When Connectivity Is Hard, Random Walks Are Easy with Non-determinism

Dean Doron
deand@bgu.ac.il
Ben-Gurion University
Beer Sheba, Israel

Edward Pyne
epyne@mit.edu
MIT
Cambridge, United States

Roei Tell
roei@cs.toronto.edu
University of Toronto
Toronto, Canada

Ryan Williams
rrw@mit.edu
MIT
Cambridge, United States

Abstract

Two fundamental problems on directed graphs are to decide s-t connectivity, and to estimate the behavior of random walks. Currently, there is no known algorithm for s-t connectivity running in polynomial time and $n^{o(1)}$ space, and no known algorithm for estimating the n-step random walk matrix running in non-deterministic logspace.

We show that for every directed graph, at least one of these problems is solvable in time and space that significantly improve on the respective state-of-the-art. In particular, there is a pair of algorithms A_1 and A_2 such that for every graph G, either:

(1) $A_1(G)$ outputs the transitive closure of G in polynomial time and polylogarithmic space.
(2) $A_2(G)$ outputs an approximation of the n-step random walk matrix of G in non-deterministic logspace.

As one application, we show surprisingly tight win-win results for space-bounded complexity. For example, for certain parameter regimes, either Savitch's theorem can be non-trivially sped up, or randomized space can be almost completely derandomized.

We also apply our techniques to significantly weaken the assumptions required to derandomize space-bounded computation, and to make non-deterministic space-bounded computation unambiguous. Specifically, we deduce such conclusions from lower bounds against uniform circuits of polynomial size, which is an exponential improvement on the required hardness in previous works (Doron–Pyne–Tell STOC 2024, Li–Pyne–Tell FOCS 2024). We further show similar results for minimal-memory derandomization (Doron–Tell CCC 2024).

To prove these results, we substantially improve the array of technical tools introduced in recent years for studying hardness-vs.-randomness for bounded-space computation. In particular, we develop derandomized distinguish-to-predict transformations for new types of distinguishers (corresponding to compositions of PRGs with weak distinguishers), we construct a derandomized logspace reconstruction procedure for the Shaltiel–Umans generator (JACM 2005) that can compress hard truth-tables to polylogarithmic size, and we design a version of the Chen–Tell generator (FOCS 2021) that is particularly suitable for the space-bounded setting.

CCS Concepts

• **Theory of computation** → **Complexity classes**; **Pseudorandomness and derandomization**; **Random walks and Markov chains**; **Graph algorithms analysis**.

Keywords

Derandomization, Complexity, Random Walks, Connectivity

ACM Reference Format:
Dean Doron, Edward Pyne, Roei Tell, and Ryan Williams. 2025. When Connectivity Is Hard, Random Walks Are Easy with Non-determinism. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3717823.3718303

1 Introduction

How much time and space is necessary to simulate randomized small-space algorithms? To simulate non-deterministic small-space algorithms? These are two of the most well-studied questions in space complexity, with particularly clean complete problems:

(1) For $\mathbf{BPL} = \mathbf{BPSPACE}[O(\log n)]$, the corresponding problem is to estimate n-step random walk probabilities on an n-vertex graph, with small additive error.
(2) For $\mathbf{NL} = \mathbf{NSPACE}[O(\log n)]$, the problem is to decide s-t connectivity on an n-vertex graph.

At the moment, we do not know how to solve either problem in polynomial time and only logarithmic space. For problem (1), it is widely conjectured that such an algorithm exists (i.e., that $\mathbf{BPL} = \mathbf{L}$, which follows from conjectured lower bounds [14, 15, 29]). However, the best known algorithms work either in super-polynomial time $2^{\log(n)^{3/2-o(1)}}$ and in space $\log(n)^{3/2-o(1)}$ [24, 42], or in polynomial time and in larger space $O(\log^2 n)$ [4, 33]. Moreover, it is not even known how to improve these algorithms using non-deterministic computation (e.g., we do not know whether $\mathbf{BPL} \subseteq \mathbf{NSPACE}[\log^{1.49} n]$).

For problem (2), there is seemingly no consensus on a widely believed conjecture, and the most important reference point is the classical algorithm of Savitch [44], which works in super-polynomial

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
STOC '25, Prague, Czechia
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718303

time $2^{\Theta(\log^2 n)}$ and in space $O(\log^2 n)$. A major open problem asks whether Savitch's algorithm can be improved. For example, already 30 years ago, Wigderson [52] asked whether an algorithm can achieve polynomial time and space $n^{1-\varepsilon}$ for some $\varepsilon > 0$: This problem is still wide open, and the best known polynomial-time algorithm uses nearly-linear space $n/2^{\Theta(\sqrt{\log n})}$ [3]. In fact, in a natural restricted model encompassing all known deterministic, randomized, and non-deterministic algorithms for directed (and undirected) connectivity,[1] there is a lower bound ruling out algorithms running in time $2^{\log^{1.99} n}$ and space $n^{0.99}$ [16].

1.1 Our Results, Part 1: A Pair of Algorithms

We prove that for every graph, *at least one* of these problems can be solved significantly more efficiently than previously known algorithms:

Theorem 1. *There are algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for every graph G on n vertices, one of the following holds:*

- *$\mathcal{A}_1(G)$ solves s-t connectivity in G in polynomial time and polylogarithmic space.*
- *$\mathcal{A}_2(G)$ estimates length-n random walk probabilities in G in non-deterministic logspace.*[2]

Moreover, both algorithms report if they fail to compute the desired answer, and do not exceed their resource bounds in any case.

We stress that the constructions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are explicit (i.e., these are specific algorithms, and their descriptions will be given in Section 2.1), and that provably, for *every graph*, at least one of the algorithms works (and both of them never return an incorrect answer). Moreover, these algorithms run in time and space that *significantly* improve on the respective state-of-the-art: The algorithm $\mathcal{A}_1$ runs in polynomial time and uses only polylogarithmic space (compared to $n/2^{\sqrt{\log n}}$ space [3]); and $\mathcal{A}_2$ uses only logarithmic space, alas it also uses non-determinism (i.e., it is akin to $\mathbf{BPL} \subseteq \mathbf{NL}$; in comparison, the algorithm of [23, 42] uses $\log(n)^{3/2-o(1)}$ space).

Application: Tight win-win results in space-bounded complexity. Since the pair of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ from Theorem 1 solve the complete problems for for **NL** and for **BPL**, respectively, we can use them to tightly connect the challenges of simulating **NL** and **BPL**. As one application, we leverage the pair of algorithms to show that either Savitch's theorem can be improved, or randomized space can be deterministically simulated *near-optimally*.

Theorem 2. *For every constant $\varepsilon > 0$, at least one of the following holds:*

- $\mathbf{NSPACE}[n] \subseteq i.o.\mathbf{TISP}\left[2^{O(n^{2-\varepsilon})}, n^{O(1)}\right]$.
- $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}\left[O(n^{1+\varepsilon})\right]$.

[1] Specifically, this is the Node-Named Jumping Automata on Graphs (NNJAG) model [12, 31, 37]. This model captures all known space-bounded directed and undirected connectivity algorithms, including Savitch, BFS, DFS, Immerman-Szelepcsényi [25, 47], Nisan et al. [35], Barnes et al. [3], Armoni et al. [2], and Reingold [40].

[2] The estimation is up to an additive $1/\mathrm{poly}(n)$ error. The algorithm runs in logspace, makes non-deterministic guesses, and either declares *fail* if the guess sequence is bad, or a single canonical matrix only depends on the graph G, or a special symbol $\perp$ indicating that $\mathcal{A}_2$ does not succeed on this input (in which case $\mathcal{A}_1$ succeeds on the input).

