2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-5519-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS54457.2022.00048

Unstructured Hardness to Average-Case Randomness

Lijie Chen
Massachusetts Institute of Technology
Cambridge, MA
wjmzbmr@gmail.com

Ron D. Rothblum
Technion
Haifa, Israel
rothblum@cs.technion.ac.il

Roei Tell
The Institute of Advanced Study and the DIMACS Center at Rutgers
Princeton, NJ
roeitell@gmail.com

***Abstract*—The leading technical approach in uniform hardness-to-randomness in the last two decades faced several well-known barriers that caused results to rely on overly strong hardness assumptions, and yet still yield suboptimal conclusions.**

In this work we show uniform hardness-to-randomness results that *simultaneously break through all of the known barriers*. Specifically, consider any one of the following three assumptions:

1) **For some $\epsilon > 0$ there exists a function f computable by uniform circuits of size $2^{O(n)}$ and depth $2^{o(n)}$ such that f is hard for probabilistic time $2^{\epsilon \cdot n}$.**
2) **For every $c \in \mathbb{N}$ there exists a function f computable by logspace-uniform circuits of polynomial size and depth n^2 such that every probabilistic algorithm running in time n^c fails to compute f on a $(1/n)$-fraction of the inputs.**
3) **For every $c \in \mathbb{N}$ there exists a logspace-uniform family of arithmetic formulas of degree n^2 over a field of size $\mathrm{poly}(n)$ such that no algorithm running in probabilistic time n^c can evaluate the family on a worst-case input.**

Assuming any of these hypotheses, where the hardness is for every sufficiently large input length $n \in \mathbb{N}$, we deduce that $\mathcal{RP}$ can be derandomized in *polynomial time* and on *all input lengths*, on average. Furthermore, under the first assumption we also show that $\mathcal{BPP}$ can be derandomized in polynomial time, on average and on all input lengths, with logarithmically many advice bits.

On the way to these results we also resolve two related open problems. First, we obtain an *optimal worst-case to average-case reduction* for computing problems in linear space by uniform probabilistic algorithms; this result builds on a new instance checker based on the doubly efficient proof system of Goldwasser, Kalai, and Rothblum (J. ACM, 2015). Secondly, we resolve the main open problem in the work of Carmosino, Impagliazzo and Sabin (ICALP 2018), by deducing derandomization from weak and general fine-grained hardness hypotheses.

The full version of this paper is available online [5].

I. Introduction

A classical line of work in complexity theory is focused on *uniform hardness vs randomness* results. These are results that connect lower bounds for uniform probabilistic algorithms to average-case derandomization. For example, as proved in the classical result of Impagliazzo and Wigderson [13], if $\mathcal{BPP} \neq \mathcal{EXP}$, then $\mathcal{BPP}$ can be simulated in sub-exponential time, on average and infinitely often.[1]

In contrast to works concerning *non-uniform* hardness vs randomness (cf., e.g., [22, 16, 30, 28, 32]), the currently known results for uniform hardness vs randomness seem suboptimal. For comparison, recall that in the non-uniform setting we know that $\mathcal{E} = \mathcal{DTIME}[2^{O(n)}]$ is hard for circuits of size $s(n)$ if and only if there exists a pseudorandom generator (PRG) for linear-sized circuits with seed length linear in $s^{-1}(\mathrm{poly}(n))$ (see [32]); in particular, for the "high-end" regime, $\mathcal{E}$ is hard for circuits of size $2^{\epsilon \cdot n}$ if and only if there exists such a PRG with seed length $O(\log n)$.[2] However, for *uniform* hardness vs randomness, the results that we know do *not* scale to the "high-end" regime. In fact, even if we assume hardness for *specific* functions in $\mathcal{SPACE}[O(n)]$ that are conductive for these results, rather than hardness for arbitrary functions in $\mathcal{E} = \mathcal{DTIME}[2^{O(n)}]$, we still only know how to deduce average-case derandomization in super-polynomial time $n^{\mathrm{polyloglog}(n)}$ (see [6]).

As one might expect, this classical challenge attracted considerable interest over the years. The main focus was improving the parameters of the hardness vs randomness tradeoff, trying to deduce faster average-case derandomization from as weak a hardness hypothesis as possible (see, e.g., [3, 17, 31, 12, 6]). Parallel lines of work studied extensions of this paradigm to derandomization of proof systems, in which case we can obtain worst-case derandomization under uniform hardness assumptions for the corresponding class of protocols (see, e.g., [20, 11, 27]); and to derandomization that relies on fine-grained hardness hypotheses for specific functions in $\mathcal{P}$, in which case we can circumvent some of the barriers above, and obtain average-case derandomization in polynomial time and on all input lengths (see [4]). The known results have been widely applied throughout complexity theory (for some applications see, e.g., [1, 24, 19, 23, 15]).

a) The main technical challenges and obstacles: Let us explain the main challenge that has been obstructing progress so far. All the results described above (with the exception of [4]) rely on reconstructive PRGs. Loosely speaking, these are generators that transform a "hard" truth-table f into a set of pseudorandom strings, and the proof of correctness relies on a reduction: A distinguisher for a pseudorandom set is converted

[1]The precise meaning of "on average" in this result is that for every $L \in \mathcal{BPP}$ and $\epsilon > 0$ there exists $L' \in \mathcal{DTIME}[2^{n^\epsilon}]$ such that for every polynomial-time samplable collection of distributions $\mathbf{x} = \{\mathbf{x}_n\}_{n \in \mathbb{N}}$ and for infinitely many n's it holds that $\Pr_{x \sim \mathbf{x}_n}[L'(x) = L(x)] \geq 1 - 1/n$.

[2]Throughout the exposition, we always assume that the running time of a PRG is exponential in its seed length. (This is because in the derandomization application we enumerate over the seeds of a PRG.)

978-1-6654-5519-0/22/$31.00 ©2022 IEEE
DOI 10.1109/FOCS54457.2022.00048

into an efficient procedure for the hard function. Indeed, there are two parts in such a construction, the generator and the reconstruction procedure.

