that the path from the root to $T_{\mathsf{O}}(X_1, \ldots, X_k)$ does not include any $\perp$ nodes. Then $\max_i \|Y_i - \mu_i\|_\infty \leq O(\varepsilon d)$ in $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \ldots, X_k)$.*

Notice that Lemma 8 does not assert that T_{O} computes the transcript of $\mathsf{O} \leftrightarrow \mathsf{S}_0$. In fact, for the analysis, we will define *another* steward S_0', and T_{O} will compute a sequence of values that arise in $\mathsf{O} \leftrightarrow \mathsf{S}_0'$. This new steward S_0' will be computationally inefficient; it will *compress and decompress* the output of S_0 (with some chance of failure) before giving it to O, as we suggested in Section 1.3.2.

Proof of Lemma 8. For an (ε, δ)-concentrated function f, define $\mu(f)$ to be vector in $\mathbb{R}^d$ at which f is concentrated.[3] For a vector $Y \in \mathbb{R}^d$, say that a value $\Delta \in [d+1]$ is *f-compatible* with Y if $Y_j = \mathsf{Round}(\mu(f)_j + 2\Delta\varepsilon)$ for every $j \in [d]$. Just for the analysis, let S_0' be the following (many-query) steward:

1. For $i = 1$ to k:
   a. Give f_i to S_0, allowing it to make its one query and choose its output vector $Y_i \in \mathbb{R}^d$.
   b. Query f_i at *every* point in its domain, thereby learning the entire function.
   c. Compute

$$\widehat{\Delta}_i = \begin{cases} \text{the smallest } \Delta \in [d+1]\ f_i\text{-compatible with } Y_i & \text{if any such } \Delta \text{ exists} \\ \perp & \text{otherwise.} \end{cases} \tag{7}$$

   d. Output $\widehat{Y}_i = (\widehat{Y}_{i1}, \ldots, \widehat{Y}_{id})$, where for each $j \in [d]$,

$$\widehat{Y}_{ij} = \begin{cases} \mathsf{Round}(\mu(f)_j + 2\widehat{\Delta}_i\varepsilon) & \text{if } \widehat{\Delta}_i \neq \perp \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

We are now ready to formally define T_{O} *as a function.* Because $\mathsf{S}_0'(X_1, \ldots, X_k)$ looks at X_i only in round i, we can sensibly speak of the first i rounds of $\mathsf{O} \leftrightarrow \mathsf{S}_0'(X_1, \ldots, X_i)$ even for $i < k$. This allows us to define $T_{\mathsf{O}}(X_1, \ldots, X_i)$ to be the node v in T_{O} such that the path from the root to v is described by the values $\widehat{\Delta}_1, \ldots, \widehat{\Delta}_i$ that arise in $\mathsf{O} \leftrightarrow \mathsf{S}_0'(X_1, \ldots, X_i)$.

Now, we must show that this function T_{O} can be realized as a block decision tree, i.e. that each internal node v can be assigned a transition function $v : \{0,1\}^n \to \Sigma$ that is compatible with the definition of T_{O} as a function. Indeed, observe that $\widehat{\Delta}_1, \ldots, \widehat{\Delta}_{i-1}$ fully determine the state of O after the first $i-1$ rounds of $\mathsf{O} \leftrightarrow \mathsf{S}_0'(X_1, \ldots, X_i)$ and hence determine the function f_i. Furthermore, S_0 is "memoryless", i.e. Y_i is fully determined by f_i and X_i. Thus, $\widehat{\Delta}_i$ is fully determined by $\widehat{\Delta}_1, \ldots, \widehat{\Delta}_{i-1}$ and X_i. So there is a function $\varphi : (\widehat{\Delta}_1, \ldots, \widehat{\Delta}_{i-1}, X_i) \mapsto \widehat{\Delta}_i$, and if the path from the root to v is described by $\widehat{\Delta}_1, \ldots, \widehat{\Delta}_{i-1}$, we can set $v(X_i) \overset{\text{def}}{=} \varphi(\widehat{\Delta}_1, \ldots, \widehat{\Delta}_{i-1}, X_i)$.

[3] To handle non-uniqueness, let $\mu(f)$ be the *smallest* such vector under the lexicographic order. This exists, because $\{0,1\}^n$ is finite, so the set of points where f is concentrated is a compact subset of $\mathbb{R}^d$.

