Deterministic Algorithms for Low Degree Factors of Constant Depth Circuits

Mrinal Kumar * Varun Ramanathan* Ramprasad Saptharishi*

Abstract

For every constant d, we design a subexponential time deterministic algorithm that takes as input a multivariate polynomial f given as a constant depth algebraic circuit over the field of rational numbers, and outputs *all* irreducible factors of f of degree at most d together with their respective multiplicities. Moreover, if f is a sparse polynomial, then the algorithm runs in quasipolynomial time.

Our results are based on a more fine-grained connection between polynomial identity testing (PIT) and polynomial factorization in the context of constant degree factors and rely on a clean connection between divisibility testing of polynomials and PIT due to Forbes [For15] and on subexponential time deterministic PIT algorithms for constant depth algebraic circuits from the recent work of Limaye, Srinivasan and Tavenas [LST21].

1 Introduction

A long line of research (cf. [vzG83, Kal85, Kal92, Kal03]) on the question of designing efficient algorithms for multivariate polynomial factorization concluded with the influential works of Kaltofen [Kal89] and Kaltofen & Trager [KT88] which gave efficient randomized algorithms for this problem in the whitebox and blackbox settings respectively.[1] These results and the technical insights discovered in the course of their proofs have since found numerous direct and indirect applications in various areas of complexity theory. This includes applications to the construction of pseudorandom generators for low degree polynomials [Bog05], algebraic algorithms [KY08], hardness-randomness tradeoffs in algebraic complexity [DSY09, GKSS19], algebraic property testing [PS94, AS03, BSCI+20], error correcting codes [BHKS20], deterministic polynomial identity tests for constant depth circuits [LST21, CKS18] among others.

Given the fundamental nature of the problem and its many applications, the question of designing efficient deterministic algorithms for multivariate polynomial factorization is of great interest and importance. Shpilka & Volkovich [SV10] observed that this question is at least as hard

*Tata Institute of Fundamental Research, Mumbai, India. Email: {mrinal, varun.ramanathan, ramprasad}@tifr.res.in. Research supported by the Department of Atomic Energy, Government of India, under project 12-R&D-TFR-5.01-0500.

[1] Throughout this paper, we use *efficient* to mean an algorithm whose time complexity is polynomially bounded in the size, bit-complexity and the degree of the input algebraic circuit.

as PIT in the sense that a deterministic factoring algorithm (in fact, an algorithm to check irreducibility suffices for this) for polynomials given by algebraic circuits implies a deterministic algorithm for PIT for algebraic circuits, a long standing open problem in computer science. In a later work, Kopparty, Saraf & Shpilka [KSS15] showed a connection in the other direction as well. They showed that an efficient deterministic algorithm for PIT for algebraic circuits implies an efficient deterministic algorithm for polynomial factorization for algebraic circuits. Thus, the questions are essentially equivalent to each other.

An intriguing aspect of the aforementioned equivalence is that while deterministic algorithms for factoring any rich enough class of circuits (for instance, constant depth circuits) lead to deterministic PIT for the same class (see Observation 1 in [SV10] for a precise statement), the connection in the other direction due to Kopparty, Saraf & Shpilka [KSS15] does not appear to be so fine-grained. In particular, even if we only wish to factor an otherwise simple class of polynomials, e.g. sparse polynomials (polynomials with a small number of non-zero monomials), the PIT required as per the proof in [KSS15] seems to be for significantly more powerful models of algebraic computation like algebraic branching programs.

As a consequence, while there has been steady progress on the state of the art of deterministic PIT algorithms in recent years for various interesting sub-classes of algebraic circuits like sparse polynomials [KS01], depth-3 circuits with constant top fan-in [SS09, SS10, KS09], read-once algebraic branching programs [FS13, FSS14, For14, GKST15, GKS16] and constant depth circuits [LST21], this progress hasn't translated to progress on the question of deterministic factoring algorithms for these circuit classes. In particular, deterministic factorization algorithms have remained elusive even for seemingly simple classes of polynomials like sparse polynomials where the corresponding PIT problem is very well understood. There are only a handful of results that make progress towards this and related problems to the best of our knowledge. Shpilka & Volkovich [SV10] showed a close connection between the problems of polynomial identity testing and that of decomposing a polynomial given by a circuit into variable disjoint factors and build on these ideas to give an efficient deterministic algorithm for factoring sparse multilinear polynomials. In subsequent works, Volkovich [Vol15, Vol17] gave an efficient deterministic algorithm to factor sparse polynomials that split into multilinear factors and sparse polynomials with individual degree at most 2. More recently, a work of Bhargava, Saraf and Volkovich [BSV18] gives a quasipolynomial time deterministic algorithm for factoring sparse polynomials with small individual degree based on some beautiful geometric insights.

In general, when the individual degree of a sparse polynomial is not small, no non-trivial deterministic factoring algorithms appear to be known, even when we have the flexibility of describing the output as algebraic circuits. As Forbes & Shpilka note in their recent survey [FS15] on polynomial factorization, we do not even have structural guarantees on the complexity of factors of sparse polynomials even for seemingly coarse measures of complexity like formula complexity. In fact, questions that might be potentially easier than factorization like checking if a given sparse polynomial is a product of constant degree polynomials or checking if a given sparse polynomial is irreducible are not known to have non-trivial deterministic algorithms. Perhaps a little surprisingly, till a recent work of Forbes [For15], we did not even have a non-trivial deterministic algorithm for checking if a given sparse polynomial is divisible by a given constant degree polynomial! Forbes gave a quasipolynomial time deterministic algorithm for this problem by reducing this question to a very structured instance of PIT for depth-4 algebraic circuits and then giving a quasipolynomial time deterministic algorithm for these resulting PIT instances.

This work is motivated by some of these problems, most notably by the question of designing efficient deterministic algorithms for factoring sparse polynomials. While we do not manage to solve this problem in this generality, we make modest progress towards this: we design a deterministic quasipolynomial time algorithm that outputs all the low degree factors of a sparse polynomial. More generally, we show that constant degree factors of a polynomial given by a constant depth circuit can be computed deterministically in subexponential time.

1.1 Our Results

Theorem 1.1 (Low degree factors of constant depth circuits)**.** *Let $\mathbb{Q}$ be the field of rational numbers and $\varepsilon > 0$, $d, k \in \mathbb{N}$ be arbitrary constants.*

Then, there is a deterministic algorithm that takes as input an algebraic circuit C of size s, bit-complexity t, degree D and depth k and outputs all the irreducible factors of C of degree at most d, along with their respective multiplicities in time $(sDt)^{O((sDt)^{\varepsilon})}$.

We note that the bit-complexity of an algebraic circuit/formula is a measure of the bit-complexities of the rational numbers appearing in the circuit. See Definition 2.1 for a formal definition.

When the input polynomial is sparse, i.e. has a small depth-2 circuit, then the time complexity of the algorithm in Theorem 1.1 can be improved to be quasipolynomially bounded in the input size. This gives us the following theorem.

Theorem 1.2 (Low degree factors of sparse polynomials)**.** *Let $d \in \mathbb{N}$ be an arbitrary constant.*

Then, there is a deterministic algorithm that takes as input a polynomial $f \in \mathbb{Q}[\mathbf{x}]$ of sparsity s, bit-complexity t, degree D, and outputs all the irreducible factors of f of degree at most d, along with their respective multiplicities in time $(sDt)^{\mathrm{poly}(\log sDt)}$.

These results immediately yield an algorithm (with comparable time complexity) to check if the polynomial computed by a given low depth circuit is a product of polynomials of degree at most d. More concretely, we have the following corollary that follows by comparison of degrees of the input polynomial and the low-degree factors (with multiplicities) listed by the algorithms in the above theorems.

Corollary 1.3. *Let $\mathbb{Q}$ be the field of rational numbers and $\varepsilon > 0$, $d, k \in \mathbb{N}$ be arbitrary constants.*

Then, there is a deterministic algorithm that takes as input an algebraic circuit C of size s , bit-complexity t, degree D and depth k and decides if C is a product of irreducibles of degree at most d in time $(sDt)^{O((sDt)^{\varepsilon})}$.

Moreover, when f is a sparse polynomial with sparsity s, then the algorithm runs in $(sDt)^{\mathrm{poly}(\log sDt)}$ time.

Note that in the constant depth regime, circuits and formulas are equivalent upto a polynomial blow-up in size. Thus we will use the terms circuits and formulas interchangeably without any loss in our final bounds, and most of our presentation will be for formulas.

Field dependence of our results

We end this section with a remark about the field dependence of our results. The field dependence in our results stems from two reasons. We need an efficient deterministic algorithm for factorization of univariate polynomials over the underlying field $\mathbb{F}$. In addition to this, our proofs also

need non-trivial deterministic algorithms for polynomial identity testing (PIT) for constant depth circuits (or very special depth-4 circuits for Theorem 1.2) over the underlying field.

The field of rational numbers satisfies both these requirements: a classical algorithm of Lenstra, Lenstra and Lovász[LLL82] solves the problem of deterministic univariate factorization efficiently over $\mathbb{Q}$ and a recent work of Limaye, Srinivasan and Tavenas [LST21] gives a subexponential time deterministic algorithm for PIT for constant depth circuits over $\mathbb{Q}$. For Theorem 1.2, the relevant PIT is for special depth-4 circuits and was given in a work of Forbes [For15]. In fact, Forbes' result holds even over finite fields.

We restrict our attention to just the field of rational numbers in the presentation although our results work over any large characteristic field that supports the above requirements.

1.2 Proof Overview

We now give an overview of some of the main ideas in our proofs. In a nutshell, our proofs are based on relatively simple structural observations on top of the existing factoring algorithms. The key is to understand the structure of circuits for which we need a PIT algorithm at every step a little better, and when looking for low degree factors, we observe that these PIT instances are relatively simple and their circuit complexity is comparable to the circuit complexity of the input polynomials themselves. We also crucially use the divisibility testing idea of Forbes [For15] in our algorithm at two stages; this helps us handle factors of large multiplicities and also lets us obtain true factors from the output of Hensel Lifting step of the factorization algorithms. This idea again helps in reducing the complexity of the PIT instance we face in these steps, and in particular, we completely avoid the linear systems solving step in a typical factorization algorithm that naively (e.g. see [KSS15]) seems to require PIT for algebraic branching programs. Once the PIT instances are shown to be relatively simple, we invoke the PIT algorithms of Forbes [For15] and Limaye, Srinivasan & Tavenas [LST21] to solve these deterministically.

Typical steps in a polynomial factorisation algorithm: Most factorisation algorithms (and ours, modulo minor deviations) follow this template:

1. **Making f monic:** Apply a suitable transformation of the form $x_i \mapsto x_i + \alpha_i y$ to ensure that f is monic in y. We may now assume that $f \in \mathbb{Q}[\mathbf{x}, y]$.
2. **Preparing for Hensel lift:** Ensure that $f(\mathbf{x}, y)$ is square-free, and further that $f(\mathbf{0}, y)$ is also square-free.
3. **Univariate factorisation:** Factorise the univariate polynomial $f(\mathbf{0}, y)$ as a product $g_0(y) \cdot h_0(y)$ where $\gcd(g_0, h_0) = 1$. This can be intepreted as a factorisation $f(\mathbf{x}, y) = g_0 \cdot h_0 \bmod \mathcal{I}$ where $\mathcal{I} = \langle \mathbf{x} \rangle$.
4. **Hensel lifting:** Compute an iterated lift to obtain $f = g_\ell \cdot h_\ell \bmod \mathcal{I}^{2^\ell}$ for a suitably large ℓ.
5. **Reconstruction:** From g_ℓ, obtain an honest-to-god factor g of f (unless f is irreducible).

