Derandomization with Minimal Memory Footprint

Dean Doron
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Roei Tell
The Institute for Advanced Study at Princeton, NJ, USA
DIMACS Center at Rutgers University, Piscataway, NJ, USA

Abstract

Existing proofs that deduce $\mathbf{BPL} = \mathbf{L}$ from circuit lower bounds convert randomized algorithms into deterministic algorithms with large constant overhead in space. We study space-bounded derandomization with minimal footprint, and ask what is the minimal possible space overhead for derandomization. We show that $\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[c \cdot S]$ for $c \approx 2$, assuming space-efficient cryptographic PRGs, and, either: (1) lower bounds against bounded-space algorithms with advice, or: (2) lower bounds against certain *uniform* compression algorithms. Under additional assumptions regarding the power of catalytic computation, in a new setting of parameters that was not studied before, we are even able to get $c \approx 1$.

Our results are constructive: Given a candidate hard function (and a candidate cryptographic PRG) we show how to transform the randomized algorithm into an efficient deterministic one. This follows from new PRGs and targeted PRGs for space-bounded algorithms, which we combine with novel space-efficient evaluation methods. A central ingredient in all our constructions is hardness amplification reductions in logspace-uniform $\mathbf{TC}^0$, that were not known before.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Pseudorandomness and derandomization; Theory of computation → Error-correcting codes

Keywords and phrases derandomization, space-bounded computation, catalytic space

Digital Object Identifier 10.4230/LIPIcs.CCC.2023.11

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2023/036/` [14]

Funding The second author is supported in part by the National Science Foundation under grant number CCF-1900460.

Acknowledgements We are grateful to Avi Wigderson for several useful conversations regarding the gap between double space blow-up and single space blow-up. We thank Lijie Chen for suggesting the idea of using catalytic space to save on complexity, early in this work, and for pointing out a gap in a previous version of the proof of Theorem 2. We also thank Ian Mertz for several useful conversations exploring the abilities of catalytic space. We are very grateful to an anonymous reviewer for a careful read of this paper and for many useful comments. Part of this work was done while the second author was visiting the Simons Institute for the Theory of Computing.

1 Introduction

One of the greatest challenges in complexity theory is the derandomization of efficient algorithms, or more broadly, understanding to what extent randomness is necessary or useful for algorithms. In the time-bounded setting, can we simulate any randomized algorithm by a deterministic one with a roughly similar runtime? In the space-bounded setting, can we derandomize with only a small factor blowup in space?

Classical hardness-to-pseudorandomness results tell us that under plausible circuit lower bounds, any randomized algorithm that runs in time T can be be simulated deterministically by an algorithm running in time T^c [37, 26], and any randomized algorithm that uses S

space can be simulated deterministically in space $c \cdot S$ [31], where c is a large constant. In the terminology of complexity classes, $\mathbf{BPP} = \mathbf{P}$ and $\mathbf{BPL} = \mathbf{L}$ follow from circuit lower bounds.[1]

The constant c in the foregoing classical results can indeed be large to the point of impracticality, for reasons that are inherent to the proof techniques. Therefore, a natural question is whether these results can be made *more efficient*, by providing an explicit small bound on the time or space overhead in derandomization. In other words, we ask what is the *precise, fine-grained value of randomness* in various computational settings. Note that this question is likely to be relevant even when the great goal of establishing $\mathbf{BPP} = \mathbf{P}$ and $\mathbf{BPL} = \mathbf{L}$ without relying on hardness assumptions is achieved.

In recent years, starting with the work of Doron, Moshkovitz, Oh, and Zuckerman [13] and continuing with the works of Chen and Tell [11, 10, 12], a study of *fine-grained derandomization* led to a series of results:[2]

- $\mathbf{BPTIME}[T] \subseteq \mathbf{DTIME}[T^{2+\alpha}]$ assuming there exists a language in $\mathbf{DTIME}[2^{(1+\alpha)n}]$ that is hard for certain randomized, non-deterministic circuits of size $2^{(1-\alpha)n}$ [13]. Chen and Tell [11] showed that one can get rid of the circuits' randomness by assuming that the language is batch-computable.
- $\mathbf{BPTIME}[T] \subseteq \mathbf{DTIME}[n^{1+\alpha} \cdot T]$, where n denotes the length of the input, assuming that one-way functions exist, and there exists a language in $\mathbf{DTIME}[2^{k \cdot n}]$ that is hard for $\mathbf{DTIME}[2^{(k-\alpha) \cdot n}]/2^{(1-\alpha) \cdot n}$ [11]. In a followup work, Chen and Tell [12] showed that one can forgo the cryptographic assumption and replace it with [13]-style ones.
- $\mathbf{BPTIME}[T] \subseteq \mathbf{heurDTIME}[n^{\alpha} \cdot T]$, meaning that the derandomization fails with negligible probability with respect to all efficiently-samplable distributions [11]. This result follows from uniform cryptographic assumptions and certain uniform hardness assumptions for multi-bit output functions.
- Derandomization of *interactive proof systems* with constantly many rounds, that either has a (bounded) polynomial time overhead that depends on the number of rounds, or has only n^{α} time overhead and yields a deterministic ($\mathbf{NP}$-style) argument system [12].

These results are often complemented with nearly matching conditional lower bounds (i.e., lower bounds assuming certain complexity-theoretic assumptions). In addition to derandomization in nearly no cost, those results gave rise to new notions, tools, and techniques in derandomization.

The space-bounded setting

In this work, we study efficient *space*-bounded derandomization under hardness assumptions, asking what is the minimal possible *space* overhead for derandomization. That is, can we transform randomized algorithms into deterministic ones that use roughly the same amount of memory?