When starting from a strong hardness assumption, such as hardness for non-uniform circuits, the reconstruction procedure may use "strong" resources, such as non-uniformity. In contrast, when only assuming hardness for uniform probabilistic algorithms, the reconstruction procedure must also be a uniform algorithm. Alas, we currently *do not know* how to construct efficient generators with uniform and efficient reconstruction procedures when the "hard" truth-table f is an arbitrary function in $\mathcal{E}$. This is because known ideas for reconstruction procedures rely on *specific structural properties* of f, namely that it is downward self-reducible and randomly self-reducible;[3] such structural properties exist only for functions in $\mathcal{PSPACE}$. (For details see the classical work [13], and for further "barrier" results see [12].)[4]

The situation gets even worse: Since we need f to admit the specific structural properties mentioned above, we cannot obtain derandomization from hardness of an *arbitrary* function in the relevant class. This leaves us with two choices – either assume hardness for specific functions, which seems an overly narrow assumption; or reduce arbitrary functions in the class to complete functions that admit the structural properties, which typically yields super-polynomial derandomization overheads. Moreover, the known $\mathcal{PSPACE}$-complete functions that admit the required structural properties can be computed in time $2^{o(n)}$, and thus are not sufficiently hard to yield derandomization in polynomial time. And to top this off, known results yield derandomization that succeeds only on infinitely many input lengths (two exceptions are [4, 6]).

b) Our contributions, in a gist: In this work we show how to *simultaneously bypass all of the obstacles mentioned above*. Specifically, we will show uniform hardness vs randomness results that:

1) Rely on hardness for functions that **do not admit the structural properties** that were required for previous results. In particular, our results start from hardness for functions that are not necessarily computable in $\mathcal{PSPACE}$.
2) Do not need to assume hardness for a specific function: It suffices to assume **hardness for any function in the relevant class** (without causing overheads in the running time of the derandomization algorithm).
3) Can yield derandomization that works in **polynomial time**, assuming that a function in the relevant class is sufficiently hard.
4) Yield derandomization algorithms that work **on all input lengths**, rather than only on infinitely many inputs lengths.

The downside of our results is that we will either derandomize $\mathcal{RP}$ (i.e., probabilistic algorithms with one-sided error), or derandomize $\mathcal{BPP}$ using a small number of advice bits (e.g., logarithmically many or less).

The main idea allowing us to break through the former obstacles is to *rely on machinery constructed for non-black-box derandomization* in the very recent work of two of the authors [7]. This machinery was previously used in a different context: In the previous work the hypothesized hardness was very strong, namely hardness on *almost all inputs*,[5] and the conclusion was a worst-case derandomization of $\mathcal{BPP}$. In contrast, in the current work we adapt this machinery to work with the weaker *standard notions of worst-case hardness*, and conclude *average-case derandomization* of $\mathcal{RP}$ and $\mathcal{BPP}$.

A. High-end results: Breaking the $\mathcal{PSPACE}$ barrier

Our first main result is the following uniform hardness to randomness tradeoff. Consider the class of logspace-uniform circuits[6] of exponential size $2^{O(n)}$ and near-exponential depth $2^{o(n)}$. Observe that this class contains $\mathcal{SPACE}[O(n)]$ and in fact seems much broader than it: Indeed, $\mathcal{SPACE}[O(n)]$ can be simulated even by logspace-uniform circuits of size $2^{O(n)}$ and smaller depth $\mathrm{poly}(n)$ rather than $2^{o(n)}$ (by the standard approach of repeated squaring). We prove that if the foregoing class contains a function hard for probabilistic algorithms with running time $2^{\epsilon \cdot n}$, then $\mathcal{RP}$ and $\mathcal{BPP}$ can be derandomized in polynomial time on average, as follows:

Theorem 1.1 (high-end hardness vs randomness beyond $\mathcal{PSPACE}$): For every $\epsilon > 0$ there exists $\delta > 0$ such that the following holds. Assume that there is a function $L \subseteq \{0,1\}^*$ computable by logspace-uniform circuits of size $2^{O(n)}$ and depth $2^{\delta \cdot n}$ such that $L \notin \mathtt{i.o.}\mathcal{BPTIME}[2^{\epsilon \cdot n}]$. Then, for every $a \in \mathbb{N}$ it holds that

$$\mathcal{RP} \subseteq \mathsf{heur}_{1-1/n^a}\text{-}\mathcal{P} \,,$$
$$\mathcal{BPP} \subseteq \mathsf{heur}_{1-1/n^a}\text{-}\mathcal{P}/O(\log n) \,.$$

The meaning of "heur_{1-1/n^a}" above is that for every $L \in \mathcal{RP}$ and *every polynomial-time samplable distribution* $\mathbf{x}$ there exists $L' \in \mathcal{P}$ such that $\Pr_{x \sim \mathbf{x}}[L(x) = L'(x)] \geq 1 - 1/|x|^a$, and ditto for $\mathcal{BPP}$ and $\mathcal{P}/O(\log(n))$ (the definition apperas in the full version of this paper, see [5, Definition 3.4]). When the depth of the circuits for the hard function is smaller, say $\mathrm{poly}(n)$, the advice for derandomizing $\mathcal{BPP}$ is shorter, say $O(\mathrm{loglog}(n))$ (see [5, Theorem 5.2] for precise details).

We stress that there are several novel features in Theorem 1.1. First, it relies on hardness for functions that

[3]Recall that a function f is downward self-reducible if we can compute f quickly (say, in small polynomial time) at any given n-bit input when given oracle access to f at inputs of length $n-1$. A function f is randomly self-reducible if we can quickly evaluate f at any given n-bit input, with high probability, given access to evaluations of f at random n-bit inputs.

[4]These obstacles were bypassed in the original work of [13] by a specific argument that introduced *significant time overheads*. Specifically, to obtain a PRG with seed length $s^{-1}(n)$ their result needs hardness for probabilistic algorithms running in time approximately $s(s(n))$; see [31, Section 1.2] for details.

[5]That is, the hard function had multiple output bits and every probabilistic algorithm running in time (say) n^{100} failed to compute this function on *each and every input* of sufficiently large length.

[6]Recall that a circuit family of size $s(n)$ is logspace-uniform if there is a machine that gets input 1^n, uses $O(\log(s(n)))$ space, and prints the n^{th} circuit in the family (see the full version of this paper for a precise definition [5, Definition 3.5]).

are (conjectured to be) *outside of* $\mathcal{PSPACE}$; in particular, these functions are not necessarily downward self-reducible. Secondly, it relies on hardness for an *arbitrary* function in the class, rather than only for specific functions with useful structure. Thirdly, the tradeoff is smooth, and in particular applies to the "high-end" regime of parameters (when hardness is $2^{\epsilon \cdot n}$ and the derandomization is in polynomial time); our result is indeed more general, covering the entire parameter range (see [5, Theorem 5.2]). And as a fourth point, the derandomization algorithm works on *all input lengths*, rather than only on infinitely many input lengths.

a) Optimal worst-case to average-case reduction: A salient feature of Theorem 1.1 is that we assume *worst-case hardness* and yet deduce derandomization that succeeds on $1 - o(1)$ of the inputs. One might suspect that the proof will go through a worst-case to average-case reduction for probabilistic algorithms (i.e., a reduction of computing a function in the worst-case to computing it on $o(1)$ of the inputs). In fact, the reduction that seems to be implicit in the result should be essentially optimal, since the conclusion of Theorem 1.1 does not have super-polynomial overheads in the algorithm's running time.

Prior to our work, *optimal* worst-case to average-case reductions for probabilistic algorithms were known either for $\mathcal{E}$ (see [31]) or for small subclasses of $\mathcal{P}$ (see, e.g., [8]). However, for classes such as the one in Theorem 1.1, the known reductions relied on hardness only for specific problems, and moreover these problems were computable in time $2^{o(n)}$ (and thus cannot have hardness $2^{\Omega(n)}$; see [31] for details).

