HARDNESS-RANDOMNESS TRADEOFFS FOR ALGEBRAIC COMPUTATION

Mrinal Kumar* Ramprasad Saptharishi†

Abstract

The interplay between the question of proving lower bounds and that of derandomization, in various settings, is one of the central themes in complexity theory. In this survey, we explore this phenomenon in the area of algebraic complexity theory. Enroute, we discuss some of the classical results, as well as some recent ones, that establish a close connection between the question of proving algebraic circuits lower bounds and that of derandomizing polynomial identity testing. We also talk about an application of this machinery to the phenomenon of *bootstrapping* for polynomial identity testing and mention some open problems.

1 Introduction

The quest of proving lower bounds for explicit functions for various computational models and that of obtaining deterministic algorithms for problems which can be solved efficiently with randomness are two of the most fundamental themes of study in theoretical computer science in general, and computational complexity in particular. The P vs NP question is perhaps the most well known instance of the former question where as the P vs BPP question is a canonical instance of the latter. An equally intriguing and significant line of research is the interaction of these themes of hardness and randomness in various contexts. We begin with a brief (and rather incomplete) discussion of some of the classical results of this kind in the set up of Boolean computation before moving on to the algebraic set up, which is the focus of this survey.

*Department of Computer Science & Engineering, IIT Bombay, Mumbai, India. Email: mrinal@cse.iitb.ac.in.

†School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India. Email: ramprasad@tifr.res.in. Research supported by Ramanujan Fellowship of DST

1.1 Hardness vs Randomness in the Boolean setting

The connection between hardness and randomness in the Boolean setting goes via the notion of a pseudorandom generator, which we now define.

Definition 1.1 (Pseudorandom generators)**.** *A family of functions* $\{\mathsf{Gen}_k : \{0,1\}^k \to \{0,1\}^{n(k)}\}$ *is said to be an* ε*-pseudorandom generator (PRG) with seed length* k *and time complexity* $t(k)$ *for a class* $\mathcal{C}$ *of Boolean circuits if, for all sufficiently large* $k \in \mathbb{N}$*, we have that* $\mathsf{Gen}_k(\mathbf{x})$ *can be computed by a deterministic Turing machine in time* $t(k)$*, and for every* $C \in \mathcal{C}$ *which takes* $n(k)$ *inputs,*

$$\left| \mathop{\mathbb{E}}_{\mathbf{y}\in\{0,1\}^n}(C(\mathbf{y})) - \mathop{\mathbb{E}}_{\mathbf{x}\in\{0,1\}^k}(C(\mathsf{Gen}_k(\mathbf{x}))) \right| < \varepsilon$$

$\Diamond$

Intuitively, the idea is that for a random string $\mathbf{x} \in \{0,1\}^k$, the behavior of a circuit in the class $\mathcal{C}$ on input $\mathsf{Gen}_k(\mathbf{x})$ closely captures its behavior on a completely random input. For k much smaller than n, this gives a substantial saving on the number of random bits needed to gauge the behavior of any circuit $C \in \mathcal{C}$ on average. For instance, if $C \in \mathcal{C}$ is a bounded error randomized algorithm which we seek to derandomize, then instead of feeding it a random seed, we can run the algorithm for all 2^k strings in the output of the generator Gen_k, and output the majority answer. The guarantee on the generator ensures the correctness of this procedure. Thus, it is desirable to have k, the seed length of the generator to be as small as possible when compared to n.

One of the most well known results in the research on hardness vs randomness is the landmark work of Nisan & Wigderson [22]. These results showed that the question of proving lower bounds on the size of Boolean circuits which approximate an *explicit*[1] function is essentially equivalent to the question of constructing fast enough *pseudorandom generators* for Boolean circuits.

This framework together with the average case lower bounds for the function PARITY for constant depth Boolean circuits (AC^0 circuits) due to Håstad [13] then immediately implied a construction of pseudorandom generators for AC^0 circuits with polylogarithmic seed length.

It was noted in [22] that while the results in [21, 22] were perhaps one of the first instances in computational complexity where lower bounds had been used for pseudorandomness, this idea had its roots in Cryptography.

[1] Here, by explicit, we just mean that the function can be computed in exponential time in its input length.

Nisan and Wigderson [22] attribute this idea to a work of Shamir [26] from a few years earlier, where he had used the security of the RSA cryptosystem to construct cryptographic pseudorandom generators (a close cousin of the objected defined in Definition 1.1). In another work around the same time, Blum & Micali [4] had shown a construction of cryptographic pseudorandom generators using the hardness of the Discrete Logarithm function. This line of work saw a vast generalization in the work of Yao [29] and that of Håstad, Impagliazzo, Levin and Luby [14] where it was shown that cryptographic pseudorandom generators can be constructed using *any* one way function.

It is worth noting that the pseudorandom generators constructed in this line of work were cryptographic in nature, where the complexity of the generator itself was much smaller than the complexity of the adversary it is fools. On the other hand, our focus in this survey is on the complexity theoretic setting, where the complexity of the generator itself is allowed to be larger than the complexity of the circuit classes it is expected to fool. In particular, due to the weaker requirements, one can hope to have constructions of complexity theoretic pseudorandom generators from much weaker assumptions than the existence of one way functions. This is all the more true if we are looking for pseudorandom generators for very structured classes of Boolean functions, and Nisan and Wigderson's generator [22] for for AC^0 circuits is one such instance. For a more thorough and detailed discussion on pseudorandom generators of both complexity theoretic and cryptographic flavor, we refer the reader to an excellent survey by Oded Goldreich [12].

1.2 Hardness vs Randomness in Algebraic Complexity

In algebraic complexity, the primary objects of study are multivariate polynomials, and the key model of computation is that of an *algebraic circuit*, which we now define.

Definition 1.2 (Algebraic circuits)**.** *An* algebraic circuit *is a directed acyclic graph whose leaves (in-degree zero nodes) are labeled by formal variables or field constants, and internal gates are labeled by $+$ and $\times$. Semantically, such a graph naturally computes polynomials of the input variables at each gate in the natural way. The root(s) (nodes of out-degree zero) are set to be the output(s) of the circuit.*

The size of an algebraic circuit is defined as the number of gates in the graph. ◊

In the algebraic setting, the hardness randomness tradeoffs study the connection between the questions of proving algebraic circuit lower bounds

for an explicit polynomial family and that of designing efficient deterministic algorithms for the question of Polynomial Identity Testing (PIT) i.e. the task of testing if a given algebraic circuit computes the identically zero polynomial. The PIT question is often asked in two flavours — the *white-box* setting where an algorithm is allowed to look inside the wiring of the input circuit to test its zeroness, or the *blackbox* setting where the algorithm just has query access to the algebraic circuit. The well known *Polynomial Identity Lemma*[2] (see Lemma 4.1) gives a very simple randomized algorithm for PIT in the blackbox setting, which proceeds by just querying the given input circuit at a random point on a large enough grid. Derandomizing this algorithm is one of the most fundamental open questions in complexity theory. And as we shall see, this question has intimate connections to the question of proving algebraic circuit lower bounds.

One of the first connections between PIT and algebraic circuit lower bounds was discovered in the work of Heintz and Schnorr [15], who in some sense formulated the question of PIT in the current form, even though in its various incarnations, the problem had been studied in multiple contexts since a much earlier time (e.g. Lemma 4.1)) and asked for efficient deterministic algorithms for this problem. They also showed that given an efficient deterministic algorithm for PIT in the blackbox setting, it is easy to give an explicit construction of polynomial families which do not have small algebraic circuits. In other words, they showed that derandomizing PIT (in the blackbox setting) implies algebraic circuit lower bounds!

This connection between hardness and randomness in the algebraic setting was further strengthened in an very influential work of Kabanets and Impagliazzo [18], who showed that even slightly non-trivial PIT algorithms (even in the supposedly weaker whitebox setting) implies circuit lower bounds[3] far beyond our current state of art. They also showed a connection in the opposite direction, by showing that an algebraic analogue of the Nisan-Wigderson pseudorandom generator can be used to get improved PIT algorithms from algebraic circuit lower bounds.

1.3 Organization of the survey

The goal of this survey is to discuss these results as well as a few more, and to mention some of the questions that remain open in this line of re-

[2] often referred to as a (suitable subset of) Ore-DeMillo-Lipton-Schwartz-Zippel lemma; the name 'Polynomial identity lemma' for this classical result is attributed to László Babai.

[3] Kabanets and Impagliazzo [18] showed that we either get better Boolean circuit lower bounds or better algebraic circuit lower bounds as a consequence.

search. In particular, we elaborate on the consequences of deterministic PIT algorithms towards lower bounds for algebraic circuits in Section 3 and discuss how algebraic circuit lower bounds imply non-trivial deterministic PIT in Section 4. To prepare for these, we set up some notation and discuss some preliminaries in Section 2.

2 Notations and Preliminaries

- We use boldface letters like $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ to denote tuples of variables. The number of variables in such a tuple will be clear from from the context.
- Unless otherwise stated, we confine our discussion to algebraic computation over fields of characteristic zero.
- We have already defined algebraic circuits in Definition 1.2. An algebraic circuit is said to be an algebraic formula if the underlying graph is a tree.
- Sometimes, it is convenient to also allow field constants on the edges of an algebraic circuit to indicate a scaling of the argument before feeding in to a gate. This allows, for example, the computation of arbitrary linear combinations of gates using a single $+$ gate.
- A polynomial $f(\mathbf{x})$ is said to have *individual degree* at most d, if for every variable x_i, the degree of f, when viewed as a univariate in x_i (with coefficients coming from the field of rational functions in the remaining variables) is at most d.
- A function $f : \mathbb{N} \to \mathbb{N}$ is said to be at most $\mathsf{poly}(n)$ if there exists a constant c such that for all large enough n, $f(n) \leq n^c$.