We stress that the second item in Theorem 2 does *not* use non-determinism (i.e., it is a scaled-up version of the statement $\mathbf{BPL} \subseteq \mathbf{SPACE}[(\log n)^{1+\varepsilon})]$), in contrast to $\mathcal{A}_2$ from Theorem 1. Theorem 2 is not an immediate corollary of Theorem 1, but it does use the techniques underlying the proof of the latter (i.e., a more general construction of a pair of algorithms).

Theorem 2 is particularly meaningful in two parameter regimes, corresponding to choices of $\varepsilon > 0$. Specifically, the following two instantiations assert that for each of the two problems we consider (i.e., s-t connectivity and estimating random walks), either we can non-trivially improve on the state-of-the-art for solving the problem, or we can *near-optimally* solve the other problem:

- With an arbitrarily small $\varepsilon > 0$: Either Savitch's algorithm can be non-trivially sped-up (i.e., replacing 2^{n^2} with $2^{n^{2-\varepsilon}}$), or we can near-optimally derandomize $\mathbf{BPSPACE}[n]$.
- With $\varepsilon = 0.49$: Either the frontier derandomization of Saks–Zhou can be non-trivially improved (i.e. $\mathbf{SPACE}[n^{1.5}]$ with $\mathbf{SPACE}[n^{1.49}]$), or Savitch's algorithm can be substantially sped up (i.e. replacing 2^{n^2} with $2^{n^{3/2+0.01}}$).

If we are willing to settle for non-deterministic simulation of **BPSPACE**, we can leverage Theorem 1 to connect *near-optimal* solutions to *both* problems (i.e., rather than connecting a slight improvement to the state-of-the-art for one problem to a near-optimal solution to the other problem). For example, either Savitch's Theorem can be optimally sped up, or we can optimally simulate probabilistic linear space using non-determinism:

Theorem 3 (informal). *It holds that either*

$$\mathbf{NSPACE}[n] \subseteq i.o.\mathbf{TISP}\left[2^{O(n)}, n^{O(1)}\right],$$

or $\mathbf{BPSPACE}[n] \subseteq \mathbf{NSPACE}[O(n)]$.

Interpretation. If one believes that $\mathbf{L} = \mathbf{NL}$ and $\mathbf{BPL} = \mathbf{L}$ (i.e., that we can solve s-t connectivity in logspace and estimate random walk probabilities in logspace), then Theorem 1 and the win-win results can be interpreted as concrete steps towards proving *both* statements. Alternatively, if one believes that (say) Savitch's algorithm cannot be sped up, then our results can be interpreted as showing that such a statement implies optimal derandomization. In any case, our results in this section provide a new algorithmic tool, and connect two fundamental problems.

1.2 Our Results, Part 2: Derandomization from Very Weak Hardness

One perspective on the win-win results above is that they convert hardness into randomness: Specifically, they deduce near-optimal derandomization of small space from hardness of improving Savitch's theorem (i.e., of solving s-t connectivity) by uniform, deterministic algorithms. We stress that typical results in hardness vs. randomness deduce derandomization from stronger assumptions (i.e., from hardness for non-uniform circuits or for probabilistic algorithms).[3]

[3] Only very recently, several works deduced derandomization of small-space computation from hardness for uniform, deterministic algrithms (see [14, 30, 39]). Jumping ahead, we build on these works and significantly develop the technical machinery introduced in them. For details, see Section 1.3.

From this perspective (i.e., if the goal is to deduce derandomization from weak hardness), we want to do better than Theorem 2 and Theorem 3, by deducing optimal derandomization – without non-determinism, and without an $n^{1+\varepsilon}$ space overhead. We are indeed able to do so.

Full derandomization of **BPSPACE** *from very weak hardness.* A classical result of Klivans and van Melkebeek [29] (following [36]) showed that **BPL** = **L** follows from sufficiently explicit lower bounds against exponential sized non-uniform circuits (c.f. [15]). Since these circuit lower bounds currently seem out of reach, a natural direction is to deduce derandomization from assumptions that are weak enough so that we hope to unconditionally prove them.

Recently, Doron, Pyne, and Tell [14] showed one such result, in which they deduced derandomization from lower bounds for exponential-sized circuits that can be printed by *uniform, space-bounded machines* (rather than non-uniform circuits). As they point out, proving such a lower bound seems significantly more tractable , since there are already known lower bounds against uniform circuits (see, e.g., Santhanam and Williams [43]).

To be more precise, in [14] they proved that $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[O(n)]$ follows from hardness of $\mathbf{SPACE}[n]$ for logspace-uniform oracle circuits of size $2^{\varepsilon n}$; that is, against exponential-sized circuits that can be printed in space $O(n)$. We deduce the same conclusion from lower bounds against uniform *polynomial-sized circuits* (equipped with an oracle that uses space εn).

Theorem 4. *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that* $\mathbf{SPACE}[n]$ *is hard for* $\mathbf{TISP}[2^{cn}, n^c]$*-uniform circuits of size n^c with oracle access to* $\mathbf{SPACE}[\varepsilon n]$.[4] *Then,* $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[O_\varepsilon(n)]$.

Theorem 4 represents a near-exponential improvement in the size of the circuits against which we need hardness, at the cost of relaxing the uniformity condition from $\mathbf{SPACE}[O(n)]$-uniformity to $\mathbf{TISP}[2^{O(n)}, n^{O(1)}]$-uniformity. The assumption in Theorem 4 strikes us as very weak, and plausibly provable: It asserts that there are N-bit strings printable in space $O(\log N)$ that cannot be deterministically compressed (in small time and space) to a circuit of size $\mathrm{polylog}(N)$ that can make oracle queries to space $\varepsilon \cdot \log(N)$.

Derandomization with minimal memory footprint. We also consider the question of derandomization with minimal memory overhead, which was introduced by Doron and Tell [15] (following [9, 13]). The classical conjecture **BPL** = **L** asserts that randomized space-S machines can be simulated in space $S' = C \cdot S$, for some (possibly large) constant $C > 1$. The more ambitious goal from [15] is to have S' as close as possible to S; for example, deduce simulation with $S' \approx 2S$ or even $S' \approx S$. Since we do not hope to show this unconditionally at the moment, the goal is to deduce it under the weakest possible assumptions.

We base minimal-memory derandomization on assumptions that are qualitatively weaker than those known to imply *standard* (i.e. not superfast) derandomization in the time-bounded setting. To see this, recall that the work of [15] deduced derandomization of randomized space-S with deterministic space $S' = 2S + O(\log n)$ under two assumptions: very efficient cryptographic PRGs, and strong circuit lower bounds. The subsequent work [14] obtained the same conclusion without the cryptographic assumption, while still requiring lower bounds for non-uniform circuits. We go further, deducing the same conclusion from hardness of compression of a multi-output function by *uniform, deterministic* machines that run in polynomial time and sublinear space:

Theorem 5 (informal). *Assume that for any large enough constant C there exists a function f mapping n bits to n^2 bits that is computable in space $(C+1) \cdot \log(n)$, but for any deterministic algorithm R that runs in space $n^{0.01}$ and time $n^{O(C)}$, there are at most finitely many $x \in \{0,1\}^n$ such that the following holds: When given input x, the algorithm $R(x)$ prints an $O(n)$-length description of a machine M that runs in space $C \cdot \log(n)$ and prints $f(x)$. Then, for any $S(n) = \Omega(\log n)$ and constant $\varepsilon > 0$,*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{SPACE}[(2+\varepsilon) \cdot S].$$

In the time-bounded regime, deducing extremely efficient (i.e., superfast) derandomization from a strong circuit lower bound is still an open problem, let alone deducing it from hardness for uniform deterministic algorithms; currently, all works require either cryptography, or lower bounds for non-deterministic non-uniform circuits (see, e.g., [8–10, 13, 46]).

Disambiguating nondeterministic logspace. The final question we consider is whether nondeterministic logspace can be made *unambiguous*, in the sense that for every **NL** language, there is a (one-way logspace) verifier that is only convinced by a *unique* witness for every $x \in L$. This is commonly known as the **NL** = **UL** question, and it is the space-bounded analogue of the **NP** vs. **UP** question. The disambiguation task reduces to a derandomization task, and specifically to derandomizing a graph-theoretic variant of the classical isolation lemma by [50], where the variant was introduced by Reinhardt and Allender [41] (see also [18]).

