Challenges in Polynomial Factorization*

Michael Forbes† Amir Shpilka‡

Abstract

Algebraic complexity theory studies the complexity of computing (multivariate) polynomials efficiently using algebraic circuits. This succinct representation leads to fundamental algorithmic challenges such as the *polynomial identity testing (PIT)* problem (decide non-zeroness of the computed polynomial) and the *polynomial factorization problem* (compute succinct representations of the factors of the circuit). While the Schwartz-Zippel-DeMillo-Lipton Lemma [Sch80, Zip79, DL78] gives an easy randomized algorithm for PIT, randomized algorithms for factorization require more ideas as given by Kaltofen [Kal89]. However, even derandomizing PIT remains a fundamental problem in understanding the power of randomness.

In this column, we survey the factorization problem, discussing the algorithmic ideas as well as the applications to other problems. We then discuss the challenges ahead, in particular focusing on the goal of obtaining deterministic factoring algorithms. While deterministic PIT algorithms have been developed for various restricted circuit classes, there are very few corresponding factoring algorithms. We discuss some recent progress on the *divisibility testing* problem (test if a given polynomial divides another given polynomial) which captures some of the difficulty of factoring. Along the way we attempt to highlight key challenges whose solutions we hope will drive progress in the area.

1 Introduction

A multivariate polynomial is an expression of the form $f(\overline{x}) = \sum_{\overline{a}} \alpha_{\overline{a}} \overline{x}^{\overline{a}}$,[1], which is a finite summation of monomials $\overline{x}^{\overline{a}} = \prod_{i=1}^{n} x_i^{a_i}$ each with a scalar coefficient $\alpha_{\overline{a}}$. Polynomials are ubiquitous in computer science because of their applications in symbolic computation, derandomization, probabilistically checkable proofs (PCPs), cryptography, secret sharing, coding theory and more. While polynomials do naturally yield *functions* by evaluating the underlying variables, *algebraic complexity* is the study of polynomials as *syntactic* expressions. This study is motivated by the observation that most natural approaches for computation of polynomials are syntactic, as they fit in the *algebraic circuit* model of computation. This model is defined over a base field $\mathbb{F}$ and uses algebraic operations such as $+$ and $\times$ to build polynomial expressions from the input variables, just as boolean circuits compute functions. For simplicity we will restrict attention here to fields of zero characteristic such as $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$, though much of the discussion can as easily be given for finite fields of large enough characteristic. For more on algebraic circuits, see for example the survey of Shpilka and Yehudayoff [SY10].

*Guest column for SIGACT.

†Department of Computer Science, Princeton University, USA, E-mail: miforbes@csail.mit.edu

‡Department of Computer Science, Tel Aviv University, Tel Aviv, Israel, shpilka@post.tau.ac.il. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257575, and from the Israel Science Foundation (grant number 339/10).

[1]We use overline to denote vectors.

A key part of understanding any given polynomial is to understand its *factorization*, for multivariate polynomials can be uniquely factored into irreducible polynomials in direct analogue with the prime factorization of integers. While integer factorization is a seemingly hard problem for (classical) computation, since the 1960's there has been significant progress on algorithms for the *polynomial factorization problem* of computing the irreducible factors of a given polynomial. This intense study has yielded efficient randomized algorithms for this problem, which has important applications in complexity theory, coding theory, and cryptography.

The most basic (but still non-trivial) setting for this problem is that of *univariate polynomials*, where the natural[2] input size of a degree-d univariate polynomial is $d+1$ (the number of possible monomials). For n-variate degree-d polynomials, the corresponding size would be $\binom{n+d}{d}$ (the *dense* representation). While factorization in this regime requires non-trivial ideas even beyond the univariate case, we will be interested in a more concise representation. The most obvious such representation is the *sparse* representation where we list only those coefficients of a polynomial that are non-zero. However, from the view of algebraic complexity theory it is most natural to consider polynomials as being given by (possibly restricted) algebraic circuits so that the input size is the size of that representation. The sparse representation is a special case of this (depth-2 formulas), but we may consider more powerful models such as (general or bounded-depth) formulas, algebraic branching programs (ABPs)[3], etc. Recall that a formula is a restricted circuit whose computation graph is a tree, and we may also further restrict the depth (the length of the longest input-output path). The general formulation of the factorization problem where we are given such a (restricted) circuit is the *white-box* model. Some algorithms can even work in the more restrictive *black-box* model, where access to the given circuit is restricted to evaluating the computed polynomial at any desired point.[4] In all settings we are interested in producing all irreducible factors of the given polynomial, where each such factor is given a succinct representation (such as via algebraic circuits).

Note that in the dense representation, it is clear that a factor of a polynomial have no larger size in this representation. As such, the challenge in this regime is to design efficient algorithms, and to a large extent this challenge has been met. However, in our primary regime of interest where polynomials are given succinctly by algebraic circuits, it is no longer even clear that there *is* a comparably succinct description.[5] However, in a foundational work, Kaltofen [Kal89] showed that not only do factors of small circuits have small circuits, but also gave a randomized (white-box) algorithm to compute these factors. Despite the 25 years since his work, we still have a very limited understanding of whether analogous results hold for more restricted classes of circuits. That is, while restricted circuits are less expressive than general circuits so that their factors are less complicated, such restricted circuits are not expressive enough to handle the level of algebraic reasoning used in Kaltofen's [Kal89] construction. The most general result thus far is that of Oliveira [Oli15] on small-depth formulas of low individual-degree, which built on the work of Dvir-Shpilka-Yehudayoff [DSY09].

[2] One can also consider a much more concise input representation where polynomials are given by the list of non-zero coefficients, where coefficients and exponents are given in binary (the *lacunary* representation). Algorithms for this regime are harder, and some algorithmic tasks are NP-hard (see for example Plaisted [Pla77] and subsequent work). For this and other reasons we will focus on polynomials where the degree d is given in unary.

[3] We omit the definition of ABPs here, but note that their computational power is between that of formulas and circuits.

[4] Some models of computation, such as sparse polynomials [BOT88, KS01], can be efficiently (and deterministically) learned in the black-box model. In such cases, the white-box and black-box models are equivalent.

[5] Note that there are natural algebraic operations that cannot (conjecturally) be performed efficiently on algebraic circuits. In particular, the partial derivative $\frac{\partial^n}{\partial y_1 \cdots \partial y_n}$ of the small formula $\prod_{i=1}^{n} \left(\sum_{j=1}^{n} y_j x_{i,j} \right)$ is the permanent of the matrix X where $X_{i,j} := x_{i,j}$, which is conjectured to be hard to compute.

Theorem 1.1 (Oliveira [Oli15])**.** *Let $f(\overline{x}) \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial with low individual-degree, that is, $\deg_{x_i}(f) \leq r$ for each i.*[6] *Suppose f is computed by a size-s and depth-D algebraic formula, then each factor of f can be computed by a depth-$(D+5)$ and size $\mathsf{poly}((nr)^r, s)$ formula.*

Intuitively, the reason this result needs low individual-degree is so that one can do induction on the number of variables. Bounding the individual degree implies each variable can only participate in a few of the factors, so this bounds the branching factor of the recursion. Unfortunately without this restriction nothing else is known, which leads us to our first open question. We pose this question for irreducible factors as these suffice for many applications.

Open Question 1.2. *Are natural restricted classes of algebraic circuits (such as formulas or algebraic branching programs) closed under taking irreducible factors?* ◊

Embarrassingly, this question is even open for *sparse* polynomials, that is, those polynomials expressible as a linear combination of a few monomials. This question is made complicated by an example of von zur Gathen and Kaltofen [vzGK85], which we slightly modify here. Consider $f(\overline{x}) := \prod_{i=1}^{n}(x_i^d - 1) + y \cdot \prod_{i=1}^{n}(x_i - 1)$. This polynomial has $2 \cdot 2^n$ monomials, and can be factorized as $f(\overline{x}) = (\prod_{i=1}^{n}(x_i - 1)) \cdot \left(y + \prod_{i=1}^{n}(1 + x_i + \cdots + x_i^{d-1})\right)$. It is not hard to verify that this second factor is irreducible (it is linear in y) and has $d^n + 1$ monomials. Taking $n = \lg d$ the example yields a polynomial of degree $d \lg d$ and sparsity $s = 2d$ that has an irreducible factor with sparsity $s^{\Theta(\lg s)}$, though this factor clearly has a $\mathsf{poly}(s)$-size formula. While this shows a *lower bound* for the sparsity of factors, non-trivial upper bounds are lacking. Shpilka and Volkovich [SV10] showed that factors of *multilinear* sparse polynomials are sparse, and Volkovich [Vol15] extended the result to sparse polynomials of individual degree at most 2. Unfortunately, for polynomials of unbounded individual degree there are no non-trivial upper bounds on the complexity of the factors aside from Kaltofen's [Kal89] result, much less non-trivial bounds on the sparsity.