Homogeneity and Homogenisation

Definition 2.1 (Homogeneity)**.** *A polynomial $f(\mathbf{x})$ is said to be homogeneous if every monomial of f has the same total degree.*

An algebraic circuit is said to be homogeneous if every gate in the compute computes a homogeneous polynomial. $\Diamond$

A well known result, due to Strassen, is that homogeneous components of polynomials with small algebraic circuits have *homogenenous* circuits small size.

Lemma 2.2. *Let $f(\mathbf{x})$ be a (not necessarily homogeneous) polynomial of degree at most d that is computable by a circuit of size s and let $g(\mathbf{x})$ be the homogeneous component of f of degree equal to k. Then, there is a homogeneous circuit of size at most $O(sk^2)$ that computes $g(\mathbf{x})$.* □

Hitting sets

As alluded to earlier, polynomial identity testing is typically studied in two flavours — whitebox and blackbox PITs. In the blackbox setting, we only have query access to the given circuit. Hence, blackbox PITs are essentially constructions of *hitting sets* for the class.

Definition 2.3 (Hitting sets)**.** *A* hitting set *for a class $\mathcal{C}$ of n-variate polynomials is a set $H \subseteq \mathbb{F}^n$ such that, for any $0 \neq P(\mathbf{x}) \in \mathcal{C}$, we have $P(\mathbf{a}) \neq 0$ for some $\mathbf{a} \in H$.*

*We shall say that a hitting set is $t(n)$-*explicit *if H can be computed in $t(n)$ time.* ◊

A variant of the notion of a pseudorandom generator, called a *hitting-set generator* will be a key object of study in Section 4. We postpone its definition to the relevant section.

VP and VNP

Two natural complexity classes of polynomials which will find some mentions in this survey are VP and VNP. We define them slightly informally below, and refer the reader to the surveys of Shpilka and Yehudayoff [27] or that of Saptharishi [24] for formal defintions of these complexity classes.

A family of polynomials $\{P_n\}$ is said to be in VP, if there is a fixed constant c such that for every large enough n, P_n has degree at most n^c and can be computed by an algebraic circuit of size at most n^c. In a nutshell, upto the degree condition, VP is a nonuniform algebraic analogue of the complexity class P.

A family of polynomials $\{P_n\}$ is said to be in VNP, if there is a family of polynomials $\{Q_m\}$ in VP and a fixed constant c such that for every large enough n,

$$P_n(x_1, x_2, \ldots, x_n) = \sum_{\mathbf{a} \in \{0,1\}^{n^c}} Q(x_1, x_2, \ldots, x_n, a_1, a_2, \ldots, a_{n^c}) \,.$$

VNP can be thought of an a nonuniform algebraic analogue of NP and in this sense, the VP vs VNP question is the nonuniform algebraic analogue of the P vs NP question.

3 Lower bounds from deterministic PIT

In this section, we focus our attention on results which show that non-trivial deterministic PIT algorithms imply algebraic circuit lower bounds. As alluded to in the introduction, the first connection of this flavour was a result of Heintz and Schnorr [15] (also observed by Agrawal [1]) who showed that non-trivial deterministic PIT algorithms in the blackbox setting imply algebraic circuit lower bounds. Showing a similar lower bound implication of non-trivial PIT algorithms in the whitebox settting turned out to be much more non-trivial, and this was eventually done in an influential work of Kabanets and Impagliazzo [18].

We start with a discussion of the result of Heintz and Schnorr [15] before moving on to the results in [18].

3.1 Lower bounds from blackbox PIT algorithms

Theorem 3.1 (Heintz and Schnorr [15], Agrawal [1])**.** *Let $H(n,d,s)$ be an explicit hitting set for circuits of size s, degree d in n variables. Then, for every $k \leq n$ and d' such that $d'k \leq d$ and $(d'+1)^k > |H(n,d,s)|$, there is a nonzero polynomial on n variables and individual degree d' that vanishes on the hitting set $H(n,d,s)$, and hence cannot be computed by a circuit of size s.*

Proof-sketch. From the definition of a hitting set, any nonzero polynomial in the class of interest must evaluate to a nonzero value at some point on the hitting set. Hence, any nonzero polynomial that evaluates to zero on *every* point on the hitting set must necessarily not lie in the class. We can find such a polynomial via interpolation, or solving a system of linear equations, as we now elaborate on.

For every k, d', a k-variate polynomial P of individual degree at most d' has $(d'+1)^k$ coefficients. Let us consider all these coefficients to be distinct formal variables, which have to be determined. Now, for every $\mathbf{a} \in \mathbb{F}^k$, if such a polynomial evaluates to 0 at $\mathbf{a}$, then its coefficients must satisfy a homogeneous linear constraint. Adding such a homogeneous linear constraint for every point $\mathbf{a} \in H(n,d,s)$ gives us a system of $|H(n,d,s)|$ homogeneous linear constraints on $(d'+1)^k$ variables. If $(d'+1)^k < |H(n,d,s)|$, then this system of homogeneous linear equations is under-determined, and has a nonzero solution. Moreover, if $k \leq n$, $d'k \leq d$, then any such nonzero solution gives us a polynomial of degree at most d on (at most) n variables which vanishes on every point of the hitting set $H(n,d,s)$, and thus, cannot be computed by an algebraic circuit of size at most s. □

Theorem 3.1 shows a direct connection between PIT algorithms and algebraic circuit lower bounds. For instance, if we have a polynomial size hitting set for circuits of size n^5, then we can construct the interpolating polynomial and that would *require* circuits of size more than n^5. Furthermore, such a polynomial has a circuit of size $\mathsf{poly}(n)$. If we could find an interpolating polynomial for a hitting set of size $s = n^{\omega(1)}$, then we would have obtained superpolynomial lower bounds. Such a polynomial can certainly be found in exponential time, but is the interpolating polynomial for a superpolynomial sized hitting set in VNP? If yes, the existence of non-trivial hitting sets would imply that $\mathsf{VP} \neq \mathsf{VNP}$. This following very natural question continues to be open.

Question 3.2 (Open problem 17 in Shpilka-Yehudayoff [27])**.** *Does a polynomial time deterministic PIT algorithm in the blackbox setting imply that* $\mathsf{VP} \neq \mathsf{VNP}$*?*

As Shpilka and Yehudayoff note in their survey [27], this question also remains open for various sub-classes of algebraic circuits. In particular, if we have efficient deterministic PIT algorithm in the blackbox setting for algebraic formulas, does this imply that the permanent polynomial does not have small formulas?

3.2 Lower bounds from whitebox PIT algorithms

Theorem 3.1 saw a substantial extension in the work of Kabanets and Impagliazzo [18] nearly twenty years after the work of Heintz and Schnorr [15]. Kabanets and Impagliazzo [18] showed that non-trivial deterministic algorithms for PIT, even in the whitebox setting have remarkable consequences towards proving new lower bounds. They showed that such algorithms show that either NEXP (Non-deterministic Exponential Time) does not have polynomial size Boolean circuits or the Permanent polynomial does not have polynomial size algebraic circuits. Connecting whitebox PIT algorithms to lower bounds turned out to be far more complicated than connecting blackbox PIT algorithms to lower bounds and indeed, the proof of [18] is much more involved than that in [15]. In particular, it relied on a set of highly non-trivial results in complexity theory, whereas as we already saw, the proof of Theorem 3.1 is completely elementary. A consequence of this potpourri of ideas in the proof of Kabanets and Impagliazzo is the somewhat surprising combination of Boolean and algebraic circuit lower bounds in their conclusion. In addition to their technical contribution, the results in [18] also contributed to a slight change in the perception about the hardness of derandomizing PIT (a problem believed to be

approachable) by connecting it to proving strong circuit lower bounds (a problem widely perceived to be much harder).

We now state the result of Kabanets and Impagliazzo and outline the main ideas in its proof.

Theorem 3.3 (Kabanets and Impagliazzo [18])**.** *If there is a deterministic algorithm for polynomial identity testing (PIT) which runs in subexponential time (or even that PIT* $\in$ NSUBEXP*), then at least one of the following is true.*

- *There is a language in non-deterministic exponential time which does not have polynomial size Boolean circuits i.e.,* NEXP *is not contained in* P/poly*.*
- *The permanent polynomial does not have polynomial size*[4] *algebraic circuits.*

Remark. *There has unfortunately been a slight overloading of the term "subexponential". In classical complexity, a function* $f : \mathbb{N} \to \mathbb{R}$ *is said to be* subexponential *if* $f(n) = 2^{o(n^\varepsilon)}$ *for* every $\varepsilon > 0$*. In algebraic complexity, sometimes* subexponential *is used to refer to functions* $f(n) = 2^{O(n^\varepsilon)}$ *for some* $0 < \varepsilon < 1$*. In this article, we shall stick to the classical definition of subexponential wherein we mean*

$$f(n) \text{ is subexponential} \quad \Leftrightarrow \quad \forall \varepsilon > 0\,,\ f(n) = 2^{o(n^\varepsilon)}.$$

$\Diamond$

Proof-sketch. The proof is via contradiction. Let us assume that we have a subexponential time deterministic algorithm for PIT, the permanent has polynomial sized algebraic circuits, and that NEXP $\subseteq$ P/poly. A result of Impagliazzo, Kabanets and Wigderson [16] shows that if NEXP $\subseteq$ P/poly implies NEXP $=$ $\mathsf{P}^{\#\mathsf{P}}$. With this in hand, and assuming that permanent has small circuits and that PIT has subexponential time deterministic algorithm, we shall contradict the non-deterministic time hierarchy theorem.