On the way to proving Theorem 1.1 we are indeed able to prove an optimal worst-case to average-case reduction for computing functions in complexity classes such as the one in Theorem 1.1. We now state what seems to us as the most interesting special case, which is an optimal worst-case to average-case reduction for computing functions in $\mathcal{SPACE}[O(n)]$ by probabilistic algorithms.

Theorem 1.2 (optimal worst-case to average-case reduction for linear space; informal, see [5, Theorem 5.4]*):* For every "nice" $\epsilon(n)$ and $T(n)$, if $\mathcal{SPACE}[O(n)] \not\subseteq \mathtt{i.o.}\mathcal{BPTIME}[T]$, then $\mathcal{SPACE}[O(n)]$ is hard to compute on more than $(1/2 + \epsilon)$ of the inputs in probabilistic time $T(n/c) \cdot (\epsilon/n)^c$, for a constant $c > 1$, on all sufficiently large input lengths $n \in \mathbb{N}$.

As a corollary of Theorem 1.2, if $\mathcal{SPACE}[O(n)] \not\subseteq \mathtt{i.o.}\mathcal{BPTIME}[2^{\delta \cdot n}]$, then $\mathcal{SPACE}[O(n)]$ cannot be successfully computed on $1/2 + 2^{-\delta' \cdot n}$ of the inputs in probabilistic time $2^{\delta' \cdot n}$, where $\delta' = \Theta(\delta)$. The main technical result underlying Theorem 1.2 is a construction of a new instance-checkable problem that is complete for $\mathcal{SPACE}[O(n)]$ under linear-time reductions (see Section II for details).

B. Fine-grained hardness for unstructured problems

As mentioned above, a second type of uniform hardness vs randomness results focuses on *fine-grained hardness*; that is, showing average-case derandomization under assumptions that functions in $\mathcal{P}$ cannot be solved in some fixed polynomial time.

Results of this type that rely on hardness for *non-uniform circuits* have been extensively studied. Specifically, following Goldreich and Wigderson [9], a sequence of works culminated in the following result by Kinne, van Melkebeek, and Shaltiel [18] (see also [21, 26, 25]): If for every k there is $L_k \in \mathcal{P}$ that is hard to compute with less than $1/n$ errors by non-uniform circuits of size n^k (for all $n \in \mathbb{N}$), then $\mathcal{BPP}$ can be derandomized in polynomial time on average (over the uniform distribution, with error $1/\mathrm{poly}(n)$; see [18, Theorem 1]).

Since the conclusion is an average-case derandomization, a natural goal is to try and relax the hypothesis and only assume hardness for *uniform* probabilistic algorithms (rather than for circuits). Recently, Carmosino, Impagliazzo and Sabin [4] showed the first result along these lines: They deduced average-case derandomization from hardness of *specific problems in* $\mathcal{P}$, namely of counting k-cliques or for k-orthogonal-vectors. Indeed, the latter problems have a structure similar to the one required in classical results, namely they are downward self-reducible in some sense (see [4, Section 2.1], following [2]). Nevertheless, their work managed to bypass some of the traditional obstacles (e.g., getting derandomization in polynomial time or on all input lengths, similarly to what we were able to obtain in Section I-A).

In this context too, our goal is to get rid of the structural requirements and of the dependency on hardness of specific problems, while simultaneously significantly improving on the parameters. Our first result starts from mild average-case hardness for *any function in a large natural subclass of* $\mathcal{P}$: Namely, the class of problems that can be decided by logspace-uniform circuits of polynomial size and fixed polynomial depth, say n^3. (Indeed, note that this upper bound refers to uniform circuits of *polynomial* depth rather than only to logspace-uniform $\mathcal{NC}$.) That is:

Theorem 1.3 (derandomization from mild average-case fine-grained hardness): Fix $d \in \mathbb{N}$, and assume that for every $c \in \mathbb{N}$ there is a problem $L \subseteq \{0,1\}^*$ computable by logspace-uniform circuits of polynomial size $n^{O_c(1)}$ and depth n^d such that $L \notin \mathtt{i.o.}\text{-}\mathsf{avg}_{(1-n^{-d})}\text{-}\mathcal{BPTIME}[n^c]$. Then, for every $a \in \mathbb{N}$ it holds that

$$\mathcal{RP} \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P} \ ,$$

where the notation avg refers to average-case simulation over the uniform distribution.[7]

Indeed, Theorem 1.3 gets very close to achieving the goal of simply replacing the non-uniform hardness assumption in the result of [18] by a uniform hardness assumption; the only remaining gaps are that we require a fixed polynomial depth upper bound and that we derandomize $\mathcal{RP}$ rather than $\mathcal{BPP}$. (This is indeed reminiscent of the gaps between Theorem 1.1 and the "ideal" result mentioned there.)

[7]That is, the notation "$\mathcal{C} \in \mathsf{avg}_{(1-\delta)}\text{-}\mathcal{C}'$" means that for every $L \in \mathcal{C}$ there exists $L \in \mathcal{C}'$ such that for every sufficiently large $n \in \mathbb{N}$ it holds that $\Pr_{x \in \{0,1\}^n}[L(x) = L'(x)] \geq 1 - \delta$.

The specific problems considered prior to our work (i.e., k-clique and k-orthogonal-vectors) belong to the class in Theorem 1.3, and in fact also to a smaller subclass that will be considered next. We stress that the hypothesis in Theorem 1.3 only assumes that every n^c-time algorithm fails on a n^{-a}-fraction of the inputs (i.e., we assume a mild average-case hardness), but the conclusion is that the derandomization succeeds on the vast majority of inputs (i.e., on a $1 - n^{-d}$ fraction).

a) Derandomization from worst-case fine-grained assumptions: While the average-case hardness assumption in Theorem 1.3 is quite mild, it is still stronger than a worst-case hardness assumption. In the following result we strike a different tradeoff. We define a natural subclass of $\mathcal{P}$ (we encourage the reader to intuitively think of it as a subclass of the one in Theorem 1.3) that consists of functions computable by logspace-uniform *arithmetic formulas of arbitrary polynomial size and fixed polynomial degree*; for example, arithmetic formulas of size $\mathrm{poly}(n)$ and degree n^2. That is:

Definition 1.4 (low-degree arithmetic formulas): Let $d \in \mathbb{N}$, let $p(n)$ be a function mapping integers to prime powers such that $n^4 \leq p(n) \leq \mathrm{poly}(n)$, and let $g = \{g_n\}$ such that $g_n \colon [p(n)] \to \{0,1\}^*$ is computable in space $O(\log(n))$. Let $F = \{F_n\}$ be a family of logspace-uniform arithmetic formulas of degree n^2 and polynomial size over $\mathbb{F}_{p(n)}$. Consider the problem $\Pi = \Pi^{F,p,g}$ in which the input is x and the output is $g(F(x))$.

