2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS) | 979-8-3503-1894-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/FOCS57990.2023.00074

Polynomial-Time Pseudodeterministic Construction of Primes

Lijie Chen
UC Berkeley
Berkeley, CA, USA
lijiechen@berkeley.edu

Zhenjian Lu
University of Oxford
Oxford, UK
zhenjian.lu@cs.ox.ac.uk

Igor C. Oliveira
University of Warwick
Coventry, UK
igor.oliveira@warwick.ac.uk

Hanlin Ren
University of Oxford
Oxford, UK
hanlin.ren@cs.ox.ac.uk

Rahul Santhanam
University of Oxford
Oxford, UK
rahul.santhanam@cs.ox.ac.uk

***Abstract*—A randomized algorithm for a search problem is *pseudodeterministic* if it produces a fixed canonical solution to the search problem with high probability. In their seminal work on the topic, Gat and Goldwasser [1] posed as their main open problem whether prime numbers can be pseudodeterministically constructed in polynomial time.**

We provide a positive solution to this question in the infinitely-often regime. In more detail, we give an *unconditional* polynomial-time randomized algorithm B such that, for infinitely many values of n, $B(1^n)$ outputs a canonical n-bit prime p_n with high probability. More generally, we prove that for every dense property Q of strings that can be decided in polynomial time, there is an infinitely-often pseudodeterministic polynomial-time construction of strings satisfying Q. This improves upon a subexponential-time construction of Oliveira and Santhanam [2].

Our construction uses several new ideas, including a novel bootstrapping technique for pseudodeterministic constructions, and a quantitative optimization of the uniform hardness-randomness framework of Chen and Tell [3], using a variant of the Shaltiel–Umans generator [4].

***Index Terms*—explicit construction, pseudodeterministic construction, hardness vs. randomness**

I. Introduction

How hard is it to construct an n-bit prime[1]? This is a fundamental problem in number theory and in complexity theory. Under reasonable assumptions, the problem is solvable in deterministic polynomial time. In more detail, Cramér's conjecture [5] in number theory asserts that the largest prime gap in any consecutive sequence of n-bit numbers is $O(n^2)$. Assuming this conjecture, we can solve the prime construction problem efficiently by testing the first $O(n^2)$ integers greater than 2^{n-1} for primality and outputting the first one, where the primality tests are done efficiently using the algorithm of Agrawal, Kayal and Saxena [6]. An independent source of evidence for the efficiency of prime construction is the complexity-theoretic conjecture that $\mathsf{DTIME}(2^{O(n)})$ requires Boolean circuits of exponential size on almost all input lengths. Under this conjecture, we can use the Impagliazzo–Wigderson pseudorandom generator [7] to *derandomize* the simple randomized algorithm that outputs a random n-bit number, using the facts that primality testing is in polynomial time and that an $\Omega(1/n)$ fraction of n-bit numbers are prime.

However, we seem very far from either settling Cramér's conjecture or proving strong complexity lower bounds. The best upper bound we can prove on the gap between consecutive n-bit primes is $2^{(0.525+o(1))n}$ [8], and no super-linear circuit lower bounds are known for $\mathsf{DTIME}(2^{O(n)})$ [9]. Indeed, the best unconditional result we have so far is that deterministic prime construction can be done in time $2^{(0.5+o(1))n}$ [10], which is very far from the polynomial-time bound we seek. The Polymath 4 project (see [11]) sought to improve this upper bound using number-theoretic techniques but did not achieve an unconditional improvement.

In contrast to the situation with deterministic prime construction, it is easy to generate an n-bit prime *randomly*, as mentioned above: simply generate a random n-bit number, test it for primality in polynomial time, and output it if it is a prime. This algorithm has success probability $\Omega(1/n)$ by the Prime Number Theorem, and the success probability can be amplified to be exponentially close to 1 by repeating the process $\mathrm{poly}(n)$ times independently, and outputting the first of these $\mathrm{poly}(n)$ numbers that is verified to be prime, assuming that there is at least one.

Gat and Goldwasser [1] asked whether it is possible to generate primes efficiently by a randomized process, such that the output is essentially *independent* of the randomness of the algorithm. In other words, is there a polynomial-time randomized algorithm, which on input 1^n, constructs a *canonical* prime of length n with high probability? They call such an algorithm a *pseudodeterministic* algorithm, since the output of the algorithm is (almost) deterministic even though the algorithm might use random bits in its operation. Note that the randomized algorithm for prime generation we described in the previous paragraph is very far from being pseudodeterministic, as different runs of the algorithm are

[1]Recall that a positive integer q is an n-bit prime if q is a prime number and $2^{n-1} \leq q \leq 2^n - 1$.

979-8-3503-1894-4/23/$31.00 ©2023 IEEE

DOI 10.1109/FOCS57990.2023.00074

unlikely to produce the same prime. It is easy to see that a pseudodeterministic construction serves as an intermediate notion between a randomized construction (which is trivial for primes) and a deterministic construction (where little progress has been made so far).

[1] initiate a general theory of pseudodeterminism for search problems, motivated by applications in cryptography and distributed computing. Since then, there have been a number of papers on pseudodeterminism, in various contexts, such as query complexity [12]–[14], streaming algorithms [15], [16], parallel computation [17], [18], learning algorithms [19], Kolmogorov complexity [20], [21], space-bounded computation [22], proof systems [23], [24], number theory and computational algebra [2], [25], approximation algorithms [26], and many other settings (see, e.g., [27]–[32]).

Despite all this progress, the main problem about pseudodeterminism posed in [1] has remained open: Is there a pseudodeterministic polynomial-time algorithm for prime construction? They describe this problem as "the most intriguing" and "perhaps the most compelling challenge for finding a unique output".

Unlike in the case of deterministic construction, number-theoretic techniques have so far not proven useful for the pseudodeterministic construction problem for primes. Using complexity-theoretic techniques, Oliveira and Santhanam [2] (see also [21]) showed that for any $\varepsilon > 0$, there is an algorithm that runs in time 2^{n^ε} and succeeds on infinitely many input lengths.

II. Our Results

In this paper, we design a significantly faster algorithm and provide an affirmative answer to the question posed by Gat and Goldwasser in the infinitely-often regime. Our main result can be stated in full generality as follows.

Theorem 1 (Infinitely-Often Polynomial-Time Pseudodeterministic Constructions)**.** *Let $Q \subseteq \{0,1\}^*$ be a language with the following properties:*

(Density.) *there is a constant $\rho \geq 1$ such that for every $n \in \mathbb{N}_{\geq 1}$, $Q_n \triangleq Q \cap \{0,1\}^n$ satisfies $|Q_n| \geq n^{-\rho} \cdot 2^n$; and*

(Easiness.) *there is a deterministic polynomial-time algorithm A_Q that decides whether an input $x \in \{0,1\}^*$ belongs to Q.*

Then there exist a probabilistic polynomial-time algorithm B and a sequence $\{x_n\}_{n \in \mathbb{N}_{\geq 1}}$ of n-bit strings in Q such that the following conditions hold:

1) *On every input length $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) \notin \{x_n, \perp\}] \leq 2^{-n}$.*
2) *On infinitely many input lengths $n \in \mathbb{N}_{\geq 1}$, $\Pr_B[B(1^n) = x_n] \geq 1 - 2^{-n}$.*

Interestingly, our construction is non-black-box, in the sense that changing the *code* of the algorithm A_Q deciding property Q affects the canonical output of the corresponding algorithm B. We will revisit this point when we discuss our techniques (see the remark at the end of Section IV-B).

Letting Q be the set of prime numbers and noticing that Q is both dense (by the Prime Number Theorem) and easy (by the AKS primality test [6]), we immediately obtain the following corollary of Theorem 1.

