Electronic Colloquium on Computational Complexity, Report No. 197 (2024)

Derandomizing Multivariate Polynomial Factoring for Low Degree Factors

Pranjal Dutta [*] Amit Sinhababu [†] Thomas Thierauf [‡]

November 26, 2024

Abstract

For a polynomial f from a class $\mathcal{C}$ of polynomials, we show that the problem to compute all the *constant degree* irreducible factors of f reduces in polynomial time to polynomial identity tests (PIT) for class $\mathcal{C}$ and divisibility tests of f by constant degree polynomials. We apply the result to several classes $\mathcal{C}$ and obtain the constant degree factors in

1. polynomial time, for $\mathcal{C}$ being polynomials that have only constant degree factors,
2. quasipolynomial time, for $\mathcal{C}$ being sparse polynomials,
3. subexponential time, for $\mathcal{C}$ being polynomials that have constant-depth circuits.

Result 2 and 3 were already shown by Kumar, Ramanathan, and Saptharishi with a different proof and their time complexities necessarily depend on black-box PITs for a related bigger class $\mathcal{C}'$. Our complexities vary on whether the input is given as a blackbox or whitebox.

We also show that the problem to compute the *sparse* factors of polynomial from a class $\mathcal{C}$ reduces in polynomial time to PIT for class $\mathcal{C}$, divisibility tests of f by sparse polynomials, and irreducibility preserving bivariate projections for sparse polynomials. For $\mathcal{C}$ being sparse polynomials, it follows that it suffices to derandomize irreducibility preserving bivariate projections for sparse polynomials in order to compute all the sparse irreducible factors efficiently. When we consider factors of sparse polynomials that are sums of univariate polynomials, a subclass of sparse polynomials, we obtain a polynomial time algorithm. This was already shown by Volkovich with a different proof.

1 Introduction

The problem of *multivariate polynomial factorization* asks to find the unique factorization of a given polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ as a product of distinct irreducible polynomials over $\mathbb{F}$. The problem reduces to *univariate polynomial factorization* over the same field, for which a deterministic polynomial time algorithm is known over the field $\mathbb{Q}$. The complexity of multivariate factorization depends on the representation of input and output polynomials. If we use *dense representation* (where all the coefficients are listed including the zero coefficients), deterministic

[*] School of Computing, National University of Singapore. Email: duttpranjal@gmail.com

[†] Dept. of Computer Science, Chennai Mathematical Institute, India. Email: amitkumarsinhababu@gmail.com

[‡] Ulm University, Germany. Email: thomas.thierauf@uni-ulm.de

ISSN 1433-8092

polynomial time algorithms for multivariate factoring are known [Kal85a]. If we use *sparse representation* (where only the nonzero coefficients are listed), only randomized polynomial time (in the total sparsity of input polynomial and the output factors) algorithms are known [vzGK85, KT90]. There are other standard representations like arithmetic circuits, and blackbox models (that gives the evaluations of the polynomial at any point, but the internal structure of the computation is hidden). Randomized polynomial time factorization algorithms are known in these models due to the classic results of Kaltofen [Kal89] and Kaltofen and Trager [KT90].

Towards derandomization of special cases of multivariate factoring, we ask the following.

Question 1 (Promise factoring[vzGK85])**.** *Let* $f = \prod_{i=1}^{m} {g_i}^{e_i}$*, where each* g_i *is an irreducible polynomial with its sparsity at most* s*. Design a* better-than-exponential *time deterministic algorithm in* s, n, d*, to output* g_i*.*

To our surprise, we do not know a deterministic *subexponential*-time algorithm even for the special case of Question 1, when the given blackbox computes the product of just *two* irreducible sparse polynomials.

Note that the factors of a sparse polynomial f might be nonsparse. Therefore, instead of finding *all* the irreducible factors, we only focus on those factors that are sparse.

Question 2. *Design a* better-than-exponential *time deterministic algorithm that outputs all the sparse irreducible factors of a sparse polynomial.*

Multivariate polynomial factoring has various applications, such as *low-degree testing* [AS03], constructions of pseudorandom generators for low-degree polynomials [Bog05, DGV24], computational algebraic geometry [HW00] and many more. Blackbox multivariate polynomial factorization is extensively used in *arithmetic circuit reconstruction* [Shp07, Sin16], and polynomial equivalence testing [Kay, Kay12, RR19]. Algebraic hardness vs randomness [KI03] results crucially use multivariate factorization.

General framework of factoring. The typical steps in factorization algorithms for a given polynomial $f \in \mathcal{C}$, from a class $\mathcal{C}$ are as follows.

1. Transform f to a polynomial $\widehat{f}$ that is monic in some newly introduced variable, using PIT; for a detailed discussion see below.

2. Project $\widehat{f}$ to a bivariate polynomial such that the factorization pattern of $\widehat{f}$ is maintained (by Hilbert Irreducibility Theorem; see Theorem 2.11).

3. Use the known algorithms to factor the projected bivariate polynomial.

4. Now often *Hensel lifting* is used to lift the projected factors back to the real factors. In our setting, we avoid Hensel lifting and argue that we can use *interpolation* instead. This interpolation trick was first used in the randomized black-box factoring algorithm of Kaltofen and Trager [KT90].

5. To make sure that the computed polynomials are indeed factors, there is a divisibility test at the end.

We now discuss some of the crucial steps and their importance in factoring a restricted class of polynomials. In our restricted setting, we completely *bypass* the step of Hensel lifting.

PIT. Given an arithmetic circuit C, polynomial identity testing (PIT) asks to test if C computes the zero polynomial. Randomization in factorization algorithms mostly stem from the fact that these algorithms use PIT as a subroutine. Further, Kopparty, Saraf and Shpilka [KSS15] showed that derandomization of white-box and blackbox multivariate circuit factoring reduces to derandomization of polynomial identity testing of arithmetic circuits in white-box and blackbox settings respectively. However, we *do not* know if sparse factorization reduces to sparse PIT or constant-depth arithmetic circuit PIT (the algorithms of [KSS15] reduce to general arithmetic circuit PIT). Special cases of depth-4 polynomial identity testing are related to questions about sparse polynomial factorization [Gup14, Vol17, BV22]. Recently, there has been some progress on these questions by [KRS24, KRSV24]. Earlier works of Volkovich [Vol15, Vol17] made progress on several special cases of sparse multivariate factoring. Shpilka and Volkovich [SV10] proved white-box/black-box factoring of *multilinear* polynomials in a class $\mathcal{C}$ is equivalent to white-box/black-box derandomization of PIT of class $\mathcal{C}$.

Divisibility testing. In a factorization algorithm, we may want to check if a candidate factor is truly a factor via *divisibility testing*. It asks to test if a polynomial $g(z)$ divides a polynomial $f(z)$. Forbes [For15] showed that the divisibility testing question can be efficiently reduced to an instance of a PIT question of a model that relates to both f and g, see Lemma 2.8. Currently, we do not know any deterministic polynomial time algorithm even when g and f are both sparse polynomials. When f is a sparse polynomial and g is a linear polynomial, the problem reduces to polynomial identity testing of any-order read-once oblivious branching programs (ROABPs), see Section 3.4. We do not know a deterministic polynomial time algorithm, even for testing if a quadratic polynomial g divides a sparse polynomial.

Irreducibility projection. For testing irreducibility of multivariate polynomials, one can use some of the effective versions of Hilbert's irreducibility theorem [vzGK85, Kal85b, Kal95]. These results can be also seen as an effective version of a classical theorem of Bertini in algebraic geometry. A multivariate irreducible polynomial $f(x, z_1, \ldots, z_n)$ may become reducible if we project the variables to make it univariate, but irreducibility is preserved if we project it in a way to make it a bivariate polynomial. Unfortunately, finding such a projection in deterministic polynomial time is hard. We do not know how to find such irreducibility preserving projections even for sparse polynomials, though we know efficient hitting set generators for them.

1.1 Our results

We show general results that exhibits properties of a class $\mathcal{C}$ of polynomials, such that we can compute the *constant-degree* factors or the *sparse* factors of polynomials $f \in \mathcal{C}$. Thereby we get a unified and simple way reprove some known results and also some new results.

Constant-degree factors. In Theorem 3.2 we show that the irreducible constant-degree factors of a polynomial $f \in \mathcal{C}$ with their multiplicities can be computed in polynomial time relative to

- PIT for $\mathcal{C}$,
- divisibility tests of $\mathcal{C}$ by constant-degree polynomials.

Considering the above general algorithmic framework, the PIT comes from step 1 and divisibility from step 5. The Hilbert Irreducibility Theorem (see Theorem 2.11) yields a randomized way to do step 2. The main point here is that we derandomize this step for constant-degree factors.

For a specific class $\mathcal{C}$, it suffices now to consider the complexity of the above two points that we already described before. For example for $\mathcal{C}$ being polynomials that have only constant degree factors, this yields a polynomial time algorithm for computing the constant degree factors, because PIT is in polynomial time and we can skip the divisibility test (Theorem 3.1). For $\mathcal{C}$ being *sparse polynomials*, we get a quasipolynomial-time algorithm (Corollary 3.3), and for $\mathcal{C}$ being polynomials computed by *constant-depth circuits*, we get subexponential-time (Corollary 3.4).

The last two results were already shown by Kumar, Ramanathan and Saptharishi [KRS24] with a different technique, using Hensel lifting. However, there is a subtle difference. In our case, we maintain the setting. That is, when the input is given whitebox, we use whitebox algorithms, and similarly for blackbox. On the other hand, the factoring algorithm by Kumar, Ramanathan and Saptharishi [KRS24] requires blackbox PIT algorithms, even when the input is given in whitebox. This is due to PIT for the *resultant polynomial* where the unknown factor is involved.

In the above two results, this difference does not matter because we have the same complexity bounds for whitebox and blackbox algorithms there. An example where this difference matters are commutative *read-once oblivious arithmetic branching programs* (ROABP). There is a polynomial time whitebox PIT algorithm for ROABPs [RS05], while the best-known blackbox PIT algorithm for commutative ROABPs runs in quasipolynomial time [GKS17, GG20]. Moreover, it follows from work of Forbes [For15] that the divisibility test for ROABPs by *linear* polynomials reduces to a PIT for ROABPs. Hence, we have again polynomial, resp. quasipolynomial time for the divisibility test in the whitebox, resp. blackbox setting. It follows that the linear factors of polynomials computed by commutative ROABPs can be computed in polynomial time in the whitebox setting, whereas it takes quasipolynomial time in the blackbox setting (Corollary 3.8).

Sparse factors. In Theorem 4.2 we show that the irreducible sparse factors of a polynomial $f \in \mathcal{C}$ with their multiplicities can be computed in polynomial time relative to

- PIT for $\mathcal{C}$,
- divisibility tests of $\mathcal{C}$ by constant-degree polynomials,
- irreducible projection to bivariate polynomials.

For $\mathcal{C}$ being *sparse polynomials*, the difficulty lies in the third point: only exponential-time algorithms are known to derandomize Hilbert Irreducibility Theorem for sparse polynomials. The hardness stems from the fact that a sparse polynomial may have both sparse and non-sparse irreducible factors. Hence, preserving irreducibility for sparse polynomials will not preserve the factorization pattern, and therefore, it may be hard to get back the actual factor. Nevertheless, Theorem 4.2 pinpoints the challenge to compute the sparse factors of sparse polynomials (Corollary 4.4 and 4.5).

For polynomials that can be written as a sum of univariate polynomials, a subclass of sparse polynomials, the irreducible projection problem can be solved in polynomial time. Therefore we can compute the sum-of-univariate factors of sparse polynomials in polynomial time (Corollary 4.9).

2 Preliminaries

We take $\mathbb{F} = \mathbb{Q}$ as the underlying field throughout the paper, although the results hold as well over fields with large characteristics.