Open Question 1.3. *Can any non-trivial upper bound be given for the complexity of irreducible factors of s-sparse polynomials, in any restricted model of computation (ideally bounding the sparsity)?* ◊

Thus far we have discussed the white-box setting, where it is at least clear that factorization is possible algorithmically (in finite time). However, factorization also makes sense in the black-box setting where we are only given oracle access to the polynomial and the algorithm must implement corresponding oracles for the irreducible factors. While intuitively it seems this setting is harder as access to the input polynomial is very limited, in some sense this question is *easier* as the output guarantee is weaker: one only needs to implement an oracle algorithmically, possibly using operations not available to algebraic circuits. For example, if the input polynomial is $f(\overline{x})^2$ for some irreducible f, it is simple to implement an oracle for $f(\overline{x})$ using a square-root operation over the field $\mathbb{F}$. However, computing a circuit for $f(\overline{x})$ is less obvious, but can be done by appealing to truncations of formal power series ([Kal87, Val80]). As such, Kaltofen and Trager [KT90] have given an efficient factoring algorithm in the black-box model. While their solution is satisfactory in many regards, it heavily uses randomness. Obtaining a deterministic factorization algorithm is the main question of this column.

Open Question 1.4. *Is there an efficient* deterministic *algorithm for the polynomial factorization problem?* ◊

In this column we shall describe the recent advances in the area that are related to the open problems mentioned above. We start by describing the ideas behind the algorithms of Kaltofen [Kal89] and Kaltofen-Trager [KT90]. We then give two applications of polynomial factorization. One is

[6] We write $\deg_{x_i}(f)$ to denote the maximal degree of x_i in any monomial in f.

an algorithm for decoding Reed-Solomon codes ([Sud97, GS99]) and another is the hardness versus randomness paradigm for algebraic circuits ([KI04]). This second application concerns derandomizing algorithms for the *polynomial identity testing (PIT)* problem, which seeks to decide whether a given input circuit computes the zero polynomial. We explain this problem and describe its equivalence ([KSS15]) with obtaining deterministic algorithms for polynomial factorization. We then focus on a very special case of this problem, that of *divisibility testing*, and discuss deterministic algorithms ([For15]) for deciding whether a given low-degree polynomial divides a given sparse polynomial.

2 Factoring algorithms

We give a high level view of the factoring algorithms of Kaltofen [Kal89] and Kaltofen-Trager [KT90]. For more detailed expositions we refer the reader to the lecture notes of Sudan [Sud98] and the book by von zur Gathen and Gerhard [vzGG13].

While these algorithms seek to factor *multi*variate polynomials, a main insight is that one can reduce this problem to factoring polynomials of few variables where we can use algorithms that work in the dense representation. Such a reduction cannot be done to univariate polynomials (as over algebraically closed fields univariates factor into linear factors), but can be done to *bi*variate polynomials (which can be non-trivially irreducible). As such, we begin by discussing the bivariate case and then comment on the reduction to bivariates.

Initialization: To factor $f(x,y)$ of degree d, where $f = g(x,y) \cdot h(x,y)$ with g irreducible, it helps to assume that f satisfies certain technical conditions. In particular, we need that f is monic in x (so that $f = \alpha x^k + \sum_{i<k} f_i(y)x^i$) and that $f(x,0)$ is square-free (note that this also implies that f is square free). By using certain transformations (such as changes of variables and translation), we can assume these conditions hold. For example, if f is not square free then by computing the greatest common divisor of f and $\frac{\partial f}{\partial x}$ we get a non-trivial factor of f. The reason we require that f is monic in x is that we start by factoring $f(x,0)$ and later "lift" is to a factorization of f. It is also important in later applications of Gauss's lemma that we have a monic polynomial at hand.

Univariate factoring: The next step is to factor the polynomial $f(x,0) = g(x,0) \cdot h(x,0)$, that is, set $y = 0$ and factor the resulting univariate polynomial. Efficient algorithms for univariate factorization were given by Lenstra-Lenstra-Lovász [LLL82] (for finite fields there are also algorithms, for example by Berlekamp [Ber70]). Note that it may be the case that $g(x,0)$ is a reducible polynomial, so that g splits into factors when we set $y = 0$ despite being originally irreducible. We consider each $g_0(x,0)$ which is a factor of $g(x,0)$ (one can try all factors of $f(x,0)$) and write $f(x,0) = g_0(x,0) \cdot h_0(x,0)$.

Lifting: By setting $y = 0$ we reduced to factoring univariates, and in this step we gradually reintroduce y by "lifting" the factor $g_0(x,0)$. By that we mean that we compute a sequence of polynomials $g_i(x,y), h_i(x,y)$ that satisfy the identity $f(x,y) = g_i(x,y) \cdot h_i(x,y) \mod y^{2^i}$. Note that for $i = 0$ this is exactly the relation $f(x,0) = g_0(x,0) \cdot h_0(x,0)$ we already obtained, so that what remains is to compute g_{i+1} and h_{i+1} inductively from g_i and h_i via a process called *Hensel lifting*.

Theorem 2.1 (Hensel Lifting)**.** *Let R be a ring and $I \subseteq R$ an ideal (in our case $R = \mathbb{F}(x)[y]$, where $\mathbb{F}(x)$ is the field of rational functions in x, and $I = y^{2^i} \cdot R$). Let $f, g, h \in R$ such that $f = g \cdot h \mod I$. Assume further that there are elements $a, b \in R$ such that $ag + bh = 1 \mod I$, that is, g and h are coprime modulo I. Then there is an efficient way to compute g', h' such that $f = g' \cdot h' \mod I^2$, $g = g' \mod I$ and $h = h' \mod I$, and g', h' are coprime modulo I^2.*

It is important to note that in principle we can run this step for infinitely many times without ever being able to get a real factor. Indeed, consider the irreducible polynomial $f(x,y) = x(1-y)+1$. Notice that we can write $f = (1-y)\cdot(x+1/(1-y))$. It is clear that $f = (1-y)(x+1/(1-y)) = (1-y)(x+1+y+\cdots+y^{i-1}) \mod y^i$. Thus, the process that we described so far will keep computing higher and higher powers of y without ever stopping. The next step tells us exactly when to stop the lifting process.

Reducing the degree: We continue the Hensel lifting step until we compute polynomials g_k, h_k such that $f = g_k \cdot h_k \mod y^{4d^2}$ (recall that $\deg(f) = d$). Intuitively, g_k is an "approximate" factor of f, however, it may have high degree in x and y. As it turns out, one can show that g_k actually divides (modulo y^{4d^2}) another polynomial, of low degree, which has a common factor with f. Thus, what we do next is compute a (non-zero) polynomial g' such that $g' = g_k \cdot r \mod y^{4d^2}$, for some polynomial r, and such that $\deg_x(g') < \deg_x(f)$ and $\deg_y(g') \leq \deg_y(f)$, that is, g' satisfies the relevant degree constraints and it still contains "information" about a factor of f. To obtain such g' we notice that all we need to do is solve a linear system of equations in the unknown coefficients of g' and an external polynomial r. Indeed, notice that given the polynomial g_k the condition above is indeed a linear system in the unknown coefficients of g' and r. The reason that this system is solvable is that if we started from a factor g_0 of g then by Hensel lifting we can find r_k such that $g = g_k \cdot r_k \mod y^{4d^2}$. Notice that g, r_k are solutions to this linear system.

Computing a non-trivial factor: The last step now is to take any solution g' from the previous step and compute a factor of f. What we do is simply compute the greatest common divisor (gcd) of g' and f. One can show using the *resultant* of f and g (which we will not define here) that if indeed we started from a factor $g_0(x,0)$ of $g(x,0)$ then f and g' will always have a non trivial gcd. Thus we can factor f and continue by induction to find all its irreducible components.

This also explains the somewhat mysterious bound $4d^2$ from the previous step. The resultant (which we did not define) of f and g with respect to x is a polynomial in y of degree at most $2d^2$ (as d is an upper bound on the degree of y in both). The resultant is zero iff f and g share a common factor. Taking g' as above should intuitively mean that we work modulo a high enough power of y so that the resultant is not affected and should thus be identically zero, meaning that f and g' have a common factor. Here we had to be careful to work modulo a high enough power of y so that we are guaranteed that the resultant is identically zero and not just zero modulo some power of y.