Corollary 2 (Infinitely-Often Polynomial-Time Pseudodeterministic Construction of Primes)**.** *There is a randomized polynomial-time algorithm B such that, for infinitely many values of n, $B(1^n)$ outputs a canonical n-bit prime p_n with high probability.*

Corollary 2 improves upon the subexponential-time infinitely-often pseudodeterministic construction of primes from [2] mentioned above. Note that the result for prime construction is a corollary of a far more general result about properties that are dense and easy. This is evidence of the surprising power of complexity theory when applied to a problem which seems to be about number theory (but where number-theoretic techniques have not so far been effective). The famous efficient primality testing algorithm of [6] similarly applied complexity-theoretic derandomization ideas to solve a longstanding open problem in computational number theory, though their argument does require more information about primes.

For a string $w \in \{0,1\}^*$ and $t\colon \mathbb{N} \to \mathbb{N}$, we let $\mathsf{rK}^t(w)$ denote the length of the smallest randomized program that runs for at most $t(|w|)$ steps and outputs w with probability at least $2/3$. (We refer to [33] for a formal definition and for an introduction to probabilistic notions of time-bounded Kolmogorov complexity.) By encoding the (constant-size) randomized polynomial-time algorithm B and each good input length n using $O(1) + \log n$ bits in total, the following result holds.

Corollary 3 (Infinitely Many Primes with Efficient Succinct Descriptions)**.** *There is a constant $c \geq 1$ such that, for $t(n) = n^c$, the following holds. For every $m \geq 1$, there is $n > m$ and an n-bit prime p_n such that $\mathsf{rK}^t(p_n) \leq \log(n) + O(1)$.*

In other words, there are infinitely many primes that admit very short efficient descriptions. The bound in Corollary 3 improves upon the sub-polynomial bound on $\mathsf{rK}^{\mathrm{poly}}(p_n)$ from [21].

In the next section, we describe at a high level the ideas in the proof of Theorem 1, and how they relate to previous work.

III. Proof Ideas

The proof of Theorem 1 relies on *uniform hardness-randomness tradeoffs* [34], [35]. For concreteness, assume that $Q = \{Q_n\}_{n \in \mathbb{N}_{\geq 1}}$, with each $Q_n \subseteq \{0,1\}^n$ consisting of the set of n-bit prime numbers. Let A_Q be a deterministic polynomial-time algorithm that decides Q (e.g., A_Q is the AKS primality test algorithm [6]). Before we present our algorithm and the main ideas underlying our result, it is instructive to discuss the approach of [2], which provides a subexponential-time pseudodeterministic construction that succeeds on infinitely many input lengths.

Subexponential-time constructions of [2]: We first recall how uniform hardness-randomness tradeoffs work. Given a presumed hard language L, a uniform hardness-randomness tradeoff for L states that either L is easy for probabilistic polynomial-time algorithms, or else we can build a *pseudorandom set* $G_n \subseteq \{0,1\}^n$ computable in subexponential time (thus also has subexponential size), which fools probabilistic polynomial-time algorithms on inputs of length n (for infinitely many n). In particular, Trevisan and Vadhan [35] give a uniform hardness-randomness tradeoff for a PSPACE-complete language L_{TV} they construct, which has certain special properties tailored to uniform hardness-randomness tradeoffs.[2]

The subexponential-time construction in [2] uses a *win-win* argument to derive an *unconditional* pseudodeterministic algorithm from the uniform hardness-randomness tradeoff of [35]. There are two cases: either $L_{\mathsf{TV}} \in \mathsf{BPP}$, or it is not. If the former is the case, then $\mathsf{PSPACE} \subseteq \mathsf{BPP}$ by the PSPACE-completeness of L_{TV}. Now, since we can in *polynomial space* test all n-bit numbers using A_Q until we find the lexicographic first prime number, we can also do it in *randomized polynomial time*, i.e., there is a randomized algorithm $B(1^n)$ that runs in polynomial time and outputs the lexicographically first n-bit prime with high probability. Thus, in this case, the lexicographically first n-bit prime is the "canonical" output of the pseudodeterministic algorithm, and the algorithm works on *every* input length n.

Suppose, on the other hand, that $L_{\mathsf{TV}} \notin \mathsf{BPP}$. Using the uniform hardness-randomness tradeoff of [35], we have that for each $\varepsilon > 0$, there is a pseudorandom set $G = \{G_n\}$, where each $G_n \subseteq \{0,1\}^n$ is of size at most 2^{n^ε}, such that for infinitely many n, G_n fools the algorithm A_Q on inputs of length n. Since A_Q accepts an $\Omega(1/n)$ fraction of strings of length n by the Prime Number Theorem, we have that the fraction of strings in G_n that are prime is $\Omega(1/n)$ (by choosing the error parameter of the uniform hardness-randomness tradeoff to be small enough). In particular, there must exist an element of G_n that is prime. Since G_n is computable in subexponential time, we can define a subexponential time *deterministic* algorithm that enumerates elements of G_n and tests each one for primality until it finds and outputs one that is prime. This algorithm is deterministic but it runs in subexponential time, and is only guaranteed to be correct for infinitely many n.

Thus, in either case, we have a pseudodeterministic algorithm for constructing primes that runs in subexponential time and works infinitely often. Note that we do not know a priori which of the two cases above holds, and therefore the argument is somewhat non-constructive. By exploiting further properties of the uniform hardness-randomness tradeoff, [2] manage to give an explicit construction algorithm that runs in subexponential time infinitely often.

[2]For the pseudorandomness experts, these special properties are *downward self-reducibility* and *random self-reducibility*.

Win-win arguments: The above argument gives a subexponential-time construction, but the win-win structure of the argument seems incapable of giving an optimal polynomial-time construction. Indeed, this is the case for many win-win arguments used in complexity theory:

- A win-win argument based on the Karp–Lipton theorem [36] gives that $\Sigma_2\mathsf{EXP}$ requires super-polynomial size Boolean circuits [37], but seems incapable of giving truly exponential ($2^{\Omega(n)}$) Boolean circuit lower bounds.
- A win-win argument based on uniform hardness-randomness tradeoffs gives that either $\mathsf{E} \subseteq \mathsf{BPP}$ or BPP can be simulated infinitely often in deterministic subexponential time on average [34], but it remains unknown if such a tradeoff holds at the "high end", i.e., whether it is the case that either E is in probabilistic subexponential-time or else BPP can be simulated infinitely often in deterministic polynomial time on average.
- A win-win argument based on the Easy Witness Lemma gives that if $\mathsf{NEXP} \subseteq \mathsf{SIZE}(\mathrm{poly})$, then $\mathsf{NEXP} = \mathsf{MA}$ [38], but it is unknown if any interesting uniform collapse follows from the simulation of NEXP by subexponential-size Boolean circuits.

In each of these cases, the win-win argument seems to have inherent limitations that prevent us from getting optimal lower bounds or tradeoffs. Indeed, a paper by Miltersen, Vinodchandran and Watanabe [39] studies the "fractional exponential" lower bounds that seem to be the best provable using win-win arguments in the context of Boolean circuit lower bounds for exponential-time classes.[3]

Thus, in order to obtain a polynomial-time pseudodeterministic algorithm for primality, it seems that we need to go beyond win-win arguments. One natural idea is to apply uniform hardness-randomness tradeoffs *recursively*. However, this seems hard to do with the uniform hardness-randomness tradeoff of [35]. Their tradeoff applies only to the special language L_{TV}. If we argue based on the hardness or other properties of L_{TV}, then in the case where $L_{\mathsf{TV}} \in \mathsf{BPP}$, we get a pseudodeterministic polynomial-time algorithm for constructing primes, but in the case where $L_{\mathsf{TV}} \notin \mathsf{BPP}$, we get a subexponential-time constructible pseudorandom set, and it is unclear how to apply the uniform hardness-randomness tradeoff to the algorithm for constructing this set.