Allender, Reinhardt, and Zhou [1] showed that if there is a problem in $\mathbf{SPACE}[O(n)]$ hard for non-uniform exponential-sized circuits, then indeed **NL** = **UL**. Very recently, Li, Pyne, and Tell [30] proved an analogous conclusion in a scaled-up regime (i.e. that $\mathbf{NSPACE}[n] = \mathbf{USPACE}[O(n)]$) from hardness against *uniform* exponential-sized circuits (where the machine printing the circuit is itself an $O(n)$ space unambiguous machine). In this context too, we deduce the same conclusion from lower bounds against uniform *polynomial-sized* circuits.

Theorem 6. *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that* $\mathbf{USPACE}[n]$ *is hard for circuits of size n^c with oracle access to* $\mathbf{USPACE}[\varepsilon cn]$*, where the circuits are uniformly generated by an algorithm that runs in* $\mathbf{TISP}[2^{O(n)}, \mathrm{poly}(n)]$ *with oracle access to* $\mathbf{USPACE}[O(n)]$.[5] *Then,* $\mathbf{NSPACE}[n] \subseteq \mathbf{USPACE}[O_\varepsilon(n)]$.

Similarly to Theorem 4, the hardness assumption in Theorem 6 represents a near-exponential improvement in the size of the circuits against which we need hardness, at the cost of mildly increasing the space allowed for the uniform machine.

[4] The input length n to the oracle is the same length as the input to the generating algorithm (so we do not let the machine write longer oracle queries).

[5] Here too, the input length n to the oracle is the same length as the input to the generating algorithm.

1.3 The Technical Contributions

Our results are based on substantial improvements to the array of technical tools that have been introduced in recent years for studying hardness-vs.-randomness for bounded-space computation. To contextualize this contribution, consider classical constructions of pseudorandom generators based on a hard function f (e.g., the Nisan-Wigderson [36] PRG). The analysis of these PRGs is based on a *reconstruction* argument: If an efficient distinguisher is not fooled by the generator (built from f), then an efficient procedure computes f.

Now, let us view both the pseudorandom generator and the reconstruction as a pair of algorithms "of equal status", rather than thinking of the reconstruction as only part of the analysis; similar perspectives have been useful for extractor theory, meta-complexity, learning, and pseudodeterministic algorithms (see, e.g., [5, 6, 22, 26, 49]). Observe that a generator with respect to f is useful for derandomization, whereas the reconstruction procedure computes f, and *at least one* of the two is guaranteed to work (cf., Theorem 1). We will use a function f such that both the output of the generator (when f is hard) and the output of f itself (when f is easy) are useful.

A key point for making this approach work is using both a generator and a reconstruction procedure with low complexity. In recent years, *deterministic* reconstruction procedures have been developed, following Pyne, Raz, and Zhan [39] (see also [14, 30]), in which case both the generator and the reconstruction algorithms are deterministic. Technically, in this work we develop new efficient generators with deterministic reconstruction procedures, as well as deterministic reconstruction procedures for known generators that work in broader contexts than before. Specifically, our results rely on the following technical contributions:

(1) **Derandomized D2Ps for PRG+Distinguisher.** All known derandomized reconstruction procedures rely on derandomized transformations of distinguishers to predictors (D2P). Informally, a D2P transformation is a mapping from circuits C into short sequences $P_1, \ldots, P_m$ of circuits, such that if C distinguishes a distribution $\mathbf{D}$ from uniform, then some P_i is a decent next-bit predictor for $\mathbf{D}$. Yao's [53] classical lemma can be thought of as a very general randomized D2P, whereas we are interested in deterministic D2Ps.
Previously, deterministic D2Ps were known either for read-once branching programs [14] or for specific distinguishers [30]. We develop deterministic D2Ps for *compositions of PRGs with distinguishers*, where both the distinguisher and the PRG may be of various types: Our D2Ps work for compositions of Nisan's [34] PRG with ROBPs, of the Forbes-Kelley [17] PRG with AOBPs, and of a PRG by van Melkebeek and Prakriya [51] with a graph-theoretic distinguisher. See Section 2.1.2 and Section 2.3 for details.

(2) **SU Generator with deterministic reconstruction.** Previous deterministic reconstruction procedures were for generators or targeted generators based on the Nisan-Wigderson generator [36] (e.g., for the targeted generator of [8] instantiated with [36]; see also [14, 30, 39]). However, the NW generator is well-known to have suboptimal parameters, and using it in our constructions would not allow us to obtain our results. We thus develop a deterministic low-space reconstruction procedure for the more efficient Shaltiel-Umans [45] generator, which we use for our results. See Section 2.1.3 for details.

(3) **A new targeted generator.** For the pair of algorithms in Theorem 1, we construct a new targeted generator with a deterministic reconstruction procedure. This generator can be thought of as a variant of the Chen–Tell [8] generator that is particularly suited for space-bounded hardness-vs.-randomness results. The construction is described in Section 2.1.

2 Overview of Proofs

In this section we present high-level overviews of our proofs, aiming to present self-contained descriptions (especially for the proof of Theorem 1). In particular, while describing the proofs we will explain the role of the new complexity-theoretic tools mentioned in Section 1.3 (i.e., the D2P transformations, the reconstructive generator, and the targeted generator for space-bounded settings), but we will present the constructions of these tools in separate subsections. In particular, in Section 2.1 we explain the proof of Theorem 1, in Section 2.2 we explain how to deduce the win-win corollaries, and in Section 2.3 we explain the proofs of results from Section 1.2.

2.1 The Pair of Algorithms

At a high level, our algorithms $\mathcal{A}_1, \mathcal{A}_2$ (for random walk estimation and s-t connectivity respectively) work as follows. Fixing a graph G on n vertices, we consider a reachability bootstrapping system (à la [8]), which is a sequence of n strings ("layers") defined as follows. For $i \in [n]$, the i^{th} layer, denoted $P_i \in \{0,1\}^{n^2}$, is defined as:

$$(P_i)_{s,t} = \mathbb{I}\left[\text{there exists a path from } s \text{ to } t \text{ of length at most } i\right].$$

We observe two critical properties about this system:

(1) **Downward self reducibility.** There is a (deterministic) logspace algorithm that, given input $(G,(s,t))$ and query access to P_i, computes $(P_{i+1})_{s,t}$.

(2) **Nondeterministic computability.** There is a nondeterministic logspace algorithm that, given (G,i,s,t), computes $(P_i)_{s,t}$.

The algorithm in Item 1 is direct, whereas the algorithm in Item 2 requires the Immerman-Szelepcsényi theorem [25, 47] that **coNL** = **NL**. Note that we could use the first algorithm to compute any entry in the bootstrapping system (by repeatedly using downward self-reducibility), but the recursion depth is n, yielding a space-inefficient algorithm. The second algorithm allows us to "shortcut", and compute each entry in the bootstrapping system using nondeterminism.

We build the pair of algorithms $\mathcal{A}_1, \mathcal{A}_2$ around the following question: Is there an i such that P_i allows us to produce pseudorandom walks on G (using complexity-theoretic tools)? Given such an i, we will estimate random walk probabilities; otherwise, we will solve s-t connectivity.

Specifically, for the purpose of producing pseudorandom walks from P_i, we build a pseudorandom generator GEN with the following properties:

(1) **Logspace computability.** Given $P \in \{0,1\}^{n^2}$ and a graph G on n vertices, $\mathsf{GEN}^P(G)$ is computable in logspace. Moreover, the output is either $\widetilde{\mathbf{G}} \in \mathbb{R}^{n\times n}$ or $\perp$, where $\widetilde{\mathbf{G}}$ is a $1/n$-approximation of $\mathbf{G}^n$, and $\mathbf{G}$ is the random walk matrix of G.
(2) **Deterministic reconstruction.** There is an algorithm REC running in polynomial time and polylogarithmic space that, given P and G such that $\mathsf{GEN}^P(G) = \perp$, outputs a polylog$(n)$-size (oracle) circuit C such that $C^G(x) = P_x$ for all $x \in [|P|]$.

