On Factorization of Sparse Polynomials of Bounded Individual Degree

Aminadav Chuyoon
School of Mathematics, Tel Aviv University, Israel

Amir Shpilka
Blavatnik School of Computer Science and AI, Tel Aviv University, Israel

Abstract

We study sparse polynomials with bounded individual degree and the class of their factors. In particular we obtain the following algorithmic and structural results:

1. A deterministic polynomial-time algorithm for finding *all the sparse divisors* of a sparse polynomial with bounded individual degree. As part of this, we establish the first upper bound on the number of non-monomial irreducible factors of such polynomials.
2. A $\mathrm{poly}(n, s^{d \log \ell})$-time algorithm for recovering ℓ irreducible s-sparse polynomials of bounded individual degree d from blackbox access to their product (which is not necessarily sparse). This partially resolves a question posed in [10]. In particular, when $\ell = O(1)$, the algorithm runs in polynomial time.
3. Deterministic algorithms for factoring a *product* of s-sparse polynomials of bounded individual degree d from blackbox access. Over fields of characteristic zero or sufficiently large, the algorithm runs in $\mathrm{poly}(n, s^{d^3 \log n})$-time; over arbitrary fields it runs in $\mathrm{poly}(n, (d^2)!, s^{d^5 \log n})$-time. This improves upon the algorithm of [2], which runs in $\mathrm{poly}(n, s^{d^7 \log n})$-time and applies only to a single sparse polynomial of bounded individual degree. In the case where the input is a single sparse polynomial, we give an algorithm that runs in $\mathrm{poly}(n, s^{d^2 \log n})$-time.
4. Given blackbox access to a *product* of (not necessarily sparse or irreducible) *factors of sparse polynomials of bounded individual degree*, we give a deterministic polynomial-time algorithm for finding all irreducible sparse multiquadratic factors of it (along with their multiplicities). This generalizes the algorithms of [26] and [27]. We also show how to decide whether such a product is a complete power (in case it is defined over a field of zero or large enough characteristic), extending the algorithm of [5].

Our algorithms most naturally apply over fields of zero or sufficiently large characteristic. To handle arbitrary fields, we introduce the notion of *primitive divisors* for a class of polynomials, which may be of independent interest. This notion enables us to adapt ideas of [5] and remove characteristic assumptions from most of our algorithms.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases algebraic complexity theory, sparse polynomials, factorization, reconstruction

Digital Object Identifier 10.4230/LIPIcs.CCC.2026.20

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2026/036/`

Funding This research was funded by the European Union (ERC, EACTP, 101142020). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

1.1 Background and Motivation

This paper is motivated by numerous open problems concerning the factorization and complexity of factors of multivariate sparse polynomials. These are n-variate polynomials of degree $\text{poly}(n)$ whose number of monomials is also $\text{poly}(n)$. In particular, this is the class of polynomials that have polynomial size $\Sigma\Pi$ circuits, which is one of the simplest classes from the algebraic complexity point of view.

Despite being structurally simple, sparse polynomials form a rich and non-trivial class of polynomials that has been extensively studied. Efficient deterministic algorithms for polynomial identity testing and reconstruction were developed for this class [1, 13, 7, 18]. In particular, the work of Klivans and Spielman [18] introduced a special reduction from multivariate sparse polynomials to univariate polynomials of polynomial degree, over any field, that has since found many applications, mainly for questions concerning polynomial identity testing and reconstruction. See the surveys [21, 24, 22, 9].

In this work we are mostly interested in understanding factors of sparse polynomials. The seminal work of Kaltofen [16] shows that factors of sparse polynomials have polynomial-size algebraic circuits. Over fields of characteristic zero, or sufficiently large characteristic, the recent breakthrough of [3] implies that factors of sparse polynomials have polynomial-size bounded-depth circuits.

However, an intriguing question that remains unsolved for over forty years concerns the complexity of factors of sparse polynomials in terms of their number of monomials. In other words, how many monomials can a factor of a sparse polynomial have. The following example of von zur Gathen and Kaltofen demonstrates that the sparsity of irreducible factors of polynomials may be super-polynomial.