Recursive application of uniform hardness-randomness tradeoffs: One of our main ideas is to exploit very recent work on uniform hardness-randomness tradeoffs [3] which applies to *generic* computations, as long as they satisfy certain mild properties. These tradeoffs yield *hitting sets* rather than pseudorandom sets based on hardness — a hitting set $\mathsf{H} \subseteq \{0,1\}^M$ is a set that has non-empty intersection with

[3]For example, a function $f : \mathbb{N} \to \mathbb{N}$ is *sub-half-exponential* if $f(f(n)^c)^c \leq O(2^n)$ for every constant c. (The exact definition of sub-half-exponential functions may be different in different papers.) Functions such as n^k and $2^{\log^k n}$ are sub-half-exponential, while $2^{\varepsilon n}$ and 2^{n^ε} are not. It is known that $\Sigma_2\mathsf{EXP}$ cannot be computed by $f(n)$-size circuits for every sub-half-exponential f, but it remains open to show that $\Sigma_2\mathsf{EXP}$ requires circuit complexity 2^{n^ε} for any constant $\varepsilon > 0$.

every $Q_M \subseteq \{0,1\}^M$ that is dense (i.e., accepts at least a $1/\mathrm{poly}(M)$ fraction of strings) and is efficiently computable. It turns out that for our application to pseudodeterministic algorithms, uniform hardness-randomness tradeoffs that yield hitting sets are sufficient.

Specifically, Chen and Tell [3] show that for any multi-output function $f\colon \{1^n\} \to \{0,1\}^n$ computed by uniform Boolean circuits of size $T = T(n)$ and depth $d = d(n)$, either there is a hitting set $\mathsf{H} \subseteq \{0,1\}^M$ computable in time $\mathrm{poly}(T)$, or $f(1^n)$ can be computed with high probability in time $(d+n)\cdot\mathrm{poly}(M)$ (which could be much less than T). Note that this tradeoff is applicable to *any* multi-output function f given bounds on its uniform circuit complexity.

Our key idea is that this more generic uniform hardness-randomness tradeoff can be applied *recursively*. Indeed, we apply it to multi-output functions which capture the very task we are trying to solve, i.e., constructing a prime! In our base case, we use the function f which does a brute-force search over n-bit numbers and outputs the lexicographically first one which is prime. This function can be computed by uniform Boolean circuits of size $2^{O(n)}$ and depth $\mathrm{poly}(n)$, and hence we can apply the Chen–Tell tradeoff to it. We set $M = n^\beta$ for some large enough constant $\beta > 1$ in the tradeoff. If we have that $f(1^n)$ is computable with high probability in time $(d+n)\cdot\mathrm{poly}(M)$, then we are done, since this gives us a pseudodeterministic algorithm for primes at length n. If not, we have that there is a hitting set $\mathsf{H} \subseteq \{0,1\}^{n^\beta}$ computable in time $2^{O(n)}$. In particular, by iterating over the elements of H and outputting the first one that is prime, we gain over the naïve brute-force search algorithm, since we are now outputting a prime of length n^β in time $2^{O(n)}$. Now *this* new algorithm can be captured by a multi-output function with output length n^β to which we apply the Chen–Tell tradeoff again. In each recursive step, we either obtain a pseudodeterministic polynomial-time construction of primes, or we obtain a significantly faster deterministic construction of primes (of a larger input length). Intuitively, analyzing this process after $O(\log n)$ steps of recursion, we can hope to show that at least one of the steps leads to a polynomial-time pseudodeterministic algorithm at the input length considered at that step.

This doesn't quite work as stated because the Chen–Tell tradeoff uses the Nisan–Wigderson generator [40], which is not known to have optimal parameters for all levels of hardness.[4] Our recursive process explores essentially all possible levels of hardness for the uniform hardness-randomness tradeoff, since each recursive step corresponds to a different level of hardness. Using the original Chen–Tell tradeoff gives a *quasi-polynomial-time* pseudodeterministic construction, but in order to get a polynomial-time pseudodeterministic construction, we need to work harder.

Another crucial idea for us is to optimize the Chen–Tell tradeoff by using the Shaltiel–Umans generator [4] rather than the Nisan–Wigderson generator. This idea comes with its own implementation challenges, since the Shaltiel–Umans generator is not known to have a crucial learnability property that is required for the uniform hardness-randomness tradeoff. We sidestep this issue using a further win-win analysis, together with some other tricks; see Section IV-C for details. This enables us to achieve an optimal polynomial-time pseudodeterministic construction on infinitely many input lengths, and thereby establish Theorem 1.[5] We note that the subexponential-time construction of [2] also only works for infinitely many input lengths, and it is still open even to get a subexponential-time construction that works on all input lengths.

The intuitive description here does not address several subtleties that arise in the proof, such as maintaining the right uniformity and depth conditions when recursively applying the uniform hardness-randomness tradeoff. We refer to Section IV for a more detailed discussion of such matters.

IV. Technical Overview

As explained above, we consider a chain of $t = O(\log n)$ recursively defined (candidate) HSGs $\mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_t$ operating over different input lengths. These HSGs are obtained from the recent construction of Chen and Tell [3], which we informally describe next. Recall that we use Q_M to denote the easy and dense property over inputs of length M.

The Chen–Tell [3] targeted HSG ("ideal version"): Let $c \geq 1$ be a large enough constant, and let $f\colon \{1^n\} \to \{0,1\}^n$ be a family of unary functions computed by (uniform) Boolean circuits of size $T = T(n)$ and depth $d = d(n)$. Then, for every $\log T \leq M \leq T$ there is a set $\mathsf{H} \subseteq \{0,1\}^M$ computable in

$$\text{time } \widetilde{T} \triangleq T^c \quad \text{and} \quad \text{depth } \widetilde{d} \triangleq d \cdot \log(T) + M^c$$

such that, if $Q_M \subseteq \{0,1\}^M$ *avoids* H, (i.e., Q_M is dense but $Q_M \cap \mathsf{H} = \varnothing$), then we can compute $f(1^n)$ with high probability in time $(d+n)\cdot M^c$.

In other words, if f admits *low-depth* circuits, we can construct a candidate HSG H over length-M inputs such that breaking the generator H allows us to compute $f(1^n)$ in time $\mathrm{poly}(n, d, M)$. For $d, M \ll T$, this can be much faster than the original time T required to compute f.

The statement above differs from the results in [3] (stated for unary functions) in two important ways. First, the claimed upper bound on $\widetilde{T}$ (the running time of the HSG) is not obtained by [3] for all choices of M. Secondly, we have not formally specified the *uniformity* of the family of circuits computing f. While these are crucial points in [3] and when proving our result, for simplicity we will assume for now that this upper bound can be achieved and omit the discussion on uniformity.

[4] Informally speaking, given a "hard truth table" of length T, we want to construct a hitting set $\mathsf{H} \subseteq \{0,1\}^M$ in $\mathrm{poly}(T)$ time; however, the Nisan–Wigderson generator requires $2^{\Theta(\log^2 T/\log M)}$ time to construct.

[5] While we do not explore this direction in the current work, we believe that our improvement on the Chen-Tell tradeoff can be used to improve the tradeoff from [41, Theorem 5.2 and Theorem 5.3], thus getting a better uniform hardness vs randomness connection in the low-end regime.