Factoring multivariate circuits: The algorithm for factoring multivariate circuits is along the same lines as the bivariate case. Consider a circuit $f(x,\overline{y})$ with factorization $f = g(x,\overline{y})\cdot h(x,\overline{y})$. We will push the variables $\overline{y}$ into the underlying field and replace them with a single new variable t. That is, let $\tilde{f}(x,t,\overline{y}) := f(x,ty_1,\ldots,ty_n)$. Then $\tilde{f}(x,t,\overline{y})$ factors as f does, so that $\tilde{f}(x,t,\overline{y}) = \tilde{g}(x,t,\overline{y})\cdot\tilde{h}(x,t,\overline{y})$ where $\tilde{g}$ and $\tilde{h}$ are defined similarly. However, now we can think of this factorization in variables x,t in the field of rational expressions $\mathbb{F}(\overline{y})$. While this field is very large, the computations in this field will all involve small circuits so that the computations will be efficient (in particular, one moves to a $\mathsf{poly}(d)$-large extension field so that non-zeroness can be tested, with randomness, using the Schwartz-Zippel-DeMillo-Lipton Lemma [Sch80, Zip79, DL78]). One can then proceed as in the bivariate case, carefully observing that the operations involved (taking gcd's, solving linear systems of equations) can all be done so that the solutions involve small circuits.

Factoring in the black-box model: In the black-box setting Kaltofen and Trager [KT90] used roughly the same strategy as above, with a few simplifications. As above, one first restricts the

black-box polynomial f to a random bivariate polynomial by restricting f to a random subspace V. The effective Hilbert Irreducibility Theorem (see Kaltofen [Kal95]) shows that the irreducible factorization pattern is preserved with high probability. One can then reconstruct this bivariate polynomial and factor it in the dense representation. So far we have learned the factorization of f over the vector space V, and now need to use this factorization to obtain oracles for the factors of f at any given point $\overline{u} \in \mathbb{F}^n$. To do so, consider the space U formed from the span of V and $\overline{u}$. We can reconstruct f restricted to U as a trivariate polynomial, and lift our bivariate factorization on V to a factorization on U, using the factorization on V to anchor the factors on U. As the factorization pattern of f is preserved on V it is also preserved on U. Thus, we have the factors of f restricted to U explicitly as trivariate polynomials and thus can evaluate them at the desired point $\overline{u}$.

3 Two applications

We now give two applications of polynomial factorization. The first, to Reed-Solomon codes, only requires factorization in the dense representation but is important enough that we include it here. The second, to hardness versus randomness in algebraic circuit complexity, will need factorization of algebraic circuits.

3.1 Reed-Solomon codes

We give a high level explanation of the role polynomial factorization plays in decoding Reed-Solomon codes. Recall that a message of an $[n, k, n-k+1]_q$ Reed-Solomon code is a univariate polynomial $f(x)$ of degree at most k over a field $\mathbb{F}_q$ of size $q \geq n$. The encoding is given by evaluating f at predetermined points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$, that is, $\mathrm{Enc}(f) = (f(\alpha_1), \ldots, f(\alpha_n))$. In the decoding problem we are given a corrupted message, that is, a vector $\overline{y} \in \mathbb{F}_q^n$ such that the Hamming distance between $\mathrm{Enc}(f)$ and $\overline{y}$ is not too large and we have to decode f from $\overline{y}$. Welch and Berlekamp [WB86] gave a decoding algorithm that can decode f from e errors for any $e < (n-k+1)/2$. That is, this algorithm can correct any number of errors up to half the minimum distance between any two codewords, which is best possible for *unique* decoding. Very roughly, the decoding algorithm works as follows. It first computes two polynomial $E(x)$ and $N(x)$, where $E(x)$ is a non-zero polynomial of degree e and N is a polynomial of degree $e+k$. Ideally we would like that E will vanish on each α_i for which $y_i \neq f(\alpha_i)$ and that $N(x) = E(x) \cdot f(x)$, so that E is the *error locator polynomial*. Since we do not know the location of the errors we cannot compute these quantities directly. Nevertheless, Welch and Berlekamp [WB86] set up a system of linear equations whose solution can be interpreted as the coefficients of two polynomials E' and N', of degree e and $e+k$, respectively. The crux is that for any nonzero solution E' and N' it holds that $N'/E' = f$. Notice that the final step of the Welch-Berlekamp [WB86] algorithm requires the division of two univariate polynomials, which can be thought of as a weak type of factorization.

The next question to ask is what if the number of errors e is larger than $(n-k+1)/2$, that is, what if the number of errors is more than half the minimum distance between any two codewords. In this case there is no unique solution to the decoding problem, but combinatorially one can show that there are not too many solutions either, if e is not too large. In this case one can ask the algorithm to output a list of all candidate polynomials f. In a seminal work Sudan [Sud97] gave an efficient algorithm for list-decoding of Reed-Solomon codes. Later with Guruswami [GS99] they gave a different algorithm that works for a larger error rate, matching the combinatorial bound. Similar to the Welch-Berlekamp [WB86] algorithm, both Sudan's [Sud97] and Guruswami-Sudan's [GS99] algorithms work by setting a system of linear equations whose solution is a *bivariate* polynomial $Q(x, y)$. They then show that any candidate solution f gives rise to a factor of Q of the form

$y - f(x)$. The final step of their algorithm is finding all irreducible components of Q that are linear in y.

3.2 Hardness versus Randomness for Algebraic Circuits

In this section, we discuss the *hardness versus randomness* paradigm as applied to algebraic circuits. Recall that for boolean circuits, this paradigm (as exemplified by Nisan-Wigderson [NW94]) says that hard functions are essentially those functions that look random to efficient computation, and in particular one can build efficient pseudorandom generators from any hard enough function. A work of Kabanets-Impagliazzo [KI04] extended this paradigm to algebraic circuits as we now explain. The notion of a hard polynomial is the same as in the boolean regime — we ask for a polynomial (computable in exponential time) that requires exponentially large algebraic circuits. The notion of pseudorandomness is slightly different however. Here, we ask for a small set of points $\mathcal{H} \subseteq \mathbb{F}^n$ such that for any efficiently computable polynomial $f(x_1, \cdots, x_n)$ it holds that f is non-zero iff it has a non-zero evaluation point within $\mathcal{H}$, that is, $f \not\equiv 0$ iff $f|_{\mathcal{H}} \not\equiv 0$. As non-zero polynomials are non-zero on random evaluation points (by the Schwartz-Zippel-DeMillo-Lipton Lemma [Sch80, Zip79, DL78]), it follows that such sets "fool" small algebraic circuits at least with respect to non-zeroness.

This notion of pseudorandomness is called a *hitting set*, and can be used to solve the *polynomial identity testing (PIT)* problem where we are given an algebraic circuit and seek to decide whether the polynomial it computes is zero or not. As efficient randomized algorithms follow easily from the Schwartz-Zippel-DeMillo-Lipton Lemma [Sch80, Zip79, DL78], the main challenge is to develop deterministic algorithms. As discussed earlier, there are white-box (where the algorithm can inspect the structure of the given circuit) and black-box (where the algorithm can only use the circuit to evaluate the polynomial) versions of this problem. An easy argument shows that deterministic black-box PIT algorithms are equivalent to constructions of small hitting sets.

PIT, as a meta-problem about algebraic circuits, is to some extent the most general algebraic problem for which randomness is helpful. Indeed, various papers ([KUW86, MVV87, SV10, DOS14, KSS15]) show that many algorithmic problems where randomness is helpful can be derandomized assuming deterministic algorithms for PIT. Derandomizing PIT even for restricted classes of circuits is a challenging problem of ongoing research.

We now discuss the result of Kabanets and Impagliazzo [KI04] for transforming a hard polynomial into a hitting set. An equivalent formulation is to design a *generator*, which is a polynomial map $\overline{\mathcal{G}} = (g_1, \ldots, g_n) : \mathbb{F}^r \to \mathbb{F}^n$, where r is much smaller than n, such that if C is a small circuit computing a nonzero polynomial then $C \circ \overline{\mathcal{G}} = C(g_1(\overline{y}), \ldots, g_n(\overline{y})) \neq 0$. A standard argument shows that such $\overline{\mathcal{G}}$ can be converted to a small hitting set (when r small) via interpolation.