2.2.0.1 Analysis of T_{O}

By the definition of T_{O} as a function, to prove Item 1 in the lemma statement, we must show that in each round of $\mathsf{O} \leftrightarrow \mathsf{S}_0'$, $\Pr[\widehat{\Delta}_i = \perp] \leq \delta$. Indeed, by concentration, with probability $1 - \delta$, for every j, $|W_{ij} - \mu(f_i)_j| \leq \varepsilon$. In this case, by the construction of S_0, $W_{ij} + 2\Delta_i\varepsilon$ and $\mu(f_i)_j + 2\Delta_i\varepsilon$ are in the same interval in $\mathcal{I}$ for every $j \in [d]$. Therefore, in this case, there is at least one Δ value that is f_i-compatible with Y_i, namely the value Δ_i used by S_0.

Finally, to prove Item 2 in the lemma statement, suppose the path from the root node to $T_{\mathsf{O}}(X_1, \dots, X_k)$ does not include any $\perp$ nodes. Then in $\mathsf{O} \leftrightarrow \mathsf{S}_0'(X_1, \dots, X_k)$, for every i, $\widehat{\Delta}_i \neq \perp$. This implies that every Y_{ij} is of the form $\mathsf{Round}(\mu(f_i)_j + 2\widehat{\Delta}_i\varepsilon)$ for some $\widehat{\Delta}_i \in [d+1]$. Therefore, $|Y_{ij} - \mu(f_i)_j| \leq 3(d+1)\varepsilon$, since $2\widehat{\Delta}_i\varepsilon \leq 2(d+1)\varepsilon$ and rounding introduces at most $(d+1)\varepsilon$ additional error.

Of course, so far the analysis has treated S_0', not S_0. But the crucial point is, for every i, since $\widehat{\Delta}_i \neq \perp$, we can be sure that $Y_i = \widehat{Y}_i$. Therefore, the values $f_1, \dots, f_k, Y_1, \dots, Y_k$ in $\mathsf{O} \leftrightarrow \mathsf{S}_0'(X_1, \dots, X_k)$ are *exactly the same* as they are in $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \dots, X_k)$! Therefore, in $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \dots, X_k)$, for every i, $\|Y_i - \mu(f_i)\|_\infty \leq (3d+3)\varepsilon$. Finally, since $\delta < 1/2$, if μ_i is *any* point where f_i is (ε, δ)-concentrated, $\|\mu(f_i) - \mu_i\|_\infty \leq 2\varepsilon$. Therefore, for every i, $\|Y_i - \mu_i\|_\infty \leq 3(d+1)\varepsilon + 2\varepsilon = (3d+5)\varepsilon$. ◀

Notice that in $\mathsf{O} \leftrightarrow \mathsf{S}_0'(X_1, \dots, X_k)$, if $\widehat{\Delta}_i = \perp$ for some i, then the interaction might diverge from $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \dots, X_k)$, in which case $T_{\mathsf{O}}(X_1, \dots, X_k)$ does not encode the transcript of $\mathsf{O} \leftrightarrow \mathsf{S}_0(X_1, \dots, X_k)$ in any way.

3 Proof of main result (Theorem 3)

Without loss of generality, assume $\delta < 1/2$. (If $\delta \geq 1/2$, then either $k = 1$ or $k\delta \geq 1$; in either case, the result is trivial.) Let S_0 be the steward of Section 2, let Σ be the alphabet of Lemma 8, and let Gen be the γ-PRG for (k, n, Σ) block decision trees of Theorem 6. The steward is $\mathsf{S}(X) \overset{\text{def}}{=} \mathsf{S}_0(\mathsf{Gen}(X))$.

Consider any owner O. We may assume without loss of generality that O is deterministic, because a randomized owner is just a distribution over deterministic owners. By Item 1 of Lemma 8 and the union bound,

$$\Pr[\text{some node in the path from the root to } T_{\mathsf{O}}(U_{nk}) \text{ is labeled } \perp] \leq k\delta. \tag{9}$$

Therefore, when T_{O} reads $\mathsf{Gen}(U_s)$ instead of U_{nk}, the probability is at most $k\delta + \gamma$. By Item 2 of Lemma 8, this proves the correctness of S. The randomness complexity of S is just the seed length of Gen, which is indeed $n + O(k \log|\Sigma| + (\log k)\log(1/\gamma)) = n + O(k\log(d+1) + (\log k)\log(1/\gamma))$. The total runtime of S is clearly $\mathrm{poly}(n, k, d, \log(1/\varepsilon), \log(1/\gamma))$.[4] ◀

4 Application: the Goldreich-Levin algorithm

▶ **Theorem 9** (Randomness-efficient Goldreich-Levin algorithm). *There is a randomized algorithm that, given oracle access to $F : \{0,1\}^n \to \{-1,1\}$ and given input parameters $\delta, \theta > 0$, outputs a list L of subsets of $[n]$ such that with probability $1 - \delta$,*

1. *every U satisfying $|\widehat{F}(U)| \geq \theta$ is in L, and*

[4] We assume here that our computational model allows the necessary arithmetic and rounding of Section 2.1 to be performed efficiently, even if the owner chooses an f_i that outputs vectors whose coordinates are very large numbers.