The first two steps typically involve the use of randomness for suitable polynomial identity tests. In the first step, we would like $\boldsymbol{\alpha}$ to be a point that keeps the highest degree homogeneous component of f non-zero, and the second step is handled by translating f by a point $\boldsymbol{\delta}$ that keeps the "discriminant" of f non-zero. The Hensel lift is a deterministic subroutine that eventually yields small circuits for the lifted factors and the reconstruction step typically involves solving a

linear system. It is mostly due to the "discriminant" that we do not have efficient deterministic factorisation algorithm even for constant-depth circuits as the best upper bound for the discriminant we have is an algebraic branching program and we do not have efficient hitting sets for them.

For our case, it is instructive to focus on a specific factor g of f and understand what would be required to make the above template yield this factor. The first observation is that the base case of Hensel Lifting does not require f to be square-free but rather that the factor g we intend to reconstruct satisfies $g|f$ and $g^2 \nmid f$. For now, let us assume this and also that f (and hence g and $h = f/g$ also) is monic in y. We have that $\gcd(g, h) = 1$ but for the Hensel lift, we also need to find a $\boldsymbol{\delta}$ that ensures that $\gcd(g_0, h_0) = 1$ where $g_0 = g(\boldsymbol{\delta}, y)$ and $h_0 = h(\boldsymbol{\delta}, y)$. The set of "good" $\boldsymbol{\delta}$'s is precisely the points that do not make the resultant $\mathrm{Res}_y(g, h)$ zero and thus we want to understand the circuit complexity of this resultant.

The resultant $\mathrm{Res}_y(g, h)$ is the determinant of a matrix of dimension $\deg_y(g) + \deg_y(h)$ and its entries are coefficients of g, h when viewed as univariates in y. However, we are only given that f is computable by a constant-depth formula and we do not have any good bound on the complexity of h. We circumvent this by working with a *pseudo-quotient* (introduced by Forbes [For15] in the context of divisibility testing) $\tilde{h}$ of f and g; we work with $\mathrm{Res}_y(g, \tilde{h})$ and show that it is also computable by constant-depth circuits of not-too-large size. Fortunately, the result of Limaye, Srinivasan and Tavenas [LST21] yields sub-exponential sized hitting sets for constant depth formulas and that enables us to avoid the use of randomness to prepare for the Hensel Lifting step.

We can then factorise the univariate polynomial $f(\boldsymbol{\delta}, y)$ and attempt all possible factors g_0 of degree at most d to begin the lifting process from $g_0 \cdot h_0$ (where $h_0 = f(\boldsymbol{\delta}, y)/g_0$). After an appropriately large lift, we have small circuits (of possibly unbounded depth) computing g_ℓ and h_ℓ such that $\tilde{f} = f(\mathbf{x} + \boldsymbol{\delta}, y) = g_\ell \cdot h_\ell \bmod \mathcal{I}^{2^\ell}$. If g_ℓ is guaranteed to be monic, and the initial choice of g_0 was indeed $g(\boldsymbol{\delta}, y)$, the uniqueness of Hensel lifting would ensure that g_ℓ is indeed equal to g (after truncating higher order terms). We can then use standard interpolation to obtain g_ℓ explicitly written as a sum of monomials. Finally, to ensure that g_ℓ is indeed a legitimate factor of $\tilde{f}$, we perform divisibility testing to check if $g_\ell \mid \tilde{f}$.

Handling factors of large multiplicity: The above overview is all we need to obtain any factor g of degree $O(1)$ that divides f with $g^2 \nmid f$. In order to handle factors with higher "factor-multiplicity", we use a simple observation that $g^{a-1} \mid f$ but $g^a \nmid f$ if and only if g divides $f, \partial_y f, \ldots, \partial_{y^{a-1}} f$ but not $\partial_{y^a} f$. We run our algorithm for each of the partial derivatives to collect the list of candidate factors, and eventually prune them via appropriate divisibility tests.

The specific case of $\Sigma\Pi$-formulas (or sparse polynomials): The above sketch yields a sub-exponential time algorithm for obtaining $O(1)$-degree factors of constant depth formulas. However, with some additional care, we obtain a quasipolynomial time algorithm in the case when f is a sparse polynomial. The key observation for this is that we do not really need f to be made monic for the above approach, but we only need g to be monic to exploit the uniqueness of Hensel lifts. Since g is a polynomial of degree at most $d = O(1)$, we can find a *low Hamming weight* vector $\boldsymbol{\alpha}$ such that $g(\mathbf{x} + y\boldsymbol{\alpha})$ is monic in y. This allows us to control the sparsity increase of f in the process and we show that the relevant resultant is a polynomial of the form

$$\sum_i \text{monomial}_i \cdot (O(1)\text{-degree})^{e_i}.$$

Forbes [For15] shows that there are quasipolynomial size hitting sets for such expressions and we use this instead of the more general hitting set of Limaye, Srinivasan and Tavenas [LST21].

Organization of the paper

The rest of the paper is organized as follows.

In the next section, we start with a discussion of some of the preliminaries and known results from algebraic complexity and previous works on polynomial factorization that we use for the design and analysis of our algorithms. In Section 3, we describe and analyze the algorithm for computing low degree factors of multiplicity one of a given constant depth formula. In Section 4, we build upon this algorithm to compute arbitrary constant degree factors and complete the proofs of Theorem 1.1 and Theorem 1.2. Finally, we conclude with some open problems in Section 5.

2 Notation and preliminaries

This section consists of all the necessary building blocks to describe and analyse (in Section 3) the main algorithm.

Fair warning: A large part of this (slightly lengthy) section is standard techniques in algebraic complexity that are relevant to this specific context, and is intended to keep the main analysis as self-contained as possible. A reader with some familiarity with standard algorithmic and structural results in algebraic complexity might be in a position to directly proceed to Section 3 and revisit this section for relevant results as required.

Notation

1. Throughout this paper, we work over the field $\mathbb{Q}$ of rational numbers. For some of the statements that are used more generally, we use $\mathbb{F}$ to denote an underlying field.
2. We use boldface lower case letters like $\mathbf{x}, \mathbf{y}, \mathbf{a}$ to denote tuples, e.g. $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. The arity of the tuple is either stated or will be clear from the context.
3. For a polynomial f and a non-negative integer k, $\mathrm{Hom}_k[f]$ denotes the homogeneous component of f of degree *equal* to k. $\mathrm{Hom}_{\leq k}[f]$ denotes the sum of homogeneous components of f of degree at most k, i.e.,
$$\mathrm{Hom}_{\leq k}[f] := \sum_{i=0}^{k} \mathrm{Hom}_i[f].$$
4. The *sparsity* of a polynomial f is the number of monomials with a non-zero coefficient in f.
5. For a parameter $k \in \mathbb{Z}_{\geq 0}$, we will use $(\Sigma\Pi)^{(k)}$ to refer to product-depth k circuits[2] with the root gate being $+$ and the deepest layer of gates being $\times$. Since any constant depth algebraic circuit of depth k and size s can be converted to a formula of depth k and size s^{k+1}

[2]We emphasize that this notation does *not* refer to the k^{th} power of a polynomial computed by a $\Sigma\Pi$ circuit.

i.e. poly(s), we will use the terms circuits and formulas interchangeably, without any loss in the final bounds we prove.

6. Let f and g be multivariate polynomials such that $g \mid f$. Then, the *multiplicity* or *factor multiplicity* of g in f is defined to be the greatest integer a such that g^a divides f.

2.1 Circuit/formula bit-complexity

Definition 2.1 (Bit-complexity of a circuit/formula)**.** *The* bit-complexity *of a circuit/formula C, denoted by* bit(C)*, is defined as the sum of* size(C) *and the bit-complexities of all the scalars*[3] *present on edges or leaves. By default, any edge that does not have a scalar on it will be assigned the scalar 1.* ♢

Lemma 2.2 (Bit-complexity of evaluations of formulas)**.** *Let C be a formula of bit-complexity s computing a polynomial $f(\mathbf{x})$. If $\mathbf{a} \in \mathbb{Q}^n$ with each entry of $\mathbf{a}$ having bit-complexity b, then the bit-complexity of $f(\mathbf{a})$ is at most $s \cdot b$.*

(Proof deferred to Appendix A)

2.2 Relevant subclasses of algebraic circuits

We briefly define subclasses of algebraic circuits that we would use often in this paper.

Definition 2.3 (Power of low-degree polynomials)**.** *For a parameter $d \in \mathbb{Z}_{\geq 0}$, let* Deg_d *refer to the class of polynomials of degree at most d. We use $(\mathrm{Deg}_d)^*$ to denote the class of polynomials that are powers of polynomials of degree at most d.* ♢

Definition 2.4 ($\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$-formulas)**.** *We will use $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ to denote the subclass of algebraic formulas that compute expressions of the form*

$$\sum_i f_i \cdot g_i^{e_i}$$

where each f_i is a $(\Sigma\Pi)^{(k)}$ formula and each g_i is a polynomial of degree at most d and e_i's are arbitrary positive integers. The size and bit-complexity of the above expression is defined as its size and bit-complexity when viewed as a general algebraic formula. ♢

Observation 2.5. *Let $\mathcal{C}$ be the class of $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formulas for fixed parameters k and d. Suppose $P_1, \ldots, P_t$ are polynomials computed by $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formulas of size s and bit-complexity b each. Then,*

- *$\sum_i P_i$ is computable by an $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formula of size at most $t \cdot s$ and bit-complexity at most $O(t \cdot b)$.*
- *$\prod_i P_i$ is computable by an $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formula of size at most $s^{O(t)}$ and bit-complexity at most $b^{O(t)}$.*

(Proof deferred to Appendix A.)

[3] For a rational number $r = p/q$, its bit-complexity bit(r) is defined as $\log(\max(|p|, |q|))$

2.3 Standard preliminaries using interpolation

Lemma 2.6 (Univariate interpolation (Lemma 5.3 [Sap15])). *Let $f(x) = f_0 + f_1 x + \cdots + f_d x^d$ be a univariate polynomial of degree at most d. Then, for any $0 \le r \le d$ and there are*[4] *field constants $\alpha_0, \ldots, \alpha_d$ and $\beta_{r0}, \ldots, \beta_{rd}$ such that*

$$f_r = \beta_{r0} f(\alpha_0) + \cdots + \beta_{rd} f(\alpha_d).$$

Furthermore, the bit-complexity of all field constants is bounded by $\mathrm{poly}(d)$. □

Lemma 2.7 (Computing homogeneous components (Lemma 5.4 [Sap15])). *Let $f \in \mathbb{Q}[\mathbf{x}]$ be an n-variate degree d polynomial. Then, for an $0 \le i \le d$, there are field constants $\alpha_0, \ldots, \alpha_d$ and β_{i0}, β_{id} of bit-complexity* $\mathrm{poly}(d)$ *such that*

$$\mathrm{Hom}_i(f) = \beta_{i0} f(\alpha_0 \cdot \mathbf{x}) + \cdots + \beta_{id} f(\alpha_d \cdot \mathbf{x}).$$

In particular for $\mathcal{C} = (\Sigma\Pi)^{(k)}$ or $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$, if f is computable by $\mathcal{C}$-formulas of size / bit-complexity at most s then $\mathrm{Hom}_i(f)$ is computable by $\mathcal{C}$-formulas of size / bit-complexity at most $\mathrm{poly}(s, d)$.