▶ **Example 1** (Example of [28]). Consider

$$g(\mathbf{x}) = \prod_{i=1}^{n}(x_i - 1) \quad \text{and} \quad h(\mathbf{x}) = \prod_{i=1}^{n}(1 + x_i + \cdots + x_i^{n-1}) + n.$$

It is not hard to prove that h is irreducible over $\mathbb{Q}$. Let

$$f(\mathbf{x}) = g \cdot h = \prod_{i=1}^{n}(x_i^n - 1) + n\prod_{i=1}^{n}(x_i - 1).$$

Then, f has $(2^{n+1} - 1)$ monomials, and an irreducible factor h of sparsity n^n, which is super-polynomial in the sparsity of f.

This led them to pose the following question.

▶ **Question 2** ([28]). *Can the output size for the factoring problem actually be more than quasi-polynomial in the sparsity of the input?*

This question is still wide open with only several special cases solved. In [26] Volkovich proved that multilinear factors of sparse polynomials are sparse. In a follow-up work [27] he proved that all factors of multiquadratic sparse polynomials are themselves sparse. This result was extended to higher individual degrees by Bhargava, Saraf, and Volkovich [2], who considered sparse polynomials with bounded individual degree d, and proved a quasi-polynomial upper bound on the sparsity of factors of such polynomials. In particular, if f is an s-sparse n-variate polynomial with individual degrees bounded by d, then every

factor of f has at most $s^{O(d^2 \log n)}$ monomials. This is the first nontrivial sparsity bound for factors in the regime $d > 2$. While the result of [2] gives a quasi-polynomial upper bound on sparsity of factors of polynomials with bounded individual degree, it is believed that the "real" bound should be polynomial. In [5], the authors provide some evidence for that by giving polynomial-time algorithms for various problems for which the existence of such algorithms is implied by the conjectured polynomial upper bound on sparsity of factors.

In their work von zur Gathen and Kaltofen gave a randomized algorithm for factorization of sparse polynomials and noted another obstacle towards obtaining deterministic factorization algorithms:

> "Also note that there does not seem to be a feasible way of checking deterministically whether the output is correct, i.e., whether $f = \prod_i f_i^{e_i}$" [28].

In particular, Dutta, Sinhababu and Thierauf asked the following question.

▶ **Question 3** ([10, Question 2]). *Can we find the factorization of polynomials whose irreducible factors are sparse?*

We note that [3] gave a subexponential time algorithm for Question 3, but obtaining a polynomial-time or even quasi-polynomial-time algorithm is still an open question.

In fact, Dutta et al. also asked the following special case of Question 3.

▶ **Question 4** ([10, Section 6, item 4]). *Given a blackbox computing the product of sparse irreducible polynomials f_i with bounded individual degree, find the f_i's in deterministic polynomial-time.*

The aforementioned results of [26, 27, 2] also have consequences for Question 3 and Question 4. In [26] Volkovich gave a deterministic polynomial-time algorithm for factorization of sparse polynomials whose irreducible factors are all multilinear (hence sparse) polynomials. In [27] he gave an efficient deterministic algorithm for factoring multiquadratic sparse polynomials. In [2] the authors combined their sparsity bound with techniques similar to [26, 27] to obtain a deterministic quasi-polynomial-time algorithm for factoring sparse polynomials of bounded individual degree. Dutta, Sinhababu, and Thierauf [10] gave efficient deterministic reconstruction of constant degree factors of sparse polynomials.

One notable limitation of the algorithm of [2] is that it does not give an efficient way to find the *sparse factors* of the input polynomial, even if such factors exist. This problem is closely related to the following question asked by Dutta, Sinhababu and Thierauf.