Bootstrapping the win-win argument: We now review the idea discussed in Section III, using notations that will be more convenient for the remainder of this technical overview. Fix an arbitrary $n \in \mathbb{N}_{\geq 1}$, and consider the corresponding property $Q_n \subseteq \{0,1\}^n$ decided by $A_Q(x)$ on inputs of length n. Our initial H_0 is trivial and set to $\{0,1\}^n$. (Intuitively, this corresponds to the first case of the argument in [2] sketched above where $L_{\mathsf{TV}} \in \mathsf{BPP}$.) Consider now a "brute-force" algorithm $\mathsf{BF}(1^n)$ that computes the first $x \in \mathsf{H}_0$ such that $A_Q(x) = 1$. We let $f(1^n) \triangleq \mathsf{BF}(1^n)$ in the Chen–Tell HSG. Note that $f(1^n)$ can be uniformly computed in time $T = 2^{O(n)}$ and depth $d = \mathrm{poly}(n)$, since $A_Q(x)$ runs in polynomial time and all elements of H_0 can be tested in parallel. We set $M(n) \triangleq n^\beta$, where $\beta > 1$ is a large enough constant. Let $\mathsf{H}_1 \subseteq \{0,1\}^M$ be the candidate HSG provided by Chen–Tell. Note that H_1 can be computed in time $\widetilde{T} = 2^{O(n)}$ and depth $\widetilde{d} = \mathrm{poly}(n)$.

Next, we consider a win-win argument based on whether Q_M avoids H_1. If this is the case, then Chen–Tell guarantees that we can compute $f(1^n) = \mathsf{BF}(1^n) \in Q_n$ with high probability in time $(d+n) \cdot M^c = \mathrm{poly}(n)$. In other words, we can pseudodeterministically produce a string in Q_n in polynomial time. On the other hand, if $\mathsf{H}_1 \cap Q_M \neq \varnothing$, we now have a set H_1 of strings of length $M = n^\beta$ that contains a string in Q_M and that can be deterministically computed in time $2^{O(n)}$. That is, we are back to the former case, except that we can compute H_1 (a set containing at least one M-bit prime) in time much faster than $2^{O(M)}$. Crucially, in contrast to the approach of [2], the Chen–Tell HSG does not limit us to the use of the special language L_{TV}, effectively allowing us to reapply the same argument (with a speedup) over a larger input length.

In the next subsection, we discuss the "bootstrapping" and its parameters in more detail and explain how it gives a polynomial-time pseudodeterministic construction, assuming we have the ideal version of [3] described above.

A. Infinitely-Often Pseudodeterministic Polynomial-Time Constructions

Let $n_0 \in \mathbb{N}$ be an "initial" input length, and $t = O(\log n_0)$ be a parameter. For each $1 \leq i \leq t$, we define the i-th input length to be $n_i \triangleq n_{i-1}^\beta$, for a large enough constant $\beta > 1$. Our goal is to design a pseudodeterministic algorithm for finding elements in Q that will be correct on *at least one of the input lengths* $n_0, n_1, \ldots, n_t$. On each input length n_i we will have:

1) the property Q_{n_i} that we want to hit;
2) a candidate hitting set generator $\mathsf{H}_i \subseteq \{0,1\}^{n_i}$; and
3) the brute-force algorithm $\mathsf{BF}_i : \{1^{n_i}\} \to \{0,1\}^{n_i}$, which iterates through all elements in H_i and outputs the first element that is in Q_{n_i}.

Note that BF_i is completely defined by H_i. Suppose that H_i can be computed (deterministically) in time T_i and depth d_i, then BF_i can also be computed (deterministically) in time $T_i' \triangleq T_i \cdot \mathrm{poly}(n_i)$ and depth $d_i' \triangleq d_i \cdot \mathrm{poly}(n_i)$. As discussed above, initially, $\mathsf{H}_0 \triangleq \{0,1\}^{n_0}$ is the trivial hitting set generator, $T_0 \triangleq 2^{O(n_0)}$, and $d_0 \triangleq \mathrm{poly}(n_0)$.

For each $0 \leq i < t$, we let $f(1^{n_i}) \triangleq \mathsf{BF}_i, M \triangleq n_{i+1}$, and invoke the Chen–Tell HSG to obtain the HSG $\mathsf{H}_{i+1} \subseteq \{0,1\}^{n_{i+1}}$. Recall that Chen–Tell guarantees the following: Suppose that $Q_M = Q_{n_{i+1}}$ avoids the HSG H_{i+1}, then one can use $Q_{n_{i+1}}$ to compute $f(1^{n_i})$ with high probability in time $\mathrm{poly}(d_i', n_i, M) \leq \mathrm{poly}(d_i, n_i)$, by our choice of parameters. Recall that if H_i indeed hits Q_{n_i}, then $f(1^{n_i})$ implements the brute-force algorithm and outputs the first element in $\mathsf{H}_i \cap Q_{n_i}$ (i.e., a *canonical* element in Q_{n_i}). To reiterate, Chen–Tell gives us the following win-win condition:

- *either* $Q_{n_{i+1}}$ avoids H_{i+1}, in which case we obtain a probabilistic algorithm that outputs a canonical element in Q_{n_i} (thus a pseudodeterministic algorithm) in $\mathrm{poly}(d_i, n_i)$ time;
- *or* H_{i+1} hits $Q_{n_{i+1}}$, in which case we obtain a hitting set H_{i+1} that hits $Q_{n_{i+1}}$, thereby making progress on input length n_{i+1}.

The HSG H_{i+1} can be computed in time $T_{i+1} \triangleq (T_i')^c$ and depth $d_{i+1} \triangleq d_i' \cdot \log T_i' + n_{i+1}^c$. Crucially, although T_0 is exponential in n_0, it is possible to show by picking a large enough $\beta > 1$ that the sequence $\{n_i\}_{i \in \mathbb{N}}$ grows faster than the sequence $\{T_i\}_{i \in \mathbb{N}}$, and eventually when $i = t = O(\log n_0)$, it will be the case that $T_t \leq \mathrm{poly}(n_t)$ and we can apply the brute-force algorithm to find the first element in H_t that is in Q_{n_t} in time polynomial in n_t.

A more precise treatment of the growth of the two sequences $\{n_i\}$ and $\{T_i\}$ are as follows. There is some absolute constant $\alpha \geq 1$ such that $T_0 \leq 2^{\alpha n_0}$ and

$$T_{i+1} \leq T_i^\alpha \text{ (for each } 0 \leq i < t).$$

We set $\beta \triangleq 2\alpha$ (recall that each $n_{i+1} = n_i^\beta$). It follows from induction that for each $0 \leq i \leq t$,

$$T_{i+1} \leq T_0^{\alpha^i} = 2^{\alpha^{i+1} n_0} \quad \text{and} \quad n_{i+1} = n_i^\beta = n_0^{\beta^{i+1}} = n_0^{(2\alpha)^{i+1}}.$$

Since

$$\frac{\log T_t}{\log n_t} \leq \frac{\alpha^t n_0}{(2\alpha)^t \log n_0} = \frac{n_0}{2^t \log n_0},$$

it follows that when $t \approx \log(n_0 / \log n_0)$, T_t will be comparable to n_t (rather than 2^{n_t}). Similarly, one can show that $d_i \leq \mathrm{poly}(n_i)$ for every $i \leq t$.

Informal description of the algorithm and correctness: To wrap up, we arrive at the following pseudodeterministic algorithm that is correct on at least one of the input lengths $n_0, n_1, \ldots, n_t$. On input length n_i, if $i = t$, then we use $\mathrm{poly}(T_t) \leq \mathrm{poly}(n_t)$ time to find the first string in H_i that is also in Q_{n_i} (i.e., simulate BF_i); otherwise, use $Q_{n_{i+1}}$ as a distinguisher for the Chen–Tell hitting set H_i and print the output of BF_i in $\mathrm{poly}(n_i, d_i) \leq \mathrm{poly}(n_i)$ time. To see that our algorithm succeeds on at least one n_i, consider the following two cases:

1) Suppose that H_t indeed hits Q_{n_t}. Then clearly, our algorithm succeeds on input length n_t.