Lemma 2.8 (Computing partial derivatives in one variable). *Let $f \in \mathbb{Q}[\mathbf{x}]$ be an n-variate degree d polynomial. Then, for an $0 \le r \le d$, there are field elements α_i's and β_{ij}'s in $\mathbb{Q}$ of bit-complexity* $\mathrm{poly}(d)$ *such that*

$$\frac{\partial^r f}{\partial x_1^r} = \sum_{i=0}^{d} x_1^i \cdot (\beta_{i0} f(\alpha_0, x_2, \ldots, x_n) + \cdots + \beta_{id} f(\alpha_d, x_2, \ldots, x_n))$$

In particular for $\mathcal{C} = (\Sigma\Pi)^{(k)}$ or $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$, if f is computable by $\mathcal{C}$-formulas of size / bit-complexity at most s then $\frac{\partial^r f}{\partial x_1^r}$ is computable by $\mathcal{C}$-formulas / bit-complexity of size at most $O(s \cdot d^3)$.

Proof. We may consider the polynomial f as a univariate in x_1, and extract each coefficient of x_1^i using Lemma 2.6 and recombine them to get the appropriate partial derivative. That justifies the claimed expression.

As for the size, note that if $\mathcal{C}$ is $(\Sigma\Pi)^{(k)}$ or $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$, multiplying a size s formula by x_1^i, by using distributivity of the top addition gate, results in a $\mathcal{C}$-formula of size at most $s \cdot d$. Thus, the overall size of the above expression for the partial derivative is at most $O(s \cdot d^3)$. □

We will be making use of the following identity, which can be proved via appropriate interpolation or by the inclusion-exclusion principle (along the lines of Lemma 2.2 [Shp02]).

Lemma 2.9 (Fischer's identity [Fis94, Ell69, Shp02]). *If $\mathbb{F}$ is a field of characteristic zero or larger than D, then for any positive integers $e_1, \ldots, e_n$ with $\sum e_i = D$ and for $r \le \prod_{i=1}^{n} (e_i + 1)$, there are homogeneous linear forms $L_1, \ldots, L_r$ and field constants $\alpha_1, \ldots, \alpha_r$ of bit-complexity* $\mathrm{poly}(d, n)$ *such that*

$$x_1^{e_1} \cdots x_n^{e_n} = \sum_{i=1}^{r} \alpha_i L_i^D.$$

[4] In fact, for any choice of distinct $\alpha_0, \ldots, \alpha_d$, there are appropriate $\beta_{r0}, \ldots, \beta_{rd}$ satisfying the equation. If the α_i's are chosen to have small bit-complexity, we can obtain a $\mathrm{poly}(d)$ bound on the bit-complexity of the associated β_{ri}'s.

2.4 Polynomial identity testing

Lemma 2.10 (Polynomial Identity Lemma [Ore22, DL78, Sch80, Zip79])**.** *Let $f \in \mathbb{Q}[\mathbf{x}]$ be a non-zero n variate polynomial of degree at most d. Then, for every set $S \subseteq \mathbb{Q}$, the number of zeroes of f in the set $S^n = S \times S \times \cdots \times S$ is at most $d|S|^{n-1}$.*

Definition 2.11 (Low Hamming weight set)**.** *Let $n \geq d \geq 0$ be integer parameters. Fix a set $T_d \subseteq \mathbb{Q}$ of size $(d+1)$. The set $\mathcal{H}(d,n)$ is defined as*

$$\mathcal{H}(d,n) = \left\{ (a_1, \ldots, a_n) \ : \ S \in \binom{[n]}{\leq d} \ , \ a_i \in T_d \text{ for all } i \in S \text{ and } a_j = 0 \text{ for all } j \notin S \right\}.$$

The size of the above set is at most $\binom{n}{\leq d} \cdot (d+1)^d = n^{O(d)}$. Furthermore, choosing T_d to consist of elements of $\mathbb{Q}$ of bit-complexity $\mathrm{poly}(d)$, the bit-complexity of the set $\mathcal{H}(d,n)$ is bounded by $n^{O(d)}$ as well. $\lozenge$

The following lemma is an easy consequence of Lemma 2.10 and will be crucial for parts of our proof. We also include a short proof sketch.

Lemma 2.12 (Hitting set for low degree polynomials)**.** *Let $f \in \mathbb{Q}[\mathbf{x}]$ be a non-zero n variate polynomial of degree at most d. Then, there exists a vector $\mathbf{a} \in \mathcal{H}(d,n) \subseteq \mathbb{Q}^n$ such that $f(\mathbf{a}) \neq 0$.*

(Proof deferred to Appendix A.)

Theorem 2.13 (PIT for constant depth formulas (modification of Corollary 6 [LST21]))**.** *Let $\varepsilon > 0$ be a real number and $\mathbb{F}$ be a field of characteristic 0. Let C be an algebraic formula of size and bit-complexity $s \leq \mathrm{poly}(n)$, depth $k = o(\log\log\log n)$ computing a polynomial on n variables, then there is a deterministic algorithm that can check whether the polynomial computed by C is identically zero or not in time $(s^{O(k)} \cdot n)^{O_\varepsilon((sD)^\varepsilon)}$.*

The original statement of Corollary 6 in [LST21] deals specifically with circuits of size $s = \mathrm{poly}(n)$. The above statement can be readily inferred from their proof.

Theorem 2.14 (PIT for $\Sigma\left((\Sigma\Pi)^{(1)} \cdot (\mathrm{Deg}_d)^*\right)$ (Corollary 6.7, [For15]))**.** *Let $t \geq 1$. Then, the class $\mathcal{C} = \Sigma\left((\Sigma\Pi)^{(1)} \cdot (\mathrm{Deg}_d)^*\right)$ that computes polynomials of the form $\sum_{i=1}^{s} f_i \cdot g_i^{d_i}$ with each f_i being s-sparse and each $\deg(g_i) \leq d$ has a $\mathrm{poly}(n, s, d\log s)$-explicit hitting set of size $\mathrm{poly}(s)^{O(d\log s)}$.*

We will also crucially use the following lemma that gives an algorithm to obtain the coefficient vector of a polynomial from an algebraic formula computing it. In our setting, we invoke this algorithm only for low degree polynomials, and in that case, we can tolerate the runtime of this algorithm within our budget.

Lemma 2.15 (Interpolating a low degree multivariate polynomial)**.** *There is a deterministic algorithm that, when given a parameter d and an n variate algebraic formula $C \in \mathbb{Q}[\mathbf{x}]$ of size at most s, bit-complexity at most b and degree at most d, outputs the coefficient vector of the polynomial computed by C.*

The algorithm runs in time $\mathrm{poly}(s, b, n^d)$.

(Proof deferred to Appendix A.)

2.5 Deterministic divisibility testing and PIT

Definition 2.16 (Pseudo-quotients)**.** *Let $f, g \in \mathbb{Q}[\mathbf{x}]$ be non-zero polynomials with $g(\mathbf{0}) = \beta \neq 0$. The* pseudo-quotient of f and g *is defined as*

$$\mathrm{Hom}_{\leq d_f - d_g}\left(\left(\frac{f(\mathbf{x})}{\beta}\right) \cdot (1 + \tilde{g} + \tilde{g}^2 + \cdots + \tilde{g}^{d_f - d_g})\right)$$

where $d_f = \deg(f)$, $d_g = \deg(g)$ and $\tilde{g} = 1 - \frac{g}{\beta}$.

More generally, if $\boldsymbol{\alpha} \in \mathbb{Q}^n$ is such that $g(\boldsymbol{\alpha}) \neq 0$, the pseudo-quotient of f and g translated by $\boldsymbol{\alpha}$ *is defined as the pseudo-quotient of $f(\mathbf{x} + \boldsymbol{\alpha})$ and $g(\mathbf{x} + \boldsymbol{\alpha})$.* ◊

The following lemma immediately follows from the above definition and Lemma 2.7.

Lemma 2.17 (Complexity of pseudo-quotients)**.** *Suppose $k \geq 1$ and $f(\mathbf{x}) \in (\Sigma\Pi)^{(k)}$ and $g(\mathbf{x}) \in \mathrm{Deg}_d$ of sizes at most s_1, s_2 respectively, and suppose $g(\mathbf{0}) \neq 0$. Then, the pseudo-quotient of f, g is computable by the $\mathcal{C}$-formulas of size at most $\mathrm{poly}(s_1, s_2)$, where $\mathcal{C} = \Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$.*

Theorem 2.18 (Divisibility testing to PIT [For15])**.** *Let $f(\mathbf{x})$ and $g(\mathbf{x})$ be non-zero n-variate polynomials over a field $\mathbb{Q}$ such that $g(\mathbf{0}) = \beta \neq 0$. Then, g divides f if and only if the polynomial $R(\mathbf{x})$ defined as*

$$R(\mathbf{x}) := f(\mathbf{x}) - g(\mathbf{x})Q(\mathbf{x})$$

is identically zero, where $Q(\mathbf{x})$ is the pseudo-quotient of f and g.

An immediate consequence of this theorem is the following corollary that takes into account the depth of an algebraic formula computing the polynomial $R(\mathbf{x})$ given above, assuming that f and g themselves can be computed by a low depth formula.

Corollary 2.19 (Divisibility testing to PIT for constant depth formulas [For15])**.** *Suppose $f(\mathbf{x})$ is a non-zero n-variate polynomial computed by a $(\Sigma\Pi)^{(k)}$ formula of size s, and suppose $g(\mathbf{x})$ is a polynomial of degree at most d with $g(\mathbf{0}) = \beta \neq 0$. Then, we can test if g divides f in time $T(k, d, s')$ where $s' = \mathrm{poly}(s, d)$ and $T(k, d, s)$ is the time required to test polynomial identities of the size s expressions of the form*

$$\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right).$$

(Proof deferred to Appendix A.)