Let $\mathcal{P}(n, d)$ be the set of n-variate polynomials of degree at most d, with variables $\boldsymbol{z} = (z_1, z_2, \ldots, z_n)$. By $\deg(f)$ we denote the total degree of f. For an exponent vector $\boldsymbol{e} = (e_1, e_2, \ldots, e_n)$, we denote the monomial

$$\boldsymbol{z}^{\boldsymbol{e}} = z_1^{e_1}\, z_2^{e_2} \,\cdots\, z_n^{e_n}\,.$$

Its degree is $\|\boldsymbol{e}\|_1 = \sum_{i=1}^{n} e_i$. The number of monomials of a polynomial f with nonzero coefficient is called the *sparsity of* f and is denoted by $\mathrm{sp}(f)$. The class of polynomials with sparsity s is denoted by

$$\mathcal{C}_{\mathrm{sp}}(s, n, d) = \{\, p \in \mathcal{P}(n, d) \mid \mathrm{sp}(p) \leq s \,\}.$$

For $\deg(f) = d$, we can write $f = \sum_{k=0}^{d} f_k$, where $f_k = \mathrm{Hom}_k[f]$ denotes the homogeneous component of f of degree k. For the highest degree component, we also skip the index, i.e. we define $\mathrm{Hom}[f] = \mathrm{Hom}_d[f] = f_d$. For a class $\mathcal{C}$ of polynomials, we define

$$\mathrm{Hom}[\mathcal{C}] = \{\, \mathrm{Hom}[f] \mid f \in \mathcal{C} \,\}.$$

We define class $\partial\mathcal{C}$ as all the partial the derivatives of polynomials in $\mathcal{C}$,

$$\partial\mathcal{C} = \left\{ \frac{\partial^e f}{\partial z^e} \mid f \in \mathcal{C},\ z \text{ a variable of } f,\ \text{and } e \geq 0 \right\}.$$

Note that $f \in \partial\mathcal{C}$, because $f = \frac{\partial^0 f}{\partial z^0}$.

Polynomial $f(\boldsymbol{z})$ *depends on* variable z_i, if $\frac{\partial f}{\partial z_i} \neq 0$. The variables that f depends on are denoted by $\mathrm{var}(f)$,

$$\mathrm{var}(f(\boldsymbol{z})) = \{\, z_i \mid f \text{ depends on } z_i \,\}.$$

A polynomial f is called *irreducible*, if it cannot be factored into the product of two non-constant polynomials.

Let x and $\boldsymbol{z} = (z_1, \ldots, z_n)$ be variables and $f(x, \boldsymbol{z})$ be a $(n+1)$-variate polynomial. Then we can view f as a univariate polynomial $f = \sum_i a_i(\boldsymbol{z})\, x^i$ over $\mathbb{K}[x]$, where $\mathbb{K} = \mathbb{F}[\boldsymbol{z}]$. The x*-degree of* f is denoted by $\deg_x(f)$. It is the highest degree of x in f. Polynomial f is called *monic in* x, if the coefficient $a_{d_x}(\boldsymbol{z})$ is the constant 1 polynomial, i.e. $a_{d_x}(\boldsymbol{z}) = 1$, where $d_x = \deg_x(f)$.

An algorithm runs in *subexponential time*, if its running time on inputs of length n and $\varepsilon > 0$ is bounded by $2^{n^{\varepsilon}}$.

2.1 Computational problems and complexity measures

For classes $\mathcal{P}, \mathcal{Q}$ of multivariate polynomials, we define the following computational problems.

- $\mathrm{PIT}(\mathcal{P})$: given $p \in \mathcal{P}$, decide whether $p \equiv 0$.
- $\mathrm{Factor}(\mathcal{P}|_{\mathcal{Q}})$: given $p \in \mathcal{P}$, compute all its irreducible factors in $\mathcal{Q}$ with their multiplicities.
- $\mathrm{Div}(\mathcal{P}/\mathcal{Q})$: given $p \in \mathcal{P}$ and $q \in \mathcal{Q}$, decide whether $q|p$.

The time complexity to solve these problems we denote by $T_{\mathrm{PIT}(\mathcal{P})}$, $T_{\mathrm{Factor}(\mathcal{P}|_{\mathcal{Q}})}$, and $T_{\mathrm{Div}(\mathcal{P}/\mathcal{Q})}$, respectively.

The problems are further distinguished according to the access to the given polynomial p. In the *whitebox* case, one gets the representation of $p \in \mathcal{P}$, for example a circuit or a formula, whereas in the *blackbox* case, one can only evaluate p. For PIT, a blackbox solution therefore means to compute a *hitting set* for a class $\mathcal{P}$. This is a set $H \subseteq \mathbb{F}^n$ such that for every nonzero $p \in \mathcal{P}$ there exists $a \in H$ such that $p(a) \neq 0$.

The blackbox PIT-algorithm then simply evaluates p at all points $a \in H$. Hence, the size of H is crucial for the running time $T_{\mathrm{PIT}(\mathcal{P})}$. The *trivial hitting set* for $\mathcal{P}(n, d)$ used in the Schwartz-Zippel PIT-Lemma is

$$H_{n,d} = [d+1]^n \tag{1}$$

of size $|H_{n,d}| = (d+1)^n$, i.e. exponential in the number n of variables. We will use it for constant-variate polynomials. Then the size is polynomial.

For the decision problems $\mathrm{PIT}(\mathcal{P})$ and $\mathrm{Div}(\mathcal{P}/\mathcal{Q})$ there is an associated *construction problem.* In case of $\mathrm{PIT}(\mathcal{P})$, a decision algorithm also yields an algorithm that computes a point $a \in (\mathbb{F}\setminus\{0\})^n$ such that $p(a) \neq 0$, in case when $p \not\equiv 0$.

Lemma 2.1. *Let $p \in \mathcal{P}$ be a nonzero polynomial. A point $a \in (\mathbb{F} \setminus \{0\})^n$ such that $p(a) \neq 0$ can be computed in time $nd\,T_{\mathrm{PIT}(\mathcal{P})}$.*

Proof. In the blackbox case, let H_0 be the queries of the decision algorithm on the input of the zero-polynomial. Note that H_0 is a hitting set for the whole class $\mathcal{P}$. Hence, we can find $a \in H_0$ in time $T_{\mathrm{PIT}(\mathcal{P})}$ such that $p(a) \neq 0$.

Still, some coordinates of a might be 0. In this case, we shift a: Let t be a new variable and consider $a+t = (a_1+t, a_2+t, \ldots, a_n+t)$. Then $p(a+t)$ is a nonzero polynomial in one variable of degree d. Let $M = |a_i|$, where a_i is the minimum coordinate of a. Then all coordiantes of $a+M+1$ are positive and there is a $t \in \{M+1, M+2, \ldots, M+d+1\}$, such that $p(a+t) \neq 0$. For the time complexity to find the right t, we have to add $(d+1)$ evaluations of p.

In the whitebox case, one can search for a by assigning values successively to the variables and do kind of a *self-reduction.* For each variable, one tries at most d values from $\{1, 2, \ldots, d\}$ for a polynomial of degree d. If they all give 0, definitely $d+1$ works because it cannot be zero at $(d+1)$ many values. With n variables, this amounts to nd calls to the PIT-decision algorithm. □

For time complexity, we assume that the polynomials are given in some model of computation, such as circuits, branching programs, or formulas. With each model, we associate a complexity measure $\mu : \mathbb{F}[z] \to \mathbb{N}$. For example, let $f \in \mathbb{F}[z]$, some of the commonly used measures in the literature are:

- $\mu(f) = \mathrm{sp}(f)$, the number of monomials with nonzero coefficients,
- $\mu(f) = \mathrm{size}_{\Delta}(f)$, the size of the *smallest* depth-Δ circuit that computes f,
- $\mu(f) = \mathrm{size}_{\mathrm{ROABP}}(f)$, the *width* of the smallest read read-once oblivious branching program (ROABP) that computes f.

We define classes of polynomials of bounded measure,

$$\mathcal{C}_{\mu}(s, n, d) \ = \{ f \in \mathcal{P}(n, d) \mid \mu(f) \leq s \}.$$

For example, when we skip the index μ, we just refer to circuit size,

$$\mathcal{C}(s, n, d) = \{ p \in \mathcal{P}(n, d) \mid p \text{ has a circuit of size } s \}.$$

We also consider polynomials that can be computed by circuits of size s and depth t,

$$\mathcal{C}_{\text{Depth-t}}(s, n, d) = \{ p \in \mathcal{C}(s, n, d) \mid p \text{ has a circuit of depth } t \}.$$

We generally assume that all polynomials in this paper can be *efficiently evaluated* at any point $\mathbf{a} \in \mathbb{F}^n$ within the respective measure, where we consider the unit-cost model for operations over $\mathbb{F}$. This holds for all the computational models usually considered in the literature.

2.2 Transformation to a monic polynomial

Algorithms for factoring polynomials often assume that the given polynomial is monic. If this is not the case for the given polynomial f, we apply a transformation τ to f that yields a monic polynomial $\tau(f)$ that we can factor. From the factors of $\tau(f)$ we can then reveal the factors of f. Although this is standard in the literature, we state it in the terms we introduced above.

Lemma 2.2 (Transformation to monic). *Let* $\mathcal{C}_{\mu} = \mathcal{C}_{\mu}(s, n, d)$ *be a class of polynomials,* $f(z) \in \mathcal{C}_{\mu}$*, and* $f_d = \mathrm{Hom}_d[f]$ *be the homogeneous degree* d *component of* f*. For a new variable* x*, and* $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in (\mathbb{F} \setminus \{0\})^n$*, define a linear transformation* τ_{α} *on the variables* z_i*:*

$$\tau_{\alpha} : \quad z_i \mapsto \alpha_i x + z_i,$$

for $i = 1, 2, \dots, n$*. Let* $f_{\alpha}(x, z)$ *be the resulting polynomial.*

We can compute $\boldsymbol{\alpha}$ *such that* $\frac{1}{f_d(\alpha)} f_{\alpha}(x, z)$ *is monic in* x *in time*

$$n d\, T_{\mathrm{PIT}(\mathrm{Hom}[\mathcal{C}])} + \mathrm{poly}(snd)\,.$$

Proof. Let $f(z) \in \mathcal{C}$ be a polynomial of degree d with n variables $z = (z_1, \dots, z_n)$.

To see what the transformation does, let

$$f = f_0 + f_1 + \cdots + f_d,$$

where $f_k = \mathrm{Hom}_k[f]$, the homogeneous degree-k component of f. Consider the degree-d component,

$$f_d(\boldsymbol{z}) = \sum_{|\beta|_1 = d} c_\beta \boldsymbol{z}^\beta.$$

Then, for f_α, we have $\deg_x(f_\alpha) = d$ and the coefficient of the leading x-term x^d in f_α is $f_d(\boldsymbol{\alpha}) = \sum_{|\beta|_1 = d} c_\beta \boldsymbol{\alpha}^\beta$.

Hence, the PIT algorithm for the homogeneous component f_d of f yields an $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})^n$ such that $f_d(\boldsymbol{\alpha}) \neq 0$, by Lemma 2.1. Then the polynomial $\frac{1}{f_d(\boldsymbol{\alpha})} f_\alpha(x, \boldsymbol{z})$ is monic in x. □

For simplicity of notation, assume in the following that $f_d(\boldsymbol{\alpha}) = 1$, so that $f_\alpha(x, \boldsymbol{z})$ is monic in x.

Since we work with the shifted polynomial, we need to ensure that the shift of variables does not affect the irreducibility of the factors; this is guaranteed by the following lemma. It is quite standard in the literature; for a nice proof, see [KRSV24, Lemma B7].