Consider an arbitrary language L in NTIME(2^n). The goal is to design a much faster algorithm to decide membership in L. Assuming that NEXP $\subseteq$ P/poly, the permanent has small algebraic circuits and that PIT has a subexponential time deterministic algorithm, we now outline such a non-deterministic algorithm for L.

[4]There is a subtle point to be made here. In this theorem, the size of the circuit is measured in terms of the total description size, which includes the constants possibly appearing on the wires. Typically, in algebraic complexity, size is defined only as the *number* of gates and does not include the bit-complexity of the constants.

As discussed above, $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ implies that $\mathsf{NEXP} = \mathsf{P}^{\#\mathsf{P}}$. Thus, membership in L (which is in NEXP) can be decided by a polynomial time algorithm with a #P oracle. Due to the #P completeness of the permanent, without loss of generality, we can assume that each of the oracle queries involves a permanent computation. We now outline a procedure to simulate this $\mathsf{P}^{\#\mathsf{P}}$ algorithm by a non-deterministic algorithm which runs in subexponential time.

- Let $m = \mathsf{poly}(n)$ be the largest sized permanent queries made by the $\mathsf{P}^{\#\mathsf{P}}$ algorithm to solve L on length n inputs. Guess polynomial sized algebraic circuits $C_1, \ldots, C_m$ such that C_a computes the $a \times a$ permanent.
- Using the *self-reducibility* of the permanent and the faster algorithm for PIT, check that $C_1(x) = x$ and

$$\forall a \in \{2, \ldots, m\}, i \in [a] \ : \ C_a(M) - \sum_{j=1}^{a} M_{i,j} \cdot C_{a-1}(\hat{M}_{i,j}) \equiv 0,$$

  where $M_{i,j}$ refers to the entry at (i, j) and $\hat{M}_{i,j}$ refers to the matrix obtained by removing the i-th row and j-th column. If all the above tests succeed, use the circuits $C_1, \ldots, C_m$ to simulate the $\mathsf{P}^{\#\mathsf{P}}$ algorithm for L.

Overall, the above sketch simulates any $L \in \mathsf{NTIME}(2^n)$ in non-deterministic subexponential time, and this contradicts the non-deterministic time hierarchy theorem. □

4 Deterministic PIT from lower bounds

We now move on to the *other* direction of this connection between hardness and randomness and discuss the design of non-trivial deterministic PIT algorithms for polynomials with small circuits assuming that we have access to explicit hard polynomials. This direction of research, as noted in the introduction started in the 80's in the set up of Cryptography [26, 4, 29, 14] where it was shown that pseudorandom generators can be constructed assuming the existence of one way functions. A strong motivation for subsequent research on this problem has been driven by the goal of obtaining the strongest possible derandomization conclusion from the weakest possible hypothesis. Indeed, the existence of one way functions is a very strong assumption, which in particular implies that $\mathsf{P} \neq \mathsf{NP}$

and it is desirable to weaken this to a lower bound hypothesis to something that is more approachable.

In algebraic complexity, an obvious plausible candidate statement to aim for would be to show that there is an efficient deterministic algorithm for PIT assuming that we have algebraic circuit lower bounds. It would be even nicer to have a fine grained connection where to obtain PIT for a class $\mathcal{C}$ of algebraic circuits, we *only* need explicit lower bounds for a class $\mathcal{C}'$ which is *close* to $\mathcal{C}$. For instance, we could take both $\mathcal{C}$ and $\mathcal{C}'$ to be small algebraic formulas, and ask whether super polynomial lower bounds for algebraic formula implies efficient deterministic PIT algorithms for small formulas. In fact, there is nothing specific about algebraic formulas here and we could ask the same question for other natural classes of algebraic circuits, like circuits of bounded depth and algebraic branching programs.

As of now, most of these questions remain far from settled. Nevertheless, as we shall see in the next few sections, substantial progress has been made towards them. We now discuss at a high level, the ideas involved in the results which obtain deterministic PIT from algebraic circuit lower bounds. We start with the crucial notion of a *hitting-set generator*. Before moving to a formal definition, we first discuss the role this notion plays in the goal of obtaining an efficient deterministic algorithm for PIT.

The most naïve (although, inefficient) deterministic algorithm for PIT is an immediate consequence of the following well known Polynomial Identity Lemma, which can be seen as a higher dimensional generalization of the fact that any nonzero degree d univariate polynomial over a field $\mathbb{F}$ has at most d zeros. The lemma has been discovered multiple times and is attributed to many different authors (see Bishnoi *et al.* [3] and also footnote 2).

Lemma 4.1 (Polynomial identity Lemma [23, 7, 25, 30])**.** *Let $\mathbb{F}$ be an arbitrary field with at least $d+1$ elements and let $P \in \mathbb{F}[\mathbf{x}]$ be a nonzero polynomial of total degree at most d on n variables. Then, for every subset S of $\mathbb{F}$ with $|S| \geq d+1$, there is an element $(b_1, b_2, \ldots, b_n)$ in the set $S^n = \{(a_1, a_2, \ldots, a_n) : \forall i \in [n], a_i \in S\}$ such that $P(b_1, b_2, \ldots, b_n)$ is nonzero.*

The lemma immediately gives a way to deterministically test if a given algebraic circuit computing a degree d polynomial is nonzero — simply evaluate the polynomial at *every* point on the grid S^n for an arbitrary subset S of the field of size at least $d+1$ [5]. Unfortunately, this deterministic algorithm makes $(d+1)^n$ queries to the circuit, and hence, is quite inefficient. In some sense, we have not used anything apart from the degree of

[5] We will always assume that the underlying field is large enough, else we will work over a large enough extension.

the input circuit so far in the algorithm, and in this generality, it is not hard to argue that such a superpolynomial running time is necessary if we confine ourselves to making non-adaptive queries to the circuit[6]. However, we have more information about the input at our hands, one of the most significant being the fact that the input polynomial has a small circuit, and in this setting a much faster blackbox algorithm is conceivable.

It is worth noting that the running time of the algorithm based on Lemma 4.1 is exponential in n, the number of variables. At a high level, the conditional PIT algorithms discussed later in this survey, and indeed most of the PIT algorithms that we know in literature [7] essentially aim to construct an efficiently computable polynomial map where every *nonzero* polynomial with a small circuit is mapped to a *nonzero* polynomial with much fewer variables and not too high degree. The next step is to just query the polynomial output by this variable reduction procedure on a grid of appropriate size. The correctness of the algorithm is ensured by the guarantees on the variable reduction map, and Lemma 4.1. The absolutely crucial point is that size the polynomial output by the variable reduction procedure has very few variables and the degree hasn't increased much in the process, the number of query points given by Lemma 4.1 is just exponential in a much smaller quantity than the total number of variables in the input, which might still be tolerable if the parameters are set appropriately. To make this discussion formal, we start with the following definition.

Definition 4.2 (Hitting-set generators)**.** *A polynomial map $G : \mathbb{F}^k \to \mathbb{F}^n$ given by $G(z_1, z_2, \ldots, z_k) = (g_1(\mathbf{z}), g_2(\mathbf{z}), \ldots, g_n(\mathbf{z}))$ is said to be a hitting-set generator (HSG) for a class $\mathcal{C} \subseteq \mathbb{F}[x_1, x_2, \ldots, x_n]$ of polynomials if for every nonzero $Q \in \mathcal{C}$, we have that $Q \circ G = Q(g_1, g_2, \ldots, g_n)$ is nonzero.*

*Here k is called the seed length of the HSG and $(n - k)$ is its stretch. The maximum of the degrees of $g_1, g_2, \ldots, g_n$ is the degree of the HSG and it is said to be $t(n)$-*explicit *if, for any input $\mathbf{a} \in \mathbb{F}^k$, we can compute $G(\mathbf{a})$ in deterministic time $t(n)$.* ♢

Intuitively, for any class $\mathcal{C}$ of polynomials, it is desirable to have an HSG which is efficiently computable, has small seed length and degree

[6] An n-variate degree d polynomial has $\binom{n+d}{d}$ coefficients, and for any set H of points in $\mathbb{F}^n$ of size at most $\binom{n+d}{d} - 1$, there exists a nonzero n-variate polynomial of degree at most d which vanishes on all of H. Thus, an algorithm which decides nonzeroness only on the basis of the values of the input polynomial on points in H makes an error on such a polynomial.

[7] Many of them can be viewed in this form, even though their original presentation might be in a slightly different language.

and has a large stretch. An intuitively nice range of parameters is to think of k as being sublinear in n, degree of G to be $\mathsf{poly}(n)$ and G is $\mathsf{poly}(n)$-explicit.

The utility of an HSG to a PIT algorithm, which we discussed earlier in this section stems from the following observation.

Lemma 4.3 (HSGs to hitting sets)**.** *Let $G : \mathbb{F}^k \to \mathbb{F}^n$ be a degree D, $t(n)$-explicit hitting-set generator for a class $\mathcal{C}$ of n-variate polynomials of degree at most d. Then, there is a deterministic algorithm which runs in time $(d(D+1))^k\, t(n)$ and outputs a set $H \subseteq \mathbb{F}^n$ of size at most $(d(D+1))^k$ such that for every nonzero $Q \in \mathcal{C}$, there is an $\mathbf{a} \in H$ that satisfies $Q(\mathbf{a}) \neq 0$.*

Thus, given a hitting-set generator, we just need to query the given input circuit at points on the corresponding hitting set H to determine nonzeroness. Our goal, for the rest of this section is to construct hitting-set generators for polynomial size algebraic circuits assuming that we have an explicit hard polynomial family. We discuss two seemingly different constructions, the first is a beautiful result of Kabanets and Impagliazzo [18] which can be thought of as an algebraic analogue of the classical pseudorandom generator of Nisan and Wigderson [22] in the Boolean setting. This construction inherits some of the inherent combinatorial nature of the construction of Nisan and Wigderson, and in particular relies on the notion of combinatorial designs, which are large uniform set families with small pairwise intersections. The second construction that we discuss is due to a recent work of Guo, Solomon and the authors [10] and appears to be more algebraic in nature. Even though these two hitting-set generators differ from each other, they can both be viewed as generalizations of the following very simple hitting-set generator which stretches the seed by just one. Moreover, the overall structure of the proof of correctness of this simple construction will also be instructive and many of the ideas from here will also be present in the generalizations. We start with the definition of the toy generator.