We note that unlike the time-bounded setting, wherein unconditional derandomization results would lead to lower bounds that currently seem out of reach (see, e.g., [25, 28, 30, 47, 34, 44, 7, 8]), in the space-bounded setting we do have unconditional partial derandomization results. Savitch's theorem [40] can be extended to show that $\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[O(S^2)]$ (see also [5]). Nisan [35, 36] devised a time-efficient derandomization with

[1] We also have equality between the promise classes. In fact, all our results in this paper will hold for the corresponding promise classes as well, but for readability we will omit the promise problems notation.
[2] In what follows, $\alpha > 0$ is an arbitrarily small constant, but different appearances of α may (or should) not be the same. We refer the reader to the relevant papers for the precise statements.

a quadratic overhead in space, namely, $\mathbf{BPL} \subseteq \mathbf{DTISP}[\text{poly}(n), O(\log^2 n)]$. Focusing solely on space, Saks and Zhou [39] cleverly built on Nisan's work to deterministically simulate space-S randomized algorithms in $\mathbf{DSPACE}[O(S^{2/3})]$. The state-of-the-art is a recent improvement by Hoza [23], giving a deterministic simulation in space $O(S^{2/3}/\sqrt{\log S})$.

Still, even *when* $\mathbf{BPL} = \mathbf{L}$ is proven, it is very likely that the minimal-footprint derandomization question would remain: What is the minimal c for which

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[c \cdot S]?$$

We will give assumptions under which c approaches 2, and further assumptions under which c approaches 1! Moreover, the results in this paper are *constructive*. Namely, given a candidate hard function (and a candidate cryptographic PRG), we show how to transform the randomized algorithm into an efficient deterministic one.

We proceed to give an overview of our results. Throughout the paper, when we refer to a nice space function, we mean a function $S(n) \geq c_0 \cdot \log(n)$ (where $c_0 \geq 1$ is a universal constant) such that there exists a Turing machine that gets input $(x, 1^t)$ where $t \leq \lceil \log(S(|x|)) \rceil$, runs in space $O(\log(|x|) + \log(S(|x|)))$, and accepts if and only if $t = \lceil \log(S(|x|)) \rceil$.

1.1 Setting the stage: A tighter hypothesis and improved local list decoding

We first revisit the Klivans–van-Malkebeek result [31] that establishes $\mathbf{BPL} = \mathbf{L}$ from standard, nonuniform hardness assumptions. The [31] result, which goes along the line of [37], states that given a language in $\mathbf{DSPACE}[O(n)]$ that is hard for circuits of size $2^{\varepsilon n}$, then $\mathbf{BPL} = \mathbf{L}$.[3] Can we do better? In particular, can we work with a more restricted class of circuits? We show:

▶ **Theorem 1** (see also [14, Theorem 5.2]). *Assume there exists a language $L \in \mathbf{DSPACE}[O(n)]$ that is hard for $\mathbf{TC}^0$ circuits of size $2^{\varepsilon n}$, for some $\varepsilon \in (0,1)$, with oracle access to read-once branching programs of length and width $2^{\varepsilon n}$. Then, for any nice space function S,*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[O(S)].$$

While Theorem 1 is not needed for our minimal-footprint results, the main ingredient that goes into the proof of Theorem 1 is a basic component in all of our results: We give a new *hardness amplification* result in $\mathbf{TC}^0$, or equivalently, a new locally list decodable code with $\mathbf{TC}^0$ decoding. We elaborate on it in Section 2.1.

1.2 Black-box derandomization with minimal footprint

Our first derandomization result follows from *worst-case nonuniform hardness assumptions* and *cryptographic assumptions*. We begin with our hardness assumption, which asserts that there is a language computable in linear space that is hard for algorithms that use smaller linear space as well as non-uniform advice.

[3] In [31] it is also stated that one can obtain $\mathbf{BPL} = \mathbf{L}$ from a size-$2^{\varepsilon n}$ lower bound on branching programs. The proof of this statement is not spelled out there in full detail, and as far as we understand, the branching programs referred to in the statement are non oblivious and non read once – a model that lies between $\mathbf{NC}^1$ and $\mathbf{AC}^1$. Theorem 1 gives a stronger statement.

▶ **Assumption 1** (nonuniform hardness assumption). *For a sufficiently large constant C there exists a language L computable in deterministic space $(C+1)\cdot n$ that is hard, on all but finitely many input lengths, for algorithms that run in deterministic space $C\cdot n$ with $2^{n/2}$ bits of advice.*

We note that the gap of $C+1$ vs. C can be replaced by any constant gap (i.e., $C+k$ vs. C for any small constant k), at the cost of allowing a relatively minor additional overhead in the derandomization; for the precise statement, see [14, Theorem 5.5]. We note that a small gap between the space complexity of L and the space for which it is hard for, is inherent for "super efficient" derandomization results merely due to space hierarchy theorems.

We continue with our cryptographic PRG.

▶ **Assumption 2.** *There exists a polynomial-stretch PRG fooling circuits of arbitrary polynomial size, computable in logarithmic space.*[4]

One appealing candidate for a cryptographic PRG satisfying Assumption 2 is Goldreich's expander-based PRG [16], instantiated with expanders whose neighbor function is logspace-computable; we elaborate on this below. However, in the assumption we can use any cryptographic PRG with polynomial stretch, as long as its space complexity is as described.

Equipped with those two assumptions, we can state our efficient derandomization from worst case hardness assumptions.

▶ **Theorem 2** (see also [14, Theorem 5.5]). *Suppose that Assumption 1 and Assumption 2 hold. Then, for any nice space function S, we have that*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[\left(2+\frac{c}{C}\right)S\right],$$

where $c>1$ is some fixed universal constant.

As the section's name suggests, the above result uses a space-efficient pseudorandom generator (along the lines of [11]), which we combine with a new method to space-efficiently evaluate a space-bounded machine over the PRG's image, utilizing the machine's own configuration. See Section 2.2 for a discussion about the techniques.