▶ **Question 5** ([10, Question 1]). *Design a better-than-exponential time deterministic algorithm that outputs all the sparse irreducible factors of a sparse polynomial.*

The result of Bhattacharjee et al. [3] gives a subexponential time algorithm for the problem, but obtaining a polynomial-time (or quasi-polynomial) time algorithm is still an open problem.

Another outstanding open problem concerns the reconstruction of rational functions whose numerator and denominator are sparse polynomials. Surprisingly, the best known deterministic algorithm for this question runs in exponential time [14]. While seemingly unrelated, special cases of this question lie at the heart of our algorithms, and we solve it in two special cases where more information is provided.

Many more open problems have been asked for sparse polynomials and the reader is referred to [10] and [12].

1.2 Our Results

Before presenting our results, let us introduce the following notation.

▶ **Definition 6.** *We denote the class of n-variate, s-sparse polynomials of bounded individual degree d, by $\mathcal{P}(n,s,d)$. We refer to polynomials in this class as (n,s,d)-sparse polynomials. It turns out that in many cases it makes sense to talk about the class of* divisors *of (n,s,d)-sparse polynomials; we denote this class of polynomials by $\mathcal{P}_{Factors}(n,s,d)$.*

We first summarize our main contributions informally as follows:

1. We prove a sharp upper bound on the number of irreducible factors of (n,s,d)-sparse polynomials (Theorem 8 and Remark 18).
2. Using this bound, we give the first deterministic polynomial-time algorithm that constructs *all* sparse divisors of (n,s,d)-sparse polynomials (Theorem 9).
3. We give a deterministic blackbox factorization algorithm for products of (n,s,d)-sparse polynomials in quasi-polynomial-time in the number of terms. This solves Question 4 when the number of terms is constant (Theorem 10). We note that the running time of this algorithm is polynomial in the number of variables n.

▶ Remark 7. In stating our results we suppress the precise dependence on the field (beyond its characteristic) and on bit complexity. As usual in factorization algorithms, an additional factor reflecting the complexity of univariate factorization over the field should be included.

We now give the formal statement of our results. Our first main result gives an upper bound for the number of sparse divisors of (n,s,d)-sparse polynomials. While its proof is not difficult, it is instrumental in our algorithms and demonstrates the strong structural limitations that sparsity imposes.

▶ **Theorem 8** (Divisor bound for (n,s,d)-sparse polynomials)**.** *Let $f \in \mathcal{P}(n,s,d)$. The number of non-monomial irreducible factors of f, counted with multiplicities, is at most $d \log s$. In particular, the number of divisors of an s-sparse polynomial f with bounded individual degree d, that are not divisible by any monomial, is at most $2^{d \log s} = s^d$.*

Our first main algorithmic result shows how to deterministically output all sparse divisors of such polynomials in polynomial-time. Earlier, only a quasi-polynomial-time algorithm was known [2].

▶ **Theorem 9** (Finding all sparse divisors of (n,s,d)-sparse polynomials)**.** *There is a deterministic, $\mathrm{poly}(n, d!, s^d)$-time algorithm which, given $f \in \mathcal{P}(n,s,d)$, outputs all the s-sparse divisors of f that are not divisible by any monomial, as well as the monomial of maximal degree dividing f.*

Our second main algorithmic result answers Question 4 when there are constantly many polynomials in the product, and in the general case it provides a quasi-polynomial-time algorithm.

▶ **Theorem 10** (Factoring products of (n,s,d)-sparse polynomials)**.** *Let $\mathbb{F}$ be a field of characteristic zero or larger than $2d$. There is a deterministic $\mathrm{poly}(n, d^d, s^{d \log \ell}, \ell^d)$-time algorithm which, given blackbox access to a product $f = \prod_{i=1}^{\ell} \phi_i$, where $\phi_1, \ldots, \phi_\ell \in \mathcal{P}(n,s,d)$ are all (not necessarily different) irreducible, returns the ϕ_i-s. Moreover, when the assumption on the field is removed, there is an algorithm that solves this problem in $\mathrm{poly}(n, (d^2)!, s^{d \log \ell + d^3}, \ell^d)$-time.*

This theorem will follow as a consequence of the next, more general, result.