Lemma 2.3. *Let $f(\boldsymbol{z}) \in \mathbb{F}[\boldsymbol{z}]$ be an n-variate irreducible polynomial. Then, for every $\boldsymbol{a} \in \mathbb{F}^n$, the polynomial $f(\boldsymbol{a}x + \boldsymbol{z})$ is also irreducible.*

2.3 Basics of factoring and interpolation and PIT

Berlekamp [Ber70] and Lenstra, Lenstra and Lovász [LLL82] gave efficient factorization algorithms for *univariate* polynomials over finite fields and $\mathbb{Q}$, respectively. Kaltofen [Kal85c] showed how to reduce the factorization of *bivariate* polynomials to univariate polynomials. In fact, the reduction works for k-variate polynomials, for any constant k. In our case, we use it for the case $k = 3$.

Via standard interpolation, one can assume that the input is given as a dense representation.

Lemma 2.4 (Trivariate Factorization). *Let $f(x, y, z)$ be a trivariate polynomial of degree d. Then there exists an algorithm that outputs all its irreducible factors and their multiplicities in time* $\mathrm{poly}(d)$.

The following lemma shows how to find the multiplicity of an irreducible factor g of a polynomial f. It holds when $\mathrm{char}(\mathbb{F}) = 0$, or, large. For a concise proof, see [KRS24, Lemma 4.1].

Lemma 2.5 (Factor multiplicity). *Let $f(\boldsymbol{z}), g(\boldsymbol{z}) \in \mathbb{F}[\boldsymbol{z}]$ be non-zero polynomials and let $z \in \{z_1, \cdots, z_n\}$ be such that $\partial_z(g) \neq 0$ and g is irreducible. Then the multiplicity of g in f is the smallest non-negative integer e such that $g \nmid \frac{\partial^e f}{\partial z^e}$.*

Klivans and Spielman [KS01] derandomized the isolation lemma for PIT of sparse polynomials. Their algorithm works over fields of 0 or large characteristics.

Theorem 2.6 (Sparse PIT and interpolation [KS01]). *Let $\mathcal{C}_{sp} = \mathcal{C}_{sp}(s, n, d)$. Then* $\mathrm{PIT}(\mathcal{C}_{sp})$ *can be solved in time*

$$T_{\mathrm{PIT}(\mathcal{C}_{sp})} = \mathrm{poly}(snd).$$

Furthermore, given $f \in \mathcal{C}_{sp}$, in time $\mathrm{poly}(snd)$ *one can compute a set of* evaluation points *$\mathcal{E} \subseteq \mathbb{F}^n$ of size* $\mathrm{poly}(snd)$ *such that given the evaluations of* f *at all points in $\mathcal{E}$, one can solve for the coefficients of* f *in time* $\mathrm{poly}(snd)$.

Limaye, Srinivasan, and Tavenas [LST21] designed a deterministic subexponential-time PIT for constant-depth circuits.

Theorem 2.7 (PIT for constant depth circuits [LST21, Corollary 6]). *Let $\epsilon > 0$ be a real number. Let $\mathcal{C}_{\text{Depth-t}} = \mathcal{C}_{\text{Depth-t}}(s, n, d)$ be such that $s \leq \text{poly}(n)$ and $t = o(\log\log\log n)$. Then $\text{PIT}(\mathcal{C}_{\text{Depth-t}})$ can be decided in time*

$$T_{\text{PIT}(\mathcal{C}_{\text{Depth-t}})} = \left(ns^{O(t)}\right)^{O((sd)^{\epsilon})}.$$

2.4 Divisibility testing reduces to PIT

Given a polynomial f to factor, our algorithms might compute a polynomial g that is a candidate for a factor, but in fact, is not a factor. Hence, we have to verify whether g is a factor of f, i.e., whether $g|f$. Therefore, we are interested in the complexity of division algorithms.

When f, g can be computed by circuits of size s, Strassen [Str73] showed that if $g|f$, then $h = f/g$ can be computed by a circuit of size $\text{poly}(sd)$, where $d = \deg(h)$. Forbes [For15] observed that even in the case when $g \not| f$, one can follow Strassen's argument and obtain a small size circuit that computes a polynomial $\tilde{h}$ such that $g|f \iff f = g\tilde{h}$. Hence, we have a reduction from divisibility testing to PIT.

Lemma 2.8 (Divisibility reduces to PIT [For15, Corollary 7.10]). *Let $g(\mathbf{z})$ and $f(\mathbf{z})$ be two polynomials of degree at most d. Let $S = [2d^2+1]$ and $\boldsymbol{\alpha} \in \mathbb{F}^n$ such that $g(\boldsymbol{\alpha}) \neq 0$. Then there are constants $\{c_{\beta,i}\}_{\beta\in S, 0\leq i\leq d}$, computable in time $\text{poly}(d)$, such that for*

$$\tilde{h}(\mathbf{z}) = \sum_{\beta\in S} f(\beta\mathbf{z}+\boldsymbol{\alpha}) \sum_{0\leq i\leq d} c_{\beta,i}\, g(\beta\mathbf{z}+\boldsymbol{\alpha})^i, \tag{2}$$

we have

$$g(\mathbf{z})|f(\mathbf{z}) \iff f(\mathbf{z}+\boldsymbol{\alpha}) = g(\mathbf{z}+\boldsymbol{\alpha})\,\tilde{h}(\mathbf{z}).$$

A consequence from Lemma 2.8 is that divisibility testing of a polynomial computed by constant-depth circuit by a sparse polynomial is in subexponential time.

Corollary 2.9 (Constant depth by sparse division). *Let $\mathcal{C}_{\text{Depth-t}} = \mathcal{C}_{\text{Depth-t}}(s, n, d)$ and $\mathcal{D}_{\text{sp}} = \mathcal{C}_{\text{sp}}(s, n, d)$. For any $\epsilon > 0$, we have that $\text{Div}(\mathcal{C}_{\text{Depth-t}}/\mathcal{D}_{\text{sp}})$ can be decided in time*

$$T_{\text{Div}(\mathcal{C}_{\text{Depth-t}}/\mathcal{D}_{\text{sp}})} = \left(n\,(sd)^{O(t)}\right)^{O((sd)^{\epsilon})}.$$

Proof. We apply Lemma 2.8 with $g \in \mathcal{C}_{\text{sp}}$ and $f \in \mathcal{C}_{\text{Depth-t}}$. Then $\tilde{h}$ in (2) can be computed by a circuit of size $O(sd^2)$ and depth $t+2$. Therefore, the polynomial

$$\tilde{f} = f(\mathbf{z}+\boldsymbol{\alpha}) - g(\mathbf{z}+\boldsymbol{\alpha}) \cdot \tilde{h}(\mathbf{z})$$

can be computed by a circuit of size $O(sd^2)$ and depth $t+4$. By Lemma 2.8, we have $\tilde{f} = 0$ iff $g|f$, and by Theorem 2.7, the identity can be checked in time $\left(n\,(sd)^{O(t)}\right)^{O((sd)^{\epsilon})}$. □

We also consider divisibility of sparse polynomials by *constant-degree* polynomials. Building on Lemma 2.8, Forbes [For15] reduced the problem to a PIT that can be solved in quasipolynomial time.

Corollary 2.10 (Sparse by constant degree division [For15, Corollary 7.17]). *Let $\mathcal{C}_{\mathrm{sp}} = \mathcal{C}_{\mathrm{sp}}(s, n, d)$ and $\mathcal{D}_\delta = \mathcal{P}(n, \delta)$. Then* $\mathrm{Div}(\mathcal{C}_{\mathrm{sp}}/\mathcal{D}_\delta)$ *can be decided in time*

$$T_{\mathrm{Div}(\mathcal{C}_{\mathrm{sp}}/\mathcal{D}_\delta)} = (snd)^{O(\delta \log s)}.$$

2.5 Effective Hilbert's Irreducibility Theorem

Factorization algorithms often start with an effective version of Hilbert's Irreducibility Theorem due to Kaltofen and von zur Gathen. It shows how to project a multivariate irreducible polynomial down to two variables, such that the projected bivariate polynomial stays irreducible. The proof shows the existence of an *irreducibility certifying polynomial* $G(\mathbf{b}, \mathbf{c})$ in $2n$ variables corresponding to the irreducible polynomial $g(x, \boldsymbol{z})$. The nonzeroness of G proves the irreducibility of $g(x, \boldsymbol{z})$ and also gives a way to find an irreducibility-preserving projection to bivariate (see [Kal85b, Kal95, KSS15]).

Theorem 2.11. *Let $g(x, \boldsymbol{z})$ be an irreducible polynomial of total degree δ with $n+1$ variables that is monic in x. There exists a nonzero polynomial $G(\mathbf{b}, \mathbf{c})$ of degree $2\delta^5$ in $2n$ variables such that for $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{F}^n$,*

$$G(\boldsymbol{\beta}, \boldsymbol{\gamma}) \neq 0 \implies \widehat{g}(x, t) = g(x, \boldsymbol{\beta} t + \boldsymbol{\gamma}) \text{ is irreducible,} \tag{3}$$

where $g(x, \boldsymbol{\beta} t + \boldsymbol{\gamma}) = g(x, \beta_1 t + \gamma_1, \ldots, \beta_n t + \gamma_n)$.

The certifying polynomial G immediately yields a randomized algorithm to construct the irreducible projection $\widehat{g}$ via PIT. The derandomization of Hilbert's Irreducibility Theorem is a challenging open problem in general. Essentially it means to find a hitting set for G.

We define a corresponding computational problem. Let $\mathcal{C} \subseteq \mathcal{P}(n, d)$ be a class of polynomials and $g(\boldsymbol{z}) \in \mathcal{C}$. Assume we have already computed an $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})$ as in Lemma 2.2 that the shifted polynomial $g_{\boldsymbol{\alpha}}(x, \boldsymbol{z})$ is monic. Now we want to find a hitting set according to (3).

- Irred-Proj($\mathcal{C}$):
  Given $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})^n$, compute a set $H_{\boldsymbol{\alpha}} \subseteq \mathbb{F}^{2n}$ such that for all $g \in \mathcal{C}$ where $g(\boldsymbol{\alpha} x + \boldsymbol{z})$ is monic in x, we have

$$g \text{ irreducible} \implies \exists (\boldsymbol{\beta}, \boldsymbol{\gamma}) \in H_{\boldsymbol{\alpha}} \quad g(\boldsymbol{\alpha} x + \boldsymbol{\beta} t + \boldsymbol{\gamma}) \in \mathbb{F}[x, t] \text{ is irreducible.}$$

By $T_{\text{Irred-Proj}(\mathcal{C})}$ we denote the time complexity to compute Irred-Proj($\mathcal{C}$).

2.6 Isolation

Let M_δ be the set of monomials in n variables $\boldsymbol{z} = (z_1, z_2, \ldots, z_n)$ of degree bounded by δ,

$$M_\delta = \{\, \boldsymbol{z}^{\boldsymbol{e}} \mid \|\boldsymbol{e}\|_1 \leq \delta \,\}.$$

Note that M_δ is polynomially bounded, for constant δ,

$$|M_\delta| \leq \binom{n+\delta}{\delta} \leq (n+\delta)^\delta \leq (\delta+1)\, n^\delta = O(n^\delta). \tag{4}$$

There is a standard way to map the multivariate monomials in M_δ in a injective way to univariate monomials of polynomial degree. For completeness, we describe the details.