2) On the other hand, suppose that H_t does not hit Q_{n_t}. Since our trivial HSG H_0 hits Q_{n_0}, there exists an index $0 \le i < t$ such that H_i hits Q_{n_i} but $Q_{n_{i+1}}$ avoids H_{i+1}. Since $Q_{n_{i+1}}$ avoids H_{i+1}, Chen–Tell guarantees that we can speed up the computation of BF_i using $Q_{n_{i+1}}$ as an oracle. Since H_i hits Q_{n_i}, the output of BF_i is indeed a canonical element in Q_{n_i}. It follows that our algorithm succeeds on input length n_i.

This completes the sketch of the algorithm and its correctness. We note that while this exposition explains how the second bullet of Theorem 1 is achieved, it does not address the behavior of the algorithm on other input lengths (i.e., the first bullet in the same statement). For simplicity, we omit this here and refer to the formal presentation in Section 3 of the full version.[6]

While the aforementioned construction conveys the gist of our approach, there are two important issues with our presentation. Firstly, as explained before, the results of [3] do not achieve the *ideal parameters* of the HSG stated above. Secondly, we have only vaguely discussed the *circuit uniformity* of the function $f(1^n)$. The uniformity of f is critical for the reconstruction procedure of [3] to run in time comparable to the circuit depth of f. On the other hand, since our HSGs and functions f (corresponding to the algorithm BF) are recursively defined, the circuit uniformity of the [3] generator itself becomes another critical complexity measure in the proof.

In the next subsection, we discuss the Chen–Tell generator in more detail and explain how to obtain an improved generator construction satisfying our requirements.

B. Improving the Chen–Tell Targeted Hitting Set Generator

The uniform hardness-to-randomness framework of Chen–Tell builds on two important ingredients:[7]

1) A *layered-polynomial representation* of a shallow uniform circuit.
2) A hitting set generator with a *uniform learning reconstruction* algorithm.

Layered-polynomial representation: We now discuss the first ingredient. Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a logspace-uniform circuit family of size $T(n)$ and depth $d(n)$.[8] Let $M\colon \mathbb{N} \to \mathbb{N}$ be the parameter for output length. Building on the doubly efficient interactive proof system by [42] (and its subsequent simplification by [43]), for any $z \in \{0,1\}^n$, [3] showed that there is a sequence of polynomials $\{P_i^z\}_{i\in[d']}$ for $d' = d \cdot \mathrm{polylog}(T)$ with the following nice properties:

- (**Arithmetic setting.**) Let $\mathbb{F}$ be a finite field of size M^c for a large universal constant $c > 1$, and let m be of order $\frac{\log T}{\log M}$. All the P_i^z map $\mathbb{F}^m$ to $\mathbb{F}$ and have total degree at most M.
- (**Base case.**) There is an algorithm Base such that, given the input $z \in \{0,1\}^n$ and $\vec{w} \in \mathbb{F}^m$, computes $P_1^z(\vec{w})$ in $\mathrm{poly}(M)$ time.
- (**Downward self-reducibility.**) There is an oracle algorithm DSR that, given input $i \in \{2,\dots,d'\}$ and $\vec{w} \in \mathbb{F}^m$, together with the oracle access to $P_{i-1}^z(\cdot)$, computes $P_i^z(\vec{w})$ in $\mathrm{poly}(M)$ time.
- (**Faithful representation.**) There is an oracle algorithm OUT that, given input $i \in [n]$ and oracle access to $P_{d'}^z$, outputs $f(z)_i$ in $\mathrm{poly}(M)$ time.

Intuitively, these polynomials form an *encoded* version of the computation of f in the sense that they admit both *downward self-reducibility* and *random self-reducibility*: every P_i^z has low degree and hence admits error correction properties; downward self-reducibility follows from definition.

We note that the proof of this result depends in a crucial way on the logspace-uniformity of the circuit family computing f. (This allows one to arithmetize a formula of bounded size that computes the direct connection language of the circuit, while also controlling the circuit uniformity of the resulting polynomials.)

Hitting set generators with a uniform learning reconstruction algorithm: The second ingredient of [3] is the Nisan-Wigderson generator combined with Reed-Muller codes [40], [44]. The most important property of this generator is that it supports a uniform learning reconstruction algorithm. In more detail, for a polynomial $P\colon \mathbb{F}^m \to \mathbb{F}$, the generator NW^P takes $s = O\left(\frac{\log^2 T}{\log M}\right)$ bits as seed, such that there is a uniform oracle algorithm R (for "reconstruction") where the following holds. Given oracle access to both P and an oracle $D\colon \{0,1\}^M \to \{0,1\}$ that distinguishes $\mathsf{NW}^P(U_s)$ from the uniform distribution, $R^{P,D}$ runs in $\mathrm{poly}(M)$ time and with high probability outputs a polynomial-size D-oracle circuit that computes P.

Now, the hitting set $H_f(z)$ is defined as

$$H_f(z) \triangleq \bigcup_{i\in[d']} \mathsf{NW}^{P_i^z}.$$

The uniform reconstruction algorithm: One key observation here is that if a distinguisher $D\colon \{0,1\}^M \to \{0,1\}$ avoids $H_f(z)$, meaning that D accepts a large fraction of inputs from $\{0,1\}^M$ but rejects all strings in $H_f(z)$, then clearly D also distinguishes all $\mathsf{NW}^{P_i^z}(U_s)$ from the uniform distribution. Following [34], [3] then shows that there is a uniform oracle algorithm R_f that takes input $z \in \{0,1\}^n$ and any "avoider" D of $H_f(z)$ as oracle, and outputs $f(z)$ with high probability. In more detail, R_f works as follows:

1) It is given input $z \in \{0,1\}^n$ and oracle access to an avoider $D\colon \{0,1\}^M \to \{0,1\}$ of $H_f(z)$.
2) For every $i \in \{2,\dots,d'\}$:

[6] Alternatively, the guarantee from the first bullet of Theorem 1 can always be achieved via a general argument. We refer to [2, Proposition 2] for the details.

[7] Below we will focus on the high-level picture of the Chen–Tell framework without diving into too many details. Our presentation is also somewhat different from the original presentation in [3].

[8] Intuitively, a circuit family is logspace-uniform if each circuit in the family can be printed by a fixed machine that runs in space that is of logarithmic order in the size of the circuits. See Section 2.3 of the full version for the precise definition of logspace-uniform circuits.

a) The goal of the i-th step is to construct a $\mathrm{poly}(M)$-size D-oracle circuit C_i that computes P_i^z.
b) It runs the learning reconstruction algorithm $R^{P_i^z,D}$ to obtain a $\mathrm{poly}(M)$-size D-oracle circuit. To answer queries to P_i^z, we first run the algorithm DSR to convert them into queries to P_{i-1}^z. Next, when $i = 2$, we answer these queries by calling Base directly, and when $i > 2$ we answer these queries by evaluating our D-oracle circuit C_{i-1}.

3) For every $i \in [n]$, output $\mathsf{OUT}_{d'}^{C^D}(i)$.