Theorem 2.20 ([For15])**.** *Let $\mathbb{F}$ be any sufficiently large field. Then, there is a deterministic algorithm that takes an input two polynomials f and g and parameters d, D, n, s, where f is an n-variate polynomial of degree at most D and sparsity s; g is an n variate polynomial of degree d, and outputs whether g divides f or not in time $\exp(O(d \log^2 snDd))$.*

2.6 Resultants

Definition 2.21 (The Resultant)**.** *Let $\mathcal{R}$ be a commutative ring. Given polynomials g and h in $\mathcal{R}[y]$, where:*

$$g(y) = g_0 + \cdots + y^d \cdot g_d$$
$$h(y) = h_0 + y \cdot h_1 + \cdots + y^D \cdot h_D$$

with g_d and $h_D \neq 0$ the Resultant *of g and h, denoted by $\mathrm{Res}_y(g,h)$, is the determinant of the $(D+d) \times (D+d)$ Sylvester matrix Γ of g and h, given by:*

$$\Gamma = \begin{bmatrix} h_0 & h_1 & \dots & & h_D & & \\ & \ddots & \ddots & & \ddots & \ddots & \\ & & h_0 & h_1 & & \dots & h_D \\ g_0 & \dots & & g_d & & & \\ & g_0 & \cdots & & g_d & & \\ & & \ddots & \ddots & & \ddots & \\ & & & g_0 & \cdots & & g_d \end{bmatrix}$$

$\Diamond$

Lemma 2.22 (Resultant and gcd (Corollary 6.20 [vzGG13]))**.** *Let $\mathcal{R}$ be a unique factorization domain and $g, h \in \mathcal{R}[y]$ be non-zero polynomials. Then:*

$$\deg_y(\gcd(g,h)) > 0 \iff \mathrm{Res}_y(g,h) = 0$$

where $\gcd(g,h) \in \mathcal{R}[y]$ and $\mathrm{Res}_y(g,h) \in \mathcal{R}$.

In this paper, $\mathcal{R}$ will be $\mathbb{Q}[\mathbf{x}]$ (which is a unique factorization domain), and $\mathrm{Res}_y(g,h)$ will denote the resultant of $g, h \in \mathbb{Q}[\mathbf{x}][y]$ when considered as polynomials in $\mathcal{R}[y]$. We might also occasionally refer to it as the *y-resultant* of g and h. For more details about the resultant as well as a proof of the above lemma, we refer the reader to von zur Gathen and Gerhard's book on computer algebra (Chapter 6, [vzGG13]). We mention a simple observation from the above definition that would be useful for this paper.

Observation 2.23 (Resultant under substitutions)**.** *Suppose $g(\mathbf{x}, y) = g_0(\mathbf{x}) + g_1(\mathbf{x})y + \cdots g_d(\mathbf{x})y^d$ and $h(\mathbf{x}, y) = h_0(\mathbf{x}) + h_1(\mathbf{x})y + \cdots + h_D(\mathbf{x})y^D$ with $g_d, h_D \neq 0$. Then, for any $\mathbf{a} \in \mathbb{Q}^{|\mathbf{x}|}$ that ensures $g_d(\mathbf{a}), h_D(\mathbf{a}) \neq 0$, we have*

$$(\mathrm{Res}_y(g,h))(\mathbf{a}) = \mathrm{Res}_y(g(\mathbf{a}, y), h(\mathbf{a}, y)).$$

$\square$

2.7 Hensel Lifting

Now we will state the definition of a *lift* and the main lemma for Hensel lifting. For more details, one can look up some of the cited papers or the standard references in computational algebra [KSS15, ST20, vzGG13, Sud98].

Definition 2.24 (Hensel lifts)**.** *Let $\mathcal{I} \subseteq \mathbb{Q}[\mathbf{x}, y]$ be an ideal. Let $f, g, h, u, v \in \mathbb{Q}[\mathbf{x}, y]$ such that $f \equiv gh \bmod \mathcal{I}$ and $ug + vh \equiv 1 \bmod \mathcal{I}$. Then, we call $g', h' \in \mathbb{Q}[\mathbf{x}, y]$ a* lift *of g and h if:*

1. *$f \equiv g'h' \bmod \mathcal{I}^2$,*
2. *$g' \equiv g \bmod \mathcal{I}$ and $h' \equiv h \bmod \mathcal{I}$, and*
3. *$\exists u', v' \in \mathbb{Q}[\mathbf{x}, y]$ s.t $u'g' + v'h' \equiv 1 \bmod \mathcal{I}^2$.*

$\Diamond$

For the rest of the section, we define $\mathcal{I}$ to be the ideal $\langle x_1, \dots, x_n \rangle$ and $\mathcal{I}_k := \mathcal{I}^{2^k}$.

Lemma 2.25 (Iterated monic Hensel lifting (Lemma 3.4 [KSS15])). *Suppose we're given $f \in \mathbb{Q}[\mathbf{x}, y]$ such that $f = gh$, g is monic in y and $\gcd(g, h) = 1$. We are also given $g_0, h_0, u_0, v_0 \in \mathbb{Q}[\mathbf{x}, y]$ such that $g_0 \equiv g \bmod \mathcal{I}$, $h_0 \equiv h \bmod \mathcal{I}$ and $u_0 g_0 + v_0 h_0 \equiv 1 \bmod \mathcal{I}$. Then, for all $k \in \mathbb{N}, k \geq 1$, there exist $g_k, h_k, u_k, v_k \in \mathbb{Q}[\mathbf{x}, y]$, with each g_k being monic, such that the following conditions hold:*

1. *The pair g_k, h_k is a lift of g_{k-1}, h_{k-1}, with $u_k g_k + v_k h_k \equiv 1 \bmod \mathcal{I}_k$; in particular, $f \equiv g_k h_k \bmod \mathcal{I}_k$*
2. *$g_k \equiv g \bmod \mathcal{I}_k$ and $h_k \equiv h \bmod \mathcal{I}_k$*

Moreover, for each k, g_k and h_k are unique polynomials modulo $\mathcal{I}_k$ satisfying the above conditions when the g_ks are monic. For each k, we will call g_k, h_k the k-th iterated lift of *g_0, h_0.*

If $\deg_{\mathbf{x}}(g) = d$, we can choose an integer k^* such that $d < 2^{k^*} \leq 2d$ and use the above Lemma to get $g_{k^*} \equiv g \bmod \mathcal{I}_{k^*}$, which means we can truncate g_{k^*} to degree d and retrieve g. The next lemma tells us that this can be done with reasonable bounds on the parameters of the underlying circuits.

Lemma 2.26 (Small circuit for Hensel lifting (Lemma 3.6 [KSS15])). *Let f be a degree D polynomial in $\mathbb{Q}[\mathbf{x}, y]$, computable by a $(\Sigma\Pi)^{(k)}$ formula of size and bit-complexity s, with a factorization $f = gh$ such that $\gcd(g, h) = 1$ and g is monic. Let $g_0 = g \bmod \mathcal{I}$ and $h_0 = h \bmod \mathcal{I}$ be univariates in $\mathbb{Q}[y]$ with $\gcd(g_0, h_0) = 1$.*

Then, there are formulas C_g, C_h of size and bit complexity $(sDk)^{O(k \log D)}$ that compute the k^{th} iterated lift g_k,h_k of g_0,h_0, where g_k is monic. More generally, if the total degree of g_k is at most d, then the size and bit complexity of the formula for g_k is at most $(sDk)^{O(\log d)}$.

Moreover, there is a deterministic algorithm, that when given the formulas for f and g_0, h_0 and integer k as input, outputs the formulas for g_k and h_k in time $(sDk)^{O(k \log D)}$ (resp. $(sDk)^{O(\log d)}$ if g_k has total degree d).

(Proof sketch deferred to Appendix A.)

2.8 Results on polynomial factorization

We rely on the following two fundamental results on polynomial factorization for our results. The first theorem is a classical algorithm of Lenstra, Lenstra and Lovász for factoring univariate polynomials over the field of rational numbers.

Theorem 2.27 (Factorizing polynomials with rational coefficients [LLL82, vzGG13]). *Let $f \in \mathbb{Q}[x]$ be a monic polynomial of degree d. Then there is a deterministic algorithm computing all the irreducible factors of f that runs in time $\mathrm{poly}(d, t)$, where t is the maximum bit-complexity of the coefficients of f.*

The second result we need is an easy consequence of the results of Kopparty, Saraf and Shpilka [KSS15]. They showed that an efficient deterministic algorithm for PIT for algebraic circuits implies an efficient deterministic algorithm for polynomial factorization. The formal statement below essentially invokes this for constant degree polynomials. In this case, the PIT instances also happen to be of constant degree and hence can be easily solved in time that is polynomial in the length of the coefficient vector of these polynomials.

Theorem 2.28 ([KSS15]). *There is a deterministic algorithm that when given as input the coefficient vector of an n variate polynomial $f(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$ of total degree d, runs in time $n^{O(d^2)}$ and decides if f is irreducible or not.*

3 Computing candidate low-degree factors of multiplicity one

We first present the algorithm for computing candidate low-degree factors of multiplicity one in Algorithm 1 below. In the next section, we use this as a subroutine in Algorithm 2 to compute factors of all multiplicity and also eliminate those candidates that were not actual factors.

Algorithm 1: Computing candidate degree d factors of factor-multiplicity one

Input : A $(\Sigma\Pi)^{(k)}$-formula of size s, bit-complexity t, degree D computing a polynomial $f(\mathbf{x})$.

Output: A list of polynomials of degree at most d, that include all factors of f with degree at most d and multiplicity 1.

Set the output list $L = \emptyset$.

Compute hitting-set $H_1 = \mathcal{H}(d, n)$ (as defined in Definition 2.11).

Compute hitting-set H_2 for the class of $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$-formulas that have size $s' \leq (sD)^{O(d)}$. (Lemma 3.2, Theorem 2.14)

for $\boldsymbol{\alpha}, \boldsymbol{\beta} \in H_1$ *and* $\boldsymbol{\delta} \in H_2$ **do**

  Define $F(\mathbf{x}, y) = f(\mathbf{x} + \boldsymbol{\alpha} \cdot y + \boldsymbol{\beta} + \boldsymbol{\delta}) = f(x_1 + \alpha_1 y + \beta_1 + \delta_1, \ldots, x_n + \alpha_n y + \beta_1 + \delta_n)$

  Using interpolation on the formula for $F(\mathbf{x}, y)$ (via Lemma 2.6), compute $F(\mathbf{0}, y)$ as a sum of monomials.

  Factorise the polynomial $F(\mathbf{0}, y)$ into irreducible factors as

$$F(\mathbf{0}, y) = \sigma \cdot F_1^{e_1} \cdots F_r^{e_r}.$$

  where $0 \neq \sigma \in \mathbb{Q}$ and each F_r is monic in y.

  for $T \subseteq [r]$ *of size at most* d **do**

    Define $g_0 = \prod_{i \in T} F_i^{e_i}$ and $h_0 = \sigma \cdot \prod_{i \notin T} F_i^{e_i}$, interpretted as polynomials in $\mathbb{Q}[\mathbf{x}, y]$ for Lemma 2.25

    if $\deg(g_0) > d$ **then**

      Continue to the next choice of T in the current loop.

    Compute polynomials u_0, v_0 such that $u_0 g_0 + v_0 h_0 = 1$.

    Use Hensel-Lifting (Lemma 2.26) to lift the factorisation $F(\mathbf{x}, y) = g_0(\mathbf{x}, y) \cdot h_0(\mathbf{x}, y) \bmod I$, where $I = \langle \mathbf{x} \rangle$, to obtain algebraic circuits for g_ℓ, h_ℓ satisfying

$$F(\mathbf{x}, y) = g_\ell(\mathbf{x}, y) \cdot h_\ell(\mathbf{x}, y) \bmod I^{2^\ell}$$

    with g_ℓ being monic and $d < 2^\ell < 2d$.