On the hardness assumptions

The combination of two assumptions – one asserting hardness for non-uniform machines, and an additional one that is either cryptographic or relies on hardness for non-deterministic non-uniform machines (as in [13]) – is in line with previous works in the area (see [11, 12]). However, previous works focused on time bounded algorithms, whereas the space bounded model turns out to be more subtle, posing several additional challenges. Thus, the particular hardness assumptions that we use above are more specialized. Let us elaborate.

First, note that the hypothesized lower bound in Assumption 1 is against space-bounded Turing machines (with advice). One could have hoped for a hardness assumption that is even closer to Theorem 1, namely, for *read-once branching programs* (or for $\mathbf{TC}^0$ circuits with oracle access to such programs). In the technical section we show that Assumption 1 can indeed be relaxed to a seemingly weaker, branching programs based assumption, which is a bit more involved to state (see [14, Section 5.3.2] for details).

[4] That is, for any constants η and k we have a PRG $C^{\mathsf{cry}}\colon \{0,1\}^{n^\eta} \to \{0,1\}^{n^k}$ computable in space $O(\log(n^\eta)+\log\log(n^k))$ (see [14, Section 3.1]).

Secondly, our cryptographic PRG is not just an arbitrary one, but has to be logspace-computable. However, as mentioned above, we propose Goldreich's PRG [16] as a natural and well-studied candidate, that works as follows. Let $\Gamma\colon [n^C] \times [d] \to [n^\eta]$ be the neighbor function of a suitable *lossless expander*, and let $P\colon \{0,1\}^d \to \{0,1\}$ be a predicate. Then, given $s \in \{0,1\}^{n^\eta}$ and $i \in [n^C]$, we define

$$G^{\mathsf{exp}}(s)_i = P\left(s{\restriction_{\Gamma(i)}}\right),$$

where $s{\restriction_{\Gamma(i)}}$ is the restriction of s to the set of right-neighbors of i the lossless expander. For Γ, we use an explicit, space-efficient expander [22, 29]. We further discuss Goldreich's PRG and its security, including possible choices for P, in [14, Section 5.3.1].

1.3 Non black-box derandomization with minimal footprint

Next, we turn to minimal-footprint derandomization under *uniform* hardness assumptions. Roughly speaking, we assume the existence of a function computable in space $(C+1)\cdot n$ that cannot be probabilistically "compressed" (even in slightly larger space) into a small Turing machine that uses only $C \cdot n$ space and computes the function. Formally:

▶ **Definition 1.** *We say that $P \in \{0,1\}^\star$ is an* S-space compressed version of $f \in \{0,1\}^\star$ *if P is a description, of length $\sqrt{|f|}$, of a Turing machine M that satisfies the following: On input $x \in [|f|]$, the machine M runs in space $S(\log(|f|))$ and outputs f_x.*

▶ **Assumption 3.** *For a sufficiently large constant C, there exists a function $f\colon \{0,1\}^\star \to \{0,1\}^\star$ mapping n bits to n^2 bits, that is computable in space $(C+1)\cdot \log n$, and satisfies the following. For every probabilistic algorithm R running in space $C \cdot \log n + O(\log n)$*[5] *there are at most finitely many $x \in \{0,1\}^\star$ for which*

$$\Pr\left[R(x) \text{ prints a } (C\cdot\log n)\text{-space compressed version of } f(x)\right] \geq \frac{2}{3}.$$

Again, similarly to our comments after Assumption 1, the precise difference of $C+1$ vs. C is not crucial (i.e., we can use $C+k$ vs. C for a fixed small universal k), and moreover the precise "amount of compression" can also be relaxed (e.g., compressing to $|f|^{0.01}$ instead of to $\sqrt{|f|}$); see [14, Section 6] for the precise details.

▶ **Theorem 3** (see also [14, Theorem 6.5])**.** *Suppose that Assumption 3 and Assumption 2 hold. Then, for any nice space function S, we have that*

$$\mathbf{BPSPACE}[S] \in \mathbf{DSPACE}\left[\left(2 + \frac{c}{C}\right)S\right],$$

where $c > 1$ is some fixed universal constant.

The derandomization algorithm in Theorem 3 does not rely on a pseudorandom generator, but instead works in a "non black-box" way that depends on the *input*. This follows an approach in a recent line of works initiated in [10] (with origins dating back to [18, 17]). As in previous works, the underlying hardness-to-randomness tradeoff is *instance-wise*, in the sense that for every space-S machine M and any *fixed input* x, if a certain machine $R_M(x)$ fails to print a compressed version of $f(x)$, then the deterministic simulation of M succeeds at the particular input x.

[5] The constant hidden in the $O()$ does not depend on C.

On the hardness assumption

The hardness assumption in Theorem 3 is different than the one in Theorem 2, but the conclusion is identical. This lends additional support for the possibility of derandomization with small footprint. Moreover, assuming hardness only for uniform algorithms (as in Theorem 3) is preferable, and the notion of hardness on all but finitely many inputs is necessary for derandomization and was used in several recent works (see, e.g., [10, 32, 33]).

Nevertheless, the particular notion of hardness of *compressing* $f(x)$ is non standard, and we elaborate on it. Recall that, by Kolmogorov-complexity-type arguments, almost all strings do not have *any* concise representation, let alone one that represents a space-bounded machine. (In particular, since such a representation does not exist, then certainly it is impossible to efficiently find it as in Assumption 3.) The crux of the assumption is that such representations are infeasible to find even for the outputs of the efficiently-computable function $f(x)$. We also note that an assumption reminiscent of "hardness of compressing $f(x)$ on all but finitely many inputs x" was recently used to *characterize time-bounded derandomization* (i.e., it is equivalent to the statement $\mathbf{prBPP} = \mathbf{prP}$); see [32] for precise details.