2. *every $U \in L$ satisfies $|\widehat{F}(U)| \geq \theta/2$.*

The number of queries made by the algorithm is $O\left(\frac{n}{\theta^{11}\log(1/\theta)}\log\left(\frac{n}{\delta\theta}\right)\right)$, the number of random bits used by the algorithm is $O(n+(\log n)\log(1/\delta))$, and the runtime of the algorithm is $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$.

For comparison, using standard techniques, the Goldreich-Levin algorithm can be implemented in a straightforward way to use $O(\frac{n}{\theta^6}\log(\frac{n}{\delta\theta}))$ queries and $O(n^2 + n\log(\frac{n}{\delta\theta}))$ random bits. So our algorithm significantly improves the randomness complexity at the expense of substantially increasing the exponent of $1/\theta$ in the query complexity.

Toward proving Theorem 9, for a string $x \in \{0,1\}^{\leq n}$, define

$$\mathcal{U}(x) = \{U \subseteq [n] : \forall j \leq |x|, j \in U \iff x_j = 1\}. \tag{10}$$

(That is, we think of $x \in \{0,1\}^\ell$ as specifying $U \cap [\ell]$ in the natural way.) Define $W_x[F] = \sum_{U \in \mathcal{U}(x)} \widehat{F}(U)^2$. One of the key facts used in the standard Goldreich-Levin algorithm is that $W_x[F]$ can be estimated using few queries to F; here, we use the GW sampler to improve the randomness efficiency of that estimation.

▶ **Lemma 10.** *There is a randomized algorithm that, given oracle access to F and inputs $x \in \{0,1\}^{\leq n}$, $\varepsilon, \delta > 0$, estimates $W_x[F]$ to within $\pm\varepsilon$ with failure probability δ. The number of queries is $O(\log(1/\delta)/\varepsilon^2)$, the number of random bits is $O(n + \log(1/\delta))$, and the runtime is* $\mathrm{poly}(n, 1/\varepsilon, \log(1/\delta))$.

Proof. Let $\ell = |x|$. As shown in the proof of [19, Proposition 3.40],

$$W_x[F] = \mathop{\mathrm{E}}_{\substack{y,y' \in \{0,1\}^\ell \\ z \in \{0,1\}^{n-\ell}}}[F(y,z) \cdot F(y',z) \cdot \chi_x(y) \cdot \chi_x(y')], \tag{11}$$

where $\chi_x(y) \stackrel{\text{def}}{=} \prod_{j:x_j=1}(-1)^{y_j}$. Let $C : \{0,1\}^{n+\ell} \to \{0,1\}$ be the function

$$C(y,y',z) = \frac{1}{2} + \frac{1}{2} \cdot F(y,z) \cdot F(y',z) \cdot \chi_x(y) \cdot \chi_x(y'), \tag{12}$$

so that $W_x[F] = 2\,\mathrm{E}_{y,y',z}[C(y,y',z)] - 1$. We can estimate the expectation of C to within $\pm\varepsilon/2$ with failure probability δ using the GW sampler [9, Theorem 6.5], which implies an estimate of $W_x[F]$ to within $\pm\varepsilon$. The number of queries made by the GW sampler is $O(\log(1/\delta)/\varepsilon^2)$, and each query to C can be evaluated by making 2 queries to F. The randomness complexity of the GW sampler is $n + \ell + O(\log(1/\delta))$, which is $O(n + \log(1/\delta))$. ◀

The standard Goldreich-Levin algorithm proceeds by finding, for $\ell = 1$ to n, the set of all x with $|x| = \ell$ such that $W_x[F] \gtrsim \theta^2$. In each round, the algorithm estimates $W_x[F]$ for all strings x formed by appending a single bit to a string x' that was previously found to satisfy $W_{x'}[F] \gtrsim \theta^2$. This adaptive structure is exactly suited for saving random bits using a steward. To further drive down the randomness complexity, we reduce the number of rounds of adaptivity by appending $\log(1/\theta)$ bits at a time instead of 1 bit.