Issue with the original Chen–Tell construction: Super-logarithmic seed length of NW*:* The main issue with the construction above is that $\mathsf{NW}^{P_i^z}$ has seed length $O\left(\frac{\log^2 T}{\log M}\right)$. In particular, this means that when $\log M \leq o(\log T)$, the hitting set $H_f(z)$ has super-polynomial size, and therefore cannot be computed in $\mathrm{poly}(T)$ time as in the "ideal version" of [3] stated above.[9] Hence, to improve the computation time of $H_f(z)$ to $\mathrm{poly}(T)$, we need an HSG with seed length $O(\log T)$ for all possible values of M, together with a uniform learning reconstruction, when it is instantiated with polynomials. Jumping ahead, we will replace NW with the Shaltiel–Umans Hitting Set Generator [4], obtaining an optimized version of the Chen–Tell generator with better parameters. However, the original generator from [4] does not provide a uniform learning reconstruction procedure. By a clever use of the classical construction of a *cryptographic pseudorandom generator from a one-way permutation* and of another idea, we managed to modify their construction to allow a uniform learning reconstruction. See the next subsection for more details.

Controlling the circuit uniformity of the optimized Chen–Tell generator: As stressed above, in order to construct a layered-polynomial representation for f with the aforementioned parameters, it is crucial that f admits a logspace-uniform circuit family. Since we will rely on multiple applications of the generator, and each new function BF on which the result is invoked contains as a subroutine the code of the previous generator, we must *upper bound the circuit uniformity* of our optimized Chen–Tell generator. This turns out to require a delicate manipulation of all circuits involved in the proof and of the Turing machines that produce them, including the components of the Shaltiel–Umans generator. For this reason, whenever we talk about a Boolean circuit in the actual proof, we also bound the description length and space complexity of its corresponding machine. Additionally, as we manipulate a super-constant number of circuits (and their corresponding machines) in our construction, we will also consider the complexity of producing the code of a machine M_2 encoding a circuit C_2 from the code of a machine M_1 encoding a circuit C_1 (see, e.g., the "Moreover" part in the statement of Theorem 3.1 in the full version). The details are quite tedious, but they are necessary for verifying the correctness and running time of our algorithm. In order to provide some intuition for it, we notice that as we move from the HSG H_i to H_{i+1}, we also increase the corresponding input length parameter from n_i to $n_{i+1} = n_i^\beta$. While there is an increase in the uniformity complexity, it remains bounded relative to the new input length. (Think of a truncated geometric series whose value is dominated by the complexity over the current input length.) We omit the details in this proof overview.

Non-black-box behavior: We note that the recursive application of the Chen–Tell generator is responsible for the *fully non-black-box* behavior of our pseudodeterministic construction. Indeed, since we invoke the Chen–Tell generator on each function BF (which contains the code of the algorithm A_Q deciding property Q as a subroutine), the collection of strings in the hitting set generator depends on the layered-polynomial representation that is obtained from the *code* of BF. As a consequence, our construction has the unusual feature that the canonical outputs of the algorithm B in Theorem 1 are affected by the code of A_Q. In other words, by using a different primality test algorithm (or by making changes to the code implementing the AKS routine), one might get a different n-bit prime!

The parameters of our hitting set generator appear in Section 3 of the full version. The proof of the result is given in Section 5 of the full version.

C. Modified Shaltiel–Umans Generator with Uniform Learning Reconstruction

As explained above, in order to complete the proof of Theorem 1 we need to design a variant of the Shaltiel–Umans generator [4] with a *uniform learning reconstruction* procedure.

The Shaltiel–Umans generator takes as input a low-degree polynomial $P : \mathbb{F}_p^m \to \mathbb{F}_p$ (in our case p will be a power of 2) and produces a set of binary strings (which is supposed to be a hitting set). The construction of this generator also relies on "generator matrices". A matrix $A \in \mathbb{F}_p^{m \times m}$ is a *generator matrix* if it satisfies $\{A^i \cdot \vec{1}\}_{1 \leq i < p^m} = \mathbb{F}_p^m \setminus \{\vec{0}\}$. Roughly put, the matrix A can be thought of as performing multiplication with a generator of the multiplicative group of $\mathbb{F}_{p^m}$.

Recall that a generator has a uniform learning reconstruction algorithm if the following holds. Given an algorithm D that avoids the output of the generator constructed using P, as well as P itself, we can *uniformly* and *efficiently* generate (with high probability) a D-oracle circuit that computes the polynomial P. (In other words, we can query P while producing the circuit, but the circuit itself does not have access to P.)

However, the reconstruction procedure provided by the original Shaltiel–Umans generator only guarantees the following: If the generator is constructed using P and some generator matrix A, then using an algorithm D that avoids the output of the generator, and *given the matrix* A and oracle access to P, one can obtain a (D-oracle) circuit $C : [p^m - 1] \to \mathbb{F}_p^m$

[9] Indeed, if we rely on the original Chen–Tell construction to implement the bootstrapping method described above, we would only obtain a quasi-polynomial-time pseudodeterministic construction, instead of a polynomial-time one.

such that $C(i) = P(A^i \cdot \vec{1})$.[10] (For the precise statement, see Theorem 4.9 of the full version.) That is, this reconstruction is not a uniform learning algorithm in the following sense:

1) It needs to know the matrix A (which can be viewed as non-uniform advice).
2) Given oracle access to P, it only learns a circuit that computes the mapping $i \mapsto P(A^i \cdot \vec{1})$, instead of a circuit that computes $P(\vec{x})$ on a given $\vec{x} \in \mathbb{F}_p^m$.

We now describe how to modify the Shaltiel–Umans generator to make its reconstruction a uniform learning algorithm.

For the first issue, our idea is that, instead of using a generator matrix that is obtained by brute-force search as in the original construction (we note that the reconstruction cannot afford to perform the brute-force search due to its time constraints), we will use a generator matrix that is from a small set of matrices that can be constructed *efficiently*. More specifically, using results about finding primitive roots of finite fields (e.g., [45]), we show that one can efficiently and deterministically construct a set S of matrices that contains at least one generator matrix. The advantage is that the reconstruction algorithm can still afford to compute this set S. Note that although we don't know which matrix in S is a valid generator matrix (as verifying whether a matrix is a generator matrix requires too much time), we can try all the matrices from S, and one of them will be the correct one. This allows us to obtain a list of candidate circuits, one of which computes P (provided that we can also handle the second issue, which will be discussed next). Then by selecting from the list a circuit that is sufficiently close to P (note that given oracle access to P, we can easily test whether a circuit is close to P by sampling) and by using the *self-correction* property of low-degree polynomials, we can obtain a circuit that computes P exactly.

With the above idea, we may now assume that in the reconstruction we know the generator matrix A used by the Shaltiel–Umans generator. Next, we describe how to handle the second issue. Recall that the reconstruction algorithm of the Shaltiel–Umans generator gives a circuit C such that $C(i) = P(A^i \cdot \vec{1})$, for $i \in [p^m - 1]$, and we want instead a circuit that given $\vec{x} \in \mathbb{F}_p^m$ computes $P(\vec{x})$. Now suppose given $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$, we can also *efficiently* compute the value $i \in [p^m - 1]$ such that $A^i \cdot \vec{1} = \vec{x}$. Then we would be able to combine this with C to get a circuit E that computes P, i.e., if $\vec{x} = \vec{0}$ then E outputs $P(\vec{0})$ (where the value $P(\vec{0})$ can be hardcoded); otherwise, E computes i for $\vec{x}$ as described above and then outputs $C(i)$. However, the task of finding such i given A and $\vec{x}$ is essentially the *discrete logarithm problem*, for which no efficient algorithm is known!