Lastly, since the underlying hardness-vs.-randomness tradeoff is instance-wise, the statement of Theorem 3 is robust, in the following sense: If the hardness holds not on all n-bit inputs, but rather only on $1 - \mu(n)$ fraction of the n-bit inputs over some distribution $\mathbf{x}_n$, then the derandomization succeeds with precisely the same probability and over the same distribution. Further details appear in [14, Section 6].

1.4 Catalytic computation towards an even smaller footprint

In the model of *catalytic computation*, introduced by Buhrman et al. [6], we enrich the space-bounded model with an *auxiliary memory*, that initially already stores some data. While we are allowed to use the auxiliary memory for *our* computation, in addition to the standard work tape, the auxiliary memory needs to be restored to its original content after use. Can such a seemingly restrictive usage be useful for computation? The work of [6] and followup works showed that it is indeed the case. Here, we give a possible *application* of catalytic space to derandomization, a connection that as far as we know, was not suggested before.

Suppose that our hard language from Assumption 1 can be computed catalytically, that is, most of the space used to compute it can be eventually restored. More concretely, consider the following assumption:

▶ **Assumption 4.** *The language from Assumption 1 is computable in space n using additional $C \cdot n$ auxiliary catalytic space.*

Then, we can show:

▶ **Theorem 4.** *Suppose that Assumption 4 and Assumption 2 hold. Then, for any nice space function S, we have that*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[\left(1 + \frac{c}{C}\right) S\right]$$

for some fixed universal constant c.

The same conclusion holds when adapting Assumption 3 in similar fashion.

Theorem 4 brings us tantalizingly close to derandomization without added memory footprint. Interestingly, the regime of parameters in Assumption 4, where the work space is only a small constant fraction of the catalytic space, has not been studied in the catalytic computation literature. (So far, the focus has been on the particular case where the catalytic space is exponential in the working space.) We thus view Assumption 3 also as a motivation to study catalytic computation in other regimes of parameters, which are useful for the study of derandomization.

2 Technical Overview

In Section 2.1 we describe the proof of Theorem 1, the main component of which (a new error-correcting code – see Theorem 2) will be used in the subsequent proofs. Then, in Section 2.2 we describe the proofs of Theorem 2 and Theorem 4. In particular, we show how to eliminate derandomization overheads, using a new algorithmic idea for derandomization, a particular type of cryptographic PRG, and an assumption about catalytic space. The proof of Theorem 3 is described in Section 2.3, and requires the strengthening of all the components described in Sections 2.1 and 2.2.

2.1 Warm-up: Hardness amplification for TC^0 circuits in linear space

The proof of Theorem 1 relies on the standard hardness-vs.-randomness approach, following [37, 26, 43]: Given an input $x \in \{0,1\}^n$ the derandomization algorithm first computes the truth-table $f \in \{0,1\}^{\mathrm{poly}(n)}$ of the hard function (on input length $O(\log n)$); then it transforms f into a truth-table $\bar{f} \in \{0,1\}^{\mathrm{poly}(|f|)}$ of a function that is hard on average, using a locally list-decodable error-correcting code; and finally it uses the Nisan–Wigderson generator to transform $\bar{f}$ into pseudorandom strings on which the probabilistic machine is evaluated with input x (see, e.g., [15, Chapters 7, 8], [4, Chapters 19, 20]).

The bottleneck in this approach is the worst-case to average-case reduction underlying the transformation of f to $\bar{f}$. To prove that the derandomization works for logspace machines, it suffices for $\bar{f}$ to be hard on $1/2 - 2^{-\varepsilon\cdot m}$ fraction of its inputs for ROBPs of width $2^{\varepsilon\cdot m}$, for some $\varepsilon > 0$ and where $m = \log(|\bar{f}|)$.[6] In order to deduce this conclusion using the standard argument of [43], we need to assume that f itself is hard (in the worst-case) for $\mathcal{C}$-procedures with oracle access to ROBPs of linear width, where $\mathcal{C}$ is the complexity of the local list-decoding algorithm of the code.

Loosely speaking, to decode from distance $1/2 - \delta$ (and deduce hardness on $1/2 - \delta$ fraction of the inputs), the procedure $\mathcal{C}$ needs to be able to compute the majority function (on $\Theta(1/\delta)$ bits, which in our setting would be $\Theta(2^{\varepsilon\cdot m})$ bits; see [19]).[7] Unfortunately, even when allowing $\mathcal{C} = \mathbf{TC}^0$, the best known decoder, from [19], only handles $\delta = 2^{-\sqrt{m}}$, which is too large for us. The codes that *are* typically used for hardness amplification with $\delta = 2^{-\varepsilon\cdot m}$ (i.e., the ones from [26, 43]) are not known to be locally list-decodable in complexity as low as $\mathbf{TC}^0$.[8]

The key observation allowing us to bridge this gap is that for our application of hardness amplification, we do not have to insist on the $\mathbf{TC}^0$ circuit being of size $\mathrm{poly}(\ell)$, where $\ell = \log(|f|)$, as in the standard setting of local coding. In fact, in our setting we can allow a circuit of size $2^{\varepsilon\cdot\ell}$. Given this relaxation, we construct the following suitable code.

[6] This is actually an over-simplification, and what we actually need is for $\bar{f}$ to be hard on $1/2 - 2^{-\varepsilon\cdot m}$ fraction of the its inputs for $\mathbf{AC}^0$ circuits that have oracle access to an ROBPs of width $2^{\varepsilon\cdot m}$ (this follows from the standard reconstruction argument of [37]). In this high-level overview we ignore the $\mathbf{AC}^0$ overhead, for simplicity of presentation.

[7] In fact, a similar statement holds for any "black-box" worst-case to average-case hardness amplification [46, 41, 20].

[8] The bottleneck in both cases is local list-decoding of the Reed-Muller code; see [10] for a recent construction of a decoder in logspace-uniform $\mathbf{NC}$.