4.1 Warming up: An HSG of stretch one

The first thing to attempt is to build a hitting-set generator with the smallest non-trivial stretch. This construction is inspired by an important concept from Boolean pseudorandomness called "next-bit-unpredictability".

Next bit unpredictability. An important concept, often discussed in the cryptography setting, is the notion of next bit unpredictability. This can be

seen in the work of Yao [29] and in the the context of the Nisan-Wigderson pseudorandom generators [22]. From [29], it is known that a sequence of bits is pseudorandom if and only if no small circuit can *predict* the next bit in the sequence with non-trivial advantage.

The following definition can be seen as an algebraic analogue of coming up with a next-bit-unpredictable sequence by using a *hard function* to create the correlation.

Definition 4.4 (HSG of stretch 1)**.** *Let $P(\mathbf{z}) \in \mathbb{F}[z_1, \ldots, z_k]$ be any k-variate polynomial. We define the map $\mathsf{Gen}_P^0 : \mathbb{F}^k \to \mathbb{F}^{k+1}$ as*

$$\mathsf{Gen}_P^0(\mathbf{z}) = (z_1, z_2, \ldots, z_k, P(\mathbf{z}))\,. \qquad \Diamond$$

In other words, the first k coordinates of the generator $G_0(P)$ are just the original variables, and the last coordinate is the polynomial $P(\mathbf{z})$. The utility of this toy construction stems from the following theorem.

Theorem 4.5 (Generator of stretch 1)**.** *Let $P(\mathbf{z}) \in \mathbb{F}[\mathbf{z}]$ be a k-variate polynomial that cannot be computed by algebraic circuits of size s. Then, for any circuit $C \in \mathbb{F}[y_1, \ldots, y_{k+1}]$ of size at most s' and degree d that satisfies $(s'd)^5 < s$, the following is true:*

$$C(\mathbf{y}) = 0 \iff C \circ \mathsf{Gen}_P^0(\mathbf{z}) = 0\,,$$

where, $C \circ \mathsf{Gen}_P^0(\mathbf{z})$ is the polynomial $C(z_1, z_2, \ldots, z_k, P(\mathbf{z}))$. Moreover, if P is $t(k)$-explicit, then $\mathsf{Gen}_P^0(\mathbf{z})$ is $O(t(k))$-explicit.

In other words, if we have a hard enough P of not-too-high degree, we have a hitting-set generator for $k+1$-variate circuits of low enough size and degree. Intuitively, if P is a hard polynomial, then a small circuit C should not be able to "discover" that z_{k+1} is actually $P(z_1, \ldots, z_k)$.

There is one key difference in the statement of Theorem 4.5 and these earlier instantiations of next-bit-unpredictability; in Theorem 4.5, we are relying on the fact that P cannot be computed by small circuits (i.e. a notion of worst case hardness) for $\mathsf{Gen}_P^0(\mathbf{z})$ to be a hitting-set generator, whereas in these earlier constructions, an average case hardness of P appears to be needed. In fact, it isn't clear what the right analogue of the notion of average case hardness would be for the algebraic setting since we are computing formal polynomials and not functions. We proceed with an overview of the proof now, and shall return to this discussion later in this section.

Proof of Theorem 4.5. We prove Theorem 4.5 via contradiction. Let us start with the assumption that $\mathsf{Gen}_P^0(\mathbf{z})$ is not an HSG as claimed in the theorem, that there is a $k+1$-variate circuit A of small enough size and degree,

which is nonzero, but vanishes on being composed with $\mathsf{Gen}_P^0(\mathbf{z})$. We will use this circuit A to show that there is a *small* circuit B which computes P, thus contradicting the alleged hardness of P.

For the sake of contradiction, let us assume that there is a circuit $A(\mathbf{y})$ on $k+1$ variables of size at most s' and degree d with $(s'd)^5 < s$ such that $A(\mathbf{y})$ is nonzero and $A \circ \mathsf{Gen}_P^0(\mathbf{z})$ is identically zero. Let us just relabel the variable y_i as z_i for every $i \in \{1, 2, \ldots, k\}$. This is an invertible linear transformation of the coordinates and clearly preserves the nonzeroness of A. Moreover, its circuit size and the degree also remain the same. Now composing A with the generator $\mathsf{Gen}_P^0(\mathbf{z})$ can be just viewed as replacing the variable y_{k+1} in $A(\mathbf{z}, y_{k+1})$ by $P(\mathbf{z})$. As per our assumption, $A(\mathbf{z}, y_{k+1})$ is nonzero whereas $A(\mathbf{z}, P(\mathbf{z}))$ is identically zero. This implies that $(y_{k+1} - P(\mathbf{z}))$ divides $A(\mathbf{z}, y_{k+1})$.

Using the fact that A has a small circuit, *if* we could conclude that P must have a small circuit, then we would be done. In particular, if it was the case that *factors* of polynomials computed by small circuits can indeed be computed by small circuits, then we would have our contradiction. Miraculously, this is indeed true!

A deeply influential and surprising result of Kaltofen [17] showed that if a low degree polynomial has a small circuit, then all[8] of its factors also have a small circuit. This truly algebraic fact is what distinguishes this proof from its counterparts in the Boolean setting. We state the following seemingly weaker version of Kaltofen's result, which will be sufficient for the applications here.

> **Theorem 4.6** (Kaltofen [17])**.** *Let $A(z_1, z_2, \ldots, z_k, y)$ be a nonzero polynomial of degree d such that A has an algebraic circuit of size $s \geq d$. Let $P(\mathbf{z})$ be a polynomial such that*
>
> $$A(\mathbf{z}, P(\mathbf{z})) = 0\,.$$
>
> *Then, P can be computed by an algebraic circuit of size at most[9] $(sd)^5$.*

By Theorem 4.6, this implies that P is computable by a circuit of size at most $(s'd)^5 < s$, thereby contradicting its hardness. □

[8] There are some technical issues in the case of small characteristic fields but let us not get into that for now.

[9] Bürgisser [5, Theorem 1.2] shows that the circuit complexity of factors is upper bounded by $O(sd^4 \log d)$; we are using $(sd)^5$ purely for brevity and the constant 5 is not important here. All the applications of this result in this survey just rely on this bound being $\mathsf{poly}(s, d)$.

4.2 The Kabanets-Impagliazzo hitting-set generator

The HSG construction in Theorem 4.5 is natural and intuitive, but it suffers from one major drawback — it just stretches the seed by one. For the goal of derandomizing PIT, even somewhat efficiently, we would need the stretch to be a much faster growing function of the seed length.

A natural attempt is to define the following new generator:

$$\mathsf{Gen}_P^{0.5} : \mathbb{F}^{tk} \to \mathbb{F}^{(t+1)k}$$
$$\mathsf{Gen}_P^{0.5}(\mathbf{z}_1, \ldots, \mathbf{z}_t) = (\mathbf{z}_1, \ldots, \mathbf{z}_t, P(\mathbf{z}_1), \ldots, P(\mathbf{z}_t))$$

In other words, partition partition the coordinates of the seed variables $\mathbf{z}$ into sets $S_1, S_2, \ldots, S_t$ and look at the polynomial map which maps $\mathbf{z}$ to $(\mathbf{z}, P(\mathbf{z}\mid_{S_1}), P(\mathbf{z}\mid_{S_2}), \ldots, P(\mathbf{z}\mid_{S_t}))$, where $\mathbf{z}\mid_{S_i}$ is the projection of $\mathbf{z}$ on the coordinates in S_i. It is fairly straightforward to see that this map is indeed an HSG as its analysis easily reduces to that of Theorem 4.5. However, the issue of stretch not being sufficiently large continues to linger, the primary reason being that the subsets $S_1, S_2, \ldots, S_t$ are disjoint from each other and we can only have (m/k) many pairwise disjoint k-sets from a universe of size m.

The HSG of Kabanets and Imapagliazzo that we discuss next gets around this issue by considering a family of subsets which are not disjoint, but are *almost* disjoint in the sense that their pairwise intersections are much smaller than their respective sizes. This allows them to take many more subsets than the seed length, thereby giving an HSG with a much larger stretch. The construction of the HSG in by Kabanets and Impagliazzo [18] is precisely the same as that in the classical pseudorandom generator of Nisan and Wigderson [22] in the Boolean setting and some of their ideas in the analysis of the generator also carries over to the algebraic setting. It is worth noting that that the HSG construction in [18] was one of the earliest such constructions in the algebraic setting.

We start by defining the generator, for which we need the following theorem of Nisan and Wigderson [22] about the almost-disjoint set families.

Theorem 4.7 (Combinatorial designs [22])**.** *Let n, m be positive integers such that $n < 2^m$. Then, there is a family of subsets $S_1, S_2, \ldots, S_n \subseteq [k]$ with the following properties.*

- $|S_i| = m$*, for each* $i \in [n]$*,*
- $\left|S_i \cap S_j\right| \leq \log n$*, for all* $i, j \in [n]$ *such that* $i \neq j$*,*

- $k = O(\frac{m^2}{\log n})$.