First we explain how to combine these ingredients to obtain Theorem 1, then go further into the description of the generator. To estimate random walk probabilities in **NL**, we enumerate over i, and use $\mathsf{GEN}(G)$ with P_i to try to produce $\widetilde{\mathbf{G}} \approx \mathbf{G}^n$. When $\mathsf{GEN}(G)$ tries to access entries of P_i, we answer using the **NL** algorithm from Item 2. This yields an **NL** algorithm $\mathcal{A}_1$ such that if there is an i for which $\mathsf{GEN}^{P_i}(G) \neq \perp$, the algorithm outputs an approximation of n-step walks on G.[6]

Otherwise, it is the case that $\mathsf{GEN}^{P_i}(G) = \perp$ for every i. In this case, we iterate from $i = 1, \ldots, n$, at each stage using REC to build a compressed representation C_i of P_i. This compressed representation is of size polylog(n), and can be evaluated in space polylog(n).[7] Once we have a compressed representation C_n of P_n, we can output the transitive closure of G using polylog(n) space and poly(n) time. In more detail, in each iteration $i \in [n]$:

(1) Assume we have a representation C_{i-1} such that $C^G_{i-1}(s,t) = (P_{i-1})_{s,t}$ for all $(s,t) \in [n]^2$. By Item 1, we can compute $(s,t) \mapsto (P_i)_{s,t}$ using queries to C_{i-1}.
(In the first iteration $i = 1$ we can compute each entry of P_1 directly in logspace.)
(2) By our assumption, $\mathsf{GEN}^{P_i}(G) = \perp$. Hence, we can use the algorithm REC from Item 2 to obtain C_i such that $C^G_i(s,t) = (P_i)_{s,t}$ for all s, t.
(3) Finally, delete the representation C_{i-1}, and increment i.

Since each of the n steps takes polylog(n) space and poly(n) time, and space is reused across steps, the algorithm $\mathcal{A}_2$ runs in $\mathbf{SC} = \mathbf{TISP}[\mathrm{poly}(n), \mathrm{polylog}(n)]$ as claimed.

Finally, note that both algorithms can detect failure. Specifically, the **NL** algorithm $\mathcal{A}_1$ outputs $\perp$ if no i allowed it to produce $\widetilde{\mathbf{G}}$ (i.e., if $\mathsf{GEN}(G)$ outputs $\perp$ with all P_i). Similarly, the **SC** algorithm $\mathcal{A}_2$ can check at each iteration i that C^G_i computes P_i (otherwise, it outputs $\perp$).

Outline of Technical Description. In Section 2.1.1, we give an overview of the construction of GEN. Then, in Section 2.1.2 and Section 2.1.3, we describe constructions of two key technical components that are necessary for the construction of GEN.

2.1.1 A walk generator with deterministic reconstruction. The construction of GEN is based on the hardness vs. randomness paradigm. As explained in Section 1.3, this paradigm yields a pair of algorithms GEN and REC such that for every string P, either GEN produces pseudorandomness from P or REC efficiently computes P (in this case, REC produces a small description of P). The pseudorandomness that we need in our setting is very specific: we just need to approximate walk probabilities on a graph.

A first attempt might be to instantiate a known PRG, such as NW [27, 36], with the truth table $f = P_i$, and use the output of the generator to take random walks on G. A line of recent work [11, 14, 15, 30, 39] developed deterministic and space-efficient reconstruction procedures for NW that work in our setting (i.e., when using the output to approximate walks), but there is a fundamental issue: If we use NW to produce n pseudorandom bits (for an n-step walk), then we can only guarantee that when NW fails, the reconstruction compresses f to size n^c (for some constant $c > 1$), which in our case would not compress the reachability matrix at all.

Indeed, we need a generator that will produce n-step pseudorandom walks, such that failure of this generator allows us to compress f to size polylog(n). This is known to be impossible when using standard black-box techniques to produce n pseudorandom bits,[8] but since we only care about producing pseudorandom walks for a given graph (rather than arbitrary pseudorandom bits), we are able to bypass this barrier using two technical components.

Technical component 1: Using an unconditional "outer" PRG. Instead of producing n-step walks, we use the PRG to produce a polylog(n)-bit seed for the pseudorandom generator of Nisan [32], which stretches its seed to an n-step pseudorandom walk.[9] Since our PRG now only needs to output polylog(n) bits, when it fails we can compress f to size polylog(n).

However, this breaks a key part of the argument. Specifically, the reconstruction REC for our PRG relies on a transformation of a *distinguisher* (for the output of the PRG) into a *next-bit-predictor*, à la Yao [53], denoted D2P. Since we need REC to be deterministic, we need the D2P to be deterministic. However, previous works [11, 14, 30, 39] constructed D2P transformations only for certain classes of distinguishers, and the distinguisher *"does the PRG output a seed for* NIS *that causes it to produce good pseudorandom walks?"* is not in these classes.

Thus, we develop a new deterministic D2P for the latter distinguisher. In fact, this is part of a broader contribution of this work, in which we develop deterministic D2Ps for various types of distinguishers of the form "composition of a PRG with a weak distinguisher". Our D2P for the current specific distinguisher is described in Section 2.1.2.

Technical component 2: A better "inner" PRG.. A well-known limitation of NW is that when producing polylog(n) output bits and compressing f to size polylog(n), its seed is polylogarithmic rather than logarithmic. Thus, it would not have the properties that we need from GEN (i.e., logspace and polytime computability). Instead of using NW, we use a generator that does not suffer from this limitation, namely the Shaltiel–Umans [45] generator SU.

[6] To ensure the output is consistent for a fixed graph, the algorithm tries $i = 1, \ldots,$ in sequence and uses the first layer that produces an output.
[7] To be more accurate, the algorithm needs to evaluate C^G_i rather than C_i. But since G is given to the algorithm as input, whenever C_i queries G, the algorithm can answer the query in logspace.
[8] To be precise, whenever using the standard hybrid argument (or, more generally, a distinguish-to-predict transform for arbitrary distinguishers) and black-box hardness amplification to produce n pseudorandom bits, an overhead of poly(n) in compression is unavoidable; see [20, 46] and [30, Appendix B].
[9] In this simplified description, we consider NIS as an algorithm that takes as input a random seed and produces a set of random walks (and for most seeds, the uniform distribution over the set of walks is pseudorandom). This idea was also used to reduce the catalytic space complexity of producing random walks [38].

However, now another key part in the argument breaks. Recent works developed deterministic logspace reconstruction algorithms for NW, but no such algorithms are known for SU. In fact, only very recently Chen *et al.* [6] showed a setting in which (a modification of) SU has reconstruction that can be computed uniformly (rather than by non-uniform circuits), and their algorithm is neither space-efficient nor randomness-efficient. Thus, we develop a new reconstruction procedure, which simultaneously achieves both these goals. This is described in Section 2.1.3.

By combining. the two foregoing technical components, we obtain GEN = SU such that GEN and REC have the properties needed to construct the pair of algorithms described in Section 2.1.

2.1.2 A D2P for Nisan's generator composed with estimating random walks. Let us recall the definitions of distinguishers, predictors, and of D2P transformations (the latter were formally introduced in [14] and studied more generally in [30]).

Definition 2.1 (distinguisher)**.** We say that $C\colon \{0,1\}^n \to \{0,1\}$ is an ε-distinguisher for a distribution $\mathbf{D}$ over $\{0,1\}^n$ if $\left|\mathbb{E}[C(\mathbf{U}_n)] - \mathbb{E}[C(\mathbf{D})]\right| \geq \varepsilon$, where $\mathbf{U}_n$ is the uniform distribution.