Consider the standard *Kronecker substitution* on M_δ. Define

$$\varphi : \; z_i \;\mapsto\; y^{(\delta+1)^{i-1}} .$$

By extending φ linearly to monomials $\boldsymbol{z}^{\boldsymbol{e}} \in M_\delta$, we get

$$\varphi : \; \boldsymbol{z}^{\boldsymbol{e}} \;\mapsto\; y^{\sum_{i=1}^{n} e_i (\delta+1)^{i-1}} ,$$

Clearly, φ is injective on M_δ. However, the degree of y can be exponentially large, up to $(\delta+1)^n$. A way around is to take the exponents modulo some small prime number p. We have to determine p in a way to keep the mapping injective on M_δ. Hence, for any two terms $y^e, y^{e'}$ we get from φ, we have to ensure that $e \not\equiv e' \pmod{p}$. Equivalently $p \not| (e - e')$.

We have $|e - e'| \leq (\delta+1)^n$ and, by (4), there are $(\delta+1)^2 n^{2\delta}$ many pairs e, e' we get from M_δ via φ. Prime p should not divide any of these differences, and hence, p should not divide their product P. The product P is bounded by

$$P \leq ((\delta+1)^n)^{(\delta+1)^2 n^{2\delta}} = (\delta+1)^{(\delta+1)^2 n^{2\delta+1}} .$$

Hence, P has at most $\log P \leq R = (\delta+1)^3\, n^{2\delta+1}$ many prime factors. By the Prime Number Theorem, there are more than $\log P$ primes in the set $[R^2]$. Hence, we can find an appropriate prime $p \leq R^2 = n^{O(\delta)}$.

Lemma 2.12. *There is a prime $p = n^{O(\delta)}$ such that the linear extension of*

$$\varphi_p : \; z_i \;\mapsto\; y^{w_i} , \quad \text{where } w_i = (\delta+1)^{i-1} \bmod p , \quad \text{for } i = 1, 2, \ldots, n,$$

to monomials is injective on M_δ. Moreover, we can find such a p in time $n^{O(\delta)}$ and compute and invert φ_p in time $n^{O(\delta)}$.

Proof. We already argued about the existence of prime p. For the running time, recall that $|M_\delta| = O(n^\delta)$. Therefore we can search for p and check whether it works on M_δ in time $n^{O(\delta)}$. At the same time we can compute pairs of exponents $(\boldsymbol{e}, k)$ such that $\varphi_p(\boldsymbol{z}^{\boldsymbol{e}}) = y^k$. These pairs can be used to invert φ_p. □

The mapping φ_p in Lemma 2.12 maintains factors of degree δ of a polynomial in the following sense.

Lemma 2.13. *Let polynomial $f(\boldsymbol{z})$ factor as $f = gh$, where $g(\boldsymbol{z})$ has degree δ. Let φ_p be the map from Lemma 2.12. Then we have $\varphi_p(f) = \varphi_p(g)\varphi_p(h)$, and g can be recovered from $\varphi_p(g)$ in time $n^{O(\delta)}$.*

Note that in Lemma 2.13, we do *not* claim that irreducibility is maintained: when g is irreducible, still $\varphi_p(g)$ might be reducible. Consider the example $n = \delta = 2$. The weights $\{1, 3\}$ make sure that each monomial z_1^2, z_1z_2, z_2^2 gets mapped to a distinct power in y. Let $g(x, \boldsymbol{z}) = x^2 - z_1z_2$. Observe that g is irreducible, however $g(x, y, y^3) = (x - y^2)(x + y^2)$ is *reducible*.

We combine Lemma 2.12 and Theorem 2.11 to obtain a projection of a multivariate polynomial to a 3-variate polynomial that maintains irreducibility of polynomials up to degree δ.

Corollary 2.14. *Let $g(x, \boldsymbol{z})$ be an irreducible polynomial of constant degree δ with $n + 1$ variables that is monic in x. There exists $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{F}^n$ with $w_i, w_i' = n^{\mathrm{poly}(\delta)}$ such that*

$$\Psi(g) = g(x, y^{w_1}t + y^{w_1'}, \ldots, y^{w_n}t + y^{w_n'}) \in \mathbb{F}[x, y, t]$$

is irreducible. Moreover, we can compute and invert $\Psi(g)$ in time $n^{\mathrm{poly}(\delta)}$.

Proof. Let $G(\boldsymbol{a}, \boldsymbol{b})$ be the polynomial of degree $2\delta^5$ in $2n$ variables provided by Theorem 2.11 for g. Let $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{F}^n$ with $w_i, w_i' = n^{\mathrm{poly}(\delta)}$ be the exponents we get from Lemma 2.12 for G. That is,

$$\widehat{G}(y) = G(y^{w_1}, \ldots, y^{w_n}, y^{w_1'}, \ldots, y^{w_n'}) \neq 0\,.$$

Now, suppose that $\Psi(g)$ is reducible. Then it would also be reducible at a point $y = \alpha$, where $\widehat{G}(\alpha) \neq 0$. But then $\widehat{g}(x, t) = \Psi(g)(x, \alpha, t)$ would be reducible too, and this would contradict Theorem 2.11. We conclude that $\Psi(g)$ is irreducible.

For the complexity, we first determine prime p from Lemma 2.12 and then get the weights $\boldsymbol{w}, \boldsymbol{w}'$ from above. For a given $g(x, \boldsymbol{z}) = \sum_{k,\boldsymbol{e}} c_{k,\boldsymbol{e}} x^k \boldsymbol{z}^{\boldsymbol{e}}$, we can compute $\Psi(g)$ in time $n^{\mathrm{poly}(\delta)}$. For a monomial of g, the mapping looks as follows:

$$c_{k,\boldsymbol{e}}\, x^k \boldsymbol{z}^{\boldsymbol{e}} \;\mapsto\; c_{k,\boldsymbol{e}}\, x^k \prod_{i=1}^{n} (y^{w_i}t + y^{w_i'})^{e_i}\,. \tag{5}$$

To compute g from $\Psi(g)$, set $t = 0$, i.e. consider $\Psi(g)(x, y, 0)$. From (5) we see that monomials then have the form

$$c_{k,\boldsymbol{e}}\, x^k\, y^{\sum_{i=1}^{n} e_i w_i'}\,.$$

From these we get the exponents k and $\boldsymbol{e}$ similar as in the proof of Lemma 2.12. □

Remark. In Corollary 2.14, when we say that we *invert* Ψ, it means that for a given $h \in \mathbb{F}[x, y, t]$ which is monic in x with x-degree $\leq \delta$, we either detect that h is not in the codomain of Ψ, or we compute $g \in \mathbb{F}[x, \boldsymbol{z}]$ such that $\Psi(g) = h$ in time $n^{\mathrm{poly}(\delta)}$.

The inversion can be done similarly as described in the proof of Corollary 2.14. One can evaluate $t = 0$, and then for every monomial $x^k y^j$, try to find $x^k \boldsymbol{z}^{\boldsymbol{e}}$ that would map to such a monomial at $t = 0$. By the property of the map, while mapping the y-degrees, $\boldsymbol{z}$-degree could be at most δ, i.e. $\deg(x^k \boldsymbol{z}^{\boldsymbol{e}}) \leq 2\delta$. We will, of course, return empty if the degree of any such monomial, after inverting, becomes $> \delta$. Finally, once we have got a candidate g of degree δ, we still have to check whether $\Psi(g) = h$, because the inversion procedure ignores the variable t. The last step can also be efficiently checked.

The polynomial g of degree δ we considered so far can be thought to be a constant-degree factor of a given polynomial f of degree d. Our goal would be to compute g. It is now easy to extend the above results to hold for all degree-δ factors of f simultaneously.

Corollary 2.15. *Let $f(x, \boldsymbol{z})$ be a polynomial of degree d with $n+1$ variables that is monic in x, and let δ be a constant. There exists $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{F}^n$ with $w_i, w_i' \leq dn^{\mathrm{poly}(\delta)}$ such that for any irreducible factor g of degree δ of f, we have that $\Psi(g)$ is an irreducible factor of $\Psi(f)$.*

Proof. The proof goes along the lines of Corollary 2.14, but we choose the weights slightly larger so that the $\widehat{G}(y)$ polynomials for *all* the degree-δ factors g of f are non-zero simultaneously. That is, we choose prime p in Lemma 2.12 as $p = dn^{\mathrm{poly}(\delta)}$. □

Finally, we conclude this subsection by a general remark that whenever n and δ are fixed, these weights are fixed and can be found efficiently.

3 Computing the low-degree factors

We show how to compute the factors of constant degree of a given polynomial f. In Section 3.2 for the case when *all* factors of f have constant degree, and in Section 3.3 for general f. In both cases, our algorithm starts by projecting the given polynomial to a 3-variate polynomial. We start with this common part.

3.1 Projected constant degree factors

The following algorithm is an initiating step in both, Algorithm 2 and 3 in Sections 3.2 and 3.3. It takes an n-variate polynomial $f(\boldsymbol{z})$ of degree d. The final goal is to compute the factors of f of degree δ, for some given constant δ. The initial steps are to project f to a trivariate monic polynomial and then factorize the projection.

In more detail, the first step is to make $f(\boldsymbol{z})$ monic in a new variable x via Lemma 2.2. That is, we compute $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})^n$ such that the transformed polynomial $f_{\boldsymbol{\alpha}}(x, \boldsymbol{z}) = f(\boldsymbol{\alpha} x + \boldsymbol{z})$ is monic.

Then we apply Corollary 2.15 to $f_{\boldsymbol{\alpha}}(x, \boldsymbol{z})$. That is, we compute the weights $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{F}^n$ bounded by $dn^{\mathrm{poly}(\delta)}$ and explicitly compute $\Psi(f_{\boldsymbol{\alpha}}) \in \mathbb{F}[x, y, t]$ of degree at most $\tilde{d} = d^2 n^{\mathrm{poly}(\delta)}$. Note that the x-degree of $f_{\boldsymbol{\alpha}}$ has not changed by mapping Ψ.

The next step is to factor 3-variate $\Psi(f_{\boldsymbol{\alpha}})$ via Lemma 2.4. The following pseudo-code summarizes the steps.

For the running time of Projected-Factoring we have $nd\, T_{\mathrm{PIT}(\mathrm{Hom}[\mathcal{C}])}$ for finding $\boldsymbol{\alpha}$ in step 1 by Lemma 2.2. Steps 2 and 3 take time $\mathrm{poly}(d\, n^{\mathrm{poly}(\delta)})$. In summary, the time complexity of Projected-Factoring is

$$nd\, T_{\mathrm{PIT}(\mathrm{Hom}[\mathcal{C}])} + \mathrm{poly}(d\, n^{\mathrm{poly}(\delta)})\,.$$

3.2 Factors of a constant degree product

Given a polynomial that is the product of constant degree polynomials. We show that all the factors can be computed in polynomial time.

```
Algorithm 1: PROJECTED-FACTORING
  Input  : $f(z) \in \mathcal{C}_\mu(s,n,d)$ and a constant $\delta$.
  Output: All projected factors of $f$ of degree $\delta$ with their multiplicities.
  Find $\alpha$ such that $f_\alpha(x,z) = f(\alpha x + z)$ is monic    /* by Lemma 2.2 */
  Find $w, w' \in \mathbb{F}^n$ as in Corollary 2.15 and compute $\Psi(f_\alpha) \in \mathbb{F}[x,y]$ in dense representation
  Factorize $\Psi(f_\alpha) = h_1^{e_1} h_2^{e_2} \cdots h_m^{e_m}$    /* by Lemma 2.4, in dense representation */
  Define $S_{\text{Proj-Fac}} = \{ h_i \mid \deg_x(h_i) \leq \delta \}$ and $S_{\text{Proj-Fac-mult}} = \{ (h_i, e_i) \mid h_i \in S_{\text{Proj-Fac}} \}$
  return $S_{\text{Proj-Fac}}, S_{\text{Proj-Fac-mult}}$
```

Theorem 3.1. *For a constant δ, let $\mathcal{D}_\delta = \mathcal{P}(n,\delta)$ and $\mathcal{C}_d \subseteq \mathcal{P}(n,d)$ be the class of polynomials that are a product of polynomials from $\mathcal{D}_\delta$, i.e. $\mathcal{C}_d = \prod \mathcal{D}_\delta$. Then* $\text{Factor}(\mathcal{C}_d|_{\mathcal{D}_\delta})$ *can be solved in time* $T_{\text{Factor}(\mathcal{C}_d|_{\mathcal{D}_\delta})} = \text{poly}(dn^{\text{poly}(\delta)})$.