Moreover, such a family of sets can be constructed via a deterministic algorithm in time $\mathsf{poly}(n, 2^k)$.

The set families as defined in Theorem 4.7 are referred to as an $(m, \log n)$ *combinatorial design*, which is a name we use in the rest of the discussion.

Definition 4.8 (HSG of Kabanets and Impagliazzo [18])**.** *Let m and n be positive integers such that $n < 2^m$, and $S_1, S_2, \ldots, S_n \subseteq [k]$ be an $(m, \log n)$ combinatorial design with $k = O\left(m^2 / \log n\right)$. Let P be an m-variate polynomial. Then, the polynomial map $\mathsf{Gen}_P^1 : \mathbb{F}^k \to \mathbb{F}^n$ is defined as*

$$\mathsf{Gen}_P^1(\mathbf{z}) = \left(P(\mathbf{z}\mid_{S_1}), P(\mathbf{z}\mid_{S_2}), \ldots, P(\mathbf{z}\mid_{S_n})\right). \qquad \Diamond$$

Note that one way in which $\mathsf{Gen}_P^1(\mathbf{z})$ differs from $\mathsf{Gen}_P^0(\mathbf{z})$ is that the original seed variables are not a part of the output. We remark that this is completely superficial. We could have appended the seed to the generator and increased the stretch to n (as opposed to the current stretch of $n - k$), and this would not affect any of the results that we discuss here. However, for a slight ease of notation, we stick with way things are defined in Definition 4.8.

A simple and key observation about the HSG above is about its explicitness, and is summarized below.

Observation 4.9. *If the polynomial P is $t(m)$-explicit, then the polynomial map $\mathsf{Gen}_P^1(\mathbf{z})$ is $\mathsf{poly}(n, 2^k, t(m))$-explicit.*

The following theorem, which is the focus of this section says that for a hard enough explicit polynomial P, the map $\mathsf{Gen}_P^1(\mathbf{z})$ is a hitting-set generator for circuits of low enough complexity. Together with Observation 4.9, this implies that for any explicit and hard enough P, $\mathsf{Gen}_P^1(\mathbf{z})$ is a hitting-set generator.

Theorem 4.10 (Kabanets and Impagliazzo [18])**.** *Let $P(\mathbf{z}) \in \mathbb{F}[\mathbf{z}]$ be an m-variate multilinear polynomial which cannot be computed by an algebraic circuit of size s. Let $C(\mathbf{y}) \in \mathbb{F}[\mathbf{y}]$ be an n-variate circuit of size at most s' and degree d. If $(s'nmd)^5 < s$ then,*

$$C(\mathbf{y}) = 0 \iff C \circ \mathsf{Gen}_P^1(\mathbf{z}) = 0\,.$$

Proof. At a very high level, the proof of this theorem is essentially via a reduction to Theorem 4.5. Analogous to the set up of the proof of Theorem 4.5, we again start with the assumption that there is a nonzero n-variate circuit $C(\mathbf{y})$ of size s' and degree d (which are small enough), with

$C \circ \mathsf{Gen}_P^1(\mathbf{z})$ being identically zero. And the eventual goal is to show that this assumption implies that P has a small circuit, thereby contradicting the hardness of P and completing the proof. This route of starting with this assumption, and getting to the contradiction is a little more non-trivial though, as compared to that in Theorem 4.5. In particular, it is far from clear how Theorem 4.10 comes into the picture (and it does!). The keyword to addressing this difficulty is *hybrid argument* which we now discuss.

Hybrid Argument. For $i = 1, 2, \ldots, n$, let C_i be the circuit obtained from $C(\mathbf{y})$ by substituting the variables $y_1, y_2, \ldots, y_i$ by $P(\mathbf{z} \mid_{S_1}), P(\mathbf{z} \mid_{S_2}), \ldots, P(\mathbf{z} \mid_{S_i})$ respectively. Clearly, $C_n \equiv C \circ \mathsf{Gen}_P^1(\mathbf{z})$. We can view the process of obtaining C_n from C one step at a time, where in the i^{th} step, the variable y_i in the circuit C_{i-1} is replaced by $P(\mathbf{z} \mid_{S_i})$. Now, from our assumption, we know that C is nonzero and C_n is identically zero. Thus, there must have been an intermediate step i, such that C_i is nonzero, and C_{i+1} is identically zero. It is this circuit C_i that we focus on next. Observe that i must be at least one.

Reduction to Theorem 4.5. Recall that C_i is defined as

$$C_i = C\left(P(\mathbf{z} \mid S_1), \ldots, P(\mathbf{z} \mid S_i), y_{i+1}, \ldots, y_n\right).$$

Thus, C_i has two sets of variables, namely $\mathbf{z}$ and $\{y_{i+1}, y_{i+2}, \ldots, y_n\}$. The idea is to just focus on the *relevant variables*, which in this case are the variables $\mathbf{z} \mid_{S_{i+1}}$ and y_{i+1}. To this end, we set every $\mathbf{y}$-variable apart from y_{i+1} and every $\mathbf{z}$-variable outside $\mathbf{z} \mid_{S_{i+1}}$ to a value in the underlying field which keeps C_i nonzero. Since, C_i is a nonzero polynomial, in particular, a random value for there additional variables picked from a large enough grid will have this property. We just work with one such value. Let C_i' and C_{i+1}' be the circuits obtained from C_i and C_{i+1} respectively at the end of this partial assignment. From the discussion above, we know that C_i' only depends on the variables $\mathbf{z} \mid_{S_{i+1}}$ and y_{i+1} and C_{i+1}' which is identically zero is obtained from C_i' by replacing y_{i+1} by $P(\mathbf{z} \mid_{S_{i+1}})$. We now claim that the size of C_i' is *not too large* when compared to that of C.

Claim 4.11. *The size of C_i' is at most $O(s' \cdot n)$ and degree at most md.*

For now, we use Claim 4.11 to complete the proof of Theorem 4.10 and then move on to its proof. Observe that C_i' is a polynomial of circuit size at most $O(s'n)$ and degree at most md on the variables $\mathbf{z} \mid_{S_{i+1}}$ and y_{i+1}, and the circuit obtained by replacing y_{i+1} by $P(\mathbf{z} \mid_{S_{i+1}})$ which equals C_{i+1}' is identically zero. Thus, we are in the setting of Theorem 4.6, which implies that the circuit size of $P(\mathbf{z} \mid_{S_{i+1}})$ is at most $(s'nmd)^5 < s$. Note that this

contradicts the hypothesis of Theorem 4.10, where we assumed that the m-variate polynomial P *cannot* be computed by circuits of size less than s. Thus, $C \circ \mathsf{Gen}^1_P(\mathbf{z})$ could not have been identically zero. □

We now prove Claim 4.11. This would complete the proof of Theorem 4.10.

Proof of Claim 4.11. Recall that the circuit C'_i was obtained from the circuit C by replacing $y_1, y_2, \ldots, y_i$ by $P(\mathbf{z}\mid_{S_1}), P(\mathbf{z}\mid_{S_2}), \ldots, P(\mathbf{z}\mid_{S_i})$ respectively (this gives us circuit C_i) and then substituting all variables apart from $\mathbf{z}\mid_{S_{i+1}}$ and y_{i+1} by some field elements. Note that since P is a hard polynomial, it is not clear that the size of the circuit C_i is small. However, as we shall see next, after the partial substitution involved in getting C'_i from C_i, we will be able to argue that the size of C'_i is indeed small.

To see this, recall that from the pairwise intersection property of the sets $S_1, S_2, \ldots, S_n$, we know that $\left|S_{i+1} \cap S_j\right| \leq \log n$ for all $j \neq (i+1)$. Thus, the polynomial obtained from the polynomial $P(\mathbf{z}\mid_{S_j})$ after setting all variables in $\mathbf{z}$ outside $\mathbf{z}\mid_{S_{i+1}}$ to field elements is a multilinear polynomial in only $\log n$ variables. Thus, such a polynomial is a sum of at most $2^{\log n} = n$ monomials and has a circuit of size $O(n)$. Thus, the circuit C'_i can be directly obtained from C by replacing the variables $y_1, y_2, \ldots, y_i$ by polynomials of circuit size at most n and the variables $y_{i+2}, y_{i+3}, \ldots, y_n$ by field elements. Therefore, the size of C'_i is at most $O(s'n)$.

The bound on the degree just follows from the fact that the degree of C is at most d and the degree of P is at most m since it is multilinear on m variables. □

A particularly insightful regime of parameters to invoke Theorem 4.10 when we have access to a family of multilinear polynomials which require superpolynomial size algebraic circuits is by setting[10] $m = \log^2 n$. As a consequence, it follows from Theorem 4.10 that the seed length of the generator $\mathsf{Gen}^1_P(\mathbf{z})$ can be taken to be at most $m^2/\log n$ which is at most $\log^3 n$. Now, combining Theorem 4.10 with Lemma 4.1,we have the following (slightly informally stated) theorem.

Corollary 4.12 ((Informal) [18])**.** *Given a polynomial family $\{f_n\}$, with f_n being an n-variate multilinear polynomial that requires $2^{\Omega(n)}$-sized circuits, there exists a deterministic algorithm for PIT which runs in $2^{\mathsf{poly}(\log s)}$ time (where s is the size of the input circuit).*

[10] We could have chosen m to be anything asymptotically large than $\log n$ for this discussion.

We remark that there is really nothing special about having hard multilinear polynomial families. In fact, we can assume without loss of generality that the hard polynomial family is multilinear, up to an appropriate tuning of parameters involved.