Definition 2.2 (next-bit predictor)**.** For $i \in [n]$, we say that $P\colon \{0,1\}^{i-1} \to \{0,1\}$ is a δ-next-bit-predictor for a distribution $\mathbf{D}$ over $\{0,1\}^n$ if $\Pr_{x \leftarrow \mathbf{D}}[P(x_{<i}) = x_i] \geq \frac{1}{2} + \delta$.

Definition 2.3 (D2P, simplified)**.** An algorithm A is a distinguish to predict (D2P) transformation for a class C if A gets as input a description of a circuit $C\colon \{0,1\}^n \to \{0,1\}$ from C, and prints a list of circuits $P_1, ..., P_m\colon \{0,1\}^\star \to \{0,1\}$ such that for *every* distribution $\mathbf{D}$ over $\{0,1\}^n$ the following holds. If C is a $(1/3)$-distinguisher for $\mathbf{D}$, then there is an $i \in [m]$ such that P_i is an $(1/O(n))$-predictor for $\mathbf{D}$.

Several prior works [11, 14, 19, 34, 39] yield deterministic D2P transformations for the "random walk" distinguisher. In more detail, this distinguisher can be modeled as a read-once branching program (ROBP), and deterministic D2Ps for ROBPs are known.

Now, recall that Nisan's PRG NIS chooses at random $\ell = \log n$ hash functions $\vec{h} = (h_1, \ldots, h_\ell)$, each over $O(\log n)$ bits, and for every graph G, for almost all collections of hash functions, the generator $\mathsf{NIS}_{\vec{h}}$ produces a pseudorandom distribution of random walks for G. Our distinguisher is therefore

$$T_G(\vec{h}) = \mathbb{I}\left[\mathsf{NIS}_{\vec{h}} \text{ produces good random walks for } G\right],$$

and this is not an ROBP (as the Nisan PRG reads each hash function repeatedly even to produce a single output). We overcome this issue by constructing a D2P transformation for this distinguisher:

Theorem 2.4 (informal)**.** *There is a deterministic logspace D2P transformation for T_G. Moreover, each candidate predictor is evaluable in logspace, given access to G.*

Our proof uses a reduction from the recent work of Li, Pyne, and Tell [30]. They show that producing a D2P transformation for a distinguisher T reduces to solving a problem called "prefix-CAPP" (PCAPP) for T.[10] In particular, we say that a logspace machine solves PCAPP for T if given $T\colon \{0,1\}^n \to \{0,1\}$ and $x \in \{0,1\}^{\leq n}$, the machine estimates $\mathbb{E}_z[T(x \circ z)]$ to within error $1/n^2$.

Theorem 2.5 ([30])**.** *Suppose there is a logspace machine that solves* PCAPP *for T. Then, there is a logspace computable D2P transformation for T.*

Naively, solving such a PCAPP problem may seem as hard as solving CAPP directly for T (which itself in general is as hard as derandomization). However, we exploit the structure of T to solve PCAPP more efficiently. In [30] they observed that this is possible if the distinguisher obeys a certain *polarization* property, which in our case is as follows. For every G, and prefix of hash functions $(h_1, \ldots, h_i)$, the following dichotomy occurs:

(1) *There is a $j \leq i$ such that h_j is a "bad" hash function.* In this case, for every suffix z, $\mathbb{E}_z[T_G(h_1, \ldots, h_i, z)] = 0$.
(2) *There is no $j \leq i$ such that h_j is a bad hash function.* In this case, $\mathbb{E}_z[T_G(h_1, \ldots, h_i, z)] \approx 1$.

We show that (a slight modification of) the generator NIS indeed obeys this polarization property. Hence, when considering the output distribution of this (modified) generator, we can solve PCAPP in a simple, deterministic way: we only need to test each of the hash functions in the given prefix (i.e., rather than estimate T on a distribution of suffixes). By the reduction of D2P to PCAPP from [30], we obtain an efficient D2P transformation for T_G.

2.1.3 A generator with uniform near-deterministic logspace reconstruction. The generator SU maps $f \in \{0,1\}^N$ to a list of strings in $\{0,1\}^M$, which are hopefully pseudorandom. It is coupled with an efficient reconstruction algorithm RSU that converts any next-bit-predictor P for the list of M-bit strings into a circuit C_f^P of size $\mathrm{poly}(M) \ll N$ that computes f. In our setting $N = n^2$ and $M = \mathrm{polylog}(n)$, and it is crucial that RSU is a small-space machine that uses only $O(\log N)$ random coins (so that we can enumerate).

A formal statement appears in the full version, and we now describe some of the ideas in our modification, at a high-level. The generator SU, which we do not change, arithmetizes its input f as a low-degree polynomial $\hat{f}\colon \mathbb{F}_q^v \to \mathbb{F}_q$, and outputs evaluations of $\hat{f}$ on "lines" going in a certain direction A in $\mathbb{F}_q^v$.[11] A simplified version of the reconstruction is as follows: Choose a random low-degree curve $C\colon \mathbb{F}_q \to \mathbb{F}_q^v$, and query $\hat{f}$ at the $m-1$ "preceding" curves going back from C in direction A^{-1} (i.e., query $\hat{f}$ at all points on the curves $A^{-i} \cdot C$ for $i \in [m-1]$). This curve C and queried "starting points" define a circuit F, which computes $\hat{f}$. Specifically, when given $\vec{z}$, the circuit F starts from $C_0 = C$ and repeatedly uses the next-element-predictor – which predict in "direction" A – to predict the next curve $A \cdot C_{i+1}$, until reaching a curve that contains $\vec{z}$. (This simplified description hides many details, among them the fact that D works in several "strides" of the form A^i for $i = q, ..., q^{v-1}$, and the fact that C actually uses two interleaved curves and relies on a list-decoding algorithm at each step.)

[10] CAPP is short for Circuit Approximation Probability Problem [28], the problem of estimating the probability of acceptance of a given circuit C to within a fixed additive error. In a "prefix-CAPP" problem, we are given a prefix x along with a device T, and wish to approximate the probability of acceptance of $T(x \circ y)$ over random suffixes y.

[11] To be more accurate, consider a matrix $A \in \mathbb{F}_q^{v \times v}$ representing multiplication by a primitive element in $\mathbb{F}_{q^v}$. Then, the generator chooses a random $\vec{x} \in \mathbb{F}_q^v$ and outputs the evaluations of $\hat{f}$ at the points $\vec{x}, A \cdot \vec{x}, A^2 \cdot \vec{x}, ..., A^{m-1} \cdot \vec{x}$.

We briefly explain how we make this reconstruction randomness-efficient. (Making the reconstruction space-efficient is relatively easier, relying on the efficiency of many of its components as well as on ideas of Doron and Tell [15].)

Randomness-efficient samplers. A la [39], we use randomness-efficient samplers to reduce the randomness complexity of various parts of the reconstruction. The underlying observation is that many of the components in the reconstruction repeat a single procedure that uses $O(\log N)$ coins multiple times (and take, for example, an OR, or the majority vote). Instead of repeating the procedure with independent coins, we can use a sampler with $O(\log N)$ coins to output a sample such that (w.h.p.) the procedure behaves on the sample approximately the same as on a uniformly chosen sample.

Pseudorandom curves, and reusing randomness for defining points. The procedure relies on the random low-degree curve having sufficiently strong sampling properties. First, it needs the curve to be a good sampler (i.e., for any subset $T \subseteq \mathbb{F}_q^v$, the points on the curve sample T approximately correctly, with high probability). A natural idea is to replace a random curve with a *curve sampler*, which pseudorandmly outputs a curve with the needed sampling properties. Indeed such a sampler was designed by Guo [21] (following [48]) for this particular purpose – and in fact with our parameter regime in mind.