▶ **Theorem 11** (Factoring products of factors of sparse polynomials)**.** *Let $\mathbb{F}$ be a field with* $\mathrm{char}(\mathbb{F}) = 0$ *or* $\mathrm{char}(\mathbb{F}) > 2d$*. There is a deterministic* $\mathrm{poly}(n, d^d, s^{d\log \ell}, \ell^d)$*-time algorithm which, given blackbox access to a product f of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d)$ over the field $\mathbb{F}$, returns all the s-sparse divisors of bounded individual degree d of f that are not divisible by any monomial, as well as the multiplicity k_i of each x_i as a factor of f. Moreover, if f is in fact a product of polynomials in $\mathcal{P}(n, s, d)$, there is such an algorithm that works over arbitrary fields, whose running time is* $\mathrm{poly}(n, (d^2)!, s^{d\log \ell + d^3}, \ell^d)$*.*

Observe that this result is slightly weaker for fields of small characteristic. Indeed, as will be discussed in Section 1.4.1.2 we will see that some of our methods only work for zero characteristic fields, or for fields with large enough characteristic.

In addition to our main results we provide several improvements and generalizations of earlier work. We start by stating results concerning factorization of sparse polynomials of bounded individual degree into irreducible factors. The first improves the best known factorization algorithm of such polynomials [2].

▶ **Theorem 12.** *There is a deterministic* $\mathrm{poly}(n, s^{d^2 \log n})$*-time algorithm which, given an s-sparse polynomial of bounded individual degree d in n-variables, returns its factorization into irreducible factors.*

The second theorem generalizes the result of [2] to a significantly larger class of polynomials, while achieving improved running time compared to their algorithm.

▶ **Theorem 13.** *Let $\mathbb{F}$ be a field with* $\mathrm{char}(\mathbb{F}) = 0$ *or* $\mathrm{char}(\mathbb{F}) > 2d$*. There is a deterministic* $\mathrm{poly}(n, s^{d^3 \log n}, (d\ell)^d)$*-time algorithm which, given blackbox access to a product f of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d)$ over $\mathbb{F}$, returns its factorization into irreducible factors. Moreover, there is an algorithm that works over arbitrary fields, whose running time is* $\mathrm{poly}(n, (d^2)!, s^{d^5 \log n}, \ell^d)$*.*

Note that this last result considers the class of *factors of sparse polynomials of bounded individual degree*. Indeed, we will see that it is more natural to consider this class in many cases, and several results of ours actually hold for this class rather than just for the class of sparse polynomials.

Finally, we generalize results of [26, 27, 5] in a similar manner to our generalization of the results of [2].

Our next algorithm finds all the multiquadratic irreducible factors of a product of polynomials from $\mathcal{P}_{\mathrm{Factors}}(n, s, d)$. Earlier, only factorization of polynomials whose irreducible factors are all multiquadratic and s-sparse was known [27, Theorem 47].

▶ **Theorem 14.** *Let $\mathbb{F}$ be a field of characteristic zero or larger than $2d$. There is a deterministic* $\mathrm{poly}(n, d!, s^d, \ell)$*-time algorithm that, given blackbox access to a product f of ℓ elements of $\mathcal{P}_{Factors}(n, s, d)$ defined over $\mathbb{F}$, returns all the s-sparse multiquadratic irreducible factors of f. In particular, the algorithm returns all the multilinear factors of f (as these are s-sparse, see for example [15, Lemma 10]). Moreover, there exists such an algorithm that works over arbitrary fields, whose running time is* $\mathrm{poly}(n, (d^2)!, s^{d^2}, \ell)$*.*

We give a similar improvement to the perfect power testing of [5].

▶ **Theorem 15.** *Let $\mathbb{F}$ be a field of characteristic zero or larger than $2d$. There is a deterministic* $\mathrm{poly}(n, d!, s^d, \ell)$*-time algorithm that, given blackbox access to a polynomial f that is a product of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d)$ (defined over $\mathbb{F}$) and a positive integer e, decides whether f is a complete e-th power.*

Finally, we extend the polynomial identity testing result of [5].