Proof of Theorem 9. Algorithm:

1. Let $u = \lfloor \log(1/\theta) \rfloor$, let $k = \lceil n/u \rceil$, and let $d = \lfloor 2^u \cdot 4/\theta^2 \rfloor$.
2. Let S be a $(\theta^2/4, \delta)$-steward for k adaptively chosen $(\varepsilon, \delta/(2n))$-concentrated functions $f_1, \ldots, f_k : \{0,1\}^m \to \mathbb{R}^d$, where $\varepsilon \geq \Omega(\theta^2/d)$ and m will become clear later.
3. Set $L_0 :=$ {empty string}.
4. For $i = 1$ to k:

   a. If $|L_{i-1}| > d/2^u$, abort and output “fail”.
   b. Observe that every string in L_{i-1} has length $\ell = u(i-1) < n$. Let $x_1, \dots, x_t$ be the set of all strings obtained from strings in L_{i-1} by appending $\min\{u, n-\ell\}$ bits, so $t \leq 2^u |L_{i-1}| \leq d$.
   c. Define $f_i : \{0,1\}^m \to \mathbb{R}^t$ by letting $f_i(X)_j$ be the estimate of $W_{x_j}[F]$ to within $\pm\varepsilon$ provided by the algorithm of Lemma 10 with failure probability $\delta/(2dn)$ using randomness X. Observe that by the union bound, f_i is $(\varepsilon, \delta/(2n))$-concentrated at $(W_{x_1}[F], \dots, W_{x_t}[F])$.
   d. By giving f_i to S, obtain estimates $\mu_1, \dots, \mu_t$ for $W_{x_1}[F], \dots, W_{x_t}[F]$.
   e. Set $L_i := \{x_j : \mu_j \geq \theta^2/2\}$.

5. Output $L \overset{\text{def}}{=} \bigcup_{x \in L_k} \mathcal{U}(x)$.

(So m is the number of random bits used by the algorithm of Lemma 10.) With probability $1-\delta$, all of the responses of S are accurate, i.e. every μ_j value is within $\pm\theta^2/4$ of the corresponding $W_{x_j}[F]$ value. Assume from now on that this has happened.

By the definition of L_i, every x in every L_i satisfies $W_x[F] \geq \theta^2/4$. By Parseval's theorem (see, e.g., [19, Section 1.4]), this implies that $|L_i| \leq 4/\theta^2 \leq d/2^u$ for every i. Therefore, the algorithm does not abort. Let ℓ_i be the length of all the strings in L_i, so $\ell_i = ui$ for $i < k$ and $\ell_k = n$. Suppose $\widehat{F}(U)^2 \geq \theta^2$. By induction on i, the unique string $x \in \{0,1\}^{\ell_i}$ with $U \in \mathcal{U}(x)$ is placed in L_i, because the estimate of $W_x[F]$ is at least $3\theta^2/4 > \theta^2/2$. This shows that $U \in L$. Conversely, if U ends up in L, then the estimate of $\widehat{F}(U)^2$ in iteration $i = n$ was at least $\theta^2/2$, so $\widehat{F}(U)^2 \geq \theta^2/4$. This completes the proof of correctness of the algorithm.

Now, observe that the total number of queries to F is at most kd times the $O(\log(nd/\delta)/\varepsilon^2)$ queries that the algorithm of Lemma 10 makes, i.e. the total number of queries to F is

$$O\left(\frac{kd^3 \log(nd/\delta)}{\theta^2}\right) = O\left(\frac{n}{\theta^{11}\log(1/\theta)} \log\left(\frac{n}{\delta\theta}\right)\right).$$

The randomness complexity of the algorithm is just the randomness complexity of S. We will use the steward of Theorem 3 with $\gamma = \delta/2$, so the randomness complexity is $m + O(k\log(d+1) + (\log k)\log(1/\delta))$. Since $m \leq O(n + \log(n/(\delta\theta)))$, the total randomness complexity is

$$O\left(n + \frac{n}{\log(1/\theta)}\log(1/\theta) + (\log n)\log(1/\delta) + \log(1/\theta)\right) = O(n + (\log n)\log(1/\delta) + \log(1/\theta)).$$

To get rid of the $\log(1/\theta)$ term as claimed in the theorem statement, just notice that we can assume without loss of generality that $\theta \geq 2^{-n+1}$, because any nonzero Fourier coefficient of a $\{-1,1\}$-valued function has absolute value at least 2^{-n+1}. The total runtime of the algorithm is clearly $\mathrm{poly}(n, 1/\theta, \log(1/\delta))$. ◀

References

1 R. Armoni. On the derandomization of space-bounded computations. In *Randomization and Approximation Techniques in Computer Science*, pages 47–59. Springer, 1998.

2 M. Bellare. A technique for upper bounding the spectral norm with applications to learning. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 62–70. ACM, 1992.

3 N. H. Bshouty, J. C. Jackson, and C. Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution. *Journal of Computer and System Sciences*, 68(1):205–234, 2004.

4 H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 100–109. Springer, 1999.

5 Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

6 C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. L. Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the 47th Annual Symposium on Theory of Computing*, STOC '15, pages 117–126. ACM, 2015.

7 E. Gat and S. Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 136, 2011.

8 O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, STOC '89, pages 25–32. ACM, 1989.

9 O. Goldreich and A. Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997. doi:10.1002/(SICI)1098-2418(199712)11:4<315::AID-RSA3>3.0.CO;2-1.