4.2.1 What more can we ask for ?

Given Corollary 4.12, we know that we can get deterministic quasipolynomial time algorithm for PIT given access to explicit hard polynomial families. To a large extent, this answers the question of derandomization from algebraic hardness. However, some very natural questions in this context continue to remain unanswered. We now focus on briefly discussing some of these.

Complete derandomization of PIT. Corollary 4.12 only gives a *quasipolynomial* time algorithm for PIT, and not a truly polynomial time algorithm. While the presentation in this survey is for a specific range of parameters, it is not hard to verify that the hitting-set generator in Theorem 4.10 cannot be used to get a polynomial time algorithm for PIT even if we have access to a polynomial family which is optimally hard. To see this observe that the stretch of $\mathsf{Gen}_P^1(\mathbf{z})$ can be at most exponential in its seed length. In other words, the seed length cannot be smaller than $\log n$ for the stretch to be n. Thus, the application of the generator reduces a nonzero polynomial of low degree and circuit size on n variables to a nonzero polynomial on $\log n$ variables. Now to test the nonzeroness of this resulting polynomial, we rely on Lemma 4.1, which gives us a hitting set of size $D^{\log n}$, where D is the degree of the resulting circuit. Now, for any growing[11] D, the final size of the hitting set is always going to be superpolynomially growing in n. Thus, a natural question is to understand is the following:

Question 4.13. *Does a superpolynomial lower bound for algebraic circuits for an explicit polynomial family imply that there is a deterministic polynomial time algorithm for PIT?*

Tradeoffs for natural subclasses of algebraic circuits. Yet another question of interest which comes up in the light of Theorem 4.10 is if an analogue of it is true for natural subclasses of algebraic circuits. For instance, the following seems to be a natural question.

[11] This is the interesting regime of parameters for the PIT problem.

Question 4.14. *Does superpolynomial lower bound for* algebraic formulas *for an explicit polynomial family imply that there is a deterministic polynomial time algorithm for PIT for* algebraic formulas*?*

Remark 4.15. *We note that a lower bound of $n^{\omega(\log d)}$ for an n-variate degree d polynomial for formulas implies algebraic circuit lower bounds. This is an immediate consequence of a celebrated depth reduction result of Valiant et al. [28] which shows that any circuit of size s circuit computing a polynomial of degree d can be converted to a formula of size at most $s^{O(\log d)}$. Thus, the above question is of interest when the superpolynomial lower bounds for formulas are not strong enough to imply superpolynomial algebraic circuit lower bounds.* ◊

Perhaps the first course of action is to understand if, the proof of Definition 4.8 just naturally carries over to the case of formulas thereby answering Question 4.14. And, most of the argument does indeed just carry over to the case of formulas, up until the point where we invoke Theorem 4.10. In particular, algebraic formulas are not known to be closed under taking factors (or more specifically taking roots as in the case here). It is worth noting that the question of closure of formulas or essentially any other natural class of polynomials is a very natural question on its own, even beyond the immediate application to Question 4.14 and also continues to remain open.

Beyond design based hitting-set generators. In the Boolean set up, it is not hard to show that we cannot have a pseudorandom generator of seed length k and stretch n which fools circuits of size larger than $n2^k$. To see this, observe that we can construct a circuit of size $O(n2^k)$ to identify the range of the generator (consisting of 2^k strings of length n each), and accept every input in the range while rejecting everything else. However, this upper bound on the stretch does not seem to extend immediately to the algebraic set up, though the argument above can be adapted to show an upper bound of $(dD)^{O(k)}$ on the size of degree d algebraic circuits which can be fooled by a HSG with seed length k, stretch n and degree D. Thus, in principle, one could expect HSGs with stretch larger than 2^k.

An intriguing feature of Theorem 4.10 is that even though it is a statement which connects two inherently algebraic questions, namely the question of proving strong lower bounds for explicit polynomial families and designing efficient deterministic algorithms for PIT, its proof crucially relies on the notion of combinatorial designs. For aesthetic reasons, it seems desirable to seek an alternate construction of a hitting-set generator from algebraic hardness which is algebraic and does not crucially rely on com-

binatorial ideas.

4.3 Fine grained analogues of the KI generator

In this section we briefly discuss some recent work broadly aimed at answering Question 4.14, and while we are still far from fully answering the question, there has been some interesting progress in this direction.

HSG for polynomials of low individual degree [9]. Dvir, Shpilka and Yehudayoff [9] showed that an explicit family of polynomials which cannot be computed by a small algebraic circuit of depth Δ can be used to derandomize PIT for algebraic circuits of depth $(\Delta - 5)$ of *bounded individual degree*. Thus, for the case of circuits of bounded individual degree and some loss in depth, they show that lower bounds for constant depth circuits for an explicit polynomial family does imply deterministic PIT for them. The hitting-set generator in [9] is precisely the same as the HSG in Definition 4.8. The key idea in the analysis is a statement about the complexity of factors of polynomials of small individual degree which have small constant depth circuits[12]. They show that the analogue of Theorem 4.6 is true for constant depth circuits of bounded individual degree. Armed with this key structural result, the proof follows the outline of the proof of Theorem 4.10.

HSG using hardness of polynomials of low degree [6]. Yet another partial result towards Question 4.14 was by Chou, Kumar and Solomon [6] who showed that a family of explicit polynomials of low degree (degree at most $\exp(O(\sqrt{\log k}))$ for a k-variate polynomial) which requires super-polynomial size algebraic formulas implies a deterministic algorithm for PIT for algebraic formulas. As in [9], the hitting-set generator in [6] is again the same as that in Definition 4.8, and the analysis again crucially relies on a statement about the complexity of low degree factors of polynomials with small formulas. Chou et al. show that factors of *low enough* degree of polynomials of with small formulas have small formulas.

4.4 A hitting-set generator without designs

In this section, we discuss a construction of hitting generators from algebraic hardness due to Guo et al. [10]. The generator constructed here uses a

[12] More precisely, [9] only show this for *roots* as in Theorem 4.6, and this was generalized to arbitrary factors by Oliveira [8].

k-variate degree d polynomial which requires circuit size s, and has stretch $s^{\Omega(1/k)}$. For this range of parameters, we know from counting arguments that there exist polynomial which require circuit size at least $d^{\Omega(k)}$. The hitting-set generator of Guo et al. uses such an explicit hard polynomial to give a polynomial map which maps $2k$ variables to $d^{\Omega(1)}$ variables, such that the map preserves the nonzeroness of any circuit of not-too-large size. The degree of this map is upper bounded by d. Thus, this generator answers Question 4.13 in a slightly weakened setting. As we shall see in Section 5, this generator also has a clean application to the phenomenon of *bootstrapping* for PIT. We start by defining the generator.

Definition 4.16 (The generator)**.** *For any k-variate polynomial $P(\mathbf{z})$, define the map $\mathsf{Gen}_P^2 : \mathbb{F}^k \times \mathbb{F}^k \to \mathbb{F}^{n+1}$ as follows:*

$$\mathsf{Gen}_P^2(\mathbf{z}, \mathbf{y}) = (\Delta_0(P)(\mathbf{z}, \mathbf{y}), \Delta_1(P)(\mathbf{z}, \mathbf{y}), \ldots, \Delta_n(P)(\mathbf{z}, \mathbf{y}))\,,$$

where $\Delta_i(P)$ is the homogeneous degree i (in $\mathbf{y}$) component in the Taylor expansion of $P(\mathbf{z} + \mathbf{y})$, i.e.

$$\Delta_i(P)(\mathbf{z}, \mathbf{y}) = \sum_{\mathbf{e} \in \mathbb{N}^k, |\mathbf{e}|_1 = i} \frac{\mathbf{y}^{\mathbf{e}}}{\mathbf{e}!} \cdot \frac{\partial P}{\partial \mathbf{z}^{\mathbf{e}}}. \qquad \textit{(here, } \mathbf{e}! := e_1! \cdots e_k!\textit{)} \qquad \Diamond$$

The following theorem which is the focus of this section states that for a sufficiently hard P, Gen_P^2 is a hitting-set generator.

Theorem 4.17 ([11, 10])**.** *Assume that the underlying field $\mathbb{F}$ has characteristic zero and let P be a k-variate polynomial of degree d. Suppose P cannot be computed by algebraic circuits of size $\tilde{s} = \left(s \cdot D \cdot d^3 \cdot n^{10k}\right)$ for parameters n, D, s. Then, for any $(n+1)$-variate algebraic circuit $C(x_0, \ldots, x_n)$ of size at most s and degree at most D, we have*

$$C \neq 0 \iff C \circ \mathsf{Gen}_P^2(\mathbf{z}, \mathbf{y}) \neq 0.$$

With a few more ideas on top of Theorem 4.17, the following slightly stronger statement can be shown. The theorem essentially shows that for any constant variate explicit polynomial family, a lower bound with polynomial dependence on the degree implies efficient deterministic PIT for algebraic circuits.

Theorem 4.18 ([10])**.** *Let $k \in \mathbb{N}$ and $\delta > 0$ be arbitrary constants. Suppose $\{P_{k,d}\}_{d \in \mathbb{N}}$ is a family[13] of explicit k-variate polynomials such that $\deg(P_{k,d}) = d$*

[13] We assume that the polynomial family contains a degree d polynomial, $P_{k,d}$, for every positive integer d. For the purposes of this theorem, it suffices to assume that the family is *sufficiently often* in the following sense: There are absolute constants a, b such that for any $t \in \mathbb{N}$, there is some $P_{k,d}$ in the family such that $t^a \leq d \leq t^b$.

and $P_{k,d}$ requires circuits of size at least d^{δ}. Then, there are explicit hitting sets of size $s^{O(k^2/\delta^2)}$ for the class of s-variate, degree s polynomials that can be computed by circuits of size s.