▶ **Theorem 16.** *Let $\mathbb{F}$ be a field of characteristic zero or larger than $2d$. There exists a deterministic* $\mathrm{poly}(n, d!, s^d, \ell)$*-time algorithm that, given blackbox access to two polynomials (over $\mathbb{F}$) f and g that are products of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d)$, decides if $f|g$. Moreover, if f and g are actually products of ℓ polynomials in $\mathcal{P}(n, s, d)$, there is an algorithm that works over arbitrary fields, whose running time is* $\mathrm{poly}(n, (d^2)!, s^{d^3}, \ell)$

As mentioned in Section 1.1, we will have to solve some rational interpolation problems. Somewhat surprisingly, it is not known how to deterministically reconstruct two coprime s-sparse polynomials f and g from blackbox access to their quotient g/f in polynomial-time. We shall see that with some additional information on f, this is in fact possible. Without this extra information, only exponential deterministic algorithms are known for solving this problem [14]. We note that efficient randomized algorithms for solving this problem are known, see e.g., [8, 25].

1.3 Related Work

The questions considered in this work have been studied extensively in recent years. We organize the prior results around three themes that are central to this paper: sparsity bounds, factorization and reconstruction algorithms, and divisibility and identity testing.

Sparsity of Factors of Sparse Polynomials

Volkovich [26] showed that multilinear factors of s-sparse polynomials are again s-sparse. In [27] he proved that factors of an s-sparse multiquadratic polynomial are s-sparse as well. For general polynomials of bounded individual degree d, Bhargava, Saraf and Volkovich [2], obtained the first non-trivial bound on the sparsity of factors of s-sparse polynomials with bounded individual degree d, namely $s^{O(d^2 \log n)}$. Bisht and Saxena [4] improved this bound to $s^{\mathrm{poly}(d)}$ for the class of *symmetric* s-sparse polynomials of bounded individual degree d. Bisht and Volkovich [5] obtained an $s^{O(d \log s)}$ bound on the sparsity of a quotient of an s-sparse polynomial of bounded individual degree d by a multilinear factor. Finally, the work of Forbes [11] and results in [5] imply that the sparsity of the quotient of $f \in \mathcal{P}(n, s, d)$ by a degree r polynomial is bounded by $(ns)^{O(dr)}$.

Deterministic factorization and reconstruction

Factorization results have largely paralleled the development of sparsity bounds. Volkovich showed in [26] how to efficiently factorize a sparse polynomial whose factors are all multilinear. In [27] he demonstrated how to efficiently compute the irreducible factors of a polynomial from a blackbox access, given that all of them are multiquadratic and sparse. In Theorem 14 we further improve this result by relaxing the condition on the input polynomial – we allow it to be a product of factors of some sparse polynomial of bounded individual degree, and not simply a product of sparse multiquadratic polynomials.

Using their sparsity bound, Bhargava, Saraf, and Volkovich [2] derived a deterministic quasi-polynomial algorithm for factoring sparse polynomials of bounded individual degree. However, their methods do not yield a polynomial-time algorithm for finding the sparse divisors of the input polynomial, so they do not imply Theorem 9. Furthermore, their algorithm does not extend to factorization of products of sparse, bounded degree polynomials, let alone products of factors of such polynomials.

Kumar, Ramanathan, and Saptharishi [19] and Dutta, Sinhababu, and Thierauf [10], gave an algorithm that outputs all the irreducible factors of constant degree of a sparse polynomial in quasi-polynomial-time. As their result applies to general sparse polynomials without restrictions on the individual degree, it does not follow from the methods of this paper.

In [5], an algorithm for deciding if a sparse polynomial of bounded individual degree is a complete power is given; in Section 1.4.1 we describe how to use our stronger methods to obtain this result when the input polynomial is a product of polynomials in $\mathcal{P}_{\text{Factors}}(n, s, d)$, given we have blackbox access to it. However, this generalization does not work when the underlying field is of small characteristic.