10 W. M. Hoza and A. R. Klivans. Preserving randomness for adaptive algorithms. *arXiv preprint arXiv:1611.00783*, 2016.

11 R. Impagliazzo. *Pseudo-random generators for cryptography and for randomized algorithms*. PhD thesis, University of California, Berkeley, 1992. URL: http://cseweb.ucsd.edu/users/russell/format.ps.

12 R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual Symposium on Theory of Computing*, STOC '94, pages 356–364. ACM, 1994.

13 R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, FOCS '89, pages 248–253. IEEE, 1989.

14 E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

15 L. A. Levin. Randomness and nondeterminism. *Journal of Symbolic Logic*, 58(3):1102–1103, 1993.

16 P. Moser. Relative to *p promise-bpp* equals *app*. In *Electronic Colloquium on Computational Complexity (ECCC) Report TR01*, volume 68, 2001.

17 J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM journal on computing*, 22(4):838–856, 1993.

18 N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

19 R. O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

20 R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, STOC '99, pages 159–168. ACM, 1999. doi:10.1145/301250.301294.

21 M. Saks and S. Zhou. $BP_H SPACE(S) \subseteq DSPACE(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

22 S. P. Vadhan. *Pseudorandomness*, volume 56. Now, 2012.

A Pseudorandomness for block decision trees

In this section, we prove Theorem 6. Recall that our goal is to modify the internal parameters of the INW generator, thereby constructing a γ-PRG for (k, n, Σ) block decision trees with seed length $n + O(k \log |\Sigma| + (\log k) \log(1/\gamma)$. The construction and analysis mimic the standard treatment of the INW generator, and the reader who is familiar with the INW generator is encouraged to skip to Section A.4 to just see the new parameters. In words, the only new feature is that we use extractors for a geometrically growing entropy deficit at each level of the recursion to match the geometrically growing width of the block decision tree.

A.1 Formal definition of a PRG

Let U_n denote the uniform distribution on $\{0,1\}^n$. For two probability distributions μ, μ' on the same measurable space, write $\mu \sim_\gamma \mu'$ to indicate that μ and μ' have total variation distance at most γ.

▶ **Definition 11.** We say that $\mathsf{Gen} : \{0,1\}^s \to \{0,1\}^{nk}$ is a *γ-PRG for (k, n, Σ) block decision trees* if for every such tree T, $T(\mathsf{Gen}(U_s)) \sim_\gamma T(U_{nk})$.

A.2 Concatenating PRGs for block decision trees

Toward proving Theorem 6, for a (k, n, Σ) block decision tree $T = (V, E)$ and a node $v \in V$, let T_v denote the subtree rooted at v, and observe that we can think of T_v as a (k', n, Σ) block decision tree, where $k' = k - \mathrm{depth}(v)$. This simple observation – after a block decision tree has been computing for a while, the remaining computation is just another block decision tree – implies that pseudorandom generators for block decision trees can be *concatenated* with mild error accumulation. This fact and its easy proof are perfectly analogous to the situation with ordered branching programs. We record the details below.

▶ **Lemma 12.** *Suppose* $\mathsf{Gen}_1 : \{0,1\}^{s_1} \to \{0,1\}^{nk_1}$ *is a γ_1-PRG for (k_1, n, Σ) block decision trees and* $\mathsf{Gen}_2 : \{0,1\}^{s_2} \to \{0,1\}^{nk_2}$ *is a γ_2-PRG for (k_2, n, Σ) block decision trees. Let* $\mathsf{Gen}(x, y) = (\mathsf{Gen}_1(x), \mathsf{Gen}_2(y))$. *Then* Gen *is a $(\gamma_1 + \gamma_2)$-PRG for $(k_1 + k_2, n, \Sigma)$ block decision trees.*

Proof. Fix a $(k_1 + k_2, n, \Sigma)$ block decision tree T. For a node u at depth k_1 and a leaf node v, define

$$\begin{aligned} p(u) &= \Pr[T(U_{nk_1}) = u] & p(v \mid u) &= \Pr[T_u(U_{nk_2}) = v] \\ \widetilde{p}(u) &= \Pr[T(\mathsf{Gen}_1(U_{s_1})) = u] & \widetilde{p}(v \mid u) &= \Pr[T_u(\mathsf{Gen}_2(U_{s_2})) = v]. \end{aligned}$$