The starting observation is to consider generators with stretch $n - r = 1$. That is, suppose $f(\overline{y})$ requires $\exp(\Omega(r))$-size circuits, where $\overline{y}$ consists of $r = n - 1$ variables. Consider then the generator $\overline{\mathcal{G}}(\overline{y}) := (\overline{y}, f(\overline{y}))$. Suppose that $C(\overline{y}, z)$ is a non-zero polynomial computable by a size-s algebraic circuit, but that $C \circ \overline{\mathcal{G}} = C(\overline{y}, f(\overline{y})) = 0$. It follows (from Gauss's lemma, see Lemma 5.1) that this implies that $z - f(\overline{y})$ divides $C(\overline{y}, z)$ (and is actually an irreducible factor), which by Kaltofen's [Kal89] result implies that $z - f(\overline{y})$, and thus $f(\overline{y})$ itself, is computable by a $\mathsf{poly}(s)$-size circuit. However, from this we see that $s \geq \exp(\Omega(r)) = \exp(\Omega(n))$. Taking the contrapositive, it follows that if $C(\overline{y}, z)$ is a non-zero polynomial computable by a $\mathsf{poly}(n)$-size circuit then it must be that $C \circ \overline{\mathcal{G}} \neq 0$, as we wanted.

This construction has two deficiencies. The first and most obvious is that the stretch $n - r$ is 1 here, while we would want an exponential stretch to obtain small hitting sets[7]. The second is that

[7] In the boolean regime, one can amplify stretch 1 to stretch $\mathsf{poly}(n)$ as shown by Yao [Yao82] using repeated composition. This idea is unfortunately not valid here as this produces a generator of *exponential* degree, which

this generator is not even efficiently computable as we need to evaluate $f(\overline{y})$, which we assumed is hard to compute! To fix these issues, Kabanets and Impagliazzo [KI04] follow the solution of Nisan and Wigderson [NW94] for these same issues in the boolean regime. This solution is to consider generators that take a hard polynomial f on ℓ variables and evaluate it on many *nearly disjoint* subsets of the r input variables $\overline{y}$. That is, to use generators of the form $\overline{\mathcal{G}}(\overline{y}) := (f(\overline{y}|_{S_i}))_{S_i \in \mathcal{F}}$, where $\mathcal{F}$ is a collection of sets in $[r]$ of size ℓ, and $\overline{y}|_{S_i}$ is the restriction of the input variables in $\overline{y}$ to those variables in S_i.

This construction deals with both of these problems, as we can choose $r \gg \ell$ so that $|\mathcal{F}|$ is exponential in r and hence we obtain a large stretch. Similarly, as we only need to evaluate the generator in $\mathsf{poly}(|\mathcal{F}|)$-time, we can afford to evaluate f on inputs of size ℓ even if this takes $\exp(\ell)$ time as this can be $\mathsf{poly}(|\mathcal{F}|)$ as $|\mathcal{F}|$ is so large. However, we must argue that this construction still has the generator property. To do so, $\mathcal{F}$ is chosen as a *design* so that for any $i \neq j$, S_i and S_j have a small intersection, that is $|S_i \cap S_j| \ll \ell$. We now argue that this construction works by using the standard *hybrid* argument. Suppose that C is non-zero polynomial computed by a small circuit and yet $C \circ \overline{\mathcal{G}} = 0$. Putting it another way, we have that $C(x_1, \ldots, x_n) \neq 0$ but that $C(g_1(\overline{y}), \ldots, g_n(\overline{y})) = 0$. In particular, this implies that there is some k such that $C(g_1(\overline{y}), \ldots, g_{k-1}(\overline{y}), x_k, x_{k+1}, \ldots, x_n) \neq 0$ but $C(g_1(\overline{y}), \ldots, g_{k-1}(\overline{y}), g_k(\overline{y}), x_{k+1}, \ldots, x_n) = 0$. As non-zero polynomials are non-zero on random values, we can fix $x_{k+1}, \ldots, x_n$ to random values $\alpha_{k+1}, \ldots, \alpha_n \in \mathbb{F}$ while preserving non-zeroness. For notational convenience, write the resulting circuit as $C'(x_1, \ldots, x_{k-1}, z) := C(x_1, \ldots, x_{k-1}, z, \alpha_{k+1}, \ldots, \alpha_n)$. We then have that $C'(g_1(\overline{y}), \ldots, g_{k-1}(\overline{y}), z) \neq 0$ but $C'(g_1(\overline{y}), \ldots, g_{k-1}(\overline{y}), g_k(\overline{y})) = 0$. This implies as before that $z - g_k(\overline{y}) = z - f(\overline{y}|_{S_k})$ divides $C'(g_1, \ldots, g_{k-1}, z)$, and it is tempting to think this implies that $f(\overline{y}|_{S_k})$ has a small circuit as before, in violation of lower bound for f. This is not immediately obvious though, as to compute $C'(g_1, \ldots, g_{k-1}, z)$ requires computing $f(\overline{y}|_{S_i})$ for $i < k$ via *brute force*, so that we do not have a small enough upper bound on the complexity of this circuit to violate the lower bound on the complexity of f.

To deal with this problem we must use the requirement that the sets S_i form a design. In particular, we partition $\overline{y} = (\overline{y}|_{S_k}, \overline{y}|_{\overline{S_k}})$ where $\overline{S_k} = [r] \setminus S_k$. We then see that we can *also* set those variables in $\overline{S_k}$ to random values β as this does not affect g_k because it does not depend on these variables. As such, we obtain that $C'(g_1(\overline{y}|_{S_k}, \beta), \ldots, g_{k-1}(\overline{y}|_{S_k}, \beta), z) \neq 0$ but $C'(g_1(\overline{y}|_{S_k}, \beta), \ldots, g_{k-1}(\overline{y}|_{S_k}, \beta), f(\overline{y}|_{S_k})) = 0$. Thus, as before $z - f(\overline{y}|_{S_k})$ divides $C'(g_1(\overline{y}|_{S_k}, \beta), \ldots, g_{k-1}(\overline{y}|_{S_k}, \beta), z)$ but we now use that this latter polynomial has a non-trivially small circuit. That is, as each $|S_i \cap S_k| \ll |S_k|$ for $i < k$ it follows that each $g_i(\overline{y}|_{S_k}, \beta)$ can be computed by a circuit of size $\exp(|S_i \cap S_k|) \ll \exp(|S_k|) = \exp(\ell)$. Thus, if C is small enough (so that C' is also) and f requires $\exp(\Omega(\ell))$-size circuits, this is a contradiction to Kaltofen's [Kal89] result that small circuits have factors with small circuits. Hence, we must have that $C \circ \overline{\mathcal{G}} \neq 0$ and as $\overline{\mathcal{G}}$ is now efficiently computable with large stretch we obtain the desired generator.

Note that this argument crucially used Kaltofen's [Kal89] argument that small circuits have factors computable by small circuits, but only used it for factors of the form $z - f(\overline{y})$. Dvir, Shpilka and Yehudayoff [DSY09] showed that such factors of low-depth formulas are also computable by low-depth formulas (of slightly larger depth), at least for polynomials of low individual-degree. As mentioned, Oliveira [Oli15] recently extended this to all factors of such formulas. However, we still lack such results for bounded depth formulas of large individual-degree, which is crucial in extending this hardness versus randomness paradigm to such formulas.

Open Question 3.1. *Can the irreducible factors of n-variate polynomials that are computed by depth-D algebraic circuits of size-s be computed by depth-$O(D)$ algebraic circuits of size $\mathsf{poly}(n, s)$?*

cannot be converted to a small hitting set using existing techniques.

$\Diamond$

4 Factorization and checking polynomial identities

In the previous sections, we described randomized algorithms for factoring polynomials and PIT, and how factorization algorithms play a role in potentially derandomizing PIT using hardness versus randomness. Indeed, these two problems seem tightly connected as Shpilka and Volkovich [SV10] observed that deterministic factorization algorithms immediately derandomize PIT, as $f(\overline{x})$ is identically zero if and only if $f(\overline{x})+yz$ has two irreducible factors. We now discuss how derandomizing PIT is in turn sufficient to derandomize these factorization algorithms, as recently shown by Kopparty, Saraf and Shpilka [KSS15].

We begin by discussing the black-box algorithm of Kaltofen and Trager [KT90]. The main use of randomness in this algorithm was the choice of the random subspace V where we initially factor f. To derandomize this step we need to fool the criteria for whether V is a good subspace. It turns out that whether V preserves the factorization pattern of f is determined by the (non)-vanishing of certain polynomials. These polynomials are resultants of the factors of f, where as mentioned above the resultant of two polynomials g and h is a third polynomial that (essentially) vanishes iff g and h have a common factor. If we are able to find a subspace where all of these resultants are non-zero then the factorization pattern has been preserved. Due to Kaltofen's [Kal89] result we know that the factors of f *have* small circuits, so that even though we do not have them (we are trying to compute them!) it follows that their resultants also have small circuits. Thus, if we have a hitting set for small circuits we can find an evaluation point in this hitting set where all resultants are non-zero, yielding the desired subspace V. Having derandomized this one main step the derandomization of the rest of the algorithm essentially follows.