    Using interpolation on the circuit for g_ℓ (via Lemma 2.15), compute g_ℓ as a sum of monomials.

    Add $\tilde{g} = g_\ell(\mathbf{x} - \boldsymbol{\delta} - \boldsymbol{\beta}, 0)$ to L.

return L

Before we discuss the proof of correctness and running time of Algorithm 1, we state two simple observations that we use in the analysis. We defer the proofs of these observations to the end of the section.

Observation 3.1 (Size growth under a translation of low Hamming weight)**.** *Let $k > 0$ be a parameter. Let $f(\mathbf{x})$ be an n-variate polynomial of degree at most D with $(\Sigma\Pi)^k$-size at most s. If $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{H}(d, n)$, the polynomial $\tilde{f}(\mathbf{x}, y) = f(\mathbf{x} + y\boldsymbol{\alpha} + \boldsymbol{\beta})$ has $(\Sigma\Pi)^k$-size at most $s \cdot D^{O(d)}$.*

Lemma 3.2. *Let $f(\mathbf{x})$ be an n-variate polynomial computed by a $(\Sigma\Pi)^{(k)}$ formula of size s, and let $g(\mathbf{x})$ be an n-variate degree d polynomial with $g(\mathbf{0}) \neq 0$. Let $Q(\mathbf{x})$ be the pseudo-quotient of f and g. Then, for any variable $y \in \mathbf{x}$, the polynomial $\mathrm{Res}_y(Q, g)$ is computable by a $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formula of size at most $s^{O(d)}$.*

3.1 Proof of correctness of the Algorithm 1

Lemma 3.3 (Correctness of Algorithm 1)**.** *For every input polynomial f computed by $(\Sigma\Pi)^{(k)}$ formulas of size s, bit-complexity t, degree D and any factor g of degree at most d with $g \mid f$ and $g^2 \nmid f$, the polynomial g is included in the output list of Algorithm 1 on input f.*

Proof. Algorithm 1 outputs a list of candidate factors; we would like to prove that every factor of f with degree $\leq d$ and factor-multiplicity one will be contained in this list. Fix any specific factor g of f, with $\deg(g) = d' \leq d$ and factor-multiplicity one, which ensures that $\gcd(g, f/g) = 1$.

1. **Make g monic and $g(\mathbf{0}) \neq 0$**

   The coefficient of $y^{d'}$ in $g'(\mathbf{x}, y) := g(\mathbf{x} + y\boldsymbol{\alpha} + \boldsymbol{\beta})$ is the evaluation of $\mathrm{Hom}_{d'}(g)$ at $\boldsymbol{\alpha}$ and the constant term of $g'(\mathbf{x}, y)$ is $g'(\mathbf{0}, 0) = g(\boldsymbol{\beta})$. Thus by Lemma 2.12, there is some $\boldsymbol{\alpha}, \boldsymbol{\beta} \in H_1$ such that $\mathrm{Hom}_{d'}(g)(\boldsymbol{\alpha}) \neq 0$ and $g(\boldsymbol{\beta}) \neq 0$. Fix this choice of $\boldsymbol{\alpha}, \boldsymbol{\beta}$. We then have that $g'(\mathbf{x}, y)$ is monic in y, has $\deg_y(g') = \deg(g) = d'$, and has non-zero constant term.

2. **Bound the size of $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formula for the resultant**

   With the above properties, the pseudo-quotient h' of $f'(\mathbf{x}, y) := f(\mathbf{x} + y\boldsymbol{\alpha} + \boldsymbol{\beta})$ and $g'(\mathbf{x}, y)$ is well-defined and is a polynomial in $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ (by Lemma 2.17) of size $\mathrm{poly}(s, D, d) \leq \mathrm{poly}(sD)$. By Lemma 3.2, $\mathrm{Res}_y(g', h') \in \mathbb{Q}[\mathbf{x}]$ is a non-zero polynomial computable by $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formulas of size $(sD)^{O(d)}$.

3. **Maintain $\gcd(g, h) = 1$ condition in the univariate setting by hitting the resultant**

   Let $\deg_y(h') = r$ and $h'(\mathbf{x}, y) = h'_0(\mathbf{x}) + \cdots + h'_r(\mathbf{x})y^r$. Since h' is computable by size $(sD)^{O(d)}$ formula from $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$, so is the leading term $h'_r(\mathbf{x})$ by Lemma 2.7. Therefore by Observation 2.5, the polynomial $\Gamma(\mathbf{x}) = \mathrm{Res}_y(g', h') \cdot h'_r(\mathbf{x})$ is also computable by $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formulas of size $s' = (sD)^{O(d)}$. Since H_2 is a hitting set for $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formulas of size s', fix a $\boldsymbol{\delta} \in H_2$ such that $\Gamma(\boldsymbol{\delta}) \neq 0$ and in particular, the conditions required in Observation 2.23 are true (note that the leading coefficient of g' is just 1 by monicness). By Lemma 2.22 and Observation 2.23, we have that $g'(\boldsymbol{\delta}, y)$ and $h'(\boldsymbol{\delta}, y)$ are coprime polynomials. Thus, if $g''(\mathbf{x}, y) = g'(\mathbf{x} + \boldsymbol{\delta}, y)$ and $h''(\mathbf{x}, y) = h'(\mathbf{x} + \boldsymbol{\delta}, y)$ (h' being the pseudo-quotient),

Theorem 2.18 implies that

$$\begin{aligned} & f(\mathbf{x}+\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta}) = g''(\mathbf{x},y)\cdot h''(\mathbf{x},y) \\ \implies\ & f(\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta}) = g''(\mathbf{0},y)\cdot h''(\mathbf{0},y) \\ & \qquad\qquad \text{with } \gcd(g''(\mathbf{0},y),h''(\mathbf{0},y)) = 1. \end{aligned}$$

4. **Univariate factorization and Hensel Lifting**

   Line 7 thus factorises the univariate polynomial $f(\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta})$ and one of the sets T in Line 8 must correspond to $g_0(y)$ chosen in Line 9 to satisfy $g_0(y) = g''(\mathbf{0},y)$ and $h_0(y) = h''(\mathbf{0},y)$. Thus, we have a factorisation of the form

$$\begin{aligned} & f(\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta}) = g''(\mathbf{0},y)\cdot h''(\mathbf{0},y) = g_0\cdot h_0 \\ \implies\ & f(\mathbf{x}+\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta}) = g_0\cdot h_0 \bmod \mathcal{I}, \quad \text{where } \mathcal{I} = \langle \mathbf{x}\rangle. \end{aligned}$$

   We are therefore set-up to apply Hensel Lifting (Lemma 2.25) to obtain g_ℓ, h_ℓ such that g_ℓ is monic in y and

$$f(\mathbf{x}+\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta}) = g_\ell(\mathbf{x},y)\cdot h_\ell(\mathbf{x},y) \bmod \mathcal{I}^{2^\ell}.$$

   From the uniqueness of Hensel Lifting (which is guaranteed by Lemma 2.25), we must have that $g_\ell(\mathbf{x},y) = g''(\mathbf{x},y) = g(\mathbf{x}+\boldsymbol{\alpha}y+\boldsymbol{\beta}+\boldsymbol{\delta})$. Thus, for this choice of $\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\delta}$ and T, we would include $g(\mathbf{x}) = g''(\mathbf{x}-\boldsymbol{\beta}-\boldsymbol{\delta},0)$ in the set of candidate factors in Line 15.

   Finally, since the lift also ensures that there exist u_ℓ and v_ℓ such that $u_\ell g_\ell + v_\ell h_\ell = 1 \bmod \mathcal{I}^{2^\ell}$, we also have that $g_\ell^2 \nmid f$.

□

3.2 Running time analysis

We now bound the time complexity of the algorithm.

Lemma 3.4 (Running time of Algorithm 1)**.** *Let $\varepsilon > 0, d, k \in \mathbb{N}$ be an arbitrary constants and let $f \in \mathbb{Q}[\mathbf{x}]$ be a polynomial computable by a $(\Sigma\Pi)^{(k)}$ formula C of size s, degree at most D and bit-complexity t. Then, on input C, Algorithm 1 terminates in time at most $(sD)^{O_\varepsilon(kd(sD)^{\varepsilon d})}\cdot t^{O(d\log d)}$.*

Moreover, if $k = 1$, i.e. f has sparsity at most s, then Algorithm 1 terminates in time at most $(sDt)^{(\mathrm{poly}(d)\log sDt)}$.

Proof. Let $T_k^{(1)}(s,d)$ be the time-complexity to output the hitting set H_1 in Line 2 and $T_k^{(2)}(s,D,d)$ be the time-complexity to output the hitting set H_2 in Line 3.

From Definition 2.11, we immediately have that $T_k^{(1)}(s,d) \le s^{O(d)}$. As for $T_k^{(2)}(s,D,d)$, in the case of $k = 1$, Theorem 2.14 shows that $T_k^{(2)}(s,D,d) \le (sD)^{(\mathrm{poly}(d)\log sD)}$. For k satisfying $2 \le k = o(\log\log\log s)$, then Theorem 2.13 shows that $T_k^{(2)}(s,D,d) \le (sD)^{O_\varepsilon(kd(sD)^{\varepsilon d})}$ for any constant $\varepsilon > 0$.

Using Lemma 2.6, we get that Line 6 takes $\mathrm{poly}(s,D,t)$-time. Now, each of the coefficients of $F(\mathbf{0},y)$ has bit-complexity at most $\mathrm{poly}(s,D,t)$. Thus, from Theorem 2.27, we get that $F(\mathbf{0},y)$ can be factorized into its irreducible factors in time at most $\mathrm{poly}(s,D,t)$.

There are at most D^d choices for the set T in Line 8. For each such choice, Lines 9 to 12 compute formulas of size $\mathrm{poly}(s, D, t)$ for g_0, h_0, u_0, v_0 in time $\mathrm{poly}(s, D, t)$. By Lemma 2.26, we have that Line 13 takes time $(sDt)^{O(\log d)}$ to compute a formula of the same size and bit-complexity for g_ℓ. From Lemma 2.15, we get that we can obtain the coefficient vector of g_ℓ in time at most $(sDt)^{O(d \log d)}$.

Therefore, the overall running time of Algorithm 1 is at most

$$T_k^{(1)}(s, d) \cdot T_k^{(2)}(s, D, d) \cdot D^d \cdot \mathrm{poly}(s, D, t) \cdot (sDt)^{O(d \log d)}.$$

Plugging in the estimates for $T_k^{(1)}(s, d)$, $T_k^{(2)}(s, D, d)$, we get the overall bound of $(sD)^{O_\varepsilon(kd(sD)^{\varepsilon d})} \cdot t^{O(d \log d)}$ for $k > 1$, which is essentially dominated by $T_k^{(2)}(s, D, d)$.