Proof. We start by invoking PROJECTED-FACTORING for f to get S', the set of factors of the transformed trivariate polynomial $\Psi(\tau_\alpha(f))$ with their multiplicities. Then we first invert the transformation Ψ on the factors using Corollary 2.14 and its remark. Then we invert τ_α. Since f and $\Psi(\tau_\alpha(f))$ have the same factorization pattern, we finally get the factors of f. We summarize the steps in Algorithm 2.

```
Algorithm 2: CONSTANT-DEGREE-FACTORIZATION
  Input  : $f(z) \in \mathcal{C}_\mu(s,n,d)$ and a constant $\delta$ such that
           all irreducible factors of $f$ have degree $\leq \delta$.
  Output: All irreducible factors of $f$ with their multiplicities.
  $L = \emptyset$    /* Initialize output list */
  $(S_{\text{Proj-Fac}}, S_{\text{Proj-Fac-mult}}) = $ PROJECTED-FACTORING$(f, \delta)$
  for $(\tilde{g}, e) \in S_{\text{Proj-Fac-mult}}$ do
      /* Computing the irreducible factors via inversion */
      Compute $\hat{g} = \Psi^{-1}(\tilde{g})$, by Corollary 2.14 and its remark
      Compute $g = \tau_\alpha^{-1}(\hat{g})$, and add $(g, e)$ to $L$
  return $L$
```

For the time complexity of CONSTANT-DEGREE-FACTORIZATION, recall that constant degree polynomials are sparse, with sparsity $s = O(n^\delta)$ by (4). Hence, we get $T_{\text{PIT}(\text{Hom}[\mathcal{C}])} = \text{poly}(dn^\delta)$ by Theorem 2.6. The **for**-loop with the inverse mappings takes time $\text{poly}(dn^{\text{poly}(\delta)})$. □

Note that Theorem 3.1 solves a *promise case*. That is, we assume that f is a product of constant degree polynomials, but we do not verify this assumption. So in case that the assumption does not hold, one can anyway run the algorithm from Theorem 3.1, but then it might output polynomials of degree δ that are not factors of f. In Section 3.3, we show how to compute the constant degree factors of an arbitrary polynomial.

3.3 Constant degree factors

In Section 3.2, we computed the constant degree factors of a polynomial f under the assumption that *all* factors of f are of constant degree. Now we skip the assumption and let f be a polynomial from some class $\mathcal{C}_\mu = \mathcal{C}_\mu(s, n, d) \subseteq \mathcal{P}(n, d)$. We still want to compute the constant degree factors of f from $\mathcal{D}_\delta = \mathcal{P}(n, \delta)$, for some constant δ. We show that the problem can be reduced in polynomial time to a PIT for $\mathrm{Hom}[\mathcal{C}_\mu]$ and divisibility tests $\mathcal{C}_\mu$ by $\mathcal{D}_\delta$.

Theorem 3.2. *Let $\mathcal{C}_\mu = \mathcal{C}_\mu(s, n, d)$ and $\mathcal{D}_\delta = \mathcal{P}(n, \delta)$, for a constant δ. Then* $\mathsf{Factor}(\mathcal{C}_\mu|_{\mathcal{D}_\delta})$ *can be solved in time*

$$T_{\mathsf{Factor}(\mathcal{C}_\mu|_{\mathcal{D}_\delta})} = nd\, T_{\mathsf{PIT}(\mathsf{Hom}[\mathcal{C}_\mu])} + d^2 n^{\mathrm{poly}(\delta)}\, T_{\mathsf{Div}(\partial\mathcal{C}_\mu/\mathcal{D}_\delta)} + \mathrm{poly}(s, n^{\mathrm{poly}(\delta)}, d).$$

Proof. Let $f(z) \in \mathcal{C}$. To compute the factors of degree δ of f, we start again by invoking Projected-Factoring for f to get S, the set of factors of the transformed trivariate polynomial $\Psi(\tau_\alpha(f))$. As in the proof of Theorem 3.1, we compute the inverses of the transformations, $g = \tau_\alpha^{-1}(\Psi^{-1}(\tilde{g}))$.

However $\tilde{g}$ might also *not* correspond to a degree-δ factor of f. In this case, either the inverse transformation does not go through properly, or the degree we get is larger than δ. In these cases, we can immediately throw away $\tilde{g}$; see the remark after Corollary 2.14. But it could also be that we actually obtain a polynomial g of degree δ, just that it is not a factor of f. For that reason, we finally do a divisibility check whether $g|f$. That way we will compute all factors of f of degree δ. The multiplicities of the factors we compute via Lemma 2.5. We summarize the steps in Algorithm 3.

```
Algorithm 3: Constant-Degree-Factors
  Input  : $f(z) \in \mathcal{C}_\mu(s, n, d)$ and a constant $\delta$.
  Output: All irreducible factors of $f$ of degree $\leq \delta$ with their multiplicities.
$L = \emptyset$, $L' = \emptyset$    /* Initialize output list and intermediate candidates list */
$(S_{\text{Proj-Fac}}, S_{\text{Proj-Fac-mult}})$ = Projected-Factoring$(f, \delta)$  /* Compute projected
    3-variate factors                                                                   */
for $\tilde{g} \in S_{\text{Proj-Fac}}$ do
    /* Computing candidate factors via divisibility                                     */
    Compute $\hat{g} = \Psi^{-1}(\tilde{g})$ (if the inverse exists) of degree $\leq \delta$ by Corollary 2.14
    Compute $g = \tau_\alpha^{-1}(\hat{g})$
    If $g|f$ then add $g$ to $L'$
for $g \in L'$ do
    /* Computing multiplicities via Lemma 2.5                                           */
    Let $z$ be a variable that $g$ depends on
    Find the smallest $e \geq 1$ such that $g \nmid \frac{\partial^e f}{\partial z^e}$ and add $(g, e)$ to list $L$
return $L$
```

For the time complexity of the factoring algorithm, we have $nd\, T_{\mathsf{PIT}(\mathsf{Hom}[\mathcal{C}])}$ for transforming f to monic f_α by Lemma 2.2. Time $\mathrm{poly}(d\, n^{\mathrm{poly}(\delta)})$ is used for map Ψ and the factoring of $\Psi(f_\alpha)$.

Similar time is taken to invert and get the candidate factors. Finally, we have at most $d^2 n^{\mathrm{poly}(\delta)}$ candidate polynomials g for which we test divisibility of $g|f$ in Line 6. For the multiplicities, we first find a variable z that g depends on in Line 8. Then we have at most d divisibility tests whether $g| \frac{\partial^e f}{\partial z^e}$ in Line 9. □

As a consequence, we get a quasipolynomial-time algorithm to compute the constant-degree factors of a sparse polynomial. With a different proof, this was already shown by Kumar, Ramanathan, and Saptharishi [KRS24].

Corollary 3.3 (Constant-degree factors of sparse [KRS24]). *Let* $\mathcal{C}_{\mathrm{sp}} = \mathcal{C}_{\mathrm{sp}}(s, n, d)$ *and* $\mathcal{D}_\delta = \mathcal{P}(n, \delta)$*, for a constant* δ*. Then* $\mathsf{Factor}(\mathcal{C}_{\mathrm{sp}}|_{\mathcal{D}_\delta})$ *can be solved in time*

$$T_{\mathsf{Factor}(\mathcal{C}_{\mathrm{sp}}|_{\mathcal{D}_\delta})} = (snd)^{\mathrm{poly}(\delta)\log s}.$$

Proof. Homogeneous components as well as the derivatives of a sparse polynomial remain sparse. Hence, we have $T_{\mathsf{PIT}(\mathrm{Hom}[\mathcal{C}_{\mathrm{sp}}])} = \mathrm{poly}(snd)$ by Theorem 2.6, and $T_{\mathsf{Div}(\partial\mathcal{C}_{\mathrm{sp}}/\mathcal{D}_\delta)} = (snd)^{O(\delta \log s)}$ by Corollary 2.10. Hence, we get the desired complexity by Theorem 3.2. □

Kumar, Ramanathan, and Saptharishi [KRS24] generalized sparse polynomials to polynomials computed by constant-depth circuits and showed that the constant-degree factors can be computed in subexponential time. We can now also derive this result via Theorem 3.2.

Recall that

$$\mathcal{C}_{\text{Depth-t}}(s, n, d) = \{p \in \mathcal{P}(n, d) \mid p \text{ has a circuit of size } s \text{ and depth } t\}.$$

We state some properties of $\mathcal{C}_{\text{Depth-t}}$.

(i) $\mathcal{C}_{\mathrm{sp}}(s, n, d) \subseteq \mathcal{C}_{\text{Depth-2}}(s + 1, n, d)$.

(ii) $\mathcal{C}_{\text{Depth-t}}$ is closed for homogeneous components: For a polynomial $f \in \mathcal{C}_{\text{Depth-t}}(s, n, d)$, all homogeneous components of f are in $\mathcal{C}_{\text{Depth-(t+1)}}(sd, n, d)$ (see [Oli16, Lemma 2.3]).

(iii) $\mathcal{C}_{\text{Depth-t}}$ is closed under derivatives: For a polynomial $f \in \mathcal{C}_{\text{Depth-t}}(s, n, d)$, a variable z and $e \geq 1$, we have $\frac{\partial^e f}{\partial z^e} \in \mathcal{C}_{\text{Depth-(t+1)}}(d^2 s, n, d)$ (see [Oli16, Lemma 2.5]).

We apply Theorem 3.2 to compute constant-degree factors of constant-depth circuits.

Corollary 3.4 (Constant-degree factors of constant-depth [KRS24]). *Let* $\mathcal{C}_{\text{Depth-t}} = \mathcal{C}_{\text{Depth-t}}(s, n, d)$ *and* $\mathcal{D}_\delta = \mathcal{P}(n, \delta)$*, for a constant* δ*. Then, for any* $\varepsilon > 0$*,* $\mathsf{Factor}(\mathcal{C}_{\text{Depth-t}}|_{\mathcal{D}_\delta})$ *can be solved in time*

$$T_{\mathsf{Factor}(\mathcal{C}_{\text{Depth-t}}|_{\mathcal{D}_\delta})} = \left(n\,(sd)^{O(t)}\right)^{O((sd)^\varepsilon)}.$$

Proof. Let $f \in \mathcal{C}_{\text{Depth-t}}$. We consider the running times from Theorem 4.2 to factor f. We argue that

$$T_{\mathsf{PIT}(\mathrm{Hom}[\mathcal{C}_{\text{Depth-t}}])} = T_{\mathsf{Div}(\partial\mathcal{C}_{\text{Depth-t}}/\mathcal{D}_\delta)} = \left(n\,(sd)^{O(t)}\right)^{O((sd)^\varepsilon)}.$$

For the homogeneous components of f, this follows from property (ii) above that the components have again bounded-depth circuits. Hence, for PIT, we can apply Theorem 2.7.