The white-box algorithm for factoring (given a white-box PIT algorithm) proceeds essentially as we described above (where our description above is actually based on Kopparty, Saraf and Shpilka [KSS15]). That is, we proceed to use the bivariate factoring algorithm on $f(x, ty_1, \ldots, ty_n)$ in the field $\mathbb{F}(\overline{y})$ in the variables t and x. This involves a reduction to a "formal" two-dimensional subspace spanned by $\overline{v} = (1, 0, \ldots, 0)$ and the formal vector $\overline{w} = (0, y_1, \ldots, y_n)$ over $\mathbb{F}(\overline{y})$. While this is a large field, the elements of the field actually encountered in the algorithm are computed by small circuits so that derandomization of PIT suffices to be able to manipulate these circuits intelligently. In particular, this allows one to (non-trivially) solve systems of linear equations involving circuits and also taking gcd's.

This high level description establishes the equivalence of derandomizing PIT and derandomizing polynomial factorization in both the white-box and the black-box models. It is instructive to note though, that in order to derandomize the factorization algorithm for a restricted circuit class, such as sparse polynomials, one needs PIT for a much larger class as the operations used (such as solving systems of linear equations or computing resultants) involve determinants of the original polynomials and thus increases their complexity. In particular, the fact that we have very efficient black-box PIT algorithms for sparse polynomials does not help us to derandomize the factorization algorithm for them.

Despite the complicated algebraic reasoning involved in the above factoring algorithms, there are simple problems here for which derandomization may be easier. The first is that of *irreducibility testing*, where we seek to decide whether a given circuit computes an irreducible polynomial or not. This is interesting as the above reduction from PIT to factoring is actually to irreducibility testing, as $f(\overline{x})$ is zero if and only if $f(\overline{x}) + yz$ is reducible. Thus, we are led to the following question.

Open Question 4.1. *Does irreducibility testing of a class of restricted algebraic circuits reduce to*

PIT of another restricted class of algebraic circuits of roughly similar complexity? ◊

Another problem is that of *divisibility testing*, which asks if a polynomial $f(\overline{x})$ divides a polynomial $g(\overline{x})$ when both polynomials are given by algebraic circuits. This problem is clearly solvable by factoring both f and g, but one can hope for a simpler algorithm. In particular, this algorithm could be relevant for factoring, particularly in the above derandomization of black-box factoring via PIT. That is, we wanted to find a subspace V where the factorization pattern of a polynomial was preserved, which can alternatively be roughly phrased as saying that certain (non)divisibilities are preserved in this subspace. Understanding when two polynomials divide each other could help in derandomizing this step in special cases, and indeed divisibility testing is the subject of our next section.

5 Divisibility Testing

As obtaining deterministic factoring algorithms seems difficult even for simple algebraic circuit classes, we now describe the weaker problem of *divisibility* testing: given two multivariate polynomials $f(\overline{x})$ and $g(\overline{x})$ we wish to decide whether $f(\overline{x})$ divides $g(\overline{x})$. Clearly if we can factor both f and g then we can compare their irreducible factors (using PIT) to decide divisibility, and in particular this shows that divisibility testing has an efficient randomized algorithm by appealing to the above factoring algorithms. However, obtaining *deterministic* divisibility testing algorithms seems perhaps simpler than derandomizing factoring algorithms.

In this section we thus discuss recent progress on deterministic divisibility testing algorithms via corresponding advances in deterministic PIT algorithms. We begin with perhaps the simplest imaginable instance of this problem, where $f(\overline{x})$ is linear and $g(\overline{x})$ is sparse, and see that the known deterministic algorithms use non-trivial PIT results ([RS05, SSS13]) on a model combining sparse polynomials and powers of a linear form. We then discuss recent progress by Forbes [For15] on the case where $f(\overline{x})$ is of constant (total) degree, and discuss the underlying PIT of a model combining sparse polynomials and powers of low-degree polynomials. Finally, we mention a general reduction of divisibility testing to PIT by Forbes [For15] which roughly preserves the complexity of the original polynomials.

5.1 Testing Divisibility of Linear into Sparse

Known deterministic algorithms for divisibility proceed via reductions to PIT. For univariate polynomials, it is a familiar fact that $x - \alpha$ divides $f(x)$ iff the evaluation $f(\alpha)$ is zero. Generalizing this via long-division of polynomials, we have the following lemma that we have already used extensively.

Lemma 5.1 (Gauss's lemma)**.** *Let $y - f(\overline{x}), g(\overline{x}, y) \in \mathbb{F}[\overline{x}, y]$. Then $y - f(\overline{x})$ divides $g(\overline{x}, y)$ iff $g(\overline{x}, f(\overline{x})) = 0$.* □

While the usage of this lemma is restricted to polynomials of the form $y - f(\overline{x})$, one can see that any non-zero linear polynomial is of this form (up to a global constant irrelevant for divisibility) so that we can write it as $y - \ell(\overline{x})$ with ℓ being linear. Thus, if $g(\overline{x})$ is a sparse polynomial $g(\overline{x}, y) = \sum_{i=1}^{s} \alpha_i \overline{x}^{\overline{a}_i} y^{b_i}$ then $y - \ell(\overline{x})$ divides g iff $\sum_{i=1}^{s} \alpha_i \overline{x}^{\overline{a}_i} \ell(\overline{x})^{b_i} = 0$, giving a reduction to PIT. While in this summation there is only a single linear function $\ell(\overline{x})$, existing algorithms seemingly[8]

[8]The algorithm of Agrawal, Saha, Saptharishi and Saxena [ASSS12] can handle PIT of polynomials that can be written as a composition of a polynomial on few variables with a small number of sparse polynomials. This does not seem immediately applicable here because we multiply powers of our linear function by many different sparse polynomials (one for each power).

cannot exploit this and thus we will actually reduce to PIT where the linear form can *vary* with the summation, as we now summarize.

Proposition 5.2. *Testing whether a linear polynomial divides a sparse polynomial reduces to PIT of sums of a monomial times a power of a linear form, that is, expressions of the form* $\sum_{i=1}^{s} \alpha_i \overline{x}^{\overline{a}_i} \ell_i(\overline{x})^{b_i}$. □

This model of sums of a monomial times powers of a linear form can be more succinctly named as "$\sum \mathrm{m} \bigwedge \sum$", where "$\bigwedge$" is used to denote (low-degree) exponentiation (to mirror the '^' character) and "m" is used to indicate a monomial. This model was first studied by Saha, Saptharishi and Saxena [SSS13] (who referred to it as *semi-diagonal depth-3*), who gave a polynomial-time deterministic white-box PIT for this class. This generalized an earlier work of Saxena [Sax08] who gave such an algorithm for sums of powers of linear forms ($\sum \bigwedge \sum$).

Both algorithms can be viewed as proceeding in two steps. The first is to convert the $\sum \mathrm{m} \bigwedge \sum$ formula to a *read-once (oblivious) algebraic branching program (roABP)* of similar size. This model of computation captures those polynomials expressible as $\mathrm{tr}(A_1(x_1) \cdots A_n(x_n))$, where the A_i are $w \times w$ matrices with univariate polynomial entries and "tr" is the trace operator. The size of this computation is its width w. While the exact definition is not crucial here, we note that these computations can be split into left- and right-halves with only w-numbers that "cross" this partition, and this is the property used to design PIT algorithms for such computations. Once the conversion from $\sum \mathrm{m} \bigwedge \sum$ to roABP is done, one can invoke the deterministic poly-time PIT for this class[9] due to Raz and Shpilka [RS05].

To motivate the first step, consider the following decomposition of a power of a linear polynomial, which follows easily from the binomial theorem.

Lemma 5.3. *Let* $\ell(\overline{x}, \overline{y})$ *be a linear polynomial. Then* $\ell(\overline{x}, \overline{y})^d$ *can be expressed as* $\ell(\overline{x}, \overline{y})^d = \sum_{i=0}^{d} f_i(\overline{x}) g_i(\overline{y})$ *for some* f_i *and* g_j *of degree* $\leq d$. □

This lemma shows that even though powers of linear polynomials such as $(x_1 + \cdots + x_n)^n$ can create many monomials, the amount of "correlation" between any two partitions of the variables can be bounded by the degree of the polynomial. This is the above mentioned property of small read-once oblivious computation, where we first "read" the $\overline{x}$ variables and then "read" the $\overline{y}$-variables, and indeed this property actually *characterizes* roABPs as shown by adapting an argument of Nisan [Nis91].

However, one can make the above reduction more explicit by using the following *duality trick* of Saxena [Sax08].