When f has sparsity s, then as discussed in the proof, $T_k^{(2)}(s, d)$ is at most $(sD)^{(\mathrm{poly}(d) \log sD)}$. Plugging this back in the above expression, we get that the running time is at most $(sDt)^{(\mathrm{poly}(d) \log sDd)}$. □

3.3 Proof of structural lemmas

In this subsection, we include the proofs of Observation 3.1 and Lemma 3.2. This completes the analysis of Algorithm 1.

Proof of Observation 3.1. By definition of $\mathcal{H}(d, n)$ (Definition 2.11), the transformation $\mathbf{x} \mapsto \mathbf{x} + y\boldsymbol{\alpha} + \boldsymbol{\beta}$ takes a monomial $\prod_{i \in [n]} x_i^{e_i}$ to $\left(\prod_{i \in T} (x_i + \alpha_i y + \beta_i)^{e_i}\right) \cdot \left(\prod_{i \in [n] \setminus T} x_i^{e_i}\right)$, for some $T \subseteq [n]$ s.t. $|T| = d$. If we expand $\prod_{i \in T} (x_i + \alpha_i y + \beta_i)^{e_i}$ into a sum of monomials, we will get at most $D^{O(d)}$ monomials (when $\sum_i e_i \leq D$). Expanding each $\prod_{i \in [n]} (x_i + \alpha_i y + \beta_i)^{e_i}$ at the bottom layer into a sum of monomials this way, we get the required $(\Sigma\Pi)^{(k)}$ formula with size at most $s \cdot D^{O(d)}$. □

Proof of Lemma 3.2. Let $\mathbf{x}' = \mathbf{x} \setminus \{y\}$ and let $\mathcal{C}$ be the class $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$. Let us assume that $\deg_y(Q) = D \leq s$ and $\deg_y(g) = d$. By Lemma 2.17, we have that $Q(\mathbf{x})$ is computable by a $\mathcal{C}$-formula of size at most $\mathrm{poly}(s, d)$. Let us consider the $(D + d) \times (D + d)$ Sylvester matrix Γ of Q and g with respect to the variable y whose determinant is $\mathrm{Res}_y(Q, g)$.

$$Q(\mathbf{x}) = Q_0(\mathbf{x}') + y \cdot Q_1(\mathbf{x}') + \cdots + y^D \cdot Q_D(\mathbf{x}')$$
$$g(\mathbf{x}) = g_0(\mathbf{x}') + \cdots + y^d \cdot g_d(\mathbf{x}')$$
$$\Gamma = \begin{bmatrix} Q_0 & Q_1 & \cdots & & Q_D & & \\ & \ddots & \ddots & & \ddots & \ddots & \\ & & Q_0 & Q_1 & & \cdots & Q_D \\ g_0 & \cdots & & g_d & & & \\ & g_0 & \cdots & & g_d & & \\ & & \ddots & \ddots & & \ddots & \\ & & & g_0 & \cdots & & g_d \end{bmatrix}$$

Note that, by Lemma 2.6, each of the Q_i's are computed by a $\mathcal{C}$-formula of size $\mathrm{poly}(s, D)$ and each g_i is a polynomial of degree at most d.

For a subset S of rows and T of columns, we will use $\Gamma(S,T)$ to refer to the submatrix restricted to the rows in S and columns in T, and let $\text{Top} = \{1,\dots,d\}$ and $\text{Bot} = \{d+1,\dots,d+D\}$. The determinant of Γ can then be expressed as

$$\det(\Gamma) = \text{Res}_y(Q,g) = \sum_{T \in \binom{[D+d]}{d}} \det(\Gamma(\text{Top},T)) \cdot \det(\Gamma(\text{Bot},\overline{T}))$$

For every choice of T, the polynomial $\det(\Gamma(\text{Top},T))$ is the determinant of a $d \times d$ matrix each of whose entries are computable by $s' = \text{poly}(s,D)$ sized $\mathcal{C}$-formulas. Therefore, using Observation 2.5, the polynomial $\det(\Gamma(\text{Top},T))$ is computable by $\mathcal{C}$-formulas of size at most $(sD)^{O(d)}$.

The polynomial $\det(\Gamma(\text{Bot},\overline{T}))$ is a degree D polynomial combination of $g_0,\dots,g_d$ and can therefore be expressed as

$$\begin{aligned}\det(\Gamma(\text{Bot},\overline{T})) &= \sum_{i=1}^{D^{d+1}} a_i \cdot g_0^{e_{i,0}} \cdots g_d^{e_{i,d}} \\ &= \sum_{i=1}^{D^{d+1}} a_i \cdot \left(\sum_{j=1}^{D^{O(d)}} b_{ij} \cdot f_{ij}^{e_{ij}} \right) \quad \text{(using Lemma 2.9).}\end{aligned}$$

for some polynomials f_j of degree at most d. Thus, using Observation 2.5 again, we have that $\text{Res}_y(Q,g)$ is computable by $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\text{Deg}_d)^*\right)$ formulas of size at most $D^{O(d)} \cdot (sD)^{O(d)} = (sD)^{O(d)}$. □

4 Computing factors of all multiplicity

The following lemma essentially shows that the multiplicity of any factor g of a given polynomial f can be reduced by working with appropriate partial derivatives of f, with respect to variables that are present in g. This naturally yields an algorithm that uses Algorithm 1 as a subroutine, and computes all irreducible factors of f.

Lemma 4.1 (Reducing factor multiplicity)**.** *Let $f(\mathbf{x}), g(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$ be non-zero polynomials and let $x \in \mathbf{x}$ be such that $\partial_x(g) \neq 0$ and g is square-free. Then, the factor-multiplicity of g in f (i.e. the integer a satisfying $g^a \mid f$ and $g^{a+1} \nmid f$) is also the smallest non-negative integer a such that $g \nmid \frac{\partial^a f}{\partial x^a}$.*

Proof. If the factor-multiplicity of g in f is zero, i.e. $g \nmid f$, then claim is clearly true. Thus let us assume that the factor-multiplicity of g in f is $a \geq 1$. It suffices to show that the factor-multiplicity of g in $\partial_x(f)$ is exactly $a-1$.

Suppose $f = g^a \cdot h$ where $\gcd(g,h) = 1$. Then,

$$\partial_x f = \partial_x(g^a) \cdot h + g^a \cdot \partial_x(h) = g^{a-1} \cdot (a \cdot \partial_x(g) \cdot h + g \cdot \partial_x(h)).$$

Hence, we have that the factor-multiplicity of g in $\partial_x(f)$ is at least $(a-1)$.

On the other hand, we have that $\partial_x(g) \neq 0$ and g is square-free and hence $\gcd(g,\partial_x(g)) = 1$. Therefore

$$\gcd(g, a \cdot g \cdot \partial_x(h) + h \cdot \partial_x(g)) = \gcd(g, h \cdot \partial_x(g)) = \gcd(g,h) = 1$$

and hence $g^a \nmid \partial_x(f)$ and therefore the factor-multiplicity of f □

We are now ready to describe the algorithm.

Algorithm 2: Computing list of all degree d irreducible factors and their multiplicities

Input : A $(\Sigma\Pi)^{(k)}$-formula of size s, bit-complexity t, degree D computing a polynomial $f(\mathbf{x})$.
Output: A list of all irreducible factors f of degree at most d and their multiplicities.

1 Set the output list $L = \emptyset$.
2 Set the intermediate candidates list $L' = \emptyset$.
3 Compute hitting-set $H_1 = \mathcal{H}(d, n)$ (as defined in Definition 2.11).
4 **for** $\boldsymbol{\alpha} \in H_1$ **do**
5 Define $F(\mathbf{x}, y) = f(\mathbf{x} + \boldsymbol{\alpha} \cdot y) = f(x_1 + \alpha_1 y, \ldots, x_n + \alpha_n y)$
6 **for** $i = 0, 1, \ldots, \deg(F)$ **do**
7 Define $\tilde{F}(\mathbf{x}, y) = \frac{\partial^i F}{\partial y^i}$.
8 Compute the list $\tilde{L}$ of all candidate degree d multiplicity-one factors of $\tilde{F}(\mathbf{x}, y)$ using Algorithm 1.
9 **foreach** $\tilde{g}(\mathbf{x}, y) \in \tilde{L}$ **do**
10 Add $g(\mathbf{x}) := \tilde{g}(\mathbf{x}, 0)$ to L'.
11 **for** $g \in L'$ **do**
12 **if** g is not irreducible **then** skip to the next iteration.
13 Let x be a variable that g depends on, so that $\partial_x(g) \neq 0$.
14 Find the smallest non-negative integer e such that $g \nmid \frac{\partial^e f}{\partial x^e}$.
15 **if** $e > 1$ **then** add (g, e) to the list L.
16 **return** L

Lemma 4.2 (Correctness of Algorithm 2)**.** *For every input polynomial f computed by a $(\Sigma\Pi)^{(k)}$ formula of size s, degree D, bit-complexity t and $d \in \mathbb{N}$, the list L output by Algorithm 2 is precisely the list of all irreducible factors of f of degree at most d (up to scalar multiplication) along with their multiplicities in f.*

Proof. From Lines 11 to 15 and Lemma 4.1, it is clear that any (g, e) in the output list ensures that g is an irreducible polynomial, $g^e \mid f$ and $g^{e+1} \nmid f$. Thus, it suffices to show that for every irreducible polynomial g such that $\deg(g) \leq d$ and $g \mid f$, some non-zero scalar multiple of g is under consideration in the list L'. Fix any such irreducible factor g of degree at most $r \leq d$ and let its factor-multiplicity be e

By Lemma 2.12, there is some $\boldsymbol{\alpha} \in H_1$ such that $\mathrm{Hom}_r(g)(\boldsymbol{\alpha}) \neq 0$, where r is the total degree of g. Thus, for this choice of $\boldsymbol{\alpha}$, we have that $g'(\mathbf{x}, y) = g(\mathbf{x} + y\boldsymbol{\alpha})$ is a factor of $F(\mathbf{x}, y) = f(\mathbf{x} + y\boldsymbol{\alpha})$ and g' is monic in y and has factor-multiplicity e. By Lemma 4.1, we have that g' has factor-multiplicity one in $\tilde{F}(\mathbf{x}, y) := \frac{\partial^{e-1} F}{\partial y^{e-1}}$. Thus, by the correctness of Algorithm 1 (Lemma 3.3), a non-zero multiple of the polynomial $g'(\mathbf{x}, y)$ must be included in the list $\tilde{L}$ in Line 8. Therefore, a non-zero multiple of $g(\mathbf{x}) = g'(\mathbf{x}, 0)$ will be added to L' in Line 10. □

Lemma 4.3 (Running time of Algorithm 2)**.** *Let $\varepsilon > 0, k, d \in \mathbb{N}$ be arbitrary constants. Let $f \in \mathbb{Q}[\mathbf{x}]$ be a polynomial computable by a $(\Sigma\Pi)^{(k)}$ formula C of size s, degree at most D and bit-complexity t. Then, on input C and $d \in \mathbb{N}$, Algorithm 1 terminates in time at most $(sDt)^{O(kd(sDt)^{\varepsilon d})}$.*

Moreover, if $k = 1$, i.e. f has sparsity at most s, then Algorithm 1 terminates in time at most $(snDt)^{O(\mathrm{poly}(d) \cdot \log snDt)}$.