For the divisibility test, this follows from property (i) and (iii) above and Corollary 2.9. □

3.4 Linear factors of ROABPs

We can apply Theorem 3.2 in the case of *read-once oblivious arithmetic branching programs*, ROABPs, and compute *linear* factors. ROABPs are arithmetic branching programs (ABPs) where there is an order on the variables and on every path of the ABP, every variable is evaluated in this order and only once. As size measure for ROABPs, usually the width w is taken. Its size as a graph is then bounded by nw, where n is the number of variables. Width and size can greatly vary depending on the variable order. We consider the *any-order* model. That is, when we say that a polynomial has an ROABP of width w, it is a bound for all orders of the variables. For a more detailed definition see for example [GKS17, GKST17]. Let

$$\mathcal{C}_{\mathrm{ROABP}}(w, n, d) = \{ p \in \mathcal{P}(n, d) \mid p \text{ has an any-order ROABP of width } w \}.$$

ROABPs generalize sparse polynomials: $\mathcal{C}_{\mathrm{sp}}(s, n, d) \subseteq \mathcal{C}_{\mathrm{ROABP}}(s, n, d)$.

For our arguments, we need some folklore properties of ROABPs.

Lemma 3.5 (Properties of ROABP). *Let* $f \in \mathcal{C}_{\mathrm{ROABP}}(w_1, n, d)$ *and* $g \in \mathcal{C}_{\mathrm{ROABP}}(w_2, n, d)$. *Then*

(i) $f + g \in \mathcal{C}_{\mathrm{ROABP}}(w_1 + w_2, n, d)$,

(ii) $fg \in \mathcal{C}_{\mathrm{ROABP}}(w_1 w_2, n, 2d)$.

(iii) $\mathrm{Hom}_d[f] \in \mathcal{C}_{\mathrm{ROABP}}((d+1)\, w_1, n, d)$.

(iv) $\partial_{z^e}(f) \in \mathcal{C}_{\mathrm{ROABP}}(dw_1, n, d-e)$, *for a variable* z *and* $e \le d$.

(v) $(a_0 + a_1 z_1 + a_2 z_2 + \cdots + a_n z_n)^d \in \mathcal{C}_{\mathrm{ROABP}}(d+1, n, d)$, *for* $a_0, a_1, a_2, \ldots, a_n \in \mathbb{F}$.

There are efficient PITs for ROABPs in the literature.

Lemma 3.6. *Let* $\mathcal{C}_{\mathrm{ROABP}} = \mathcal{C}_{\mathrm{ROABP}}(w, n, d)$. *Then* $\mathrm{PIT}_{\mathcal{C}_{\mathrm{ROABP}}}$ *can be solved in*

- *polynomial time* $\mathrm{poly}(ndw)$, *in the whitebox setting* [RS05],
- *quasipolynomial time* $(ndw)^{O(\log\log w)}$, *in the blackbox setting* [GKS17, GG20].

For factorization of a polynomial f computed by an ROABP, we apply Theorem 3.2. The time for PIT of homogeneous components of f follows from Lemma 3.5 (iii) and Lemma 3.6.

For divisions, we consider *linear* polynomials. This is the case $\delta = 1$ in the above terminology. That is, we consider class $\mathcal{D}_1$. We use Lemma 2.8 to reduce the divisibility test to a PIT instance of a polynomial-size ROABP, which can be solved by Lemma 3.6. By Lemma 3.5 (iv), this holds similarly for the partial derivatives.

Lemma 3.7 (ROABP by linear division). *Let* $\mathcal{C}_{\mathrm{ROABP}} = \mathcal{C}_{\mathrm{ROABP}}(w, n, d)$ *and* $\mathcal{D}_1 = \mathcal{P}(n, 1)$. *Then* $\mathrm{Div}(\mathcal{C}_{\mathrm{ROABP}}/\mathcal{D}_1)$ *can be solved in time*

- $\mathrm{poly}(ndw)$, *in the whitebox setting,*
- $(ndw)^{O(\log\log dw)}$, *in the black-box setting.*

Proof. Let $f(z) \in \mathcal{C}_{\mathrm{ROABP}}(w,n,d)$ and $\ell(z)$ be linear. We apply Lemma 2.8. Let $S = [2d^2+1]$ and find an $\alpha \in \mathbb{F}^n$ such that $\ell(\alpha) \neq 0$. Such an α can be found in time $O(n)$. Let $c_{\beta,i} \in \mathbb{F}$ be constants such that $\ell | f \iff \widehat{f} = f(z+\alpha) - \ell(z+\alpha)\,\widetilde{h} = 0$, where

$$\widetilde{h} = \sum_{\beta \in S} f(\beta z + \alpha) \sum_{0 \le i \le d} c_{\beta,i}\ell(\beta z + \alpha)^i \,.$$

We consider the width of an ROABP that computes $\widehat{f}$. The terms $\ell(\beta z + \alpha)^i$ in $\widetilde{h}$ have ROABPs of width $i+1 \le d+1$ by Lemma 3.5 (v). Applying Lemma 3.5 (i) and (ii), we get that $\widetilde{h}$ has an ROABP of width $O(wd^4)$. Hence, for $\widehat{f}$, we get an ROABP of width $O(w^2d^4)$. Now the claim follows from Lemma 3.6. □

We plug the time bounds from Lemma 3.6 and 3.7 in Theorem 3.2.

Corollary 3.8 (Linear factors of ROABPs). *Let $\mathcal{C}_{\mathrm{ROABP}} = \mathcal{C}_{\mathrm{ROABP}}(w,n,d)$ and $\mathcal{D}_1 = \mathcal{P}(n,1)$. Then* $\mathrm{Factor}(\mathcal{C}_{\mathrm{ROABP}}|_{\mathcal{D}_1})$ *can be solved in time*

- $\mathrm{poly}(ndw)$ *in the whitebox setting,*
- $(ndw)^{O(\log\log dw)}$ *in the blackbox setting.*

When the input f is s-sparse, this result is already known due to Volkovich [Vol15, Theorem 4].

4 Computing the sparse factors

Recall the class of sparse polynomials,

$$\mathcal{C}_{\mathrm{sp}}(s,n,d) = \{\, p \in \mathcal{P}(n,d) \mid \mathrm{sp}(p) \le s \,\}.$$

We want to compute the sparse factors of a given polynomial. For a polynomial f from a class $\mathcal{C}_\mu$, we show that the sparse factors of f can be computed efficiently relative to a PIT for $\mathrm{Hom}[\mathcal{C}_\mu]$, a divisibility test of f by sparse candidate factors, and an irreducibility preserving projection of sparse polynomials, $\mathrm{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})$ (see Section 2.5).

Lemma 4.1. *Given $f \in \mathcal{C}_{\mathrm{sp}}$, we can test whether f is irreducible in time* $\mathrm{poly}(snd)\,T_{\mathrm{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})}$.

Proof. The irreducibiltiy test first finds an $\alpha \in (\mathbb{F}\backslash\{0\})^n$ such that $\mathrm{Hom}_d[f](\alpha) \neq 0$ in time $\mathrm{poly}(snd)$ by Theorem 2.6. Then, by solving $\mathrm{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})$, one can find a set $\mathcal{H}_\alpha$ such that $f(z)$ is reducible if and only if $f(\alpha x + \beta t + \gamma)$ is reducible, for every $(\beta,\gamma) \in \mathcal{H}_\alpha$. Whether a bivariate polynomial is reducible can be checked in time $\mathrm{poly}(d)$ by Lemma 2.4. □

Theorem 4.2 (Sparse factors). *Let $\mathcal{C}_\mu = \mathcal{C}_\mu(s,n,d)$ and $\mathcal{C}_{\mathrm{sp}} = \mathcal{C}_{\mathrm{sp}}(s,n,d)$. Then* $\mathrm{Factor}(\mathcal{C}_\mu|_{\mathcal{C}_{\mathrm{sp}}})$ *can be solved in time*

$$T_{\mathrm{Factor}(\mathcal{C}_\mu|_{\mathcal{C}_{\mathrm{sp}}})} = nd\,T_{\mathrm{PIT}(\mathrm{Hom}[\mathcal{C}_\mu])} + \mathrm{poly}(snd)\,T_{\mathrm{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})}\,T_{\mathrm{Div}(\partial\mathcal{C}_\mu/\mathcal{C}_{\mathrm{sp}})}\,.$$

Proof. Let $f(z) \in \mathcal{C}_\mu(s, n, d)$ and let g be a s-sparse irreducible factor of f with multiplicity e, that is $f = g^e h$, where $\gcd(g, h) = 1$. Let $\deg(g) = d_g$ and $\deg(h) = d_h$.

The first step is to transform f to a monic polynomial by finding an $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})^n$ such that $\mathrm{Hom}_d[f](\boldsymbol{\alpha}) \neq 0$. Observe that $\mathrm{Hom}_d[f] = (\mathrm{Hom}_{d_g}[g])^e \, \mathrm{Hom}_{d_h}[h]$. Therefore, we also have $\mathrm{Hom}_{d_g}[g](\boldsymbol{\alpha}) \neq 0$.

The second step is to solve Irred-Proj$(\mathcal{C}_{\mathrm{sp}})$ and compute a set $\mathcal{H}_\alpha$ such that $g(\boldsymbol{\alpha} x + \boldsymbol{\beta} t + \boldsymbol{\gamma})$ remains irreducible for some $(\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathcal{H}_\alpha$. We call such a $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ a *good point* for g.

Since we do not know g for now, we also do not know which pair $(\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathcal{H}_\alpha$ is a good point for g. For that reason, we iteratively try all pairs $(\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathcal{H}_\alpha$ in the following. Our algorithm will detect when a pair is not good at some point and then ignore that pair. Consider a good point $(\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathcal{H}_\alpha$ and compute bivariate polynomial $\widehat{f}$,

$$\widehat{f}(x, t) = f(\boldsymbol{\alpha} x + \boldsymbol{\beta} t + \boldsymbol{\gamma})$$

in dense representation by interpolation. Then factor g is mapped to $\widehat{g}(x, t) = g(\boldsymbol{\alpha} x + \boldsymbol{\beta} t + \boldsymbol{\gamma})$, a factor of $\widehat{f}$. We factorize $\widehat{f}$ over $\mathbb{F}$ by Lemma 2.4. Let $F = \{\widehat{g}_1, \widehat{g}_2, \ldots, \widehat{g}_m\}$ be the set of irreducible factors of $\widehat{f}$. Note that for a good point $(\boldsymbol{\beta}, \boldsymbol{\gamma})$, we have $\widehat{g} \in F$.

Our task now is to find out which polynomials in F are coming from sparse factors of f and to recover these factors. Let $\mathcal{E} = \mathcal{E}(s, n, d) \subseteq \mathbb{F}^n$ be the set of evaluation points to interpolate s-sparse n-variate polynomials of degree d from Theorem 2.6. For any $\boldsymbol{\omega} \in \mathcal{E}$, we compute the trivariate polynomial $\widehat{f}_\omega$,

$$\widehat{f}_\omega(x, t_1, t_2) = f(\boldsymbol{\alpha} x + \boldsymbol{\beta} t_1 + (\boldsymbol{\omega} - \boldsymbol{\gamma}) t_2 + \boldsymbol{\gamma})$$

in dense representation by interpolation. Factor g is mapped correspondingly to $\widehat{g}_\omega(x, t_1, t_2) = g(\boldsymbol{\alpha} x + \boldsymbol{\beta} t_1 + (\boldsymbol{\omega} - \boldsymbol{\gamma}) t_2 + \boldsymbol{\gamma})$. Note that $\widehat{f}_\omega$ and $\widehat{g}_\omega$ are monic polynomial in x and $\widehat{g}_\omega$ remains irreducible, since $\widehat{g}_\omega(x, t, 0) = g(\boldsymbol{\alpha} x + \boldsymbol{\beta} t + \boldsymbol{\gamma})$ is irreducible, for a good point $(\boldsymbol{\beta}, \boldsymbol{\gamma})$. We factorize $\widehat{f}_\omega$ over $\mathbb{F}$ by Lemma 2.4. Let $F_\omega = \{\widehat{g}_{\omega,1}, \widehat{g}_{\omega,2}, \ldots, \widehat{g}_{\omega,r}\}$ be the set of irreducible factors of $\widehat{f}_\omega$. Then we have $\widehat{g}_\omega \in F_\omega$.