▶ **Theorem 2** (see also [14, Theorem 4.2]). *There exists a universal constant $c > 1$ such that for any constant $\gamma \in (0, 1)$ the following holds. For every $k \in \mathbb{N}$ and $\varepsilon > 0$, there exists a logspace-computable code $\mathcal{C} \colon \{0, 1\}^k \to \{0, 1\}^n$, for $n = (\frac{k}{\varepsilon})^{c/\gamma}$, that is locally list decodable from $1/2 + \varepsilon$ fraction of agreement by constant-depth threshold circuits of size $k^\gamma \cdot (1/\varepsilon)^c$.*

At a high level, the proof of Theorem 2 (wherein one should think of $k = 2^m$) combines three known code constructions:

1. A small modification of the code of [19], which uniquely decodes from agreement $1 - \frac{1}{25}$ using $\mathbf{TC}^0$ circuits;
2. the derandomized direct-product code of [26], which $(1 - \frac{1}{25})$-approximately list-decodes from agreement $\eta = 2^{-O(\varepsilon \cdot m)}$ using a $\mathbf{TC}^0$ circuit of size $2^{O(\varepsilon \cdot m)}$; and,
3. the Hadamard code, which we concatenate with the direct product code and is list-decodable from agreement $\frac{1}{2} + 2^{-\varepsilon \cdot m}$ by $\mathbf{TC}^0$ circuits of size $\text{poly}(m)$.

As a corollary, we obtain a $\mathbf{TC}^0$-computable worst-case to average-case reduction for computing functions in $\mathbf{DSPACE}[O(n)]$; see [14, Corollary 4.1]. This reduction handles the "high-end" parameter regime, which previous reductions for functions in $\mathbf{DSPACE}[O(n)]$ did not handle (see [42, 21, 19]), and is incomparable to reductions computable by probabilistic (uniform) algorithms [45, 9].

The decoder's complexity

For our results we crucially rely on the fact that the decoder can be implemented in $\mathbf{TC}^0$ (e.g., when deducing black-box derandomization from hardness for $\mathbf{TC}^0$ circuits with oracle access to branching programs, or when deducing non-black-box derandomization). We suspect that it is possible to construct a code with weaker guarantees – namely, a logspace decoder, rather than a $\mathbf{TC}^0$ decoder – using simpler techniques (i.e., replacing the "outer" code of [19] by more classical tools).

2.2 Derandomization with minimal footprint using PRGs

Let $S = C \cdot \log n$ denote the space complexity of the machine M we wish to derandomize (the result for arbitrary S will follow from padding). At a high-level, our construction follows an approach first introduced in [11], which composes two "low-cost" PRGs:

- An inner PRG that stretches $O(\log n)$ bits to n^η bits for some tiny constant $\eta > 0$, and,
- an outer PRG that stretches n^η bits to n^C bits.

Specifically, as in [10], we take the inner PRG, denoted by NW, to be an appropriately parameterized Nisan-Wigderson PRG [37] with a hard truth-table $f \in \{0, 1\}^{n^2}$, and the outer PRG, denoted by G^{cry}, to be one that relies on a cryptographic assumption.

Unfortunately, materializing this approach in the current setting turns out to be significantly more subtle than in [11]. To see this, observe that the final computation iterates over seeds $s \in \{0, 1\}^{O(\log n)}$, and for each s computes

$$M(x, G^{\mathsf{cry}}(\mathsf{NW}^f(s))),$$

where f is the truth-table of the hypothesized hard function. To compute this using space-efficient composition, we use the following chain of simulations:

1. Simulate $M(x, \cdot)$, and whenever it queries its second input –
2. Simulate G^{cry}, and whenever it queries its input –
3. Simulate $\mathsf{NW}^f(s)$, and whenever it queries f –
4. Compute the corresponding bit of f.

Recall that, using space-efficient (emulative) composition, the complexity of the final construction is additive in the space complexity of each of its components, plus additional overheads that are logarithmic in the output length of each component. (The latter logarithmic overhead is caused by the fact that we are simulating a virtual input head for each component. See [14, Proposition 3.2].) A naive implementation of this approach yields space complexity of $3S + \mathsf{Space}(G^{\mathsf{cry}}) + \mathsf{Space}(\mathsf{NW})$, where $\mathsf{Space}(\cdot)$ denotes the space complexity of the corresponding algorithm, and we ignore factors of the form $c \cdot \log n$ where $c > 1$ is a universal constant that doesn't depend on S.

A more efficient derandomization

Our first observation is that the standard way of derandomizing probabilistic space-S machine is wasteful. There, we think of the probabilistic machine as reading a tape of random bits, sequentially; and when simulating it deterministically, we keep track of a counter $i \in [2^S]$, and whenever the machine wishes to read a random bit, we answer using the i-th bit in the pseudorandom output of the generator, and update $i \leftarrow i + 1$.

It might (mistakenly) seem that using a dedicated counter is necessary, because we must ensure that the machine reads each bit in the random (or pseudorandom) sequence exactly once. However, this intuition turns out to be false: Instead of keeping a dedicated counter $i \in [2^S]$, *we can use the machine's own configuration as a counter.* Specifically, recall that at each step the machine has some configuration $\sigma \in \{0,1\}^S$ describing the contents of its work tapes, its current state, and the locations of its heads.[9] Moreover, since for every input x and fixed sequence r of coins, the execution of $M(x,r)$ halts, any configuration $\sigma \in \{0,1\}^S$ is encountered *at most once* during the execution of M (see [14, Claim 3.3]). Thus, we consider the following machine $\bar{M}$, which simulates M using oracle access to a sequence of random coins but without the overhead of keeping a counter:

> Simulate M, and whenever M tries to flip a random coin, access the sequence of random coins at location σ, where σ is M's current configuration.

Since the functionality of $\bar{M}$ and of M at any input x, with uniform coins, is identical, it suffices to faithfully simulate $\bar{M}$ with pseudorandom coins.