Proof. From Definition 2.11, we have the size of the set H_1 is $n^{O(d)}$. The time complexity of computing a formula for F from the given formula for f is at most $O(sD)$. From Lemma 2.8, we have

that $(\Sigma\Pi)^{k+1}$ formulas for all the y derivatives of F can be computed in time at most $\text{poly}(s, D, t)$, which is also a bound on the bit-complexity and the size of these formulas. Algorithm 1 is invoked at most D times.

The total time taken to construct the list L' is at most $D \cdot T_1$, where T_1 is the time taken by Algorithm 1 on inputs with formula size and bit-complexity $\text{poly}(s, D, t)$, and degree parameter d. $D \cdot T_1$ is also an upper bound on the size of the list of candidate factors L'.

Now, for each $g \in L'$, from Theorem 2.28, we have that the irreducibility test in Line 12 takes at most $(sDt)^{O(d^2)}$ time. There are at most D instances of divisibility test performed to determine the exact multiplicity in f of each $g \in L'$. This requires computing the corresponding derivatives, which as discussed in the previous paragraph, takes time $\text{poly}(s, D, t)$ and outputs a formula of size and bit-complexity $\text{poly}(s, D, t)$ for the derivatives, and then doing a divisibility test, the time complexity of which we denote by T_2.

Therefore, the total time taken by the algorithm is at most $(n^{O(d)} \cdot \text{poly}(s, D, t) \cdot D \cdot T_1) + (D \cdot T_1 \cdot (sDt)^{O(d^2)} \cdot \text{poly}(s, D, t) \cdot T_2)$.

Now, if f is s sparse, i.e. $k = 1$, then from Definition 2.11, we have that every vector in H_1 has at most d non-zero coordinates. Thus, from Observation 3.1, for every $\boldsymbol{\alpha} \in H_1$, $F(\mathbf{x}, y) = f(\mathbf{x} + \boldsymbol{\alpha} \cdot y)$ has sparsity and bit-complexity at most $s' \leq s \cdot D^d$. Note that the derivatives of arbitrary order of F with respect to any variable also have the same bound on their sparsity and bit-complexity of coefficients. Thus, in this case, from Lemma 3.4, $T_1 \leq (sDt)^{\text{poly}(d) \log sDt}$. From Theorem 2.20, we have that $T_2 \leq (snD)^{O(d \log^2 snD)}$. Therefore, the overall running time of the algorithm is at most $(snDt)^{O(\text{poly}(d) \cdot \log snDt)}$.

On the other hand, if $k > 1$, then from Lemma 3.4, $T_1 \leq (sD)^{O_\varepsilon(kd(sD)^{\varepsilon d})} \cdot t^{O(d \log d)}$. To bound T_2 in this case, we note from Corollary 2.19, this divisibility testing instances reduce to PIT instances for $(\Sigma\Pi)^{(k+1)}$ formula of size and bit-complexity at most $\text{poly}(s, D, t)$ and from Theorem 2.13, this can be done in at most $(sDt)^{O(k(sDt)^{\varepsilon})}$ time for the arbitrary constant ε chosen in the beginning. Thus, the total time taken is at most $(sDt)^{O_\varepsilon(kd(sDt)^{\varepsilon d})}$. □

Lemma 4.2 and Lemma 4.3 together imply our main theorems Theorem 1.1 and Theorem 1.2.

5 Open problems

We conclude with some open problems.

- Perhaps the most natural open problem here is to obtain efficient deterministic algorithms that completely factor sparse polynomials or more generally, polynomials with constant depth formulas (and not just obtain low degree factors). In the absence of better structural guarantees for the factors (for instance, if they are sparse or have small constant depth formulas), we can seek algorithms that output general algebraic circuits for these factors.

- Obtaining improved structural guarantees on the factors of polynomials that are sparse or have small constant depth formulas as mentioned in the first open problem is another very interesting open problem.

- A first step towards obtaining deterministic algorithms for general factorization of polynomials with small constant depth formulas could be to design deterministic algorithms for computing *simple* factors of such polynomials. While the notion of simplicity discussed in

this paper is that of low degree factors, there are other natural notions that seem very interesting. For instance, can we design an efficient deterministic algorithm that outputs all the sparse irreducible factors of a constant depth formula ?

- As alluded to in the introduction, polynomial factorization algorithms have found numerous applications in computer science. It would be interesting to understand if there are applications of deterministic factorization algorithms in general, and in particular the algorithms for computing low degree factors described in this paper.

Acknowledgements

A part of this work was done while the first two authors were at the Workshop on Algebraic Complexity organised at the University of Warwick in March 2023 by Christian Ikenmeyer. We thank Christian for the invitation and the delightful and stimulating atmosphere at the workshop.

References

[AS03] Sanjeev Arora and Madhu Sudan. Improved Low-Degree Testing and its Applications. *Comb.*, 23(3):365–426, 2003.

[BHKS20] Siddharth Bhandari, Prahladh Harsha, Mrinal Kumar, and Madhu Sudan. Decoding Multivariate Multiplicity Codes on Product Sets. *Electron. Colloquium Comput. Complex.*, TR20-179, 2020. Pre-print available at `arXiv:TR20-179`.

[Bog05] Andrej Bogdanov. Pseudorandom generators for low degree polynomials. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 21–30. ACM, 2005.

[BSCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity Gaps for Reed–Solomon Codes. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 900–909, 2020.

[BSV18] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic Factorization of Sparse Polynomials with Bounded Individual Degree. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 485–496. IEEE Computer Society, 2018.

[CKS18] Chi-Ning Chou, Mrinal Kumar, and Noam Solomon. Hardness vs Randomness for Bounded Depth Arithmetic Circuits. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[DL78] Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters*, 7(4):193–195, 1978.

[DSY09] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-Randomness Tradeoffs for Bounded Depth Arithmetic Circuits. *SIAM J. Comput.*, 39(4):1279–1293, 2009.

[Ell69] W.J. Ellison. A 'Waring's Problem' for homogeneous forms. *Proceedings of the Cambridge Philosophical Society*, 65:663–672, 1969.

[Fis94] Ismor Fischer. Sums of like powers of multivariate linear forms. *Mathematics Magazine*, 67(1):59–61, 1994.

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

[For15] Michael A. Forbes. Deterministic Divisibility Testing via Shifted Partial Derivatives. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, page 451–465, USA, 2015. IEEE Computer Society.

[FS13] 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`.

[FS15] Michael A. Forbes and Amir Shpilka. Complexity Theory Column 88: Challenges in Polynomial Factorization1. *SIGACT News*, 46(4):32–49, dec 2015.

[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.

[GKS16] Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity Testing for Constant-Width, and Commutative, Read-Once Oblivious ABPs. In *Proceedings of the 31st Annual Computational Complexity Conference (CCC 2016)*, pages 29:1–29:16, 2016. `arXiv:1601.08031`.

[GKSS19] Zeyu Guo, Mrinal Kumar, Ramprasad Saptharishi, and Noam Solomon. Derandomization from Algebraic Hardness: Treading the Borders. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 147–157. IEEE Computer Society, 2019.

[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)*, pages 323–346, 2015. `arXiv:1411.7341`.

[Kal85] Erich Kaltofen. Polynomial-Time Reductions from Multivariate to Bi- and Univariate Integral Polynomial Factorization. *SIAM Journal of Computing*, 14(2):469–489, 1985.

[Kal89] Erich Kaltofen. Factorization of Polynomials Given by Straight-Line Programs. In *Randomness and Computation*, pages 375–412. JAI Press, 1989.

[Kal92] Erich L. Kaltofen. Polynomial Factorization 1987-1991. In *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*, volume 583 of *Lecture Notes in Computer Science*, pages 294–313. Springer, 1992.

[Kal03] Erich L. Kaltofen. Polynomial factorization: a success story. In *Symbolic and Algebraic Computation, International Symposium ISSAC 2003, Drexel University, Philadelphia, Pennsylvania, USA, August 3-6, 2003, Proceedings*, pages 3–4. ACM, 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.

[KS09] Neeraj Kayal and Shubhangi Saraf. Blackbox polynomial identity testing for depth-3 circuits. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, 2009.

[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)*.

[KT88] Erich L. Kaltofen and Barry M. Trager. Computing with Polynomials Given By Black Boxes for Their Evaluation: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 296–305. IEEE Computer Society, 1988.

[KY08] Swastik Kopparty and Sergey Yekhanin. Detecting Rational Points on Hypersurfaces over Finite Fields. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC 2008)*, pages 311–320, 2008.

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

[LST21] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial Lower Bounds Against Low-Depth Algebraic Circuits. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2021)*, pages 804–814. IEEE, 2021. Preliminary version in the Electronic Colloquium on Computational Complexity (ECCC), Technical Report TR21-081.

[Ore22] Øystein Ore. Über höhere Kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-Linear Size Holographic Proofs. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, page 194–203, New York, NY, USA, 1994. Association for Computing Machinery.

[Sap15] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015.

[Sch80] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, 1980.

[Shp02] Amir Shpilka. Affine projections of symmetric polynomials. *Journal of Computer and System Sciences*, 65(4):639–659, 2002. Special Issue on Complexity 2001.

[SS09] Nitin Saxena and C. Seshadhri. An Almost Optimal Rank Bound for Depth-3 Identities. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC 2009)*, pages 137–148, 2009.

[SS10] Nitin Saxena and C. Seshadhri. From Sylvester-Gallai Configurations to Rank Bounds: Improved Black-Box Identity Test for Depth-3 Circuits. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 21–29, 2010.

[ST20] Amit Sinhababu and Thomas Thierauf. Factorization of Polynomials Given By Arithmetic Branching Programs. In *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[Sud98] Madhu Sudan. Lecture notes for the course 'Algebra and Computation', 1998. Available from `http://people.csail.mit.edu/madhu/FT98/`.

[SV10] Amir Shpilka and Ilya Volkovich. On the Relation between Polynomial Identity Testing and Finding Variable Disjoint Factors. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 408–419. Springer, 2010.

[Vol15] Ilya Volkovich. Deterministically Factoring Sparse Polynomials into Multilinear Factors and Sums of Univariate Polynomials. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 943–958. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[Vol17] Ilya Volkovich. On Some Computations on Sparse Polynomials. volume 81 of *LIPIcs*, pages 48:1–48:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[VSBR83] 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)*.

[vzG83] Joachim von zur Gathen. Factoring Sparse Multivariate Polynomials. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1983)*, pages 172–179, 1983.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013.

[Zip79] 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.

A Deferred proofs

Circuit/formula bit-complexity

Proof of Lemma 2.2. We will prove an equivalent statement: the numerator and denominator of $f(\mathbf{a})$ have absolute value at most $2^{s \cdot b}$. We prove this by induction on the size of the formula. We will use $N(\cdot)$ and $D(\cdot)$ to denote the numerator and denominator of some rational number.