Divisibility and identity testing

In [27] it is also shown how to efficiently check if two s-sparse polynomials of bounded individual degree d divide each other; this is improved in Theorem 16, where we show a divisibility testing result for two products of factors of sparse polynomials of bounded individual degree, given blackbox access to them. In the same work, Volkovich also gave a deterministic algorithm for testing a factorization of sparse polynomials, with constant individual degrees, into sparse irreducible factors. That is, testing if $f = \prod g_i$ when f has constant individual degrees and g_i-s are irreducible and sparse. We note that, as a simple corollary of Theorem 9, we are actually able to efficiently find the g_i's and verify that $f = \prod g_i$ in polynomial time.

Forbes [11] gave a quasi-polynomial-time algorithm for checking if a constant degree polynomial divides a sparse polynomial. Note that while our result allows the dividing polynomial to be much more complicated, Forbes' result requires no restriction on the individual degree of the given sparse polynomial. Therefore, our methods fall short in proving his result, or any other result on sparse polynomials with no restriction on their individual degree.

In [5], the authors design an efficient blackbox PIT algorithm for the class $\Sigma^{[2]}\Pi\Sigma\Pi^{[\text{ind-deg d}]}$; while doing so, they obtain some technical results about the class $\mathcal{P}_{\text{Factors}}(n, s, d)$. In Section 1.4.1.1 we show that their methods imply a stronger result than this PIT. Showing PIT for this class is essentially equivalent to giving an algorithm for determining if two products f, g of (n, s, d)-sparse polynomials are equal, given blackbox access to them. We show that the methods of [5] can be used for determining if f divides g. In Section 1.4.1.2 we further generalize and strengthen the technical results of [5], but only for fields of zero or sufficiently large characteristic.

1.4 Proof Overview

We start by describing some technical results about the class of sparse polynomials of bounded individual degree needed for our proofs. We then turn to give an overview of our results regarding factorization of sparse polynomials. We first discuss our result on the recovery of all irreducible sparse multiquadratic factors of a product of (n, s, d)-sparse polynomials, as the proof contains some of the important ideas used in the proofs of our main results. We then move on to describe the recovery of all sparse divisors of sparse polynomials of bounded individual degree and our partial solution to Question 4. Finally, we briefly describe how to use our methods for factoring general (n, s, d)-sparse polynomials and products of them, making use of the bound of [2].

1.4.1 Sparse Polynomials of Bounded Individual Degree and Their Factors

The main tool we use for obtaining our results is a generator $G : \mathbb{F}^6 \to \mathbb{F}^n$ that preserves important properties of polynomials in $\mathcal{P}_{\text{Factors}}(n, s, d)$. Roughly speaking, we want this map G to satisfy the following two properties:

1. Its image $\text{Im}(G)$ is an interpolation set for polynomials in $\mathcal{P}(n, s, d)$.
2. If ϕ and ψ are non-associate[1] irreducible elements of $\mathcal{P}_{\text{Factors}}(n, s, d)$, then $\phi(G_i)$ and $\psi(G_i)$ are coprime as polynomials in x_i, where G_i should be thought of as "G when reviving the i-th variable" (i.e., keeping x_i alive and substituting the remaining variables according to G).

The first property is easily achieved by the basic properties of the Klivans-Spielman generator (KS-generator), introduced in [18]. We show that the second property follows from the first, together with the fact that resultants of sparse polynomials of bounded individual degree are again sparse. However, we are able to do so only when the underlying field $\mathbb{F}$ is of zero characteristic, or if its characteristic is larger than twice the bound on the individual degree. If this is not the case then we are only able to prove something a little weaker – we prove that the second property holds if one replaces the notion of irreducible polynomials with the new notion of *primitive divisors*, which we first define in this paper (we describe this notion below).