Assuming that this class is hard, in the *worst-case*, for probabilistic algorithms running in any fixed polynomial time n^c, we deduce that $\mathcal{RP} = \mathcal{P}$ on average:

Theorem 1.5 (derandomization from worst-case fine-grained hardness for low-degree arithmetic formulas): Assuming that for every $c \in \mathbb{N}$ there are some g and F and p (as in Definition 1.4) such that $\Pi^{F,p,g} \notin \mathtt{i.o.}\mathcal{BPTIME}[n^c]$. Then for every $a \in \mathbb{N}$

$$\mathcal{RP} \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P} \ .$$

Intuitively, one should think of Theorem 1.5 as starting from hardness in a smaller subclass than that of Theorem 1.3, but requiring only worst-case hardness rather than mild average-case hardness. (The reason that we intuitively think of the class in Theorem 1.5 as a subclass of the one in Theorem 1.3 is that formulas of fixed polynomial degree can be evaluated in small depth; see Section II for details.)

b) A comparison of the parameters to previous work: As mentioned above, beyond the fact that we start from hardness of arbitrary functions in natural subclasses of $\mathcal{P}$, our results also significantly improve on the *parameters* of previous work. To see this, recall that [4] proved that if counting k-cliques in a given n-vertex graph requires probabilistic time $n^{(1/2+\epsilon)\cdot k}$ for some $\epsilon > 0$, then $\mathcal{BPP} = \mathcal{P}$ on average over the uniform distribution. Instantiating Theorem 1.5 for the special case of this problem (indeed, one can count k-cliques with low-degree arithmetic formulas as in Definition 1.4), we obtain the following corollary:

Corollary 1.6 (derandomization from hardness of k-clique, for comparison): Assume that for every $c \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that counting k-cliques is hard for probabilistic time n^c on all input lengths. Then, for every $a \in \mathbb{N}$

$$\mathcal{RP} \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P} \ .$$

The difference in hypotheses between Corollary 1.6 and the result of [4] is that the latter requires the hardness $n^{h(k)}$ of k-clique to grow as $h(k) = (1/2+\epsilon)\cdot k$, whereas we only require that $h(k)$ will be an unbounded function. As a consequence of our improved parameters, our results also immediately imply an affirmative answer to the main open problem in [4], which asked to obtain similar results for problems such as k-SUM that can be solved in time $O(n^{\lceil k/2 \rceil})$.

As demonstrated by the special case of Corollary 1.6, the assumptions in Theorems 1.3 and 1.5 are arguably among the most believable assumptions that are currently known to imply polynomial-time derandomization. Indeed, the only caveat is that we derandomize $\mathcal{RP}$ rather than $\mathcal{BPP}$ (see Section II-D for an explanation why).

II. Technical overview

The technical starting-point for our work is a non-black-box derandomization algorithm from [7], called a reconstructive targeted HSG. This algorithm H_f relies on a hard function f that is computable by logspace-uniform circuits of size T and bounded depth $d \ll T$ to solve the following task: The algorithm *gets input* $x \in \{0,1\}^n$, and prints a set $H_f(x)$ of n-bit strings that is, hopefully, pseudorandom for every efficient algorithm *that also gets access to the same input* x.

The analysis of this algorithm works via a reconstruction argument: Any efficient algorithm that gets input x and distinguishes (the uniform distribution on) $H_f(x)$ from uniformly random strings can be converted into an algorithm that computes f quickly *at the same input* x.[8] Thus, the hardness of f is converted into randomness "instance-wise", for every fixed input. Indeed, a caveat here is that pseudorandomness is only guaranteed for probabilistic algorithms with one-sided error – the reconstruction relies on the assumption that $A_x(\cdot)$ accepts a uniformly random string, with high probability, but rejects all strings in $H_f(x)$. (See [5, Theorem 4.5] for precise details.)

a) A recurring challenge: Worst-case to average-case reductions: At a high-level, in this work we start with worst-case hardness assumptions (or with mild average-case hardness assumptions); that is, we assume that every algorithm fails to compute f at *one input* (or on a small fraction of inputs). However, since H_f translates hardness into randomness "instance-wise", if we want to use H_f to obtain derandomization that succeeds on $1 - o(1)$ of inputs, we need a function that is hard on $1 - o(1)$ of the inputs. Thus, many of our results will include worst-case to average-case reductions, which imply that if a function f as above is hard on the worst-case,

[8] Indeed, more formally, for every efficient algorithm A there exists an efficient algorithm F such that for every fixed x, if $A(x,\cdot) = A_x(\cdot)$ is a distinguisher for $H_f(x)$ then $F(x) = f(x)$.

then there is another function f' with similar complexity that is hard on $1 - o(1)$ of the inputs. (Other results will include reductions of computing a function successfully on $1 - o(1)$ of the inputs to computing it on $o(1)$ of the inputs.)

A. Proofs of Theorems 1.1 and 1.2

The first technical result in our work is a construction of a new instance-checkable problem. Recall that a problem L is instance-checkable if there is a probabilistic algorithm M that gets input x and oracle $\tilde{L}$, and with high probability, if $\tilde{L} = L$ then $M(x) = L(x)$, and for any $\tilde{L}$ satisfies $M(x) \in \{L(x), \perp\}$ (see [5, Definition 3.11]). An instance checker is useful for reductions of computing f in the worst-case to computing some $\bar{f}$ in the average-case: This is because such reductions usually rely on local list-decoding of error-correcting codes to produce a *list of candidate procedures* for f, and an instance checker allows us to test each candidate and only "trust" the answer of ones who are correct (see, e.g., [31, Section 5] for further explanation).

a) The basic version of our instance checker: For every logspace-uniform circuit C of size $T(n) \leq 2^{O(n)}$ and depth $d \ll T$, we construct a problem $L_C \subseteq \{0,1\}^*$ such that:

1) Computing C reduces in linear time to computing L_C.
2) L_C has approximately the same complexity as C.
3) L_C has an instance checker that runs in time $\mathrm{poly}(n, d, \log(T))$ and given x only makes queries of length $|x|$.

Crucially, since the reduction runs in linear time, if C is hard for probabilistic algorithms running in time $T(n)$, then L_C is also hard for probabilistic algorithms running in similar time $T(\Omega(n))$. And since we can construct L_C for any C of complexity as above, it means that if *any* such C is hard for time $T(n)$, then there is an instance-checkable problem L_C with similar hardness $T(\Omega(n))$.

The construction of $L = L_C$ is based on ideas from the doubly efficient interactive proof system of Goldwasser, Kalai, and Rothblum [10]. Loosely speaking, for any logspace-uniform circuit family C of size T and depth d and any input $x \in \{0,1\}^n$, they showed a way to encode the computation of $C(x)$ as a matrix M_x whose entries are in a field $\mathbb{F}$ of size $\mathrm{poly}(T)$ such that the following holds: Verifying a claimed value for the $(i,j)^{th}$ entry in M_x reduces in probabilistic time $\mathrm{poly}(n, d, \log(T))$, and via additional queries to M_x, to a predicate on the input x that is also computable in time $\mathrm{poly}(n, \log(T))$.