We remark that, analogous to the case of the Kabanets-Impagliazzo generator (Gen^1_P), one could also include the original seed variables in the output. This does not affect the correctness of Theorem 4.17. However, in this case, the first $2k+1$ coordinates of the generator Gen^2_P are $(z_1, \ldots, z_k, y_1, \ldots, y_k, \Delta_0(P))$. From the definition of Δ_0, note that this is the same as the tuple $(z_1, z_2, \ldots, z_k, y_1, y_2, \ldots, y_k, P(\mathbf{z}))$. This is essentially the generator Gen^0_P up to some relabeling of variables. Therefore, in this sense, the generator Gen^2_P is also another generalization of the generator Gen^0_P. As we shall see, even though the proof of Theorem 4.17 does not directly use Theorem 4.6 as a blackbox as is used in Gen^0_P, it does rely on generalization of some of the ideas in the proof of Theorem 4.6. In the rest of this section, we discuss the main ideas in the proof of Theorem 4.17. In the interest of keeping the length of this survey short, we will skip some of the formal details, and refer the interested reader to the original papers [11, 10].

4.4.1 Reconstruction from annihilators

Rather than quickly dive into the proof of the correctness of this generator, let us focus on *how* we might go about proving this; hopefully this would shed more light on the correctness. Suppose we have a generator, built using a hard polynomial P, of the form $\mathsf{Gen}_P = (P_1, \ldots, P_n)$ where each P_i is related to P in some way. We wish to show that Gen_P is indeed a hitting-set generator for the class of small-enough circuits.

The standard way we would argue this would be via contradiction. Let us assume that there is indeed a circuit C such that $C \neq 0$ but $C \circ \mathsf{Gen}_P = 0$. Then, we would like to somehow, using the circuit C, reconstruct a *not-too-large* circuit for P thus contradicting its hardness. In other words, given access to an *annihilator* C of Gen_P, we wish to reconstruct a circuit for P. In the case of Theorem 4.5 and Theorem 4.10, we made minor modifications to C to obtain a circuit C', showed that $(y - P)$ had to be a factor of C' and eventually appealed to Kaltofen's factorisation result (Theorem 4.10) to obtain a not-too-large circuit for P. With this in mind, the challenge of constructing a hitting set generator can be viewed as the quest of constructing a suitable map Gen_P such that any *annihilator* can be used to *reconstruct* P!

This approach is akin to many algebraic decoding algorithms in coding theory. For instance, the Reed-Solomon decoders proceeds by first con-

structing a suitable bivariate polynomial $Q(x, y)$ (based on the received message) such that all candidate messages $m(x)$ satisfy $Q(x, m(x)) = 0$. For Parvaresh-Vardy codes for instance, the algorithm proceeds by finding a polynomial $Q(y, z_0, \ldots, z_s)$ such that $Q(x, m(x), m(x)^h, \ldots, m(x)^{h^{s-1}}) = 0$ (roughly speaking) and efficiently recovering all $m(x)$'s that satisfy the above equation. The generator Gen_P^2 is very closely related to the decoding algorithm for *multiplicity codes* or *derivative codes*, due to Kopparty [19]. We shall illustrate the key aspects of the proof of Theorem 4.17 by the following related map instead which is essentially the list-decoding algorithm of Kopparty [19].

Reconstruction for univariate derivatives maps: Let $P(x) = p_0 + p_1 z + \cdots + p_d z^d \in \mathbb{F}[z]$ be a univariate polynomial. We shall define the map $\widetilde{\mathsf{Gen}_P} : \mathbb{F} \to \mathbb{F}^3$ as

$$\widetilde{\mathsf{Gen}_P}(z) = (P(z), P'(z), P''(z)),$$

where P', P'' refers to the first and second order derivative of P respectively. Suppose we are given a circuit $C(x_0, x_1, x_2)$ such that $C \circ \widetilde{\mathsf{Gen}_P} = 0$, how do we recover P from C?

Let us assume that we somehow knew the first three coefficients p_0, p_1, p_2. The goal is to recover all the other coefficients of P. The key observation is the following:

$$\begin{aligned} 0 = C(P, P', P'') &= C(p_0 + p_1 z + \cdots\ ,\ p_1 + 2p_2 z + \cdots\ ,\ 2p_2 + 6p_3 z + \cdots) \\ &= C(p_0 + p_1 z\ ,\ p_1 + 2p_2 z\ ,\ 2p_2 + 6p_3 z) \bmod z^2 \\ &= C(p_0 + p_1 z\ ,\ p_1 + 2p_2 z\ ,\ 2p_2) \quad + \\ &\qquad \left(\frac{\partial C}{\partial x_2}\right)(p_0 + p_1 z, p_1 + 2p_2 z, 2p_2) \cdot (6p_3 z) \bmod z^2, \end{aligned}$$

which is a linear equation in p_3! This last step is just an application of Taylor's expansion to the polynomial C around the point $(p_0 + p_1 z, p_1 + 2p_2 z, 2p_2)$. Therefore,

$$p_3 = \left(\frac{\text{Coefficient of } z \text{ in } C(p_0 + p_1 z, p_1 + 2p_2 z, 2p_2)}{6 \cdot \partial_{x_2} C(p_0, p_1, 2p_2)}\right)$$

Thus, if $\frac{\partial}{\partial x_2}(C)(p_0, p_1, 2p_2) \neq 0$, we can use the above equation to solve for p_3. Once we have found p_3, the same argument can be used to find p_4, and so on, to recover all the coefficients of P. Thus, we can indeed recover

P from C.

The proof of correctness of Theorem 4.17 is, to a large extent, a multivariate analogue of the above sketch but requires a careful navigation around many subtleties. The most important of the subtleties is that we need to keep track of the *algebraic circuit complexity* of the recovery process. For instance, if recovering one additional coefficient incurs a multiplicative blow-up of just two in the circuit size, the eventual circuit size of the entire recovery process becomes 2^d, which is not helpful to contradict the hardness of P. Fortunately, a careful analysis of the recovery process allows us to infer that the increase in size is merely additive instead of multiplicative. More details, we refer the interested reader to the original papers [11, 10].

5 Application: Bootstrapping for PIT

We saw in Section 3 that non-trivial deterministic algorithms for PIT are known to imply lower bounds which are much better than the current state of art. In this section, we discuss another somewhat surprising consequence of even slightly non-trivial deterministic (blackbox) algorithm for PIT. This line of research started with a work of Agrawal, Ghosh and Saxena [2] who showed that PIT is amenable to the phenomenon of *bootstrapping*.

Repeated applications of the KI generator ([2]). Let us revisit the argument of Kabanets and Impagliazzo. Suppose we wish to build a generator for, say, circuits of size n^2, computing n-variate polynomial of degree at most n. Theorem 4.10 shows that if we have a hard multilinear polynomial P on m variables that cannot be computed by circuits of size roughly $O(n^{10})$, and if we have an $(m, \log n)$-design $S_1, \ldots, S_n$, then $\mathsf{Gen}_P^1 : \mathbb{F}^k \to \mathbb{F}^n$ is a hitting-set generator. And if $m = c \log n$ for some large enough c (say 100) and $k = O(\log^2 n)$, then, indeed asking for an $\Omega(n^{10})$ lower bound seems reasonable.

Now let's consider the new circuit $C' = C \circ \mathsf{Gen}_P^1$. This now computes an k-variate polynomial, of degree $\tilde{O}(n)$, that is computable by circuits of size at most n^{10c}. What is preventing us from running through the same argument again? For starters, we are working with a larger circuit C' and hence would require a m'-variate polynomial P' with hardness n^{1000c} or so, with $m' \ll k$. The issue is that any multilinear polynomial on m' variables

trivially has a circuit of size $2^{m'}$ and if this is much smaller than n^{1000c} we seem to be stuck. This is indeed the reason why such an argument does not work in the Boolean setting since any Boolean function on m' has circuit complexity at most $2^{m'}$.

The key observation here is that we can indeed have m'-variate polynomials of any hardness that we want because we have one additional parameter to play with — the degree of the polynomial! From the proof of Theorem 4.10, it is not hard to see that multilinearity is not essential for the proof, and we could have worked with hard polynomials of higher individual degree, and upto an appropriate change in the quantitative parameters, the argument continues to be applicable. Thus, by considering polynomials of suitable large degree, we can hope for m'-variate polynomials of hardness n^{1000c} or whatever is required for the argument. This is essentially the idea behind the hitting-set generator implicit in a work of Agrawal, Ghosh and Saxena [2].

It appears that we need multiple families of hard polynomials for this generator to work, with incrementally increasing hardness. This appears to require *many* hard polynomial families as opposed to one as in Theorem 4.10. But how do we obtain such collections of hard polynomial families? If we are promised that there is a *slightly better than trivial* explicit hitting set, we can use Theorem 3.1 to get a hard polynomial from that! On understanding the evolution of parameters on repeated use of the same generator, we obtain following theorem of Agrawal, Ghosh and Saxena [2].

Theorem 5.1 (Agrawal, Ghosh and Saxena [2])**.** *Let $\varepsilon > 0$ and let n be a large enough constant. Suppose, for all $s \geq n$ we have that the class $\mathcal{C}(n, s, s)$ of n-variate polynomials of degree at most s and circuit size at most s has an explicit hitting set of size $s^{n^{0.5-\varepsilon}}$. Then, for all growing s, the class $\mathcal{C}(s, s, s)$ has hitting sets of size $s^{\exp(\exp(\log^* s))}$.*

Here, $\log^* s$ is the defined as the number of iterated logs required to obtain a number less than 2. Subsequently, this result was strengthened by Tengse and the authors [20] to the following form.