To prove correctness of Gen, recall that ℓ_1 distance is twice total variation distance. The ℓ_1 distance between $T(\mathsf{Gen}(U_{s_1+s_2}))$ and $T(U_{n(k_1+k_2)})$ is precisely $\sum_{u,v} |p(u)p(v \mid u) - \widetilde{p}(u)\widetilde{p}(v \mid u)|$. By the triangle inequality, this is bounded by

$$\begin{aligned} &\sum_{u,v} |p(u)p(v \mid u) - p(u)\widetilde{p}(v \mid u)| + \sum_{u,v} |p(u)\widetilde{p}(v \mid u) - \widetilde{p}(u)\widetilde{p}(v \mid u)| \\ &= \sum_{u,v} p(u) \cdot |p(v \mid u) - \widetilde{p}(v \mid u)| + \sum_{u,v} |p(u) - \widetilde{p}(u)| \cdot \widetilde{p}(v \mid u) \\ &= \sum_u p(u) \sum_v |p(v \mid u) - \widetilde{p}(v \mid u)| + \sum_u |p(u) - \widetilde{p}(u)|. \end{aligned}$$

By the correctness of Gen_1 and Gen_2, this is bounded by $(\sum_u p(u) \cdot 2\gamma_2) + 2\gamma_1 = 2(\gamma_1 + \gamma_2)$. ◀

A.3 Recycling randomness

We find it most enlightening to think of the INW generator in terms of extractors, as suggested by Raz and Reingold [20] and in the spirit of the Nisan-Zuckerman generator [18]. The analysis is particularly clean if we work with *average-case extractors*, a concept introduced by Dodis et al. [5].

▶ **Definition 13.** For discrete random variables X, V, the *average-case conditional min-entropy* of X given V is

$$\widetilde{H}_\infty(X \mid V) = -\log_2\left(\mathop{\mathrm{E}}_{v\sim V}\left[2^{-H_\infty(X|V=v)}\right]\right), \tag{13}$$

where H_∞ is (standard) min-entropy.

Intuitively, $\widetilde{H}_\infty(X \mid V)$ measures the amount of randomness in X from the perspective of someone who knows V. The output of an *average-case extractor* is required to look uniform even from the perspective of someone who knows V, as long as its first input is sampled from a distribution that has high min-entropy conditioned on V:

▶ **Definition 14.** We say that $\mathsf{Ext} : \{0,1\}^s \times \{0,1\}^d \to \{0,1\}^m$ is an *average-case* $(s-t, \beta)$-*extractor* if for every X distributed on $\{0,1\}^s$ and every discrete random variable V such that $\widetilde{H}_\infty(X \mid V) \geq s - t$, if we let $Y \sim U_d$ be independent of (X, V) and let $Z \sim U_m$ be independent of V, then $(V, \mathsf{Ext}(X, Y)) \sim_\beta (V, Z)$.

Average-case extractors are the perfect tools for *recycling randomness* in space-bounded computation. We record the details for block decision trees below.

▶ **Lemma 15** (Randomness recycling lemma for block decision trees)**.** *Suppose* $\mathsf{Gen} : \{0,1\}^s \to \{0,1\}^{nk}$ *is a* γ*-PRG for* (k, n, Σ) *block decision trees and* $\mathsf{Ext} : \{0,1\}^s \times \{0,1\}^d \to \{0,1\}^s$ *is an average-case* $(s - k\log|\Sigma|, \beta)$*-extractor. Define*

$$\mathsf{Gen}'(x, y) = (\mathsf{Gen}(x), \mathsf{Gen}(\mathsf{Ext}(x, y))). \tag{14}$$

Then Gen' *is a* $(2\gamma + \beta)$*-PRG for* $(2k, n, \Sigma)$ *block decision trees.*

Proof. Let T be a $(2k, n, \Sigma)$ block decision tree. Let $X \sim U_s$ and let $V = T(\mathsf{Gen}(X))$. By [5, Lemma 2.2b], the fact that V can be described using $k\log|\Sigma|$ bits implies that $\widetilde{H}_\infty(X \mid V) \geq s - k\log|\Sigma|$. Therefore, by the average-case extractor condition, if we let $Y \sim U_d$ be independent of X and $Z \sim U_d$ be independent of V, then

$$(V, \mathsf{Ext}(X, Y)) \sim_\beta (V, Z). \tag{15}$$

Applying a (deterministic) function can only make the distributions closer. Apply the function $(v, z) \mapsto T_v(\mathsf{Gen}(z))$:

$$T(\mathsf{Gen}'(X, Y)) \sim_\beta T(\mathsf{Gen}(X), \mathsf{Gen}(Z)). \tag{16}$$

By Lemma 12, the right-hand side is (2γ)-close to $T(U_{2nk})$. The triangle inequality completes the proof. ◀

To actually construct a generator, we will need to instantiate this randomness recycling lemma with an explicit average-case extractor:

▶ **Lemma 16.** *For every* $s, t \in \mathbb{N}$ *and every* $\beta > 0$*, there exists an average-case* $(s-t, \beta)$*-extractor* $\mathsf{Ext} : \{0,1\}^s \times \{0,1\}^d \to \{0,1\}^s$ *with seed length* $d \leq O(t + \log(1/\beta))$ *computable in time* $\mathrm{poly}(s, \log(1/\beta))$.