At this point the space complexity of the derandomization is essentially

$$2S + \mathsf{Space}(G^{\mathsf{cry}}) + \mathsf{Space}(\mathsf{NW}).$$

Since NW maps a truth-table f of length n^2 to pseudorandom strings of length n^η, it can be computed in space $c' \cdot \log n$ for a universal $c' > 1$ (see [14, Section 5.1]). Thus, our last step is to bound the space complexity of G^{cry}.

We do so by relying on a specific PRG whose space complexity is logarithmic in its input length n^η and sub-logarithmic in its output length n^C. A natural candidate for such a PRG arises from the "cryptography in $\mathbf{NC}^0$" literature (see, e.g., [2, 27, 1, 38]), and in particular we can use Goldreich's PRG [16]. The latter PRG relies on a bipartite lossless expander $\Gamma\colon [n^C] \times [d] \to [n^\eta]$ with a small left-degree d, and on a predicate $P\colon \{0,1\}^d \to \{0,1\}$. For $s \in \{0,1\}^{n^\eta}$ and $i \in [n^C]$, the i-th output of $G^{\mathsf{cry}}(s)$ is

$$P\left(s_{\Gamma(i,1)}, \ldots, s_{\Gamma(i,d)}\right),$$

where $\Gamma(i,1), \ldots, \Gamma(i,d)$ are the d neighbors of i in the expander.

[9] Indeed, we count the location of the heads and the state in the configuration of the machine, and in fact assume that they are written on dedicated worktapes; see [14, Section 3] for the precise details.

In the cryptography literature, the graph is often taken to be a random one, but in our setting we need a lossless expander whose neighbor function is computable in space $c'' \cdot \log n$, and also a predicate known to withstand existing attacks that is computable in space $c'' \cdot \log n$, where in both cases $c'' > 1$ is a universal constant. We use the recent expander construction of Kalev and Ta-Shma [29], whose degree is $d = \text{polylog}(n)$ and whose neighbor function is computable in space $O(\text{loglog} n)$, and a predicate introduced by Applebaum and Raykov [3] that is computable in space $O(\log d) = O(\text{loglog} n)$. See [14, Section 5.3.1] for further details.

This brings the complexity of the deterministic simulation of M on each particular seed to be $2S + c \cdot \log n$ for some universal constant c, and after enumerating over all seeds and taking the majority output, the space complexity increases only by an additive factor of, say, at most $c \cdot \log n$.

The reconstruction argument

We prove that the derandomization works using a reconstruction argument. Specifically, in the derandomization, we instantiate the NW generator with the code from Theorem 2, and rely on the reconstruction argument of NW and on the local list-decoding algorithm of the code to transform any ROBP distinguisher for the PRG (which arises from the computation of the space-S machine M on a fixed input x) into an efficient procedure that computes f.

The details of the reconstruction procedure appear in [14, Theorem 5.1], so let us only highlight the important points in the argument. First, our distinguisher is actually derived from $\bar{M}$, the machine that reads bits according to its configuration. Secondly, by a standard analysis of PRG composition, the distinguisher for NW^f is not just the ROBP derived from $\bar{M}$ and x, but actually the composed procedure

$$D(r) = \bar{M}^{G^{\mathsf{cry}}(r)}(x).$$

This increases the complexity of the distinguisher from a simple ROBP to a bounded-space machine. Lastly, while the machine implementing D uses space at least $S = C \cdot \log n$, the amount of non-uniform advice that it uses is much smaller than $2^S = n^C$. Specifically, it uses only $|x| = n = 2^{\log(|f|)/2}$ bits of advice.

To sum up, if the derandomization fails on some input x, then there exists a $\mathbf{TC}^0$ circuit C of size n^{ε}, and a function D that is computable in space $\approx C \cdot \log n$ with n bits of advice,[10] such that $C^D(i) = f_i$ for all $i \in [n^2]$. Finally, using the fact that C is a $\mathbf{TC}^0$ circuit, we show that C^D itself can be computed by a TM with advice, whose space complexity is only slightly larger than the space complexity of D. This contradicts the hardness of f.

Obtaining a sub-double space overhead

The derandomization above takes $2S + c \cdot \log n$ space, where the increase from S to $2S$ is caused by the space complexity of computing f. Indeed, it seems unavoidable that we will need space larger than S to compute f, because we are assuming that it is hard for algorithms running in space S.

The key observation to reducing this overhead is that if f is computable in *catalytic* space S, we can roughly use the existing used worktape cells – which, at any point in time, contain the current configuration of the derandomized machine – in order to compute each query of

[10] The precise complexity of D is $(C + 1 + \varepsilon) \cdot \log n$, but in the high-level overview we ignore these minor overheads, for simplicity of presentation.

NW to f. Specifically, whenever the derandomization machine queries f at location $i \in [n^2]$, we compute f_i using (mostly) the existing space in a catalytic way. After the computation of the bit f_i ends, the original content of the worktapes is restored. The key point is that since the configuration of the machine does not change during the computation of $i \mapsto f_i$, nor is this computation dependent on the configuration in any way, the correctness of the procedure is maintained.

Thus, under this strengthened hypothesis that f is computable in small catalytic space, the final space complexity of the derandomization algorithm is just $S + c \cdot \log n$.

An alternative to Assumption 1

Recall that $D(r)$ above can be computed by a bounded-space machine that uses $|x|$ bits of advice. While indeed D is not a read-*once* branching program, the computation of $D(r)$ is *oblivious*. Namely, computing C^{cry} can be done by a bounded-width branching program that at each layer queries several locations of r, however these locations are determined only by the underlying expander Γ. Thus, we can model C^D as a $\mathbf{TC}^0$ circuit with (non-adaptive) oracle access to branching programs of the aforementioned type. Moreover, we will later see that the $\mathbf{TC}^0$ circuit can be space-efficiently generated using a short advice. The formal assumption is given in [14, Assumption 5.10], and can replace Assumption 1 for both the double and sub-double overhead results.