Before sketching the proof that the first property of G implies the second, we note that proving that will automatically imply some algorithms for sparse polynomials. For example, it is straightforward to see that by factoring the composition of the input polynomials with G_i, the following two results can be obtained from the (general) second property of G.

1. Given blackbox access to two products of elements of $\mathcal{P}_{\text{Factors}}(n, s, d)$, we can decide if one divides the other in deterministic polynomial-time (Theorem 16).
2. Given blackbox access to a product of elements of $\mathcal{P}_{\text{Factors}}(n, s, d)$, we can decide if it is a complete power in deterministic polynomial-time (Theorem 15).

This already generalizes results from [27] and [5]. As we are able to obtain the general second property of G only for some fields, these results do not apply to arbitrary fields. It turns out, however, that the aforementioned weaker version of the second property of G is strong enough to obtain Theorem 16 over arbitrary fields, in the case we replace $\mathcal{P}_{\text{Factors}}(n, s, d)$ by $\mathcal{P}(n, s, d)$.

In the rest of this subsection we describe our proofs of the second property, both for arbitrary fields and for fields with zero or large enough characteristic.

1.4.1.1 Arbitrary Fields

In [5, Lemmas 4.5,4.6] it is shown that the second property of G holds for each pair of irreducible polynomials ϕ and ψ that satisfy a certain non-degeneracy condition: there exist (n, s, d)-sparse polynomials f, g for which $\phi|f, \psi|g$ and the matrix $\begin{pmatrix} v_\phi(f) & v_\phi(g) \\ v_\psi(f) & v_\psi(g) \end{pmatrix}$ is invertible, where $v_\phi(f)$ is the multiplicity of ϕ as a factor of f, and so are the other entries, respectively. That is, a pair of polynomials $\phi, \psi \in \mathcal{P}_{\text{Factors}}(n, s, d)$ does not satisfy this condition if the multiplicity of ϕ as a factor of an element f of $\mathcal{P}(n, s, d)$ determines the multiplicity of ψ as a factor of f. In this sense, ϕ and ψ are "inseparable" from the point of view of polynomials in $\mathcal{P}(n, s, d)$.

[1] Two polynomials are associate if one is a scalar multiple of the other.

For each irreducible $\phi \in \mathcal{P}_{\text{Factors}}(n,s,d)$, consider all ψs that are inseparable from it, in the sense that ϕ, ψ do not satisfy the aforementioned non-degeneracy condition. For each $f \in \mathcal{P}(n,s,d)$, consider the vector of multiplicities of these ψs as factors of f - $(v_\psi(f))_\psi$. By the degeneracy assumption, these vectors are all proportional. Consider the $\mathbb{Z}$-span of these vectors, and let $(a_\psi)_\psi$ be a generator for this one-dimensional lattice. The polynomial $\prod_\psi \psi^{a_\psi}$ is called a primitive divisor for the class $\mathcal{P}(n,s,d)$. Note that by construction, as the property of being "inseparable" is an equivalence relation on the class of irreducible polynomials in $\mathcal{P}_{\text{Factors}}(n,s,d)$, it must hold that every two primitive divisors are coprime.

The notion of a primitive divisor is a general notion, that applies to a general class of polynomials $\mathcal{P}$, and therefore may be of independent interest. The formal definition of a primitive divisor for a class of polynomials is given in Definition 31.

Primitive divisors for a class of polynomials $\mathcal{P}$ are for the class $\mathcal{P}$ what irreducible polynomials are for the class of all polynomials, in the sense that every element of the class $\mathcal{P}$ can be written as a product of primitive divisors for $\mathcal{P}$ in a unique way. This is not too difficult to see by the construction given in the previous paragraph, and is formulated in the following lemma.

▶ **Lemma 17.** *Let $\mathcal{P}$ be some class of polynomials, and let $f \in \mathcal{P}$. Then, f can be uniquely written as a product of primitive divisors for $\mathcal{P}$.*

Note that Lemma 17 suggests that it makes sense to talk about the multiplicity $v_\phi(f)$ of a primitive divisor ϕ as a factor of $f \in \mathcal{P}$ (see Definition 32).