A classical result in cryptography is that one can construct a pseudorandom generator based on the hardness of the discrete logarithm problem (see, e.g., [46], [47]). More generally, given a permutation f whose inverse admits *random self-reducibility*[11], one can construct a generator G based on f so that if there is a distinguisher D that breaks G, then it can be used to invert f via a uniform reduction. Our idea is to consider the bijection $f : [p^m - 1] \to \mathbb{F}_p^m \setminus \{\vec{0}\}$ such that for each $i \in [p^m - 1]$, $f(i) = A^i \cdot \vec{1}$ (where the random self-reducibility of f^{-1} follows easily from that of the discrete logarithm problem), and try to construct a pseudorandom generator G based on f. We then combine the output of G with that of the Shaltiel–Umans generator constructed with the polynomial P and the generator matrix A. Now if there is an algorithm D that avoids this combined generator, which means D *simultaneously* avoids both the Shaltiel–Umans generator and the generator G, then D can be used to obtain

- a circuit C such that $C(i) = P(A^i \cdot \vec{1})$ for every $i \in [p^m - 1]$, and
- a circuit C' that inverts f, i.e., $C'(\vec{x})$ outputs i such that $A^i \cdot \vec{1} = \vec{x}$ for every $\vec{x} \in \mathbb{F}_p^m \setminus \{\vec{0}\}$.

Then it is easy to combine C and C' to obtain a circuit that computes P.

A careful implementation of these ideas allows us to obtain a variant of the Shaltiel–Umans generator with uniform learning reconstruction, as needed in our optimized Chen–Tell generator. We refer to Theorem 4.1 in the full version for more details.

This completes the sketch of the proof of Theorem 1.

Further remarks about the proof: We note that in our proof the gap between two good input lengths on which the algorithm outputs a canonical prime can be exponentially large. It would be interesting to develop techniques to reduce this gap.

Additionally, the proof assumes the existence of a deterministic polynomial-time algorithm that decides the dense property. In contrast, the sub-exponential time algorithm from [2] also works with a dense property that is decidable by a randomized polynomial-time algorithm. This is caused by the non-black-box nature of our approach via the Chen-Tell generator, which employs the code of the algorithm A deciding the property as part of the description of the generator. Consequently, as alluded to above, changing the code of A could result in a different canonical output on a given input length. If A is randomized, fixing the randomness of A is similar to the consideration of a different algorithm that decides the property, and it is not immediately clear how to maintain the pseudodeterministic behaviour in this case.

Finally, we note that the most important guarantee on the output of the algorithm obtained in Theorem 1 comes from Item 2. It is possible to achieve the guarantee from Item 1 in a generic way through a simple argument (see Proposition 2.2 in [2]).

ACKNOWLEDGMENT

Lijie Chen is supported by a Miller Research Fellowship. Zhenjian Lu is partly supported by an NSERC postdoctoral

[10] In fact, the circuit only computes $P(A^i \cdot \vec{v})$ for some $\vec{v}$ output by the reconstruction algorithm. We assume $\vec{v} = \vec{1}$ here for simplicity.

[11] Roughly speaking, a function has random self-reducibility if computing the function on a given instance can be efficiently reduced to computing the function for uniformly random instances.

fellowship. Igor C. Oliveira received support from the EPSRC New Horizons Grant EP/V048201/1, the Royal Society University Research Fellowship URF\R1\191059, and the Centre for Discrete Mathematics and its Applications (DIMAP) at the University of Warwick. Hanlin Ren received support from DIMACS through grant number CCF-1836666 from the National Science Foundation. Rahul Santhanam received support from the EPSRC New Horizons Grant EP/V048201/1. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

REFERENCES

[1] E. Gat and S. Goldwasser, "Probabilistic search algorithms with unique answers and their cryptographic applications," *Electronic Colloquium on Computational Complexity* (ECCC), vol. 18, p. 136, 2011. [Online]. Available: https://eccc.weizmann.ac.il/report/2011/136/

[2] I. C. Oliveira and R. Santhanam, "Pseudodeterministic constructions in subexponential time," in *Symposium on Theory of Computing* (STOC), 2017, pp. 665–677. [Online]. Available: https://doi.org/10.1145/3055399.3055500

[3] L. Chen and R. Tell, "Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise," in *IEEE Symposium on Foundations of Computer Science* (FOCS), 2021, pp. 125–136. [Online]. Available: https://doi.org/10.1109/FOCS52979.2021.00021

[4] R. Shaltiel and C. Umans, "Simple extractors for all min-entropies and a new pseudorandom generator," *J. ACM*, vol. 52, no. 2, pp. 172–216, 2005. [Online]. Available: https://doi.org/10.1145/1059513.1059516

[5] H. Cramér, "On the order of magnitude of the difference between consecutive prime numbers," *Acta Arithmetica*, vol. 2, pp. 23–46, 1936.

[6] M. Agrawal, N. Kayal, and N. Saxena, "PRIMES is in P," *Annals of Mathematics*, vol. 160, no. 2, pp. 781–793, 2004. [Online]. Available: https://doi.org/10.4007/annals.2004.160.781

[7] R. Impagliazzo and A. Wigderson, "$\mathsf{P} = \mathsf{BPP}$ if E requires exponential circuits: Derandomizing the XOR lemma," in *ACM Symposium on Theory of Computing* (STOC). ACM, 1997, pp. 220–229. [Online]. Available: https://doi.org/10.1145/258533.258590

[8] R. C. Baker, G. Harman, and J. Pintz, "The difference between consecutive primes. II," *Proc. London Math. Soc. (3)*, vol. 83, no. 3, pp. 532–562, 2001. [Online]. Available: https://doi.org/10.1112/plms/83.3.532

[9] J. Li and T. Yang, "$3.1n - o(n)$ circuit lower bounds for explicit functions," in *STOC*. ACM, 2022, pp. 1180–1193. [Online]. Available: https://doi.org/10.1145/3519935.3519976

[10] J. C. Lagarias and A. M. Odlyzko, "Computing $\pi(x)$: An analytic method," *J. Algorithms*, vol. 8, no. 2, pp. 173–191, 1987. [Online]. Available: https://doi.org/10.1016/0196-6774(87)90037-X

[11] T. Tao, E. Croot, III, and H. Helfgott, "Deterministic methods to find primes," *Math. Comp.*, vol. 81, no. 278, pp. 1233–1246, 2012. [Online]. Available: https://doi.org/10.1090/S0025-5718-2011-02542-1

[12] O. Goldreich, S. Goldwasser, and D. Ron, "On the possibilities and limitations of pseudodeterministic algorithms," in *Innovations in Theoretical Computer Science* (ITCS), 2013, pp. 127–138. [Online]. Available: https://doi.org/10.1145/2422436.2422453

[13] S. Goldwasser, R. Impagliazzo, T. Pitassi, and R. Santhanam, "On the pseudo-deterministic query complexity of NP search problems," in *Computational Complexity Conference* (CCC), 2021, pp. 36:1–36:22. [Online]. Available: https://doi.org/10.4230/LIPIcs.CCC.2021.36

[14] A. Chattopadhyay, Y. Dahiya, and M. Mahajan, "Query complexity of search problems," *Electronic Colloquium on Computational Complexity* (ECCC), 2023. [Online]. Available: https://eccc.weizmann.ac.il/report/2023/039/

[15] S. Goldwasser, O. Grossman, S. Mohanty, and D. P. Woodruff, "Pseudo-deterministic streaming," in *Innovations in Theoretical Computer Science* (ITCS), 2020, pp. 79:1–79:25. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITCS.2020.79

[16] V. Braverman, R. Krauthgamer, A. Krishnan, and S. Sapir, "Lower bounds for pseudo-deterministic counting in a stream," *CoRR*, vol. abs/2303.16287, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.16287

[17] S. Goldwasser and O. Grossman, "Bipartite perfect matching in pseudo-deterministic NC," in *International Colloquium on Automata, Languages, and Programming* (ICALP), 2017, pp. 87:1–87:13. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2017.87