Secondly, when considering the defining points for the ℓ-degree curve, which are the points $t_1, ..., t_\ell \in \mathbb{F}_q$ such that we interpolate the curve according to certain values on $t_1, ..., t_\ell$, the procedure splits these points into $O(\log N)$ blocks, and needs the points inside each block to be good samplers (in $\mathbb{F}_q$).[12] A naive approach is to choose the points in each block using a sampler, but this seemingly requires too much randomness (i.e., $O(\log N)$ coins, times the sampler's randomness complexity). However, we show that the same randomness can be reused across blocks, since the analysis boils down to a union-bound over events that each depend on a single block.

Pseudorandom interleaving. The last part of our modification is more subtle. Loosely speaking, in [45] they actually use two low-degree curves C_1 and C_2, where C_1 is a random curve and C_2 is obtained by "shifting" the values of C_1 on some of the defining points, by a small number $O(\log N)$ of predetermined shift values.[13] In their argument, both C_1 and C_2 have sufficient sampling properties, since the marginal distribution over each of the curves is that of a random low-degree curve. However, in our argument, C_2 is obtained by applying a sequence of predetermined shifts to the values of a curve sampler C_1 on the defining points (and then interpolating), and it is not clear that this operation preserves the sampling properties of C_1.

If C_2 would have been obtained by applying shifts to *all* of the points of C_1 (rather than applying them to the defining points and then interpolating), then we would be able to prove that C_2 is indeed a sampler. Of course, we cannot enforce that all of the points of C_2 will be various shifts of C_1, since that is not necessarily a low-degree curve. To get around this, we partition $\mathbb{F}_q$ into a small number $O(\log N)$ of large subfields, and for each subfield, we use a sampler to choose defining points in the subfield, and define C_2 using an appropriate shift of C_1 on these defining points. As above, we reuse randomness for the sampler across subfields.[14] Since C_2 is a shift of C_1 on a pseudorandom set of points within each subfield, we can argue that C_2 behaves sufficiently similar to a shift of C_1 on the entire subfield. Further details appear in the full version.

2.2 Tight Win-Win Results in Space-Bounded Complexity

We now explain how to obtain the win-win results in space complexity, based on (the proof of) Theorem 1. For Theorem 2, we first modify the pair of algorithms. Rather than attempting to compute reachability and random walks on the same graph (equivalently, on two graphs of comparable size), we instead take a small graph G_1 (of size $2^{\log^{1/2+\varepsilon/2} n}$), and a large graph G_2, and attempt to either compute reachability on G_1, or estimate random walk probabilities on G_2. Also, instead of computing the reachability bootstrapping system in **NL**, we use Savitch's Theorem [44] to compute the system deterministically. This yields the following pair of algorithms:

Theorem 2.6 (informal)**.** *For every $\varepsilon > 0$, there are algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for every pair of graphs G_1 on $2^{\log^{1/2+\varepsilon/2} n}$ vertices, and G_2 on n vertices, at least one of the following holds:*

- *$\mathcal{A}_1(G_1, G_2)$ computes s-t connectivity in G_1 in* **SC**.
- *$\mathcal{A}_2(G_1, G_2)$ estimates length-n random walk probabilities in G_2 in* $\mathbf{SPACE}[\log^{1+\varepsilon} n]$.

Moreover, both algorithms report if they fail to compute the desired answer, and do not exceed their resource bounds in any case.

Indeed, in the proof of Theorem 2.6, the bootstrapping system can be computed in nondeterministic space $O(\log^{1/2+\varepsilon/2} n)$, and hence in *deterministic* space $O(\log^{1+\varepsilon} n)$ by Savitch's Theorem [44]. As such, in the case that the reachability bootstrapping system does contain a hard truth table, we can compute this truth table (and hence compute random walks on G_2) in space $O(\log^{1+\varepsilon} n)$, obtaining a small overhead even for deterministic derandomization.

We use the pair of algorithms from Theorem 2.6 to prove Theorem 2. Following the approach of [14, 30], we fix a $\mathbf{BPSPACE}[n]$ machine $\mathcal{B}$ and a $\mathbf{NSPACE}[n^{1/2+\varepsilon/2}]$ machine $\mathcal{N}$. For each pair $(x, y) \in \{0,1\}^n \times \{0,1\}^n$, consider the following two graphs:

$$G_1(y) = \text{the configuration graph of } \mathcal{N}(y)$$
$$G_2(x) = \text{the configuration graph of } \mathcal{B}(x).$$

Let us try and simulate $\mathcal{B}$ in $\mathbf{SPACE}[n^{1+\varepsilon}]$ (if we fail, we will show another algorithm that simulates $\mathcal{N}$ in $\mathbf{TISP}[2^m, \mathrm{poly}(m)]$). Our algorithm gets $x \in \{0,1\}^n$ and enumerates over all $y \in \{0,1\}^n$ (which it can, since it is allowed to use super-linear space). It then runs $\mathcal{A}_2$ on the pair of graphs $G_1(y), G_2(x)$; if $\mathcal{A}_2$ outputs an estimation of random walks on $G_2(x)$, we are done, and otherwise we continue to the next y. The point is that one of two things happened:

[12] This sampling property of the defining points of the curve C is used to argue that, with high probability, the predictor succeeds in predicting sufficiently many points on C (and on each $A^i \cdot C$).

[13] That is, if C_1 is defined by a small number of conditions of the form "$C_1(t) = \vec{z}_t$" (for a small number of $t \in \mathbb{F}_q$ and $\vec{z}_t \in \mathbb{F}_q^v$), then C_2 is defined by the conditions "$C_2(t) = A_t \cdot \vec{z}_t$", where A_t is an invertible matrix in $\mathbb{F}_q^v \times \mathbb{F}_q^v$.

[14] In fact, we set things up so that the $O(\log N)$ subfields we use for pseudorandom interleaving are also exactly the $O(\log N)$ blocks mentioned above (when discussing pseudorandom curves).

Either for every x there is y such that this algorithm succeeds, in which case we simulated $\mathcal{B}$ in $\mathbf{SPACE}[n^{1+\varepsilon}]$ on this input length; or there exists x such that for every y this algorithm fails. In the latter case, a symmetric argument shows that for all $y \in \{0,1\}^n$ we can simulate $\mathcal{A}_1$ in $\mathbf{TISP}[2^m, \mathrm{poly}(m)]$ (i.e., given y, we enumerate over all x and simulate $\mathcal{A}_1$ with the two graphs). For further details see the full version.

2.3 Derandomization from Very Weak Hardness

The basic idea behind all of our results that deduce derandomization from very weak hardness (i.e., Theorem 4, Theorem 6 and Theorem 2.7) is the same. Let us describe the idea in general terms, and then later focus on describing one particular result in more technical detail.

To demonstrate the idea, consider trying to derandomize the class $\mathbf{BPSPACE}[O(n)]$ in $\mathbf{SPACE}[O(n)]$. We instantiate a generator with a hard problem $L \in \mathbf{SPACE}[O(n)]$, say, the Shaltiel–Umans generator SU from Section 2.1.3. Now, if the derandomization fails on some input, then we can compress the hard truth-table, as follows. We enumerate over inputs $x \in \{0,1\}^n$ (indeed, we can do this since we are working in a scaled-up regime of linear space) and run the reconstruction algorithm. This algorithm is deterministic, and whenever the derandomization fails at x, the reconstruction manages to compress the truth-table of $L_{O(n)}$. (For simplicity, we ignore for a moment the distinguisher that is not fooled by SU and whose description is part of the compressed version of $L_{O(n)}$.)

The key question is what is the *size* of the compressed version of $L_{O(n)}$. For this question, a main bottleneck is the number of output bits that we ask SU to output; loosely speaking, if it outputs M bits, then the compressed version will be of size $\mathrm{poly}(M)$.[15] In settings concerning derandomization in polynomial time or logarithmic space, we have $M = N^{\Omega(1)}$, and thus $\mathrm{poly}(M) = N^{\Theta(1)}$. For our results, we are interesting in obtaining a compressed representation of size only $\mathrm{polylog}(N)$.