Theorem 5.2 (Kumar, Saptharishi and Tengse [20])**.** *Let $\varepsilon > 0$ and let n be a large enough constant. Suppose, for all $s \geq n$ we have that the class $\mathcal{C}(n, s, s)$ of n-variate polynomials of degree at most s and circuit size at most s has an explicit hitting set of size $s^{n-\varepsilon}$. Then, for all growing s, the class $\mathcal{C}(s, s, s)$ has hitting sets of size $s^{\exp(\exp(\log^* s))}$.*

Furthermore, the same result holds if the $\mathcal{C}$ is replaced by several natural subclasses of circuits such formulas, or algebraic branching programs etc.

In this section, we sketch a proof of the following result of Guo et al. [10], which is a further quantitative improvement on Theorem 5.2. Moreover, the proof is a fairly direct consequence of Theorem 4.17 and we sketch the details below.

Theorem 5.3 (Guo, Kumar, Saptharishi, Solomon [10]). *Assume that the underlying field $\mathbb{F}$ has characteristic zero. Let $k \in \mathbb{N}$ and $\delta > 0$ be arbitrary constants. Suppose that, for all large enough s, there is an explicit hitting set of size $(s+1)^k - 1$ for the k-variate algebraic circuits of size at most s^δ and individual degree at most s. Then for all large enough s, there is an explicit hitting set of size $s^{O((k/\delta)^2)}$ for s-variate circuits of size and degree at most s.*

Observe that it is not very hard to construct an explicit hitting set of size at most $(s+1)^k$ for k-variate polynomials of individual degree at most s. In particular, for every subset S of the field $\mathbb{F}$ of size at least $s+1$, the set of points in S^k forms a hitting set of size $(s+1)^k$ for k-variate polynomials of individual degree at most s. This is a fairly straightforward consequence of the usual induction based proof of Lemma 4.1. Thus, Theorem 5.3 says that if we can even save a single point from this trivial hitting set for constant variate polynomials, then we get a truly polynomial time deterministic algorithm for PIT. In the rest of this section, we present the fairly short and succinct proof of Theorem 5.3 which is based on the hitting-set generator in Theorem 4.17.

Proof of Theorem 5.3. Let s be large enough and let H_s be the hitting set for k-variate circuits of size s^δ and individual degree at most s. From the hypothesis of the theorem, we know that $|H_s| \leq (s+1)^k - 1 < (s+1)^k$ and H_s is explicit. From Theorem 3.1, we can obtain a polynomial $P_s(z_1, \ldots, z_k)$ of individual degree at most s that cannot be computed by circuits of size s^δ. In other words, we get that P_s is a degree $d \leq ks$ polynomial that requires circuits of size at least

$$\left(\frac{d}{k}\right)^\delta \gg d^{\delta/2}.$$

Thus, $\{P_s\}$ is an explicit family of k-variate, degree-d polynomials that require circuits of size $d^{\delta/2}$. Now, applying Theorem 4.18 with this hard polynomial family, we get an explicit $s^{O((k/\delta)^2)}$-size hitting set for s-variate circuits of size and degree at most s. □

6 Conclusion and open problems

In this survey, we studied the interplay between hardness and derandomization in algebraic complexity.

- We know from Theorem 3.1 that construction of explicit hitting sets for a class of circuits imply lower bound for the same class.
- Theorem 4.10 and Theorem 4.17 shows that suitably hard polynomials can also be used to construct hitting-set generators.
- Theorem 5.1 and Theorem 5.2 used both directions of the hardness-randomness interplay to show that even slightly non-trivial hitting sets can be bootstrapped to yield much nearly optimal hitting sets.
- Theorem 5.3 showed that even saving *one* point from the trivial hitting set in the constant-variate setting would yield a complete derandomization of PIT.

Although we have made considerable progress in our understanding of this interplay, several natural questions remain open. In particular, Question 3.2, Question 4.13 and Question 4.14 which we restate below.

Question. *Does a polynomial time deterministic PIT algorithm in the blackbox setting imply that* $\mathsf{VP} \neq \mathsf{VNP}$*?*

Theorem 3.1 yields a lower bound, but the polynomial obtained is not known to be in VNP and hence the above question continues to remain open.

Question. *Does a superpolynomial lower bound for algebraic circuits for an explicit polynomial family imply that there is a deterministic polynomial time algorithm for PIT?*

Theorem 4.18 provides an answer to a related question where the hypothesis provides a suitable lower bound for a family of *constant-variate* polynomials where the growing parameter is the degree. Thus, the above statement, as stated, continues to remain open.

Question. *Does superpolynomial lower bound for* algebraic formulas *for an explicit polynomial family imply that there is a deterministic polynomial time algorithm for PIT for* algebraic formulas*?*

In the case of Theorem 4.10, the theorem is not known to extend to the class of algebraic formulas since we do not know if Kaltofen's factorisation

result (Theorem 4.6) extends to algebraic formulas. In the setting of Theorem 4.17, the reconstruction argument heavily reuses prior computation and hence does not seem to be adaptable to the class of algebraic formulas.

Another nagging open question is that several of the results discussed in this survey require the underlying field to be zero, or larger than the degree of the polynomials involved. This is primarily due to the ubiquitous use of partial derivatives in both Kaltofen's factorisation result (Theorem 4.6) in the case of Theorem 4.10 and Theorem 5.1, and both the construction and proof argument in generator of Theorem 4.17. An exception to this is the result of Tengse and the authors Theorem 5.2 [20] which continues to hold over all fields.

Question. *Are there results analogous to Theorem 4.10, Theorem 4.17 over fields of small characteristic ?*

We are hopeful that the next few years will bring to resolution at least some of the above questions.

Acknowledgements

We are thankful to Chi-Ning Chou, Zeyu Guo, Swastik Kopparty, Noam Solomon, Anamay Tengse and Ben Lee Volk for insightful discussions on many of the themes addressed in this survey.

References

[1] Manindra Agrawal. Proving Lower Bounds Via Pseudo-random Generators. In *Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2005)*, pages 92–105, 2005.

[2] Manindra Agrawal, Sumanta Ghosh, and Nitin Saxena. Bootstrapping variables in algebraic circuits. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 1166–1179, 2018. eccc:TR18-035.

[3] Anurag Bishnoi, Pete L. Clark, Aditya Potukuchi, and John R. Schmitt. On Zeros of a Polynomial in a Finite Grid. *Combinatorics, Probability & Computing*, 27(3):310–333, 2018.

[4] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudorandom Bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[5] Peter Bürgisser. The Complexity of Factors of Multivariate Polynomials. *Foundations of Computational Mathematics*, 4(4):369–396, 2004.

[6] Chi-Ning Chou, Mrinal Kumar, and Noam Solomon. Some Closure Results for Polynomial Factorization and Applications. *CoRR*, abs/1803.05933, 2018. Pre-print available at `arXiv:1803.05933`.

[7] Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters*, 7(4):193–195, 1978.

[8] Rafael Mendes de Oliveira. Factors of Low Individual Degree Polynomials. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC 2015)*, pages 198–216, 2015.

[9] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-Randomness Tradeoffs for Bounded Depth Arithmetic Circuits. *SIAM J. Comput.*, 39(4):1279–1293, 2009.

[10] Zeyu Guo, Mrinal Kumar, Ramprasad Saptharishi, and Noam Solomon. Derandomization from algebraic hardness. Online version: `https://mrinalkr.bitbucket.io/papers/newprg.pdf`, 2019.

[11] Zeyu Guo, Mrinal Kumar, Ramprasad Saptharishi, and Noam Solomon. Derandomization from Algebraic Hardness: Treading the Borders. *To appear in Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, 2019. Online version: `https://mrinalkr.bitbucket.io/papers/newprgconf.pdf`.

[12] Oded Goldreich. Pseudorandom Generators: A Primer, 2010.

[13] Johan Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC 1986)*, pages 6–20, 1986.

[14] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A Pseudorandom Generator from Any One-way Function. *SIAM J. Comput.*, 28(4):1364–1396, March 1999.

[15] Joos Heintz and Claus-Peter Schnorr. Testing Polynomials which Are Easy to Compute (Extended Abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 262–272, 1980.

[16] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[17] Erich Kaltofen. Factorization of Polynomials Given by Straight-Line Programs. In *Randomness and Computation*, pages 375–412. JAI Press, 1989.

[18] Valentine Kabanets and Russell Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, 13(1-2):1–46, 2004. Preliminary version in the *35th Annual ACM Symposium on Theory of Computing (STOC 2003)*.

[19] Swastik Kopparty. List-Decoding Multiplicity Codes. *Theory of Computing*, 11(5):149–182, 2015.

[20] Mrinal Kumar, Ramprasad Saptharishi, and Anamay Tengse. Near-optimal Bootstrapping of Hitting Sets for Algebraic Circuits. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 639–646, 2019.

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

[22] Noam Nisan and Avi Wigderson. Hardness vs Randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. Available on `citeseer:10.1.1.83.8416`.

[23] Øystein Ore. Über höhere Kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922.

[24] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015.

[25] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, 1980.

[26] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. In *Automata, Languages and Programming*, pages 544–550, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.

[27] Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010.

[28] Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM Journal of Computing*, 12(4):641–644, 1983. Preliminary version in the *6th Internationl Symposium on the Mathematical Foundations of Computer Science (MFCS 1981)*.

[29] Andrew Chi-Chih Yao. Theory and Applications of Trapdoor Functions (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91, 1982.

[30] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.