2.3 Non black-box derandomization with minimal memory footprint

Set $S = C \cdot \log n$ and recall that we wish to obtain the same conclusions for $\mathbf{BPSPACE}[S]$ as in Section 2.2, but from different assumptions. Specifically, we assume the existence of a function $f\colon \{0,1\}^n \to \{0,1\}^{n^2}$ computable in space $S' = S + O(\log n)$ such that for every probabilistic algorithm R running in space $S_R = S' + O(\log n)$, and every $x \in \{0,1\}^n$, the algorithm $R(x)$ fails to print a *compressed* version of $f(x)$ (except with small probability). In this context, a compressed version means a Turing machine of description size $O(n) = O(\sqrt{|f(x)|})$ that runs in space roughly $S + \log n < S'$.

Following ideas from [10, 32], we will construct a *targeted* PRG, which is an algorithm that maps any input x to a set of pseudorandom strings that will fool the machine M with this particular input x. As in those previous works, our targeted PRG is based on the Nisan–Wigderson generator, and we analyze it using an *instance-wise* hardness vs. randomness tradeoff. Specifically, we show that if the derandomization fails on an input x, then a probabilistic space-S_R machine R succeeds in mapping the same fixed x to a compressed version of $f(x)$. This yields Theorem 3, and also the more general version mentioned after the theorem's statement: For every distribution $\mathbf{x}$ over the inputs, if the probability over $x \sim \mathbf{x}$ that R fails to print compressed version of $f(x)$ is $1-\mu$, then the derandomization succeeds on $1-\mu$ of the inputs $x \sim \mathbf{x}$.

The derandomization itself is similar to the one from Section 2.2, with a minor difference that is nevertheless crucial. Instead of instantiating NW with f that is the truth-table of a hypothesized hard function, we instantiate NW with $f = f(x)$ obtained from the input x. That is, we compute the majority, over seeds $s \in \{0,1\}^{O(\log n)}$, of

$$\bar{M}^{G^{\mathsf{cry}}(\mathsf{NW}^{f(x)}(s))}(x) \; .$$

Note that the complexity of the derandomization algorithm is essentially identical to that of the algorithm from Section 2.2. Thus, the only question is – why does it work?

Analysis

The main argument underlying Theorem 3 is proving that there exists a space-S_R algorithm $R = R_M$ satisfying the following: For any fixed $x \in \{0,1\}^n$, when NW is instantiated with the code from Theorem 2, if $\mathsf{NW}^{f(x)}$ does not fool M, then $R(x)$ prints a compressed version of $f(x)$.

The intuition for why this might be possible dates back to [24], who showed that the reconstruction algorithm of NW, which maps a distinguisher to a small circuit for the hard truth-table, can be made *uniform* – as long as it is allowed to make queries to the hard truth-table. Recall that in our setting, the algorithm R explicitly gets the input x, and we are allowing R to run in space that is slightly larger than the space complexity of computing $f(x)$. Therefore, R can simulate the reconstruction algorithm, and whenever the latter queries an index i of $f(x)$, the algorithm R simply computes $f(x)$ and returns the relevant bit.

The main technical challenge that we are faced with is making the algorithm R not only uniform, but also a *small-space algorithm*, and doing so when the underlying *code for hardness amplification* is the one from Theorem 2. The resulting statement appears in [14, Theorem 6.1], and its proof is the most technically subtle argument in this paper. In the rest of the section we describe the proof, at a high-level.

Low-space uniform reconstruction and decoding

We first strengthen the analysis of the code $\mathcal{C}$ from Theorem 2, to show that not only is it locally list-decodable, but that it also has a *probabilistic space-$O(\log n)$ uniform decoder, which does not need non-uniform advice* (but rather uses queries to the corrupt codeword). The algorithm R will answer this decoder's queries to the corrupt codeword $\mathcal{C}(f(x))$ by computing $f(x)$ and then $\mathcal{C}$, which it can do in its allotted space. We compose this uniform decoder for $\mathcal{C}$ with a space-$O(\log n)$ reconstruction algorithm for NW.

We stress that the two algorithms underlying R – the decoder, and the NW reconstruction – run in space $O(\log n)$, but print a procedure of description size $n^{\Omega(1)}$. Thus, not only do the two algorithms need to print a description of a procedure without remembering most of the functionality of the machine that they printed so far – but also the algorithms cannot even *evaluate* the procedures that they print.

The key observation is that in both cases, the decoding/reconstruction prints a procedure *almost all of which is a large, static, truth-table.* To see this, let us focus for simplicity on the NW reconstruction algorithm.[11] Recall that this algorithm implements very simple functionality, which can be described by a logspace-uniform constant-depth circuit of size polylog(n), and that is hard-wired with "static" information of size n^{ε} that is mostly obtained from queries to $\mathcal{C}(f(x))$.[12] With some low-level care, we can design an algorithm that queries $\mathcal{C}(f(x))$ and prints a machine that implements that functionality, and has states encoding the foregoing static information. Thus, the algorithm which prints the machine does not need to remember static information that is already printed.

A related complication arises because both the decoding algorithm of $\mathcal{C}$ and the reconstruction algorithm of NW actually succeed only with small probability. The standard approach (e.g., in [24, 10, 32]) is for R to use queries to $\mathcal{C}(f(x))$ to estimate the agreement of the

[11] The computational bottleneck in the decoder for $\mathcal{C}$ is the decoder for the derandomized direct product code of [26], which acts in a similar way to the reconstruction of NW. Thus, we use similar ideas to handle both the reconstruction of NW and the decoder of $\mathcal{C}$.