The main idea in our construction of L is to define its inputs as (x, i, j, k), where x is an input to C and (i, j) is an index in M_x (and $k \in [\log(|\mathbb{F}|)]$ is the index of a bit in the representation of $\mathbb{F}$-elements). The instance checker simulates the verifier of [10], reducing the computation of L at any given (x, i, j, k) to verification of L at other points corresponding to M_x, and then finally to an efficient computation on the input x. Since the matrix M_x is of size $\mathrm{poly}(T) \leq 2^{O(n)}$, the length of an index (i, j) is at most $O(n)$, and thus the blow-up in input length from inputs for C to inputs for L is only linear.[9] And indeed, the encoding of $C(x)$ into the matrix M_x is not computationally expensive, which means that the complexity of L_C is not much larger than that of C;[10] see [5, Proposition 4.4] for a precise statement and a proof.

b) Proof of Theorem 1.1: The proof of Theorem 1.1 will use the instance checkable L above, but it does not explicitly rely on a worst-case to average-case reduction. Assume that some C of size $2^{O(n)}$ and depth $d = 2^{o(n)}$ computes a function that is hard for $\mathcal{BPTIME}[2^{\epsilon \cdot n}]$, and let $L = L_C$ be the problem above. The main idea in the proof is to apply the generator H_f to the function f that maps any input $x \in \{0,1\}^n$ to the *truth-table of L on $\ell = O(\log(n))$ input bits*; that is, the hard function $f\colon \{0,1\}^n \to \{0,1\}^{2^\ell}$ prints the entire truth-table of L_ℓ (where L_ℓ denotes the restriction of L to inputs of size ℓ). Since L_ℓ is computable by logspace-uniform circuits with similar complexity to that of C, we can also compute f with a circuit of approximately the same complexity (by computing the output bits in parallel).

Now, to simulate a probabilistic linear-time algorithm A on input $x \in \{0,1\}^n$, we compute $H_f(x)$ and output $\vee_{s \in H_f(x)} A(x, s)$. [11] Why does this derandomization work, on average, over any polynomial-time samplable distribution? Assume that an efficient sampling algorithm S succeeds, with probability $1/n$, in finding x such that $\Pr_r[M(x, r) = 1] \geq 1/2$ but $M(x, s) = 0$ for every $s \in H_f(x)$. (For simplicity let us assume that S runs in linear time too.) For any such x, the reconstruction algorithm R for $H_f(x)$ asserts that in this case we can compute $f(x)$ in time $|f(x)| \cdot n^c$, where n^c is much smaller than the hardness $2^{\epsilon \cdot \ell}$ of L_ℓ.

We would like to use this to contradict the worst-case hardness of L_ℓ. There are two problems, however. First, the output size of f is much larger than the hardness of L_ℓ (i.e., $|f(x)| = 2^{O(\ell)} \gg 2^{\epsilon \cdot \ell}$), making the reconstruction R too inefficient to yield a contradiction. To handle this problem, we observe that the reconstruction algorithm of H_f satisfies a stronger property: Not only can it print $f(x)$ in time $|f(x)| \cdot n^c$, it can actually print a circuit $C_{f(x)}$ of size n^c whose truth-table is $f(x)$ (see [5, Theorem 4.5]).[12] Thus, we can compute L_ℓ at any input $q \in \{0,1\}^\ell$ (i.e., compute the q^{th} bit of $f(x)$) by running R to obtain $C_{f(x)}$ and outputting $C_{f(x)}(q)$.

The second problem is that the procedure above succeeds only with low probability $1/n$ (i.e., the probability that S finds an x such that $M(x, \cdot)$ is a distinguisher). We overcome this

[9] In fact, the blow-up is additive $n \mapsto n + O(\log(T))$, where $T \leq 2^{O(n)}$.

[10] Loosely speaking, the encoding M_x of $C(x)$ involves arithmetizing each layer of $C(x)$ via a low-degree extension, and adding a small number of intermediary low-degree polynomials between each pair of layers. Both the low-degree extensions and the intermediary low-degree polynomials can be efficiently computed from the original layers of $C(x)$.

[11] In the overview we focus on derandomizing $\mathcal{RTIME}[O(n)]$, for simplicity.

[12] This is the case because the reconstruction argument iteratively reconstructs circuits of small size (i.e., less than n^c) for each of the $2^{o(n)}$ layers of the circuit for f, starting from the bottom (input) layer and working its way up to the top (output) layer. Thus, in the last step it obtains a circuit whose truth-table is (a low-degree extension of) the string $f(x)$. See [5, Proof of Theorem 4.5] for precise details.

using the instance checker: Given input $z \in \{0,1\}^\ell$ we run S for $k = O(n)$ times obtaining $x_1, ..., x_k$, and for each $i \in [k]$ we run the instance checker with input z, while answering each of its queries $q \in \{0,1\}^\ell$ with the reconstruction $R(q)$ and the distinguisher $D_{x_i}(r) = M(x_i, r)$. Assuming that at least one x is "good", and that all invocations of the instance checker and of R were correct, the instance checker will output $L_\ell(z)$ for some $i \in [k]$, and will either output $\perp$ or $L_\ell(z)$ for all $i \in [k]$, allowing us to deduce $L_\ell(z)$. The running time of this procedure is some fixed polynomial, we can ensure that it is less than $2^{\epsilon \cdot \ell}$ by taking $\ell = O(\log(n))$ to be sufficiently large.

c) Derandomization of $\mathcal{BPP}$: The foregoing argument yields derandomization of $\mathcal{RP}$. To deduce derandomization of $\mathcal{BPP}$ with short advice, we observe that the targeted generator H_f is not only a targeted hitting-set generator, but also a *targeted somewhere-PRG*; that is, it outputs a collection of $d'(\ell) \approx d(\ell) = n^{o(1)}$ lists $W_1, ..., W_{d'}$ of strings, and for every efficient algorithm D there exists $i \in [d']$ such that W_i is pseudorandom for D, where pseudorandomness here is in the usual sense of *two-sided error*.

We want to use this targeted somewhere-PRG to argue that for every machine M and sampling algorithm S there exists $i \in [d']$ such that the probability that S samples an input x for which $M(x, \cdot)$ is a distinguisher for W_i is at most $1/n$. Given this claim, we can hard-wire i into the derandomization algorithm as advice of length $\log(d')$, and the derandomization algorithm will only use the pseudorandom strings in W_i.