Proof sketch. It is standard (and can be proven using expanders, see, e.g., [22]) that there exists an *ordinary* $(s - t - \log(2/\beta), \beta/2)$-extractor $\mathsf{Ext} : \{0,1\}^s \times \{0,1\}^d \to \{0,1\}^s$ with seed length $d \leq O(t + \log(1/\beta))$ computable in time $\mathrm{poly}(s, \log(1/\beta))$. By the same argument as that used to prove [5, Lemma 2.3], Ext is automatically an average-case $(s-t, \beta)$-extractor. ◀

A.4 The recursive construction

Proof of Theorem 6. Define $\beta = \gamma/2^{\lceil \log k \rceil}$. For $i \geq 0$, define $s_i \in \mathbb{N}$, $d_i \in \mathbb{N}$, $G_i : \{0,1\}^{s_i} \to \{0,1\}^{n \cdot 2^i}$, and $\mathsf{Ext}_i : \{0,1\}^{s_i} \times \{0,1\}^{d_i} \to \{0,1\}^{s_i}$ through mutual recursion as follows. Start with $s_0 = n$ and $G_0(x) = x$. Having already defined s_i and G_i, let Ext_i be the average-case $(s_i - 2^i \log|\Sigma|, \beta)$-extractor of Lemma 16, and let d_i be its seed length. Then let $s_{i+1} = s_i + d_i$, and let

$$G_{i+1}(x, y) = (G_i(x), G_i(\mathsf{Ext}_i(x, y))). \tag{17}$$

We show by induction on i that G_i is a $(\beta \cdot (2^i - 1))$-PRG for $(2^i, n, \Sigma)$ block decision trees. In the base case $i = 0$, this is trivial. For the inductive step, apply Lemma 15, and note that $2\beta(2^i - 1) + \beta = \beta(2^{i+1} - 1)$. This completes the induction. Therefore, we can let $\mathsf{Gen} = G_{\lceil \log k \rceil}$, since $\beta \cdot (2^{\lceil \log k \rceil} - 1) < \gamma$. The seed length $s_{\lceil \log k \rceil}$ of Gen is

$$\begin{aligned} n + \sum_{i=0}^{\lceil \log k \rceil} d_i &\leq n + O\left(\sum_{i=0}^{\lceil \log k \rceil} (2^i \log|\Sigma| + \log k + \log(1/\gamma))\right) \\ &\leq n + O(k \log|\Sigma| + (\log k) \log(1/\gamma)). \end{aligned}$$

The time needed to compute $\mathsf{Gen}(x)$ is just the time needed for $O(k)$ applications of Ext_i for various $i \leq O(\log k)$, which is $\mathrm{poly}(n, k, \log|\Sigma|, \log(1/\gamma))$. ◀

B Acceptance probabilities of Boolean circuits

In this section, we prove Corollary 4. To begin, we recall the definition of a *sampler*, which we used already in Section 4. A (ε, δ)-*sampler* for Boolean functions on n bits is a randomized oracle algorithm Samp such that for any Boolean function $C : \{0,1\}^n \to \{0,1\}$, if we let $\mu(C) \overset{\text{def}}{=} 2^{-n} \sum_x C(x)$, then

$$\Pr[|\mathsf{Samp}^C - \mu(C)| > \varepsilon] \leq \delta. \tag{18}$$

Proof of Corollary 4. Let c be the constant under the $O(\cdot)$ of the error ε' in the steward of Theorem 3. When given parameters $n, k, \varepsilon, \delta$, let Samp be the Boolean $(\varepsilon/c, \delta/(2k))$-sampler by Goldreich and Wigderson [9], and say it uses m coins. Let S be the (ε, δ)-steward of Theorem 3 for k adaptively chosen $(\varepsilon/c, \delta/(2k))$-concentrated functions $f_1, \dots, f_k : \{0,1\}^m \to \mathbb{R}$. (So $\gamma = \delta/2$.) When given circuit C_i, define $f_i(X) = \mathsf{Samp}^{C_i}(X)$, i.e. the output of Samp^{C_i} with randomness X. Give f_i to S, and output the value Y_i that it returns.

Proof of correctness: The definition of a sampler implies that each f_i is $(\varepsilon/c, \delta/(2k))$-concentrated at $\mu(C_i)$. Furthermore, each f_i is defined purely in terms of C_i, which is chosen based only on $Y_1, \dots, Y_{i-1}$. Therefore, the steward guarantee implies that with probability $1 - \delta$, every Y_i is within $\pm\varepsilon$ of $\mu(C_i)$.