[18] S. Ghosh and R. Gurjar, "Matroid intersection: A pseudo-deterministic parallel reduction from search to weighted-decision," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (APPROX/RANDOM), 2021, pp. 41:1–41:16. [Online]. Available: https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.41

[19] I. C. Oliveira and R. Santhanam, "Pseudo-derandomizing learning and approximation," in *International Conference on Randomization and Computation* (RANDOM), 2018, pp. 55:1–55:19. [Online]. Available: https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2018.55

[20] I. C. Oliveira, "Randomness and intractability in Kolmogorov complexity," in *International Colloquium on Automata, Languages, and Programming* (ICALP), 2019, pp. 32:1–32:14. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2019.32

[21] Z. Lu, I. C. Oliveira, and R. Santhanam, "Pseudodeterministic algorithms and the structure of probabilistic time," in *ACM Symposium on Theory of Computing* (STOC), 2021, pp. 303–316. [Online]. Available: https://doi.org/10.1145/3406325.3451085

[22] O. Grossman and Y. P. Liu, "Reproducibility and pseudo-determinism in Log-Space," in *Symposium on Discrete Algorithms* (SODA), 2019, pp. 606–620. [Online]. Available: https://doi.org/10.1137/1.9781611975482.38

[23] S. Goldwasser, O. Grossman, and D. Holden, "Pseudo-deterministic proofs," in *Innovations in Theoretical Computer Science,* (ITCS), 2018, pp. 17:1–17:18. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITCS.2018.17

[24] M. X. Goemans, S. Goldwasser, and D. Holden, "Doubly-efficient pseudo-deterministic proofs," *Electronic Colloquium on Computational Complexity* (ECCC), vol. 26, p. 135, 2019. [Online]. Available: https://eccc.weizmann.ac.il/report/2019/135

[25] O. Grossman, "Finding primitive roots pseudo-deterministically," *Electronic Colloquium on Computational Complexity* (ECCC), vol. 22, p. 207, 2015. [Online]. Available: https://eccc.weizmann.ac.il/report/2015/207

[26] P. Dixon, A. Pavan, and N. V. Vinodchandran, "On pseudodeterministic approximation algorithms," in *Symposium on Mathematical Foundations of Computer Science* (MFCS), 2018, pp. 61:1–61:11. [Online]. Available: https://doi.org/10.4230/LIPIcs.MFCS.2018.61

[27] B. Berger and Z. Brakerski, "Zero-knowledge protocols for search problems," in *International Conference on Security and Cryptography for Networks* (SCN), 2018, pp. 292–309. [Online]. Available: https://doi.org/10.1007/978-3-319-98113-0_16

[28] O. Goldreich, "Multi-pseudodeterministic algorithms," *Electronic Colloquium on Computational Complexity* (ECCC), vol. 26, p. 12, 2019. [Online]. Available: https://eccc.weizmann.ac.il/report/2019/012

[29] P. Dixon, A. Pavan, and N. V. Vinodchandran, "Complete problems for multi-pseudodeterministic computations," in *Innovations in Theoretical Computer Science* (ITCS), 2021. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITCS.2021.66

[30] P. Dixon, A. Pavan, J. V. Woude, and N. V. Vinodchandran, "Pseudodeterminism: promises and lowerbounds," in *ACM Symposium on Theory of Computing* (STOC), 2022, pp. 1552–1565. [Online]. Available: https://doi.org/10.1145/3519935.3520043

[31] J. V. Woude, P. Dixon, A. Pavan, J. Radcliffe, and N. V. Vinodchandran, "The geometry of rounding," *Electronic Colloquium on Computational Complexity* (ECCC), vol. TR22-160, 2022. [Online]. Available: https://eccc.weizmann.ac.il/report/2022/160

[32] S. Chakraborty, M. Prabhakaran, and D. Wichs, "A map of witness maps: New definitions and connections," Cryptology ePrint Archive, Paper 2023/343, 2023. [Online]. Available: https://eprint.iacr.org/2023/343

[33] Z. Lu and I. C. Oliveira, "Theory and applications of probabilistic Kolmogorov complexity," *Bull. EATCS*, vol. 137, 2022. [Online]. Available: http://bulletin.eatcs.org/index.php/beatcs/article/view/700

[34] R. Impagliazzo and A. Wigderson, "Randomness vs time: Derandomization under a uniform assumption," *Journal of Computer and System Sciences*, vol. 63, no. 4, pp. 672–688, 2001. [Online]. Available: https://doi.org/10.1006/jcss.2001.1780

[35] L. Trevisan and S. P. Vadhan, "Pseudorandomness and average-case complexity via uniform reductions," *Computational Complexity*, vol. 16, no. 4, pp. 331–364, 2007. [Online]. Available: https://doi.org/10.1007/s00037-007-0233-x

[36] R. M. Karp and R. J. Lipton, "Some connections between nonuniform and uniform complexity classes," in *ACM Symposium on Theory of Computing* (STOC), 1980, pp. 302–309. [Online]. Available: https://doi.org/10.1145/800141.804678

[37] R. Kannan, "Circuit-size lower bounds and non-reducibility to sparse sets," *Inf. Control.*, vol. 55, no. 1-3, pp. 40–56, 1982. [Online]. Available: https://doi.org/10.1016/S0019-9958(82)90382-5

[38] R. Impagliazzo, V. Kabanets, and A. Wigderson, "In search of an easy witness: exponential time vs. probabilistic polynomial time," *J. Comput. Syst. Sci.*, vol. 65, no. 4, pp. 672–694, 2002. [Online]. Available: https://doi.org/10.1016/S0022-0000(02)00024-7

[39] P. B. Miltersen, N. V. Vinodchandran, and O. Watanabe, "Super-polynomial versus half-exponential circuit size in the exponential hierarchy," in *International Computing and Combinatorics Conference* (COCOON), ser. Lecture Notes in Computer Science, vol. 1627. Springer, 1999, pp. 210–220. [Online]. Available: https://doi.org/10.1007/3-540-48686-0_21

[40] N. Nisan and A. Wigderson, "Hardness vs randomness," *Journal of Computer and System Sciences*, vol. 49, no. 2, pp. 149–167, 1994. [Online]. Available: https://doi.org/10.1016/S0022-0000(05)80043-1

[41] L. Chen, R. D. Rothblum, and R. Tell, "Unstructured hardness to average-case randomness," in *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 2022, pp. 429–437. [Online]. Available: https://doi.org/10.1109/FOCS54457.2022.00048

[42] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," *Journal of the ACM*, vol. 62, no. 4, pp. 27:1–27:64, 2015. [Online]. Available: https://doi.org/10.1145/2699436

[43] O. Goldreich, "On the doubly-efficient interactive proof systems of GKR," *Electronic Colloquium on Computational Complexity* (ECCC), vol. 24, p. 101, 2017. [Online]. Available: https://eccc.weizmann.ac.il/report/2017/101

[44] M. Sudan, L. Trevisan, and S. P. Vadhan, "Pseudorandom generators without the XOR lemma," *J. Comput. Syst. Sci.*, vol. 62, no. 2, pp. 236–266, 2001. [Online]. Available: https://doi.org/10.1006/jcss.2000.1730

[45] V. Shoup, "Searching for primitive roots in finite fields," *Mathematics of Computation*, vol. 58, no. 197, pp. 369–380, Jan. 1992.

[46] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.*, vol. 13, no. 4, pp. 850–864, 1984. [Online]. Available: https://doi.org/10.1137/0213053

[47] A. C. Yao, "Theory and applications of trapdoor functions (extended abstract)," in *IEEE Symposium on Foundations of Computer Science* (FOCS), 1982, pp. 80–91. [Online]. Available: https://doi.org/10.1109/SFCS.1982.45