To show the claim above, assume the opposite: For each $i \in [d']$, with probability at least $1/n$ the sampling algorithm S outputs x such that $M(x, \cdot)$ is a distinguisher for W_i. Recall that the pseudorandomness of the generator was established by a reconstruction argument, asserting that a distinguisher D can be used to compute L_ℓ too quickly. We show a *stronger reconstruction procedure*, which works not only when it is given a distinguisher D, but also when it is given a sequence of d' sets of functions such that for $i \in [d']$, the i^{th} set contains a distinguisher for W_i. (Intuitively, this reconstruction procedure implicitly performs iterative "instance-checking": It works in d' iterations, and in each iteration it is able to find the "good" distinguisher among the candidate functions in the corresponding set.) For each $i \in [d']$, we call S for $O(n \cdot \log(d'))$ times to sample a set X_i of inputs, such that with high probability, for every $i \in [d']$ there is $x_i \in X_i$ such that $D_{x_i}(\cdot) = M(x_i, \cdot)$ is a distinguisher for W_i. This satisfies the hypothesis of the stronger reconstruction procedure, allowing us to contradict the hardness of L_ℓ. For precise details see [5, Proofs of Theorems 4.5 and 5.3].

d) Proof of Theorem 1.2: We want to prove an optimal worst-case to average-case reduction for computing $\mathcal{SPACE}[O(n)]$ by probabilistic algorithms, and the main challenge will be to refine the instance checker above. At a high-level, our reduction follows a standard plan: Given $L^{(0)} \in \mathcal{SPACE}[O(n)]$ that is hard for probabilistic algorithms in the worst case, we reduce $L^{(0)}$ to an instance-checkable L, encode the truth-table of L by a locally list-decodable code Enc that is computable in space $O(n)$ (see [5, Theorem 3.8]), and the reduction applies the instance checker with each of the candidate circuits that the local decoder outputs (we do not elaborate on this, since the general approach is well-known; see the proof of Theorem 1.2 in [5] for details).

The challenge is that L above is "complete" for logspace-uniform circuits of size T and depth $d \ll T$,[13] whereas *we want L to be complete for $\mathcal{SPACE}[O(n)]$* (both notions of completeness here refer to linear-time reductions). Indeed, any function $L^{(0)} \in \mathcal{SPACE}[O(n)]$ has circuits of size $2^{O(n)}$ and depth $\mathrm{poly}(n)$, using the standard technique of repeatedly squaring the transition matrix of the linear-space machine M for $L^{(0)}$, and moreover these circuits are logspace-uniform. The crucial observation is that given an input $x \in \{0,1\}^n$ and an index of a gate $g \in [2^{O(n)}]$ in this circuit, *we can compute in linear space the value of $g(x)$*. This is because every gate g is associated with two instantaneous configuration γ, γ' of M, and $g(x)$ indicates whether or not running M for $i \leq 2^{O(n)}$ steps, starting from the configuration γ, results in the configuration γ'. Thus, to compute $g(x)$ we can simply simulate M starting from configuration γ, and check whether its configuration after i steps is γ'.

Given this property, we observe that all the steps required to compute L (i.e., to compute an entry in M_x) maintain the linear-space complexity. Intuitively, this is because these steps mainly involve computing low-degree extensions of the layers of the circuit for $L^{(0)}$ (or simple reductions between a constant number of low-degree extensions), and these can be carried out in space $O(n)$ with oracle access to the gates of the original circuit. See [5, Proposition 4.3] for further details.

B. Proof of Theorem 1.3

At a high-level, the plan for proving Theorem 1.3 is as follows. We assume that there is a problem $L^{(0)}$ computable by logspace-uniform circuits of polynomial size and depth n^2 such that $L^{(0)} \notin \mathsf{avg}_{(1-1/n)}\text{-}\mathcal{BPTIME}[n^c]$.[14] Since $L^{(0)}$ is reducible in linear time to the instance-checkable problem L described in the beginning of Section II-A, we hope to prove that L will also have essentially the same hardness. We then encode L via the k-wise direct product code with $k = \tilde{O}(n^2)$ repetitions, to obtain a problem $L^{\otimes k}$ with essentially the same computational complexity,[15] and use the instance checker as well as the celebrated direct product theorem of Impagliazzo, Jaiswal, Kabanets, and Wigderson [14] to argue that $L^{\otimes k}$ cannot be computed in fixed polynomial time even on (say) $1/n^3$ of the inputs (see below). Finally, we use $L^{\otimes k}$ as the hard function for the targeted HSG H_f, obtaining derandomization that runs in polynomial time and succeeds on $1 - 1/n^3$ of the inputs.

There are two parts in the plan above that we left vague: The claim that L is mildly hard on average (supposedly, because of

[13] We write "complete" because the circuit for L_C is somewhat larger and deeper than the circuit C.

[14] We use convenient parameters in the current section, for simplicity.

[15] Recall that the k-wise direct-product of L takes input $\bar{x} = (x_1, ..., x_n) \in (\{0,1\}^n)^k$ and outputs the k bits $L(x_1), ..., L(x_k)$. In particular, we can compute the k output bits in parallel.

the reduction from $L^{(0)}$, which is mildly hard on average); and the claim that $L^{\otimes k}$ is hard on $1-1/n^3$ of inputs (supposedly, because it is a k-wise direct product of L). The challenges that underlie the proofs of both claims are similar, so for simplicity we focus on the claim that $L^{\otimes k}$ cannot be computed in time close to n^c even on $1/n^3$ of the inputs.

For a large enough $k = \tilde{O}(n^3)$, assuming towards a contradiction that $L^{\otimes k}$ can be computed on at least $1/n^3$ of the inputs in time $n^{c'}$, we want to contradict the hardness of L. Recall that [14] yields a list-decoder that, with probability $\Omega(n^{-3})$, outputs a circuit of size $\mathrm{poly}(n^{c'})$ that computes L correctly on $1-1/n^2$ of the inputs. Given an input (x,i,j) for L,[16] we can repeatedly invoke the list-decoder to obtain a list of $t = O(n^3)$ circuits $C_1,...,C_t$, and run the instance checker with each C_i, hoping to be "convinced" by the good C_i and not misled by all other C_i's.

The gap in the foregoing argument is that C_i only computes L correctly on $1-n^{-2}$ of the inputs rather than on all inputs, and our instance checker is not guaranteed to work with such C_i's. The reason is that, in contrast to what one might expect when thinking of instance checkers, the queries of our instance checker are *not* uniform. (Indeed, one can design an adversarial C_i that fails this instance checker.)

a) Tolerant instance checkers: To bridge the foregoing gap we modify the instance checkable problem to a problem whose instance checker is more resilient. Specifically, we introduce the notion of tolerant instance checkers, which are instance checkers that, when given an oracle that agrees with the target problem L on $1-\epsilon$ of the inputs, satisfy the completeness requirement of a standard instance checker on at least $1-\epsilon'$ of the inputs, for $\epsilon' \approx \epsilon$ (see [5, Definition 3.12]).

We then refine the instance checkable problem L above so that it indeed has a *tolerant* instance checker, rather than only a standard one. Specifically, recall that the matrix M_x in the definition of L consists of $d' = \tilde{O}(d)$ rows where each row is a low-degree polynomial $\mathbb{F}^m \to \mathbb{F}$ (for a suitable choice of $m \in \mathbb{N}$), and in entry (i,j) we have the evaluation of the polynomial $\hat{\alpha}_i$ at the input indexed by j, denoted $\vec{j} \in \mathbb{F}^m$. For every fixed x we define a polynomial $p_x \colon \mathbb{F} \times \mathbb{F}^m \mapsto \mathbb{F}$ that interpolates all the d' polynomials; that is, when p_x gets as input (i,j) where $i \in [d']$ it outputs $\hat{\alpha}_i(\vec{j})$, and otherwise (when $i \notin [d']$) it outputs an interpolation of the d' polynomials. Since the number d' of polynomials is sufficiently small, the polynomial p_x is of low degree.