The key: Unconditional PRGs and corresponding D2Ps. The way to achieve this goal will be similar to an idea from the proof of Theorem 1. In all of our settings, we consider relatively weak distinguishers, for which unconditional PRGs are known. For example, when derandomizing $\mathbf{BPSPACE}[O(n)]$, the distinguisher D is an ROBP, and we can compose it with Nisan's [32] PRG to obtain a distinguisher $D \circ \mathsf{NIS}$ over $M = \mathrm{polylog}(N)$ bits.

The main challenge is that we now need to develop derandomized D2P transformations for distinguishers of the form "compose a weak distinguisher with an unconditional PRG". Indeed, one such D2P was presented in Section 2.1.2, for a composition of ROBPs with NIS. To further demonstrate out approach, we now describe another derandomized D2P, for the composition of an AOBP with (a modified version of) the Forbes-Kelley [17] PRG. The AOBP distinguisher – which is an *any-order* branching program, defined in [7] – comes up when constructing derandomization algorithms with minimal memory overhead (and when using an idea from [15]), and the FK PRG fools such distinguishers with polylogarithmic seed.

[15] This is since the reconstruction overhead is affected by the prediction advantage, and the prediction advantage is at most $1/M$ whenever using a hybrid argument (or a deterministic D2P; see [30, Appendix B]).

2.3.1 *Derandomized D2P for* AOBP $\circ$ FK.

Consider an AOBP denoted A and the Forbes-Kelley PRG FK. Note that even if A would have been an ROBP, the composition $A \circ \mathsf{FK}$ is not an ROBP, and thus we cannot use the known D2P transformations for ROBPs (e.g., from [14]) for this composition.

We construct a derandomized D2P transform for a modified version of the Forbes-Kelley generator, where the modification facilitates the D2P transform. For our goal of minimal-memory overhead it will be crucial that the modified version of FK remains strongly explicit (as is the original generator), but we can afford a seed length that is n^{ε} (rather than $\mathrm{polylog}(n)$). We give an informal statement of the result here:

Theorem 2.7 (Forbes–Kelley D2P, informal)**.** *There is a generator* $\mathsf{FK}\colon \{0,1\}^{n^{\varepsilon}} \to \{0,1\}^n$ *with the following properties.*

1. ***Strong Explicitness.*** *The map* $(x, j) \to \mathsf{FK}(x)_j$ *is computable in space* $O(\varepsilon \log n)$ *with catalytic access to* j.[16]
2. ***Fooling.*** *The generator fools AOBPs of size* n *to error* $1/n$.
3. ***White-Box D2P.*** *There is a white-box D2P transform that can be computed in time* $\mathrm{poly}(n)$ *and space* $O(n^{\varepsilon})$*, where each predictor can be evaluated in space* $\log(n) + O(\varepsilon \log n)$.[17]

Our actual construction makes several changes to the Forbes–Kelley generator, but the idea is the following. The generator is constructed as a sequence of *random restrictions*, each of which eliminate some variables while approximately preserving the expectation of the branching program A. We think of the generator's input as $(A_1, B_1, \ldots, A_\ell, B_\ell)$, where each A_i, B_i is the output of a k-wise independent generator over $\{0,1\}^n$. We then define $\mathsf{FK}_{\ell+1} = 0^n$, and

$$\mathsf{FK}_i = A_i \oplus B_i \wedge \mathsf{FK}_{i+1}$$

In particular, for $j \in [n]$ where $(B_i)_j = 0$, we say a variable has been eliminated by level i, and further levels of the generator do not affect the output of the generator on that bit. Recall that by [30], to produce a D2P transform, it suffices to solve prefix-CAPP, which in this case is the following:

Question 2.8. Given an AOBP A and

$$(\vec{a}, \vec{b}) = (A_1, B_1, \ldots, A_{i-1}, B_{i-1}),$$

estimate

$$\mathop{\mathbb{E}}_{\vec{a}', \vec{b}'} [A(\mathsf{FK}(\vec{a}, \vec{b}, \vec{a}', \vec{b}')].$$

To see how the prefix affects the problem, consider the branching program $A_{\vec{a},\vec{b}}$ wherein every variable that has been eliminated is simply fixed to the value output by the generator at that bit. Thus, we are now being asked to estimate

$$\mathbb{E}[A_{\vec{a},\vec{b}}(z \oplus \mathsf{FK}_i(\mathbf{U}))].$$

where z is a fixed vector that accounts for the prior levels of the generator.

The key observation is that the Forbes-Kelley generator *fools branching programs*, and so this expectation should itself be close

[16] The algorithm is given access to a special read-write tape, initialized to (the binary representation of) j. When the algorithm halts and returns $\mathsf{FK}(x)_j$, the tape must be restored to that initial configuration.

[17] For technical reasons we construct a white-box Yao derandomization [30], where we are given access to the distribution $\mathbf{D}$ that does not fool $A \circ \mathsf{FK}$ and construct a predictor for this distribution.

to $\mathbb{E}[A_{\vec{a},\vec{b}}(\mathbf{U})]$, i.e., filling in all non-eliminated variables with independent true randomness. Estimating this quantity is easily seen to be in **BPL** $\subseteq$ **SC**, and so we obtain a **SC**-computable D2P transform with this complexity.

We remark that our actual construction is substantially different from the above due to the following technical issues: First, we cannot afford $k = O(\log n)$-wise independent restrictions, as at several steps in our argument we must enumerate over all seeds in a single level of a generator, which would take time $2^{\log^2 n} \gg n$. Because of this, we can only afford $O(1/\varepsilon)$-wise independence, which requires a generator with n^ε levels, and a much smaller restriction probability.

The second and larger issue is that it is not actually true that for *every* prefix of a seed to the Forbes-Kelley generator, the output of the generator on a random suffix of the generator is approximately the same as filling in bits uniformly. For example, consider a pathological prefix that eliminates far fewer variables than it should; then, a random suffix of a seed for the generator will not eliminate all the variables (with high probability), in which case the generator fills in many bits with $\mathsf{FK}_{\ell+1} = 0^n$. (Needless to say, this is very far from uniform.)

Our solution is to construct an *auxiliary* D2P transform such that, if many prefixes are in deficient in this way, then we can solve PCAPP on these prefixes very efficiently. This modification requires changing the final level of the generator to behave differently when few variables are left alive. By itself, this change would destroy strong explicitness, but we are able to use ideas from catalytic computation to obtain a strongly explicit PRG with *catalytic access* to the input (which suffices for the minimal memory overhead application, as explained in the full version). The interested reader is referred to the full version for further details about the D2P.

References

[1] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. 1999. Isolation, Matching, and Counting Uniform and Nonuniform Upper Bounds. *J. Comput. Syst. Sci.* 59, 2 (1999), 164–181.

[2] Roy Armoni, Amnon Ta-Shma, Avi Wigderson, and Shiyu Zhou. 2000. An $O(\log(n)^{4/3})$ space algorithm for (s, t) connectivity in undirected graphs. *J. ACM* 47, 2 (2000), 294–311. doi:10.1145/333979.333984

[3] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. 1998. A Sublinear Space, Polynomial Time Algorithm for Directed *s-t* Connectivity. *SIAM J. Comput.* 27, 5 (1998), 1273–1282.

[4] Jin-yi Cai, Venkatesan T. Chakaravarthy, and Dieter van Melkebeek. 2006. Time-Space Tradeoff in Derandomizing Probabilistic Logspace. *Theory Comput. Syst.* 39, 1 (2006), 189–208. doi:10.1007/S00224-005-1264-9

[5] Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. 2015. Tighter Connections between Derandomization and Circuit Lower Bounds. In *Proc. 19th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 645–658.