Note that $\widehat{f}$ and $\widehat{g}$ are the projections of $\widehat{f}_\omega$ and $\widehat{g}_\omega$ for $t_2 = 0$. That is

$$\begin{aligned} \widehat{f}(x, t) &= \widehat{f}_\omega(x, t, 0), \\ \widehat{g}(x, t) &= \widehat{g}_\omega(x, t, 0). \end{aligned}$$

The reason that we introduced the trivariate polynomials is that we can use them to get evaluation points of g: We have

$$\widehat{g}_\omega(0, 0, 1) = g(\boldsymbol{\omega}).$$

This we will use to get g via sparse interpolation by Theorem 2.6.

Recall that we know that $\widehat{g} \in F$ and $\widehat{g}_\omega \in F_\omega$. However, we first need to know which polynomial in F_ω corresponds to $\widehat{g}$, for every $\boldsymbol{\omega} \in \mathcal{E}$. The polynomials in F are used as reference. That is, for every $\widehat{g}_j \in F$, we compute a list L_j of pairs

$$L_j = \{\, (\boldsymbol{\omega}, \Delta) \mid \boldsymbol{\omega} \in \mathcal{E} \text{ and } \exists \widehat{g}_{\omega,i} \in F_\omega \;\; \widehat{g}_j(x, t) = \widehat{g}_{\omega,i}(x, t, 0) \text{ and } \Delta = \widehat{g}_{\omega,i}(0, 0, 1) \,\}$$

Observe that when $\widehat{g} = \widehat{g}_j$, then $L_j = \{\, (\boldsymbol{\omega}, g(\boldsymbol{\omega})) \mid \boldsymbol{\omega} \in \mathcal{E} \,\}$ is a list of point-values for g. Hence we can compute g by interpolation from the points in L_j by Theorem 2.6.

In case we consider a polynomial $\widehat{g}_{j'} \in F$ such that $\widehat{g} \neq \widehat{g}_{j'}$ and do interpolation for list $L_{j'}$, we might still end up in a sparse polynomial, say g', but g' might be reducible or not be a factor of f. Hence, after interpolation, we check whether g' is reducible via Lemma 4.1 and whether g' divides f to rule out the wrong choices.

Once we have the correct factor g in hand, we compute its multiplicity via Lemma 2.5. Algorithm 4 summarizes the steps.

```
Algorithm 4: Sparse-Factors
   Input  : $f(z) \in \mathcal{C}_\mu(s, n, d)$.
   Output: All irreducible $s$-sparse factors of $f$ with their multiplicities.
 1 $L = \emptyset$, $L' = \emptyset$    /* Initialize output list and intermediate candidates list */
 2 Find $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})^n$ such that $\mathrm{Hom}_d[f](\boldsymbol{\alpha}) \neq 0$    /* by Lemma 2.2 */
 3 Compute $\mathcal{H}_\alpha = \text{Irred-Proj}(\mathcal{C}_{sp})(\boldsymbol{\alpha})$
 4 for each $(\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathcal{H}_\alpha$ do
 5     Compute $\widehat{f}(x, t) = f(\boldsymbol{\alpha} x + \boldsymbol{\beta} t + \boldsymbol{\gamma})$ in dense representation
 6     Factorize $\widehat{f}$. Let $F = \{\widehat{g}_1, \widehat{g}_2, \ldots, \widehat{g}_m\}$ be the set of its irreducible factors
 7     $L_j = \emptyset$, for $j \in [m]$    /* Initialize point-value lists */
 8     Let $\mathcal{E}$ be the evaluation points for $\mathcal{C}_{sp}(s, n, d)$    /* see Theorem 2.6 */
 9     for each $\boldsymbol{\omega} \in \mathcal{E}$ do
10         Compute $\widehat{f}_\omega(x, t_1, t_2) = f(\boldsymbol{\alpha} x + \boldsymbol{\beta} t_1 + (\boldsymbol{\omega} - \boldsymbol{\gamma}) t_2 + \boldsymbol{\gamma})$ in dense representation
11         Factorize $\widehat{f}_\omega$. Let $F_\omega = \{\widehat{g}_{\omega,1}, \widehat{g}_{\omega,2}, \ldots, \widehat{g}_{\omega,r}\}$ be the set of its irreducible factors
12         for each $j \in [m]$ do
13             Search for $i \in [r]$ such that $\widehat{g}_j(x, t) = \widehat{g}_{\omega,i}(x, t, 0)$
14             $\Omega = \widehat{g}_{\omega,i}(0, 0, 1)$
15             add $(\boldsymbol{\omega}, \Omega)$ to $L_j$
16     for each $j \in [m]$ do
           /* Compute irreducible factors using Theorem 2.6 and Lemma 4.1 */
17         Compute polynomial $P_j$ by sparse interpolation on the points in $L_j$
18         If $P_j$ is $s$-sparse, irreducible and $P_j | f$ then add $P_j$ to $L'$
19 for each $P \in L'$ do
       /* Compute multiplicities via Lemma 2.5 */
20     Let $z$ be a variable that $P$ depends on
21     Find the smallest $e \geq 1$ such that $P \nmid \frac{\partial^e f}{\partial z^e}$ and add $(P, e)$ to list $L$
22 return $L$
```

The time complexity is now easy to see. Steps 2 and 3 take time $T_{\mathrm{PIT}(\mathrm{Hom}[\mathcal{C}_\mu])}$ and $T_{\text{Irred-Proj}(\mathcal{C}_{sp})}$, respectively. The **for**-loop line 4 - 18 is executed $|\mathcal{H}_\alpha| \leq T_{\text{Irred-Proj}(\mathcal{C}_{sp})}$ times. Steps 5 - 17 take time $\mathrm{poly}(snd)$ by Lemma 2.4 and Theorem 2.6. Line 18 takes time $\mathrm{poly}(snd)\, T_{\text{Irred-Proj}(\mathcal{C}_{sp})}$ to test irreducibility by Lemma 4.1, plus the time to check divisibility, $T_{\mathrm{Div}(\mathcal{C}_\mu/\mathcal{C}_{sp})}$.

In Line 20, since we have P computed explicitely, we can choose a variable z that occurs in P. Then we have at most d divisibility tests with the derivatives of P in Line 21. In summary, we get the time bound claimed in the theorem statement. □

Let $\mathcal{D} \subseteq C_{\mathrm{sp}}$ be a class of polynomials that are sparse. We observe that the proof of Theorem 4.2 works completely analogous when we replace C_{sp} by $\mathcal{D}$. The only change is in line 18 of Sparse-Factors when we have computed a sparse representation of P_j. There we also need to check the membership of P_j in $\mathcal{D}$. This is mostly trivial, for example for $\mathcal{D} = \mathcal{D}_\delta$. However, in the general setting $\mathcal{D} \subseteq \mathcal{C}_{\mathrm{sp}}(s, n, d)$, we need to assume that, given a polynomial in sparse representation, membership in $\mathcal{D}$ can be efficiently decided, i.e. in time $\mathsf{poly}(snd)$.

Corollary 4.3. *Let $\mathcal{C}_\mu = \mathcal{C}_\mu(s, n, d)$ and $\mathcal{D} \subseteq \mathcal{C}_{\mathrm{sp}}(s, n, d)$ with efficient membership tests. Then* $\mathsf{Factor}(\mathcal{C}_\mu|_\mathcal{D})$ *can be solved in time*

$$T_{\mathsf{Factor}(\mathcal{C}_\mu|_\mathcal{D})} = nd\, T_{\mathsf{PIT}(\mathsf{Hom}[\mathcal{C}_\mu])} + \mathsf{poly}(snd)\, T_{\mathsf{Irred\text{-}Proj}(\mathcal{D})}\, T_{\mathsf{Div}(\partial\mathcal{C}_\mu/\mathcal{D})}\,.$$

By choosing $\mathcal{D} = \mathcal{D}_\delta$, the polynomials of degree δ in Corollary 4.3, we get an alternative way to derive Theorem 3.2. Just observe that $T_{\mathsf{Irred\text{-}Proj}(\mathcal{D}_\delta)} = \mathsf{poly}(dn^{\mathsf{poly}(\delta)})$. In the following, we show further applications of Theorem 4.2 and Corollary 4.3.

4.1 Sparse factors of constant-depth circuits

A challenging open problem is to compute the sparse factors of a given sparse polynomial, or more generally of a given polynomial computed by a constant-depth circuit. Only exponential-time algorithms are known. It follows now that to make progress on this problem, it suffices to derandomize the irreducibility preserving projection of sparse polynomials in better than exponential time.

We apply Theorem 4.2 to output sparse factors of constant-depth circuits.

Corollary 4.4. *Let $\mathcal{C}_{\text{Depth-}t} = \mathcal{C}_{\text{Depth-}t}(s, n, d)$ and $\mathcal{C}_{\mathrm{sp}} = \mathcal{C}_{\mathrm{sp}}(s, n, d)$. Then, for any $\varepsilon > 0$,* $\mathsf{Factor}(\mathcal{C}_{\text{Depth-}t}|_{\mathcal{C}_{\mathrm{sp}}})$ *can be solved in time*

$$T_{\mathsf{Factor}(\mathcal{C}_{\text{Depth-}t}|_{\mathcal{C}_{\mathrm{sp}}})} = \left(n\,(sd)^{O(t)}\right)^{O((sd)^\epsilon)}\, T_{\mathsf{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})}\,.$$

Proof. Let $f \in \mathcal{C}_{\text{Depth-}t}$. We consider the running times from Theorem 4.2 to factor f. We can use a similar argument as in the proof of Corollary 3.4. For PIT, we have $T_{\mathsf{PIT}(\mathsf{Hom}[\mathcal{C}_{\text{Depth-}t}])} = \left(n\,(sd)^{O(t)}\right)^{O((sd)^\epsilon)}$ and the same time bound we have for $T_{\mathsf{Div}(\partial\mathcal{C}_{\text{Depth-}t}/\mathcal{C}_{\mathrm{sp}})}$ by Corollary 2.9. □

It follows that a subexponential bound on $T_{\mathsf{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})}$ would yield a subexponential algorithm to compute the sparse factors of a sparse polynomial, or, more general, of a polynomial computed by a constant-depth circuit.

Corollary 4.5. *If irreducibility preserving projection for sparse polynomials is in subexponential time, then the sparse factors of a sparse polynomial can be computed in subexponential time,*

$$T_{\mathsf{Irred\text{-}Proj}(\mathcal{C}_{\mathrm{sp}})} = (snd)^{O((sd)^\epsilon)} \implies \mathsf{Factor}(\mathcal{C}_{\mathrm{sp}}|_{\mathcal{C}_{\mathrm{sp}}}) = (snd)^{O((sd)^\epsilon)}\,.$$

4.2 Sum-of-univariate factors of sparse polynomials

Let us consider the family of polynomials that can be written as a sum of univariate polynomials,

$$\mathcal{C}_{\mathrm{SU}}(n, d) = \left\{ \sum_{i=1}^{n} p_i(z_i) \mid p_i \in \mathcal{P}(1, d), \text{ for } i = 1, 2, \ldots, n \right\} .$$

Note that $\mathcal{C}_{\mathrm{SU}}(n, d) \subsetneq \mathcal{C}_{\mathrm{sp}}(nd + 1, n, d)$.

Given a sparse polyomial, we show that its factors that are sums of univariates can be computed in polynomial time. The result was already shown by Volokovich [Vol15] with a different technique. We show that it also follows via Corollary 4.3.