Now, we modify the definition of L such that it gets input (x,i,j) where $i \in \mathbb{F}$ may also be outside $[d']$, and we prove that this new version has logspace-uniform circuits with essentially the same depth as the previous version and with only a polynomial size overhead, and that also has an instance checker with the same time complexity as the previous version. The reason that these two claims hold is that d' is small (given that we start from $L^{(0)}$ whose circuits have fixed polynomial depth but larger polynomial size), and hence to compute L we just need to interpolate a small number of polynomials (see [5, Proposition 4.4] for details). We obtain the following two properties:

1) Given input (x,i,j), the instance checker only makes queries of the form (x,i',j'); that is, all queries have the same first component x as the input.[17]
2) For every fixed x, the function $p_x(i,j) = L(x,i,j)$ is a low-degree polynomial.

To see that this problem has a tolerant instance checker, note that if an oracle agrees with L on most inputs (x,i,j), then for most x it agrees with p_x with high probability over (i,j), say $9/10$. Thus, for most x the instance checker can use *self-correction of the low-degree polynomial* p_x, and run the original instance checker while simulating access to the actual polynomial p_x (again, see [5, Proposition 4.4] for precise details).

b) Using the refined instance checker to bridge the gaps: Let us see how we use these properties to bridge the gaps in our proof. Recall that in our "towards a contradiction" argument (when we assumed that $L^{\otimes k}$ was "too easy"), when repeating the list-decoder we obtained a list of circuits $C_1,...,C_t$, and at least one C_w computes L on $1-n^{-2}$ of the inputs. We can thus run the tolerant instance checker with each of these circuits C_i as oracle: The soundness condition holds on every input and with each oracle, whereas the tolerant completeness condition guarantees that there is a set of approximately $1-n^{-2}$ inputs such that when the instance checker uses oracle C_w it is able to compute L correctly. This yields the contradiction that we wanted.

(The proof is actually a bit more cumbersome technically, since we want to preserve hardness on almost all input lengths. This requires us to also use a tolerant instance checker for $L^{\otimes k}$, which tolerates very high corruption; such a tolerant instance checker can be obtained directly from the tolerant instance checker for L. For details see [5, Claim 3.12.1 and Lemma A.5].)

c) Strongly tolerant instance checkers: A similar argument allows us to prove that L is mildly hard on average, based on the mild average-case hardness of $L^{(0)}$. However, since we are now trying to preserve very mild hardness on all input lengths under reductions, the argument turns out to be more subtle, and requires us to introduce a more refined notion of strongly tolerant instance checkers. The instance checker presented above is already strongly tolerant, and using it the argument carries through. For technical details see [5, Definition 3.13 and Lemma A.2].

C. Proof of Theorem 1.5

Recall that we now want to prove derandomization assuming *worst-case* hardness of a function computable by low-degree arithmetic formulas of polynomial size. The intuition

[16] In this section, for simplicity of presentation, we ignore the fourth component in inputs to L, whose only function is to transform L into a Boolean function.

[17] Indeed, our previous construction of the instance checker already has this property, and it is maintained when interpolating the polynomials into p_x; see [5, Proof of Proposition 4.4]. Also, for simplicity of presentation we ignore the additional input k that converts L into a Boolean function.

underlying the proof of Theorem 1.5 is that arithmetic formulas can be *balanced* to be of logarithmic depth, by a very efficient algorithm; hence, this class of formulas is essentially a subclass of the one from Theorem 1.3. Moreover, since the formulas have low-degree, this class supports a worst-case to mild average-case reduction.

Thus, our goal is to start from worst-case hardness for our class of arithmetic formulas, argue that the formulas can be balanced while maintaining their complexity, deduce mild average-case hardness, and then invoke Theorem 1.3 as a black-box.

Balancing the formula by low-depth circuits: For any logspace-uniform arithmetic formula F_n of degree n^2, we show that the corresponding polynomial P_n can be computed in logspace-uniform $\mathcal{NC}$ (i.e., the circuit computing P_n has depth $\mathrm{polylog}(n)$).

By a standard argument (see, e.g., [29, Theorem 2.6]), any arithmetic formula of polynomial size can be converted into an equivalent arithmetic circuit of polynomial size and depth $O(\log(n))$. Our key observation is that this balancing algorithm is quite simple: In particular, the bottlenecks of the procedure are finding a "center of mass" of a binary tree,[18] and computing a certain partial derivative, both of which can be done in logspace-uniform $\mathcal{NC}$. With this observation in mind, the "balancing" procedure can be carried out in $O(\log(n))$ stages, with each stage implementable in logspace-uniform $\mathcal{NC}$. After the balancing, we evaluate the $O(\log(n))$ depth arithmetic circuit in logspace-uniform $\mathcal{NC}$ to compute P_n (see [5, Lemma 7.3.2 and Section 7.1.1] for details).[19]

Technical complications when working with prime fields: In some settings we will need to consider the formula as a polynomial over a *large prime field*; this happens, for example, when considering arithmetic formulas for counting problems (such as counting k-cliques). A standard complication in this setting is that the average-case complexity of the problem is sensitive to the Boolean encoding of field elements (see [5, Proof of Lemma 7.3] for details). An additional complication in this setting is that in the worst-case to average-case reduction, we need to deterministically and quickly find such a prime (e.g., find a prime of size n^{100} in deterministic time n^2), but such an algorithm isn't known. Thus, in our worst-case to average-case reduction we actually define an auxiliary problem in which the prime is incorporated into the truth-table. (See [5, Proof of Lemma 7.3] for a careful implementation of this idea.)

D. Why only $\mathcal{RP}$?

Let us explain the technical challenge due to which we were only able to derandomize $\mathcal{RP}$ in Theorems 1.3 and 1.5, rather than $\mathcal{BPP}$. The same technical challenge also existed in [7], and in fact it dates back at least 25 years, to the work of Impagliazzo and Wigderson [13] that founded the area of uniform hardness vs randomness.

Fix a uniform probabilistic linear-time machine M whose coins we wish to replace by pseudorandom coins on a given input x. Assume that we can produce, in time $\mathrm{poly}(n)$, a sequence of n sets $S_1, ..., S_n \subseteq \{0,1\}^n$, each consisting of $\mathrm{poly}(n)$ strings, and we are guaranteed that for every x there *exists* $i \in [n]$ such that S_i is pseudorandom for M with input x. Can we *combine* the n sets, perhaps using additional $O(\log(n))$ random bits, into a single set S that is guaranteed to be pseudorandom for M with x?