Base case: when $\text{size}(C) = 1 = s$, there is a single leaf node in the formula that reads and outputs a single rational number of bit-complexity b, thus $\text{bit}(f(\mathbf{a})) = b \leq s \cdot b$. The induction hypothesis is that for all formulas C with $\text{size}(C) \leq S$ (for some $S \geq 1$), $\text{bit}(f(\mathbf{a})) \leq \text{bit}(C) \cdot b$. For the induction step, we look at formulas C with $\text{size}(C) = S + 1$, and we consider two cases:

1. When the top gate is a sum gate: $f = \sum_{i=1}^{k} \alpha_i g_i(\mathbf{x})$, with C_i being the formula computing g_i and α_is being scalars from $\mathbb{Q}$.

$$
\begin{aligned}
f(\mathbf{a}) &= \sum_{i=1}^{k} \alpha_i g_i(\mathbf{a}) \\
|D(f(\mathbf{a}))| &= \left| \prod_{i=1}^{k} D(\alpha_i) D(g_i(\mathbf{a})) \right| \\
&\leq \prod_{i=1}^{k} 2^{\text{bit}(\alpha_i)} 2^{\text{bit}(g_i(\mathbf{a}))} \\
&\leq \prod_{i=1}^{k} 2^{\text{bit}(\alpha_i)} 2^{\text{bit}(C_i) \cdot b} && \text{(induction hypothesis)} \\
&= 2^{\sum_{i=1}^{k} (\text{bit}(\alpha_i) + \text{bit}(C_i) \cdot b)} \leq 2^{\text{bit}(C) \cdot b}
\end{aligned}
$$

$$
\begin{aligned}
|N(f(\mathbf{a}))| &\leq \sum_{i=1}^{k} |N(\alpha_i)| \, |N(g_i(\mathbf{a}))| \prod_{j \neq i} \left|D(\alpha_j)\right| \left|D(g_j(\mathbf{a}))\right| \\
&\leq \sum_{i=1}^{k} 2^{\text{bit}(\alpha_i) + \text{bit}(g_i(\mathbf{a}))} 2^{\sum_{j \neq i} \text{bit}(\alpha_j) + \text{bit}(g_j(\mathbf{a}))} \\
&\leq \sum_{i=1}^{k} 2^{\text{bit}(\alpha_i) + \text{bit}(C_i) \cdot b} 2^{\sum_{j \neq i} \text{bit}(\alpha_j) + \text{bit}(C_j) \cdot b} && \text{(induction hypothesis)} \\
&\leq \sum_{i=1}^{k} 2^{\sum_{j=1}^{k} \text{bit}(\alpha_j) + \text{bit}(C_j) \cdot b} \leq 2^{k + \sum_{j=1}^{k} \text{bit}(\alpha_j) + \text{bit}(C_j) \cdot b} \leq 2^{\text{bit}(C) \cdot b}
\end{aligned}
$$

   Thus, $\text{bit}(f(\mathbf{a})) = \max\{\text{bit}(N(f(\mathbf{a})), D(f(\mathbf{a})))\} \leq \text{bit}(C) \cdot b$.

2. When the top gate is a product gate: $f = \prod_{i=1}^{k} \alpha_i g_i(\mathbf{x})$. The proof for the denominator in the case of sum gate will work here for both the numerator and the denominator. The required bound follows.

□

Relevant subclasses of algebraic circuits

Proof of Observation 2.5. We prove the size upper bounds here; the bit-complexity upper bounds proceed along exactly the same lines. The size upper bound for the sum is immediate and hence we only need to focus on the product. Let the expression for each P_r be

$$P_r = \sum_i P_{r,i}^{(r)} \cdot g_{r,i}^{a_{r,i}}$$

$$\implies \prod P_r = \sum_{r_1,\ldots,r_t} (P_{1,r_1} \cdots P_{t,r_t}) \cdot \left(g_{1,r_1}^{a_{1,r_1}} \cdots g_{t,r_t}^{a_{t,r_t}} \right)$$

where each $P_{i,j}$ is computed by $(\Sigma\Pi)^{(k)}$ formulas of size at most s, and each $g_{i,j}$ is a polynomial of degree at most d.

Each $(P_{1,r_1} \cdots P_{t,r_t})$ is computed by a $(\Sigma\Pi)^{(k)}$ formula of size at most s^t. By Lemma 2.9, $g_{1,r_1}^{a_{1,r_1}} \cdots g_{t,r_t}^{a_{t,r_t}}$ can be expressed as a sum $\sum_{\ell=1}^{s^t} f_\ell^D$ where $D = \sum_j a_{j,r_j}$ and each f_ℓ is a degree polynomial of degree at most d. Thus, $\prod_r P_r$ is computable by a $\Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formula of size at most $s^{O(t)}$. □

Polynomial identity testing

Proof of Lemma 2.12. Since f is a non-zero polynomial of degree at most d, there is a monomial $\mathbf{x}^\mathbf{e}$ of degree at most d with a non-zero coefficient in f. Let S be the support of the monomial $\mathbf{x}^\mathbf{e}$, i.e., $S = \{x_i : e_i \neq 0\}$. Clearly, $|S| \leq d$. We now consider the polynomial $\tilde{f}$ obtained from f by setting all the variables x_j not in the set S to zero. Since f has a non-zero monomial with support contained in the set S, $\tilde{f}$ continues to be a non-zero polynomial of degree at most d. Moreover, it is a d variate polynomial since it only depends on the variables in S. From Lemma 2.10, we get that for any subset T_d of $\mathbb{Q}$ of cardinality at least $d+1$, there exists a vector $\mathbf{b} \in {T_d}^d$ such that $\tilde{f}(\mathbf{b}) \neq 0$. Let $\mathbf{a} \in \mathbb{Q}^n$ to be such that for every $i \in S$, $a_i = b_i$ and for every $i \notin S$, $a_i = 0$. Then, $f(\mathbf{a}) = \tilde{f}(\mathbf{b}) \neq 0$. Moreover, $\mathbf{a}$ is in $\mathcal{H}(d,n)$. □

Proof of Lemma 2.15. Let T_d be the set $\{0,1,2,3,\ldots,d\}$ and let $\mathcal{H}(d,n)$ be the set of points defined in Definition 2.11, i.e.,

$$\mathcal{H}(d,n) = \left\{ (a_1,\ldots,a_n) \;:\; S \in \binom{[n]}{\leq d},\; a_i \in T_d \text{ for all } i \in S \text{ and } a_j = 0 \text{ for all } j \notin S \right\}.$$

From Lemma 2.12, we know that every non-zero polynomial f of degree at most d must evaluate to zero on some point of $\mathcal{H}(d,n)$. In other words, two distinct degree d polynomials f and g cannot agree on every point of $\mathcal{H}(d,n)$. An immediate consequence of this is that if we are given the evaluations of an unknown polynomial f on all points of $\mathcal{H}(d,n)$, and we view each of these evaluations as a linear constraint on the unknown coefficients of f, then this linear system has a unique solution.

Based on this observation, a natural algorithm for computing the coefficient vector of C is the following, we evaluate the given formula on every input in $\mathcal{H}(d,n)$, set up the linear system on the coefficients of C obtained from these evaluations, and use any standard linear system solver over $\mathbb{Q}$ to solve this system.

Note that the size of this linear system is at most $n^{O(d)}$, and from Lemma 2.2, of the constants in this linear system is at most $\mathrm{poly}(s,b,d)$. Thus, this linear system can be solved in time $\mathrm{poly}(s,b,d,n^d) \leq \mathrm{poly}(s,b,n^d)$ time as claimed. □

Deterministic divisibility testing and PIT

Proof of Corollary 2.19. The proof essentially follows immediately from Theorem 2.18. From Theorem 2.18, we have that g divides f if and only if $R(\mathbf{x}) := f(\mathbf{x}) - g(\mathbf{x})Q(\mathbf{x}) \equiv 0$, where Q is the pseudo-quotient of f and g. It suffices to show that $R(\mathbf{x})$ has $\mathcal{C} = \Sigma\left((\Sigma\Pi)^{(k)} \cdot (\mathrm{Deg}_d)^*\right)$ formulas of size $\mathrm{poly}(s,d)$, and since $f(\mathbf{x}) \in (\Sigma\Pi)^{(k)}$, it suffices to bound the size of $\mathcal{C}$-formulas computing $g(\mathbf{x}) \cdot Q(\mathbf{x})$.

By Lemma 2.17, the pseudo-quotient $Q(\mathbf{x})$ is computable by $\mathcal{C}$-formulas of size $\mathrm{poly}(s,d)$. Let one such computation be of the form

$$Q(\mathbf{x}) = \sum_i f_i \cdot g_i^{e_i} \quad \text{where each } f_i \in (\Sigma\Pi)^{(k)} \text{ and } \deg(g_i) \leq d \text{ and each } e_i \leq s$$
$$\implies g(\mathbf{x})Q(\mathbf{x}) = \sum_i f_i \cdot (g \cdot g_i^{e_i})$$

From Lemma 2.9, note that any term of the form $(g \cdot h^e)$ can be expressed as

$$g \cdot h^e = \sum_{i=1}^{\mathrm{poly}(e)} \beta_i \cdot (g + \alpha_i h)^{e+1}$$

for field constants α_i's and β_i's. Thus, feeding this in the above expression for $g \cdot Q$, we have

$$g(\mathbf{x}) \cdot Q(\mathbf{x}) = \sum_i \sum_j f_i \cdot \tilde{g}_{ij}^{e_{ij}}$$

for polynomial $\tilde{g}_{ij}$ of degree at most d, and thus is also a $\mathcal{C}$-formula of size at most $\mathrm{poly}(s,d)$. Therefore, $R(\mathbf{x}) = f(\mathbf{x}) - g(\mathbf{x})Q(\mathbf{x})$ is also computable by $\mathcal{C}$-formulas of size $s' = \mathrm{poly}(s,d)$. Thus, we can check if g divides f by checking if $R(\mathbf{x}) \equiv 0$ (by Theorem 2.18) which can be done in $T(k,d,s')$ time as claimed. □

Hensel lifting

Proof sketch of Lemma 2.26. As indicated earlier, the lemma is almost an immediate consequence of Lemma 3.6 in [KSS15]. The precise statement there gives a circuit $\tilde{C}_k$ of size and bit-complexity $\mathrm{poly}(s, D, 2^k)$ for g_k, h_k. We notice that without loss of generality, the degree of g_k, h_k and hence of $\tilde{C}_k$ can be assumed to be at most $(D + 2^k)$ since the y degree is at most D and the x degree is at most 2^k. This incurs at most a polynomial blow up in the circuit size.

Now, to go from circuits for g_k, h_k to formulas computing these polynomials, we just invoke the classic depth reduction result of Valiant, Skyum, Berkowitz and Rackoff [VSBR83], which states that given an n-variate degree-Δ polynomial f with an arithmetic circuit Φ of size s, there is an arithmetic circuit Φ' that computes f, has size $\mathrm{poly}(s, n, \Delta)$ and depth $O(\log \Delta)$.

Thus we have a formula of size (and bit-complexity) at most $\mathrm{poly}(s, D, 2^k)^{\log(D+2^k)} \leq (sDk)^{k \log D}$. Note that a better bound of d on the total degree of g_k implies that the size and bit-complexity of the formula for g_k is at most $(sDk)^{O(\log d)}$. □