[12] The information consists of an index i (used for a hybrid argument), of a combinatorial design, of values for the seed outside the i-th set in the design, and of n^{η} partial truth-tables.

procedure that it outputs with $\mathcal{C}(f(x))$, repeating the experiment until it gets a procedure with sufficiently large agreement. Since in our case R cannot evaluate the procedure that it prints, it cannot take this approach. Instead, R prints a procedure that performs this "success amplification" functionality by itself. We leave the details to the technical section.

Composing the two algorithms

The description above refers vaguely to "the procedure" that R print, and being more accurate, this procedure is a $\mathbf{TC}^0$ circuit C of size n^{ε} making queries to a space-S machine D that uses n bits of advice. This is not enough, since our goal is for R to print a single Turing machine of description size $O(\sqrt{|f(x)|}) = O(n)$ running in space $S + \log n < S'$.

Bridging this gap requires more care in composing the two algorithms. Specifically, our algorithm R prints a machine whose states encode the circuit C, and that implements the standard DFS-style emulation of $\mathbf{NC}^1 \supseteq \mathbf{TC}^0$ circuits in logspace, while reading the description of the hard-coded C out of its own states. The space overhead of the emulation itself is $O(\log |C|) = O(\log(n^{\varepsilon}))$, and the machine also needs to compute the values of the gates along each DFS path. In particular, this means that we need to ensure that each path contains at most one oracle call to D (otherwise the machine's space complexity will be larger than $2S$). For this purpose, in our strengthened analysis of $\mathcal{C}$ we ensure that its decoding procedure only makes *non-adaptive queries*. This allows us to bound the space complexity of the machine that R prints by $S + O(\varepsilon \cdot \log n) \leq S + \log n$, as desired.

References

1 Benny Applebaum. *Cryptography in constant parallel time*. Information Security and Cryptography. Springer, 2014.

2 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36(4):845–888, 2006.

3 Benny Applebaum and Pavel Raykov. Fast pseudorandom functions based on expander graphs. In *Theory of Cryptography. Part I*, volume 9985, pages 27–56. Springer, 2016.

4 Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, Cambridge, 2009.

5 Allan Borodin, Stephen Cook, and Nicholas Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58(1-3):113–136, 1983.

6 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014.

7 Lijie Chen. Non-deterministic quasi-polynomial time is average-case hard for *ACC* circuits. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2019.

8 Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020.

9 Lijie Chen, Ron D. Rothblum, and Roei Tell. Unstructured hardness to average-case randomness. In *Proc. 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 429–437. IEEE, 2022.

10 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.

11 Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. In *Proc. 53st Annual ACM Symposium on Theory of Computing (STOC)*, 2021.

12 Lijie Chen and Roei Tell. When arthur has neither random coins nor time to spare: Superfast derandomization of proof systems. *Electronic Colloquium on Computational Complexity: ECCC*, 2022.

13 Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. In *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020.

14 Dean Doron and Roei Tell. Derandomization with minimal memory footprint. *Electronic Colloquium on Computational Complexity: ECCC*, 30:036, 2023.

15 Oded Goldreich. *Computational complexity: A conceptual perspective*. Cambridge University Press, New York, NY, USA, 2008.

16 Oded Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 76–87. Springer, 2011.

17 Oded Goldreich. In a world of $P = BPP$. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*, pages 191–232. Springer, 2011.

18 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)*, pages 209–223, 2002.

19 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 440–449, 2007.

20 Aryeh Grinberg, Ronen Shaltiel, and Emanuele Viola. Indistinguishability by adaptive procedures with advice, and lower bounds on hardness amplification proofs. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 956–966, 2018.

21 Venkatesan Guruswami and Valentine Kabanets. Hardness amplification via space-efficient direct products. *Computational Complexity*, 17(4):475–500, 2008.

22 Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *Journal of the ACM*, 56(4):Art. 20, 34, 2009.

23 William M. Hoza. Better pseudodistributions and derandomization for space-bounded computation. In *Proc. 25th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 28:1–28:23, 2021.

24 Russel Impagliazzo and Avi Wigderson. Randomness vs. time: Derandomization under a uniform assumption. *Journal of Computer and System Sciences*, 63(4):672–688, 2001.

25 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

26 Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 220–229. Association for Computing Machinery, 1997.

27 Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 433–442, 2008.

28 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

29 Itay Kalev and Amnon Ta-Shma. Unbalanced expanders from multiplicity codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*, volume 245 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

30 Jeff Kinne, Dieter van Melkebeek, and Ronen Shaltiel. Pseudorandom generators, typically-correct derandomization, and circuit lower bounds. *Computational Complexity*, 21(1):3–61, 2012.

31 Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

32 Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of Levin-Kolmogorov complexity. In *37th Computational Complexity Conference (CCC 2022)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

33 Yanyi Liu and Rafael Pass. Leakage-resilient hardness vs. randomness. *Electronic Colloquium on Computational Complexity: ECCC*, 30:113, 2022.

34 Cody Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: An easy witness lemma for NP and NQP. In *Proc. 50th Annual ACM Symposium on Theory of Computing (STOC)*, 2018.

35 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

36 Noam Nisan. $RL \subseteq SC$. *Computational Complexity*, 4(1):1–11, 1994.

37 Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

38 Hanlin Ren and Rahul Santhanam. Hardness of KT characterizes parallel cryptography. In *36th Computational Complexity Conference (CCC 2021)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:58. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

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

40 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

41 Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 589–598, 2008.

42 Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *toit*, 42(6):1723–1731, 1996. Codes and complexity.

43 Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

44 Roei Tell. Proving that $prBPP = P$ is as hard as proving that “almost NP” is not contained in $P/poly$. *Information Processing Letters*, 152:105841, 2019.

45 Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

46 Emanuele Viola. Hardness vs. randomness within alternating time. In *Proc. 18th Annual IEEE Conference on Computational Complexity (CCC)*, pages 53–69, 2003.

47 R. Ryan Williams. Non-uniform acc circuit lower bounds. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 115–125, 2011.