We already know that PIT for sparse polynomials is in polynomial time. Moreover, Saha, Saptharishi, and Saxena [SSS13] showed that divisibility of a sparse polynomial by a sum of univariates is in polynomial time.

Theorem 4.6 ([SSS13, Theorem 5.2]). *Let $\mathcal{C}_{\mathrm{sp}} = \mathcal{C}_{\mathrm{sp}}(s, n, d)$ and $\mathcal{C}_{\mathrm{SU}} = \mathcal{C}_{\mathrm{SU}}(n, d)$. Then $\mathrm{Div}(\mathcal{C}_{\mathrm{sp}}/\mathcal{C}_{\mathrm{SU}})$ can be computed in time $\mathrm{T}_{\mathrm{Div}(\mathcal{C}_{\mathrm{sp}}/\mathcal{C}_{\mathrm{SU}})} = \mathrm{poly}(snd)$.*

Therefore, it suffices to give bounds on the time for irreducible projection $\text{Irred-Proj}(\mathcal{C}_{\mathrm{SU}})$ (see Section 2.5). We show that $\mathrm{T}_{\text{Irred-Proj}(\mathcal{C}_{\mathrm{SU}})}$ is polynomially bounded. We crucially use the following theorem.

Theorem 4.7 ([SSS13, Theorem 5.2]). *Let $p \in \mathcal{C}_{\mathrm{SU}}(n, d)$ with $|\mathrm{var}(p)| \geq 3$. Then p is irreducible.*

Lemma 4.8. $\mathrm{T}_{\text{Irred-Proj}(\mathcal{C}_{\mathrm{SU}})} \leq O(n^3 d^{30})$.

Proof. Let $p(\boldsymbol{z}) = \sum_{i \in [n]} p_i(z_i) \in \mathcal{C}_{\mathrm{SU}}(n, d)$ be irreducible and $\boldsymbol{\alpha} \in (\mathbb{F} \setminus \{0\})^n$ such that $p(\boldsymbol{\alpha} x + \boldsymbol{z})$ is monic in x. In order to apply Theorem 4.7, we consider $\mathrm{var}(p)$, the variables that p depends on.

If $|\mathrm{var}(p)| \leq 2$, we can directly derandomize Theorem 2.11: Suppose $\mathrm{var}(p) = \{z_1, z_2\}$ so that $p(\boldsymbol{z}) = p(z_1, z_2)$. There is a polynomial P in $2 \cdot 2 = 4$ variables of degree $2d^5$ such that any non-root of P gives an irreducible projection of p, where we set the other variables of p to zero, $z_i = 0$ for $i = 3, 4, \ldots, n$. Hence, the trivial hitting set (1) of size $(2d^5 + 1)^4 = O(d^{20})$ can be used for PIT for P. Since we want to handle the blackbox case, we do not assume that we have p in hand. Hence, we do not know whether actually $\mathrm{var}(p) = \{z_1, z_2\}$. So finally we will try all $\binom{n}{2}$ possibilities to choose 2 variables. For each possibility, say $\{z_i, z_j\}$, we take the same hitting set, but put the values at z_i and z_j and set the remaining z-variables to 0. That way we get $\binom{n}{2}$ hitting sets, and take their union to get the final hitting set of size $O(n^2 d^{20})$.

Now let $|\mathrm{var}(p)| \geq 3$. Similar as in the above case, assume for now that $\{z_1, z_2, z_3\} \subseteq \mathrm{var}(p)$, and in the end, we will try all $\binom{n}{3}$ possibilities to choose 3 variables.

Consider the substituting $\psi : z_i \mapsto \alpha_i x$, for $i \geq 4$. Then $\psi(p) \in \mathcal{C}_{\mathrm{SU}}(4, d)$, since $\sum_{i=4}^{n} p_i(\alpha_i x)$ is a univariate polynomial in x. Because $\mathrm{var}(\psi(p)| \geq 3$, polynomial $\psi(p)$ is irreducible by Theorem 4.7.

Now we substitute also the first 3 variables by $\phi : z_i \mapsto \alpha_i x + z_i$, for $i \in [3]$. Then

$$\widehat{p}(x, z_1, z_2, z_3) = \phi(\psi(p)) = p_1(\alpha_1 x + z_1) + p_2(\alpha_2 x + z_2) + p_3(\alpha_3 x + z_3) + \sum_{i=4}^{n} p_i(\alpha_i x)$$

is also irreducible and monic in x. Therefore, by Theorem 2.11, there exists a polynomial P of degree $2d^5$ in $2 \cdot 3 = 6$ variables such that any non-root of P gives an irreducible projection of p, where we set the other variables of p to zero, $z_i = 0$ for $i \geq 4$. Hence, we may again take the trivial hitting set (1) of size $(2d^5 + 1)^6 = O(d^{30})$. Now we can argue similarly as above and build the union of hitting sets over all $\binom{n}{3}$ possibilities to choose 3 variables. Hence, we end up with a hitting set of size $O(n^3 d^{30})$. □

Theorem 2.6, Theorem 4.6, and Lemma 4.8 give all the time complexities used in Corollary 4.3. We conclude that the sum-of-univariate factors of a sparse polynomial can be computed in polynomial time.

Corollary 4.9 ([Vol15] Sum-of-univariate factors). *Let* $\mathcal{C}_{\mathrm{sp}} = \mathcal{C}_{\mathrm{sp}}(s, n, d)$ *and* $\mathcal{C}_{\mathrm{SU}} = \mathcal{C}_{\mathrm{SU}}(n, d)$. *Then* $\mathrm{Factor}(\mathcal{C}_{\mathrm{sp}}|_{\mathcal{C}_{\mathrm{SU}}})$ *can be computed in time* $T_{\mathrm{Factor}(\mathcal{C}_{\mathrm{sp}}|_{\mathcal{C}_{\mathrm{SU}}})} = \mathrm{poly}(snd)$.

5 Conclusion

We conclude with some open questions.

1. Can we decide whether a given sparse polynomial is irreducible in deterministic subexponential time? The proof may already give a good bivariate projection that preserves irreducibility. Then Theorem 4.2 would give us a deterministic subexponential-time algorithm to find irreducible sparse factors of a sparse polynomial.

2. Can we find *bounded individual degree* sparse factors of a sparse polynomial (that is not restricted as bounded individual degree) in deterministic quasipolynomial time? Volkovich asked if multilinear factors of a sparse polynomial can be found in deterministic polynomial time [Vol15]. We do not even know a deterministic polynomial time algorithm to test if a sparse polynomial is divisible by a multilinear polynomial.

3. Can one compute all the factors of a sparse polynomial/constant depth circuit by constant depth circuits of small size? At least, can one find all the factors that are computable in constant depth? The recent result in [KRSV24] gives a deterministic subexponential-time algorithm that outputs a list of circuits (of unbounded depth and possibly with division gates) that includes all such factors, but there might be spurious factors as well in the list.

4. Given a blackbox computing the product of sparse irreducible polynomials f_i with bounded individual degree, find f_i's in deterministic polynomial time. [BSV20] gives a quasipolynomial time algorithm, when the input is sparse with constant individual degree and the factors are all sparse (polynomially upper bounded with respect to input polynomial's sparsity).

Acknowledgements P.D. is supported by the project titled "Computational Hardness of Lattice Problems and Implications", funded by National Research Foundation (NRF) Singapore. T.T's research supported by DFG grant TH 472/5-2. A.S. was supported by DFG grant TH 472/5-1 when the work started.

References

[AS03] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Combinatorica*, 3(23), 2003.

[Ber70] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713–735, 1970.

[Bog05] Andrej Bogdanov. Pseudorandom generators for low degree polynomials. In *37th ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2005.

[BSV20] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *Journal of the ACM (JACM)*, 67(2):1–28, 2020.

[BV22] Pranav Bisht and Ilya Volkovich. On solving sparse polynomial factorization related problems. In *42nd IARCS Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022.

[DGV24] Ashish Dwivedi, Zeyu Guo, and Ben Lee Volk. Optimal pseudorandom generators for low-degree polynomials over moderately large fields. *arXiv preprint arXiv:2402.11915*, 2024.

[For15] Michael A. Forbes. Deterministic divisibility testing via shifted partial derivatives. In *56th Symposium on Foundations of Computer Science (FOCS)*, pages 451–465. IEEE, 2015.

[GG20] Zeyu Guo and Rohit Gurjar. Improved explicit hitting-sets for ROABPs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2020.

[GKS17] Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity testing for constant-width, and any-order, read-once oblivious arithmetic branching programs. *Theory Comput.*, 13(1):1–21, 2017.

[GKST17] Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. *computational complexity*, 26:835–880, 2017.

[Gup14] Ankit Gupta. Algebraic geometric techniques for depth-4 pit & Sylvester-Gallai conjectures for varieties. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 21, 2014.

[HW00] Ming-Deh Huang and Yiu-Chung Wong. Extended Hilbert irreducibility and its applications. *Journal of Algorithms*, 37(1):121–145, 2000.

[Kal85a] Erich Kaltofen. Computing with polynomials given by straight-line programs I: greatest common divisors. In *17th ACM Symposium on Theory of Computing (STOC)*, pages 131–142, 1985.

[Kal85b] Erich Kaltofen. Effective Hilbert irreducibility. *Information and Control*, 66(3):123–137, 1985.

[Kal85c] Erich Kaltofen. Polynomial-time reductions from multivariate to bi-and univariate integral polynomial factorization. *SIAM Journal on Computing*, 14(2):469–489, 1985.

[Kal89] Erich Kaltofen. Factorization of polynomials given by straight-line programs. *Randomness and Computation*, 5:375–412, 1989.

[Kal95] Erich Kaltofen. Effective Noether irreducibility forms and applications. *Journal of Computer and System Sciences*, 50(2):274–295, 1995.

[Kay] Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1409–1421.

[Kay12] Neeraj Kayal. Affine projections of polynomials. In *44th ACM Symposium on Theory of Computing (STOC)*, pages 643–662, 2012.

[KI03] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *35th ACM Symposium on Theory of Computing (STOC)*, pages 355–364, 2003.

[KRS24] Mrinal Kumar, Varun Ramanathan, and Ramprasad Saptharishi. Deterministic algorithms for low degree factors of constant depth circuits. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3901–3918. SIAM, 2024.

[KRSV24] Mrinal Kumar, Varun Ramanathan, Ramprasad Saptharishi, and Ben Lee Volk. Towards deterministic algorithms for constant-depth factors of constant-depth circuits. *arXiv preprint arXiv:2403.01965*, 2024.

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

[KSS15] Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and polynomial factorization. *computational complexity*, 24(2):295–331, 2015.

[KT90] Erich Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

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

[LST21] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *62nd Symposium on Foundations of Computer Science (FOCS)*, pages 804–814. IEEE, 2021.

[Oli16] Rafael Oliveira. Factors of low individual degree polynomials. *computational complexity*, 2(25):507–561, 2016.

[RR19] C. Ramya and BV Raghavendra Rao. Linear projections of the Vandermonde polynomial. *Theoretical Computer Science*, 795:165–182, 2019.

[RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *computational complexity*, 14:1–19, 2005.

[Shp07] Amir Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. In *39th ACM Symposium on Theory of Computing (STOC)*, pages 284–293, 2007.

[Sin16] Gaurav Sinha. Reconstruction of real depth-3 circuits with top fan-in 2. In *31st Conference on Computational Complexity (CCC)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2016.

[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.

[Str73] Volker Strassen. Vermeidung von Divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973.

[SV10] Amir Shpilka and Ilya Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *International Colloquium on Automata, Languages, and Programming*, 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)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[Vol17] Ilya Volkovich. On some computations on sparse polynomials. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[vzGK85] Joachim von zur Gathen and Erich Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, 1985.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il