Lemma 5.4 (Saxena [Sax08])**.** *Let* $\ell(x_1, \ldots, x_n)$ *be a linear polynomial. Then* $\ell(\overline{x})^d$ *can be expressed as a small sum of products of univariate polynomials, that is*

$$\ell(\overline{x})^d = \sum_{i=1}^{s} \prod_{j=1}^{n} g_{i,j}(x_j) \ ,$$

where $s \leq \mathsf{poly}(n, d)$ *and each univariate* $g_{i,j}$ *is of degree at most* d*. Furthermore, this decomposition can be computed in* $\mathsf{poly}(n, d)$ *time.*

It should be intuitively clear that such (small) sums of products of univariates can be computed in a "read-once oblivious fashion" by reading variables in the order $x_1 \prec \cdots \prec x_n$. Further, as the above computational model is preserved under multiplication by monomials and closed under a small number of additions, this yields the desired structural result (made explicit by Forbes and Shpilka [FS13b]).

[9] Their PIT algorithm is stated for non-commutative algebraic branching programs, but it is easily adapted to roABPs (see for example the thesis of Forbes [For14]).

Proposition 5.5. *Small sums of monomials times powers of linear polynomials ($\sum \mathrm{m} \bigwedge \sum$) can be computed by small sum of products of univariates, and thus a small read-once oblivious algebraic branching program.* □

While there is the poly-time white-box PIT for roABPs due to Raz-Shpilka [RS05], the best black-box algorithms run in $s^{O(\lg n)}$ for roABPs of size-s in n-variables ([FS13b, FSS14, AGKS15, GKST15]). There is even no such poly-time black-box PIT even for this $\sum \mathrm{m} \bigwedge \sum$ model (see for example [FS13b, ASS13, For15], the last of which gives a $s^{O(\lg \lg s)}$-time black-box algorithm for size s).

We now give a vignette illustrating some of the ideas entering the PIT algorithms mentioned above. In particular, we mention the $s^{O(\lg s)}$-time black-box PIT algorithm of Forbes and Shpilka [FS13a] for the $\sum \bigwedge \sum$ model, that is, expressions of the form $\sum_{i=1}^{s} \ell_i(\overline{x})^{d_i}$ where each ℓ_i is linear. This is weaker than the model needed for divisibility testing (as we do not multiply by monomials inside the summation), but also slightly stronger (as we allow the linear function to vary with the summation). The PIT algorithm follows from first understanding lower bounds for this class, making this lower bound *robust*, and then deriving the algorithm.

Lower bounds for this model can be analyzed by using the *partial derivative method* of Nisan and Wigderson [NW96] (which is basically the method of *catalecticants* of Sylvester [Syl51]). This method studies the action of taking partial derivatives from calculus, which maps a polynomial to another polynomial. For a polynomial $f(x_1, \ldots, x_n)$, we define $\boldsymbol{\partial}(f)$ to be the set of *all* partial derivatives to *all* orders. For example, $x^2 + xy$ has a set of derivatives (excluding zeroes) of $\boldsymbol{\partial}(x^2 + xy) = \{2x + y, x, 2, 1\}$ corresponding to the $\partial/\partial x$, $\partial/\partial y$, $\partial^2/\partial^2 x$, and $\partial^2/\partial x \partial y$ derivatives. While a polynomial can have exponentially many distinct derivatives, we can more accurately measure the complexity of the polynomial by the *dimension* of the space of derivatives, denoted $\mu(f) := \dim(\operatorname{span} \boldsymbol{\partial}(f))$, as these polynomials naturally live in the vector space of all polynomials.

This measure μ is a natural *complexity measure* as it obeys subadditivity ($\mu(f+g) \leq \mu(f)+\mu(g)$), which follows from the fact that taking partial derivatives is a linear operation and that dimension of vector spaces is subadditive. More importantly, the chain rule shows that for any f we have that $\frac{\partial}{\partial x} f^d = d \cdot f^{d-1} \cdot \frac{\partial}{\partial x} f$. In particular, if f is a linear function ℓ then $\frac{\partial}{\partial x}\ell$ is constant so that $\frac{\partial}{\partial x}\ell^d$ is a scalar multiple of ℓ^{d-1}. By induction, this implies that the derivatives of ℓ^d are scalar multiples of some ℓ^{d-i}, so that $\mu(\ell^d) \leq d+1$ and that $\mu(\sum_{i=1}^{s} \ell_i(\overline{x})^{d_i}) \leq s(d+1)$ if $d_i \leq d$ for all i. As algebraic complexity theory is typically concerned with low-degree polynomials ($d \leq \mathsf{poly}(n)$) this shows that μ is small for small $\sum \bigwedge \sum$ formulas.

In contrast, one can easily see that $\boldsymbol{\partial}(x_1 \cdots x_n) = \{\prod_{i \in S} x_i\}_{S \subseteq [n]}$ so that $\mu(x_1 \cdots x_n) = 2^n$. This easily yields the lower bound of $2^{\Omega(n)}$ for the size of $x_1 \cdots x_n$ as a $\sum \bigwedge \sum$ formula (which is tight).

We [FS13a] observed, following a slightly weaker result of Agrawal-Saha-Saxena [ASS13], that this lower bound can be extended by using the notion of a *monomial order* "$\prec$" (see for example [CLO07]), which is a method of totally ordering monomials. Each (non-zero) polynomial then has a *leading monomial*, which is the largest monomial with respect to $\prec$ that has a non-zero coefficient in f. One can then easily show that if f has a leading monomial $\overline{x}^{\overline{a}} = \prod_i x_i^{a_i}$ then $\mu(f) \geq \mu(\overline{x}^{\overline{a}}) \geq 2^{|\operatorname{Supp}(\overline{a})|}$, where $|\operatorname{Supp}(\overline{a})|$ is the number of variables appearing in $\overline{x}^{\overline{a}}$. This fact can be thought of as saying that the measure μ is *robust*, in that it ignores lower order terms. This then yields the following structural result.

Theorem 5.6 (Forbes-Shpilka [FS13a])**.** *Let $f \in \mathbb{F}[\overline{x}]$ be a non-zero polynomial computed by a size-s degree-d $\sum \bigwedge \sum$ formula, that is $f(\overline{x}) = \sum_{i=1}^{s} \ell_i(\overline{x})^{d_i}$ where the ℓ_i are linear and $d_i \leq d$. Then for any monomial order, the leading monomial of f involves $O(\lg sd)$ variables.*

This structural result easily implies a black-box PIT algorithm running in time $\mathsf{poly}(s, n, d)^{O(\lg sd)}$. First, one can brute-force guess the $O(\lg sd)$ variables in the leading monomial in $n^{O(\lg sd)}$ time.

Zero-ing out variables outside this set leaves a low-degree polynomial on $O(\lg sd)$ variables, which was non-zero iff the original polynomial was non-zero. One can then brute-force the black-box PIT in $d^{O(\lg sd)}$ evaluations, for an overall hitting set of size $\mathsf{poly}(n,d)^{O(\lg sd)}$.

We remark that while a similar size hitting sets can also be obtained for the even more general model of roABPs, Forbes-Shpilka-Saptharishi [FSS14] showed how to use the above ideas to reduce PIT of size-s $\sum\bigwedge\sum$ on n variables to similar sized roABPs on $O(\lg sd)$ variables. By then applying the black-box PIT for roABPs of Forbes-Shpilka [FS13b] yields hitting sets of size $s^{O(\lg\lg s)}$ for size-s $\sum\bigwedge\sum$, which is currently the best known hitting set.

Open Question 5.7. *Give an explicit polynomial-size hitting set for $\sum\bigwedge\sum$ formulas, that is, polynomials expressed as $\sum_i \ell_i(\overline{x})^{d_i}$.* ◊

5.2 Testing Divisibility of Constant-Degree into Sparse

The previous section discussed testing whether a polynomial $f(\overline{x})$ divides a polynomial $g(\overline{x})$ when f is linear and g is sparse. We now consider the generalization to when f is of *constant (total) degree*, and for concreteness we focus on when f is quadratic as this contains all of the main ideas. While this may seem like an innocuous generalization, it requires significant new ideas. In particular, one needs to first obtain a reduction of this divisibility testing to PIT for a suitably restricted model of computation, then obtain lower bounds for this model, and finally hope that the lower bounds can be turned into a PIT algorithm.

The first idea is to generalize the reduction from divisibility testing to PIT. When f is linear, we could without loss of generality express $f(\overline{x}) = z - \ell(\overline{y})$ for linear ℓ and some variable partition $\overline{x} = (\overline{y}, z)$. When f is quadratic and expressible as $f(\overline{x}) = z - q(\overline{y})$ for quadratic q, testing whether f divides $g(\overline{y}, z) = \sum_{i=1}^{s} \alpha_i \overline{y}^{\overline{a}_i} z^{b_i}$ reduces to PIT of $\sum_{i=1}^{s} \alpha_i \overline{y}^{\overline{a}_i} q(\overline{y})^{b_i}$ as was done when f was linear. This resulting model is that *sums of monomials times powers of quadratics*, denoted $\sum \mathrm{m} \bigwedge \sum \prod^2$.