[6] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. 2023. Polynomial-Time Pseudodeterministic Construction of Primes. *arXiv preprint arXiv:2305.15140* (2023).

[7] Lijie Chen, Xin Lyu, Avishay Tal, and Hongxun Wu. 2023. New PRGs for Unbounded-Width/Adaptive-Order Read-Once Branching Programs. In *Proc. 50 International Colloquium on Automata, Languages and Programming (ICALP) (LIPIcs, Vol. 261)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:20.

[8] Lijie Chen and Roei Tell. 2021. Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 125–136.

[9] Lijie Chen and Roei Tell. 2021. 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)*. 283–291.

[10] Lijie Chen and Roei Tell. 2023. When Arthur has Neither Random Coins nor Time to Spare: Superfast Derandomization of Proof Systems. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*. 60–69.

[11] Kuan Cheng and William M. Hoza. 2022. Hitting Sets Give Two-Sided Derandomization of Small Space. *Theory Comput.* 18 (2022), 1–32.

[12] Stephen A. Cook and Charles Rackoff. 1980. Space Lower Bounds for Maze Threadability on Restricted Machines. *SIAM J. Comput.* 9, 3 (1980), 636–652. doi:10.1137/0209048

[13] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. 2022. Nearly Optimal Pseudorandomness From Hardness. *Journal of the ACM* 69, 6 (2022), 1–55.

[14] Dean Doron, Edward Pyne, and Roei Tell. 2024. Opening Up the Distinguisher: A Hardness to Randomness Approach for **BPL** = **L** that Uses Properties of **BPL**. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*.

[15] Dean Doron and Roei Tell. 2023. Derandomization with Minimal Memory Footprint. In *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Article 11.

[16] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. 1999. Tight Lower Bounds for st-Connectivity on the NNJAG Model. *SIAM J. Comput.* 28, 6 (1999), 2257–2284.

[17] Michael A. Forbes and Zander Kelley. 2018. Pseudorandom generators for read-once branching programs, in any order. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 946–955.

[18] Anna Gál and Avi Wigderson. 1996. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms* 9, 1-2 (1996), 99–111.

[19] Uma Girish, Ran Raz, and Wei Zhan. 2023. Is Untrusted Randomness Helpful?. In *Proc. 14 Conference on Innovations in Theoretical Computer Science (ITCS) (LIPIcs, Vol. 251)*. 56:1–56:18.

[20] Aryeh Grinberg, Ronen Shaltiel, and Emanuele Viola. 2018. 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)*. 956–966.

[21] Zeyu Guo. 2013. Randomness-Efficient Curve Samplers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. Springer, 575–590.

[22] Shuichi Hirahara. 2023. Non-black-box worst-case to average-case reductions within NP. *SIAM Journal on Computing* 52, 6 (2023), FOCS18–349–FOCS18–382.

[23] William M. Hoza. 2021. Better Pseudodistributions and Derandomization for Space-Bounded Computation. In *Proceedings of the 25th International Conference on Randomization and Computation (RANDOM)*. 28:1–28:23.

[24] William M. Hoza. 2022. Recent Progress on Derandomizing Space-Bounded Computation. *Bull. EATCS* 138 (2022).

[25] Neil Immerman. 1988. Nondeterministic Space is Closed Under Complementation. *SIAM J. Comput.* 17, 5 (1988), 935–938. doi:10.1137/0217058

[26] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. 2006. Reducing the seed length in the Nisan-Wigderson generator. *Combinatorica* 26, 6 (2006), 647–681.

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

[28] Valentine Kabanets, Charles Rackoff, and Stephen A. Cook. 2000. Efficiently Approximable Real-Valued Functions. *Electron. Colloquium Comput. Complex.* TR00-034 (2000). ECCC:TR00-034 https://eccc.weizmann.ac.il/eccc-reports/2000/TR00-034/index.html

[29] Adam R. Klivans and Dieter van Melkebeek. 2002. Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. *SIAM Journal on Computing* 31, 5 (2002), 1501–1526.

[30] Jiatu Li, Edward Pyne, and Roei Tell. 2024. Distinguishing, Predicting, and Certifying: On the Long Reach of Partial Notions of Pseudorandomness.

[31] Pinyan Lu, Jialin Zhang, Chung Keung Poon, and Jin-yi Cai. 2005. Simulating Undirected *st*-Connectivity Algorithms on Uniform JAGs and NNJAGs. In *Proceedings of 16th International Symposium on Algorithms and Computation (ISAAC) (Lecture Notes in Computer Science, Vol. 3827)*. Springer, 767–776. doi:10.1007/11602613_77

[32] Noam Nisan. 1991. Pseudorandom bits for constant depth circuits. *Combinatorica* 11, 1 (1991), 63–70.

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

[34] Noam Nisan. 1994. **RL** $\subseteq$ **SC**. *Computational Complexity* 4 (1994), 1–11.

[35] Noam Nisan, Endre Szemerédi, and Avi Wigderson. 1992. Undirected Connectivity in $O(\log^{1.5} n)$ Space. In *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 24–29.

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

[37] Chung Keung Poon. 1993. Space Bounds for Graph Connectivity Problems on Node-named JAGs and Node-ordered JAGs. In *34th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 218–227. doi:10.1109/SFCS.1993.366865

[38] Edward Pyne. 2024. Derandomizing Logspace with a Small Shared Hard Drive. In *Proc. 39th Annual IEEE Conference on Computational Complexity (CCC)*. 4:1–4:20.

[39] Edward Pyne, Ran Raz, and Wei Zhan. 2023. Certified Hardness vs. Randomness for Log-Space. In *64th IEEE Annual Symposium on Foundations of Computer Science,*

FOCS 2023.

[40] Omer Reingold. 2008. Undirected connectivity in log-space. *Journal of the ACM* 55, 4 (2008), 17:1–17:24.

[41] Klaus Reinhardt and Eric Allender. 2000. Making Nondeterminism Unambiguous. *SIAM J. Comput.* 29, 4 (2000), 1118–1131.

[42] Michael E. Saks and Shiyu Zhou. 1999. $\mathbf{BP_HSPACE}[S] \subseteq \mathbf{DSPACE}[S^{3/2}]$. *Journal of Computer and System Sciences* 58, 2 (1999), 376–403.

[43] Rahul Santhanam and R. Ryan Williams. 2013. On medium-uniformity and circuit lower bounds. In *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*. IEEE, 15–23.

[44] Walter J. Savitch. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.* 4 (1970), 177–192.

[45] Ronen Shaltiel and Christopher Umans. 2005. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM* 52, 2 (2005), 172–216.

[46] Ronen Shaltiel and Emanuele Viola. 2022. On Hardness Assumptions Needed for "Extreme High-End" PRGs and Fast Derandomization. In *Proc. 13 Conference on Innovations in Theoretical Computer Science (ITCS)*.

[47] Róbert Szelepcsényi. 1988. The Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica* 26, 3 (1988), 279–284.

[48] Amnon Ta-Shma and Christopher Umans. 2006. Better lossless condensers through derandomized curve samplers. In *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 177–186.

[49] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. 2007. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica* 27, 2 (2007), 213–240.

[50] Leslie G. Valiant and Vijay V. Vazirani. 1986. NP is as Easy as Detecting Unique Solutions. *Theor. Comput. Sci.* 47, 3 (1986), 85–93.

[51] Dieter van Melkebeek and Gautam Prakriya. 2019. Derandomizing Isolation in Space-Bounded Settings. *SIAM J. Comput.* 48, 3 (2019), 979–1021.

[52] Avi Wigderson. 1992. The Complexity of Graph Connectivity. In *Mathematical Foundations of Computer Science (MFCS) (Lecture Notes in Computer Science, Vol. 629)*. Springer, 112–132.

[53] Andrew C. Yao. 1982. Theory and Application of Trapdoor Functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 80–91.

Received 2024-11-04; accepted 2025-02-01