Indeed, this challenge refers to the *computational version* of an object known in extractor theory as *mergers*; it is thus apt to refer to it as considering computational mergers. While we know how to construct computational mergers in other setting – for example, when the distinguisher class is non-uniform – in our setting where M is uniform (and does not have enough time to compute the strings in the S_i's by itself), we do not know how to solve this. This obstacle prevented many previous works from obtaining average-case derandomization on all input lengths (see, e.g., [13, 3, 31, 6]), and significantly increased the running time of the worst-case derandomization in [7] when it was scaled to the "low-end" parameter setting.

[18]Given a binary tree (meaning that each node has at most two children) T of n nodes, a node u is called a "center of mass", if the size of the sub-tree rooted at u has size between $[n/3, 2n/3]$.

[19]We remind the reader that arithmetic *circuits* can also be balanced, albeit by a more complicated algorithm (see [33]). We did not try to extend our results to hold for this model.

ACKNOWLEDGEMENTS

Lijie Chen is supported by NSF CCF-2127597 and an IBM Fellowship. Ron Rothblum was funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Part of this work was conducted while Roei Tell was supported by the National Science Foundation under grant number CCF-1445755 and under grant number CCF-1900460.

REFERENCES

[1] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. "Power from random strings". In: *SIAM Journal of Computing* 35.6 (2006), pp. 1467–1493.

[2] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. "Average-case fine-grained hardness". In: *Proc. 49th Annual ACM Symposium on Theory of Computing (STOC)*. 2017, pp. 483–496.

[3] Jin-Yi Cai, Ajay Nerurkar, and D. Sivakumar. "Hardness and hierarchy theorems for probabilistic quasi-polynomial time". In: *Proc. 31st Annual ACM Symposium on Theory of Computing (STOC)*. 1999, pp. 726–735.

[4] Marco L. Carmosino, Russell Impagliazzo, and Manuel Sabin. "Fine-grained derandomization: from problem-centric to resource-centric complexity". In: *Proc. 45th International Colloquium on Automata, Languages and Programming (ICALP)*. 2018, Art. No. 27, 16.

[5] Lijie Chen, Ron D. Rothblum, and Roei Tell. "Unstructured Hardness to Average-Case Randomness". In: *Electronic Colloquium on Computational Complexity: ECCC* 29 (2022), p. 097.

[6] Lijie Chen, Ron D. Rothblum, Roei Tell, and Eylon Yogev. "On Exponential-Time Hypotheses, Derandomization, and Circuit Lower Bounds". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 13–23.

[7] Lijie Chen and Roei Tell. "Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise". In: *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2021.

[8] Oded Goldreich and Guy N. Rothblum. "Worst-case to Average-case reductions for subclasses of P". In: *Electronic Colloquium on Computational Complexity: ECCC* 26 (2017), p. 130.

[9] Oded Goldreich and Avi Wigderson. "Derandomization that is rarely wrong from short advice that is typically good". In: *Proc. 6th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2002, pp. 209–223.

[10] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for muggles". In: *Journal of the ACM* 62.4 (2015), Art. 27, 64.

[11] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. "Uniform hardness versus randomness tradeoffs for Arthur-Merlin games". In: *Computational Complexity* 12.3-4 (2003), pp. 85–130.

[12] Dan Gutfreund and Salil Vadhan. "Limitations of hardness vs. randomness under uniform reductions". In: *Proc. 12th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2008, pp. 469–482.

[13] R. Impagliazzo and A. Wigderson. "Randomness vs. Time: De-Randomization Under a Uniform Assumption". In: *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1998, pp. 734–.

[14] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. "Uniform direct product theorems: simplified, optimized, and derandomized". In: *SIAM Journal of Computing* 39.4 (2010), pp. 1637–1665.

[15] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. "The power of natural properties as oracles". In: *Proc. 33rd Annual IEEE Conference on Computational Complexity (CCC)*. 2018, Art. No. 7, 20.

[16] Russell Impagliazzo and Avi Wigderson. "P = BPP if E requires exponential circuits: derandomizing the XOR lemma". In: *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*. 1997, pp. 220–229.

[17] Valentine Kabanets. "Easiness assumptions and hardness tests: trading time for zero error". In: vol. 63. 2. 2001, pp. 236–252.

[18] Jeff Kinne, Dieter van Melkebeek, and Ronen Shaltiel. "Pseudorandom generators, typically-correct derandomization, and circuit lower bounds". In: *Computational Complexity* 21.1 (2012), pp. 3–61.

[19] Adam Klivans, Pravesh Kothari, and Igor Oliveira. "Constructing Hard Functions Using Learning Algorithms". In: *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*. 2013, pp. 86–97.

[20] Chi-Jen Lu. "Derandomizing Arthur-Merlin games under uniform assumptions". In: *Computational Complexity* 10.3 (2001), pp. 247–259.

[21] Dieter van Melkebeek and Rahul Santhanam. "Holographic proofs and derandomization". In: *SIAM Journal of Computing* 35.1 (2005), pp. 59–90.

[22] Noam Nisan and Avi Wigderson. "Hardness vs. randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167.

[23] Igor C. Oliveira and Rahul Santhanam. "Pseudodeterministic constructions in subexponential time". In: *Proc. 49th Annual ACM Symposium on Theory of Computing (STOC)*. 2017, pp. 665–677.

[24] Rahul Santhanam. "Circuit lower bounds for Merlin-Arthur classes". In: *SIAM Journal of Computing* 39.3 (2009), pp. 1038–1061.

[25] Ronen Shaltiel. "Typically-Correct Derandomization". In: *SIGACT News* 41.2 (2010), 57–72.

[26] Ronen Shaltiel. "Weak derandomization of weak algorithms: explicit versions of Yao's lemma". In: *Computational Complexity* 20.1 (2011), pp. 87–143.

[27] Ronen Shaltiel and Christopher Umans. "Low-end uniform hardness vs. randomness tradeoffs for AM". In: *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*. 2007, pp. 430–439.

[28] Ronen Shaltiel and Christopher Umans. "Simple extractors for all min-entropies and a new pseudorandom generator". In: *Journal of the ACM* 52.2 (2005), pp. 172–216.

[29] Amir Shpilka and Amir Yehudayoff. "Arithmetic Circuits: A survey of recent results and open questions". In: *Found. Trends Theor. Comput. Sci.* 5.3-4 (2010), pp. 207–388.

[30] Madhu Sudan, Luca Trevisan, and Salil Vadhan. "Pseudorandom generators without the XOR lemma". In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 236–266.

[31] Luca Trevisan and Salil P. Vadhan. "Pseudorandomness and Average-Case Complexity Via Uniform Reductions". In: *Computational Complexity* 16.4 (2007), pp. 331–364.

[32] Christopher Umans. "Pseudo-random generators for all hardnesses". In: *Journal of Computer and System Sciences* 67.2 (2003), pp. 419–440.

[33] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. "Fast parallel computation of polynomials using few processors". In: *SIAM Journal of Computing* 12.4 (1983), pp. 641–644.