However, general quadratics f are not expressible as "$f(\overline{x}) = z - q(\overline{y})$" (even under a possible change of basis). However, one can *still* reduce the quadratic dividing sparse question to PIT of $\sum \mathrm{m} \bigwedge \sum \prod^2$. One method is to observe that one *can* assume (possibly after a change of variables) that a general quadratic is of the form "$f(\overline{x}) = z^2 + \ell(\overline{y})z + q(\overline{y})$", where ℓ is linear and q is quadratic. By appealing to long-division of polynomials with remainder, one can then obtain the desired reduction.

Proposition 5.8 (Forbes [For15])**.** *Testing divisibility of a quadratic $f(\overline{x})$ into a sparse $g(\overline{x})$ reduces to PIT of sums of monomials times powers of quadratics, that is, $\sum \mathrm{m} \bigwedge \sum \prod^2$.* □

Unfortunately, the efficiency of this reduction is exponential in the degree of f so that it cannot handle super-constant degree within polynomial time. However, given the above results showing that factoring (and thus divisibility) reduces to PIT, one could hope for a *general* reduction from divisibility testing to PIT that roughly preserves the complexity of the original polynomials, which was indeed recently observed for "nice" classes of circuits.

Proposition 5.9 (Forbes [For15])**.** *Let $\mathcal{C}$ and $\mathcal{D}$ be "nice" classes of algebraic circuits. Testing divisibility of $f \in \mathcal{C}$ to $g \in \mathcal{D}$ reduces to PIT of $\sum \mathcal{D} \bigwedge \mathcal{C}$, that is, expressions of the form $\sum_i g_i f_i^{d_i}$ for $f_i \in \mathcal{C}$ and $g_i \in \mathcal{D}$.* □

This reduction can be shown by observing that *if* f divided g, then $h = g/f$ is a polynomial and a circuit for h can be constructed using Strassen's [Str73] procedure for eliminating divisions. However, it turns out that this procedure can be executed *regardless* of whether f divides g (possibly yielding a garbage polynomial). Thus, the procedure outputs some quotient candidate $\widetilde{g/f}$ so that f divides g iff $g - f \cdot \widetilde{g/f} = 0$. Studying the complexity of the resulting expression yields the above result, and gives an alternate proof of the reduction of quadratics dividing sparse to PIT of $\sum \mathrm{m} \bigwedge \sum \prod^2$.

With this reduction in hand, one can then turn to obtaining lower bounds (and hopefully PIT) for $\sum \mathrm{m} \bigwedge \sum \prod^2$ formulas. However, even the subclass of $\sum \bigwedge \sum \prod^2$ requires non-trivial analysis as the two techniques discussed so far (the partial derivative technique of Nisan and Wigderson [NW96], and the roABP techniques essentially due to Nisan [Nis91]) fail to give any lower bound for $\sum \bigwedge \sum \prod^2$ formulas. Thankfully, Kayal [Kay12] introduced the method of *shifted partial derivatives* to obtain lower bounds for $\sum \bigwedge \sum \prod^2$ formulas. This method was further refined by Gupta, Kayal, Kamath and Saptharishi [GKKS14] who gave breakthrough lower bounds for non-trivial classes of depth-4 formulas, spurring numerous follow-up works (see the survey of Saptharishi [Sap15] for more on these developments).

In particular, we have an exponential lower bound again for the monomial.

Theorem 5.10 (Kayal [Kay12], Gupta-Kayal-Kamath-Saptharishi [GKKS14])**.** *The monomial $x_1 \cdots x_n$ requires $\exp(n/t)$ size to be computed as a $\sum \bigwedge \sum \prod^t$ formula, that is, expressions of the form $\sum_i f_i(\overline{x})^{d_i}$, where $\deg f_i \leq t$.*

Mirroring the above strategy of making the lower bound robust, Forbes [For15] showed that this lower bound implies that the leading monomial of a $\sum \bigwedge \sum \prod^2$ formula involves few variables, which as above implies a quasipolynomial-time black-box PIT algorithm for this class of formulas. Mahajan, Rao and Sreenivasaiah [MRS14] also obtained a similar result for the multilinear polynomials computed by $\sum \bigwedge \sum \prod^2$ formula (appealing to the hardness of representation approach of Shpilka-Volkovich [SV09]).

While the above results cover a large part of the difficulty of PIT for $\sum \bigwedge \sum \prod^2$ formulas, note that there is an inherent obstacle in extending these ideas to $\sum \mathrm{m} \bigwedge \sum \prod^2$. Specifically, the above lower bounds (and PIT) use that the monomial $x_1 \cdots x_n$ is hard for $\sum \bigwedge \sum \prod^2$, however the monomial is trivially easy for $\sum \mathrm{m} \bigwedge \sum \prod^2$ as this model can compute any sparse polynomial. This suggests that perhaps a different approach is needed.

For the $\sum \mathrm{m} \bigwedge \sum$ model, the alternate approach is to appeal to the techniques for the much stronger class of roABPs. Unfortunately these techniques are not relevant for even $\sum \bigwedge \sum \prod^2$ as these are hard for roABPs to compute. Nevertheless, the idea of *translations* of Agrawal, Saha and Saxena [ASS13] (introduced for PIT of subclasses of roABPs) can be applied here. That is, while the monomial $x_1 \cdots x_n$ is easy for $\sum \mathrm{m} \bigwedge \sum \prod^2$, the translate $(x_1 + 1) \cdots (x_n + 1)$ has no easy such upper bound as the model $\sum \mathrm{m} \bigwedge \sum \prod^2$ is not (obviously) closed under translation. Indeed, Forbes [For15] was able to extend the method of shifted partial derivatives in this case to obtain the desired robust lower bound, implying the desired PIT and divisibility testing result.

Theorem 5.11 (Forbes [For15])**.** *If $f(\overline{x})$ is computed by a size-s $\sum \mathrm{m} \bigwedge \sum \prod^2$ formula, then the leading monomial of $f(x_1 + 1, \cdots, x_n + 1)$ involves $O(\lg s)$ variables. In particular, this yields deterministic $s^{O(\lg s)}$-time algorithms: black-box PIT of size-s $\sum \mathrm{m} \bigwedge \sum \prod^2$ formulas, and testing divisibility of a quadratic polynomial into a s-sparse polynomial.*

The above result prompts several directions for future research. To begin with, while many recent black-box PIT results capture subclasses of the models handled by the white-box algorithm of Raz-Shpilka [RS05], there is no better white-box algorithm for $\sum \bigwedge \sum \prod^2$ than the above black-box result, yielding our first question.

Open Question 5.12. *Develop a deterministic (possibly white-box) polynomial-time algorithm for PIT of $\sum \bigwedge \sum \prod^2$ formulas.* ◊

In particular, such an algorithm would either have to generalize Raz-Shpilka [RS05] (which seems hard as $\sum \bigwedge \sum \prod^2$ formulas cannot be computed by roABPs, and this algorithm seems tied

to roABPs) or would have to give a completely new PIT algorithm for even the class of $\sum\bigwedge\sum$ formulas (which would be interesting in and of itself).

While the above question aims to improving existing results, our second question is more ambitious. That is, the above techniques are limited in testing whether $f(\overline{x})$ divides $g(\overline{x})$ when f is of constant-degree and $g(\overline{x})$ is sparse, and thus we can ask to obtain algorithms for more complicated f and g. While increasing the complexity of g is interesting, increasing the complexity of f seems more fundamental as the above reduction from divisibility to PIT will include powers of f while only being linear in g.

Specifically, a natural goal is to obtain divisibility algorithms when f and g are both sparse. While sparse polynomials are not "nice" enough to use the above reduction to PIT (in particular sparse polynomials are not closed under translation), the above reduction does yield some restricted class of formulas for which PIT algorithms of this class implies the desired divisibility algorithm. However, rather than describing this class, we state the following open question for a natural subclass.

Open Question 5.13. *Develop a deterministic subexponential-time algorithm for PIT for sums of powers of sparse polynomials, that is, expressions of the form $f(\overline{x}) = \sum_i f_i(\overline{x})^{d_i}$ where all f_i are sparse (denoted $\sum\bigwedge\sum\prod$).* ◊

In asking for PIT of this class, we should note that the method of *projected* shifted partial derivatives (see for example the work of Kumar and Saraf [KS14] and related works) can give exponential lower bounds for this class [Kum15]. Unfortunately, these lower bounds are for relatively complicated polynomials and do not seem to yield insight into PIT questions. In light of the above robust lower bounds framework, it seems likely that progress could be made on the above question by obtaining lower bounds on *simpler* polynomials.