Randomness complexity analysis: The number of bits m used by the sampler is $n + O(\log(k/\delta))$. Therefore, the number of bits used by the steward is

$$n + O(\log(k/\delta)) + O(k + (\log k)\log(1/\delta)) = n + O(k + (\log k)\log(1/\delta)). \tag{19}$$

Runtime analysis: The runtime of the steward is

$$\mathrm{poly}(m, k, \log(1/\gamma)) = \mathrm{poly}(n, k, \log(1/\delta)). \tag{20}$$

The runtime of the sampler is $\mathrm{poly}(n, 1/\varepsilon, \log k, \log(1/\delta))$. The time required to evaluate each query of the sampler in round i is $O(\mathrm{size}(C_i))$ (assuming we work with a suitable computational model and a suitable encoding of Boolean circuits.) The number of queries that the sampler makes in each round is $O(\log(k/\delta)/\varepsilon^2)$. Therefore, the total runtime of this algorithm is

$$O\left(\frac{\log k + \log(1/\delta)}{\varepsilon^2} \cdot \sum_{i=1}^{k} \mathrm{size}(C_i)\right) + \mathrm{poly}(n, k, 1/\varepsilon, \log(1/\delta)). \tag{21}$$

C Simulating an oracle for promise-BPP

Recall that **promise-BPP** is the class of promise problems that can be decided in probabilistic polynomial time with bounded failure probability. When an algorithm is given oracle access to a promise problem, it is allowed to make queries that violate the promise, and several models have been considered for dealing with such queries. Following Moser [16], we will stipulate that the oracle may respond in any arbitrary way to such queries. (See, e.g., [4] for two other models.) From these definitions, it is easy to show, for example, that $\mathbf{BPP}^{\mathbf{promise\text{-}BPP}} = \mathbf{BPP}$. In this section, using our steward, we give a time- and randomness-efficient simulation of any algorithm with an oracle for **promise-BPP**.

▶ **Theorem 17.** *Suppose a search problem Π can be solved by a deterministic* **promise-BPP**-*oracle algorithm that runs in time T and makes k queries, and suppose that (regardless of previous oracle responses) each query of this algorithm can be decided by a randomized algorithm that runs in time T', uses n coins, and has failure probability $1/3$. Then for any δ, Π can be solved by a randomized (non-oracle) algorithm that runs in time*

$$T + O(T' \cdot k \log(k/\delta)) + \mathrm{poly}(n, k, \log(1/\delta)),$$

has randomness complexity

$$n + O(k + (\log k)\log(1/\delta)),$$

and has failure probability δ.

(Recall that search problems generalize decision problems and function problems. In reality, the theorem generalizes to just about any kind of "problem", but we restrict ourselves to search problems for concreteness.) The theorem can easily be extended to randomized oracle algorithms by considering the problem of executing the randomized oracle algorithm using a given randomness string.

Note that Theorem 17 would be trivial if it involved a **BPP** oracle instead of a **promise-BPP** oracle. Indeed, in the **BPP** case, the randomness can be reduced to just $n + O(\log k + \log(1/\delta))$. This is because a **BPP** algorithm is pseudodeterministic, so the randomness can be safely reused from one query to the next as discussed in Section 1.4. A **promise-BPP** algorithm is not pseudodeterministic in general – it is only guaranteed to be pseudodeterministic on inputs that satisfy the promise.

Proof sketch of Theorem 17. Let B be the algorithm of Corollary 4 with $\varepsilon = 1/10$ and the desired failure probability δ. When the oracle algorithm makes query i, define $f_i(X)$ to be the value outputted by the **promise-BPP** algorithm on that query string using randomness X. Give B the "circuit" f_i. (The algorithm B treats the circuits as black boxes, so we don't need to bother implementing f_i as a literal Boolean circuit; the important thing is that $f_i(X)$ can be evaluated in time T'.) When B outputs a value Y_i, give the oracle algorithm the response 0 if $Y_i < 1/2$ and 1 if $Y_i \geq 1/2$. ◀

D Directions for further research

The problem of randomness stewardship is fundamental, and the main open problem left by this work is to construct optimal stewards. The following are examples of concrete questions along these lines.

- Does every one-query steward with failure probability $\delta' \leq O(k\delta)$ have randomness complexity $n + \Omega(k \log(d+1))$? (Is the randomness complexity of our main steward near-optimal?)
- Does there exist a one-query $(O(\varepsilon), k\delta + 0.1)$-steward with randomness complexity $n + O(k \log(d+1))$? (Can the error of our main steward be improved?)

We explained in this work how the steward model captures some older derandomization constructions, and we gave new applications of stewards. We hope that future researchers find more connections and applications.