Open Question 5.14. *What is the complexity of $(x_1+1)\cdots(x_n+1)$ as a sum of powers of sparse polynomials ($\sum\bigwedge\sum\prod$)?* ◊

It is worth noting that this question is subtle as while one might expect an $\exp(\Omega(n))$ lower bound, there are in fact $\exp(O(\sqrt{n}))$ *upper* bounds. In particular, Saptharishi and Shpilka [SS14] observed that this essentially follows from the work of Shpilka and Wigderson [SW01] (see Saptharishi [Sap15, Section 12] for an exposition).

Acknowledgments

We are grateful to Ramprasad Saptharishi, Ben Lee Volk and Yuval Wigderson for helpful comments on an earlier version. The second author would also like to thank Swastik Kopparty, Shubhangi Saraf and Ilya Volkovich for many discussions on the factorization problem.

References

[AGKS15] Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM J. Comput.*, 44(3):669–697, 2015. Preliminary version at arXiv:1406.7535.

[ASS13] Manindra Agrawal, Chandan Saha, and Nitin Saxena. Quasi-polynomial hitting-set for set-depth-Δ formulas. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 321–330, 2013. Full version at arXiv:1209.2333.

[ASSS12] Manindra Agrawal, Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. Jacobian hits circuits: hitting-sets, lower bounds for depth-D occur-k formulas & depth-3 transcendence degree-k circuits. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC 2012)*, pages 599–614, 2012. Full version at arXiv:1111.0582.

[Ber70] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24:713–735, 1970.

[BOT88] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation (extended abstract). In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC 1998)*, pages 301–309, 1988.

[CLO07] David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, New York, third edition, 2007. An introduction to computational algebraic geometry and commutative algebra.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[DOS14] Zeev Dvir, Rafael Oliveira, and Amir Shpilka. Testing equivalence of polynomials under shifts. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP 2014)*, pages 417–428, 2014. Full version at arXiv:1401.3714.

[DSY09] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. *SIAM J. Comput.*, 39(4):1279–1293, 2009. Preliminary version in the *40th Annual ACM Symposium on Theory of Computing (STOC 2008)*.

[For14] Michael A. Forbes. *Polynomial Identity Testing of Read-Once Oblivious Algebraic Branching Programs*. PhD thesis, Massachusetts Institute of Technology, June 2014.

[For15] Michael A. Forbes. Deterministic divisibility testing via shifted partial derivatives. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, 2015.

[FS13a] Michael A. Forbes and Amir Shpilka. Explicit Noether Normalization for simultaneous conjugation via polynomial identity testing. In *Proceedings of the 17th International Workshop on Randomization and Computation (RANDOM 2013)*, pages 527–542, 2013. Full version at arXiv:1303.0084.

[FS13b] Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 243–252, 2013. Full version at arXiv:1209.2408.

[FSS14] Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 867–875, 2014. Full version at arXiv:1309.5668.

[GKKS14] Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Approaching the chasm at depth four. *J. ACM*, 61(6):33:1–33:16, December 2014. Preliminary version in the *28th Annual IEEE Conference on Computational Complexity (CCC 2013)*.

[GKST15] Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC 2015)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 323–346, 2015. Full version at arXiv:1411.7341.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999. Preliminary version in the *39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*.

[Kal87] Erich L. Kaltofen. Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*, pages 443–452, 1987.

[Kal89] Erich L. Kaltofen. Factorization of polynomials given by straight-line programs. In Silvio Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press, Inc., Greenwich, CT, USA, 1989.

[Kal95] Erich L. Kaltofen. Effective Noether irreducibility forms and applications. *J. Comput. Syst. Sci.*, 50(2):274–295, 1995. Preliminary version in the *23rd Annual ACM Symposium on Theory of Computing (STOC 1991)*.

[Kay12] Neeraj Kayal. An exponential lower bound for the sum of powers of bounded degree polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 19(81), 2012.

[KI04] 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)*.

[KS01] Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001)*, pages 216–223, 2001.

[KS14] Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, pages 364–373, 2014. Preliminary version at arXiv:1404.1950.

[KSS15] Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and polynomial factorization. *Computational Complexity*, 24(2):295–331, 2015. Preliminary version in the *29th Annual IEEE Conference on Computational Complexity (CCC 2014)*.

[KT90] Erich L. Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Comput.*, 9(3):301–320, 1990. Preliminary version in the *29th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1988)*.

[Kum15] Mrinal Kumar. Personal Communication, 2015.

[KUW86] Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986. Preliminary version in the *17th Annual ACM Symposium on Theory of Computing (STOC 1985)*.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[MRS14] Meena Mahajan, B.V. Raghavendra Rao, and Karteek Sreenivasaiah. Building above read-once polynomials: Identity testing and hardness of representation. In *Proceedings of Computing and Combinatorics - 20th International Conference (COCOON 2014)*, volume 8591 of *Lecture Notes in Computer Science*, pages 1–12, 2014.

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. Preliminary version in the *19th Annual ACM Symposium on Theory of Computing (STOC 1987)*.

[Nis91] Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 1991)*, pages 410–418, 1991.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. Preliminary version in the *29th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1988)*.

[NW96] Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1996. Preliminary version in the *36th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1995)*.

[Oli15] Rafael Oliveira. Factors of low individual degree polynomials. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC 2015)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 198–216, 2015.

[Pla77] David A. Plaisted. Sparse complex polynomials and polynomial reducibility. *J. Comput. Syst. Sci.*, 14(2):210–221, 1977. Preliminary version in the *18th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1977)*.

[RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Comput. Complex.*, 14(1):1–19, April 2005. Preliminary version in the *19th Annual IEEE Conference on Computational Complexity (CCC 2004)*.

[Sap15] Ramprasad Saptharishi. A survey of known lower bounds for arithmetic circuit complexity. https://github.com/dasarpmar/lowerbounds-survey, 2015.

[Sax08] Nitin Saxena. Diagonal circuit identity testing and lower bounds. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, pages 60–71, 2008. Preliminary version in the Electronic Colloquium on Computational Complexity (ECCC), Technical Report TR07-124.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. Preliminary version in the *International Symposium on Symbolic and Algebraic Computation (EUROSAM 1979)*.

[SS14] Ramprasad Saptharishi and Amir Shpilka. Personal Communication, 2014.

[SSS13] Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. A case of depth-3 identity testing, sparse factorization and duality. *Computational Complexity*, 22(1):39–69, 2013. Preliminary version in the Electronic Colloquium on Computational Complexity (ECCC), Technical Report TR11-021.

[Str73] Volker Strassen. Vermeidung von divisionen. *J. Reine Angew. Math.*, 264:184–202, 1973.

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997. Preliminary version in the *37th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1996)*.

[Sud98] Madhu Sudan. Algebra and computation — lecture notes. http://people.csail.mit.edu/madhu/FT98/course.html, 1998.

[SV09] Amir Shpilka and Ilya Volkovich. Improved polynomial identity testing for read-once formulas. In *Proceedings of the 13th International Workshop on Randomization and Computation (RANDOM 2009)*, volume 5687 of *Lecture Notes in Computer Science*, pages 700–713, 2009. Full version in the Electronic Colloquium on Computational Complexity (ECCC), Technical Report TR10-011.

[SV10] Amir Shpilka and Ilya Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, volume 6198 of *Lecture Notes in Computer Science*, pages 408–419, 2010.

[SW01] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001. Preliminary version in the *14th Annual IEEE Conference on Computational Complexity (CCC 1999)*.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):2070–388, 2010.

[Syl51] James J. Sylvester. On a remarkable discovery in the theory of canonical forms and of hyperdeterminants. *Philosophical Magazine Series 4*, 2(12):391–410, 1851.

[Val80] Leslie G. Valiant. Negation can be exponentially powerful. *Theor. Comput. Sci.*, 12:303–314, 1980. Preliminary version in the *11th Annual ACM Symposium on Theory of Computing (STOC 1979)*.

[Vol15] Ilya Volkovich. Computations beyond exponentiation gates and applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:42, 2015.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, third edition, 2013.

[vzGK85] Joachim von zur Gathen and Erich L. Kaltofen. Factoring sparse multivariate polynomials. *J. Comput. Syst. Sci.*, 31(2):265–287, 1985. Preliminary version in the *24th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1983)*.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction for algebraic block codes, December 1986. US Patent 4,633,470.

[Yao82] Andrew Chi-Chi Yao. Theory and applications of trapdoor functions (extended abstract). In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*, pages 80–91, 1982.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM 1979)*, pages 216–226. Springer-Verlag, 1979.