With the notion of primitive divisors in hand, one can show that every two non-associate primitive divisors ϕ and ψ for a class $\mathcal{P}$ satisfy the non-degeneracy condition of [5]. As a result, we are able to apply the proof of the aforementioned result of [5] to obtain the weaker version of the second property of G, namely, for every two non-associate primitive divisors ϕ, ψ for $\mathcal{P}(n,s,d)$ it holds that $\gcd_{x_i}(\phi(G_i), \psi(G_i)) = 1$.

We further note that for every two products f, g of elements of $\mathcal{P}(n,s,d)$ we have that $f|g$ if and only if $v_\phi(f) \leq v_\phi(g)$ for every primitive divisor ϕ for $\mathcal{P}(n,s,d)$; this is a simple corollary of our unique-factorization theorem. As a result, the weaker second property of G is enough for giving a divisibility testing for f and g: just check if $f(G_i)|g(G_i)$ as polynomials in x_i for every i.

1.4.1.2 Zero or Large Characteristic

In the proof of [5, Lemma 4.5], Bisht and Volkovich consider the Sylvester matrix $M_{x_i}(f^\alpha, g^\beta)$ of some suitably chosen small powers of the polynomials f, g, viewed as polynomials in x_i. They then use the first property of G to prove that the rank of this matrix does not change when composing with the generator G_i, that is, $\text{rank}(M_{x_i}(f^\alpha, g^\beta)) = \text{rank}(M_{x_i}(f(G_i)^\alpha, g(G_i)^\beta))$. From that, they deduce that it must be the case that $\phi(G_i)$ and $\psi(G_i)$ do not share factors, as the existence of such a factor would result in rank decrease.

To obtain our result, instead of considering the Sylvester matrix $M_{x_i}(f^\alpha, g^\beta)$, we consider the matrix defining the discriminant $\Delta(fg)$ of the polynomial fg. That is, we consider the Sylvester matrix $M_{x_i}(fg, (fg)')$ of fg and its formal derivative with respect to x_i, $(fg)'$. We next explain the idea behind it.

Suppose $\phi|f, \psi|g$ are two irreducible elements of $\mathcal{P}_{\text{Factors}}(n,s,d)$. View all polynomials as polynomials in some variable x_i, and consider the Sylvester matrix $M_{x_i}(fg, (fg)')$. It is not hard to prove that its rank essentially counts the number of distinct irreducible factors of fg (when appropriately counted – an irreducible factor of degree d should contribute d distinct irreducible factors). As all the determinants of minors of $M_{x_i}(fg, (fg)')$ are sparse

polynomials of low degree, we are able to show, using the first property of G, that the rank of this matrix does not change when composing with G_i. In particular, this means that $\phi(G_i), \psi(G_i)$ cannot share a common factor, as otherwise the number of unique factors of $fg(G_i)$ would be smaller than that of fg, in contradiction to the fact that the rank remains unchanged. With this result in hand, we obtain Theorem 16 and Theorem 15 by considering the factorization of the inputs composed with the generators G_i.

1.4.2 Finding Multiquadratic Factors of Products of Sparse Polynomials

We next show how to use the results of Section 1.4.1 to obtain Theorem 14. For the sake of presentation we assume here that $\mathrm{char}(\mathbb{F}) = 0$.

We first consider the following general setting. Let f be a product of polynomials in $\mathcal{P}_{\mathrm{Factors}}(n, s, d)$; that is, f is a product of factors of s-sparse polynomials of bounded individual degree d. Consider some irreducible $\phi \in \mathcal{P}_{\mathrm{Factors}}(n, s, d)$. By the second property of G it follows that for every i for which $x_i \in \mathrm{var}(\phi)$, the highest power of $\phi(G_i)$ that divides $f(G_i)$ is precisely the multiplicity of ϕ as a factor of f. In particular, this is true whenever ϕ is an irreducible s-sparse polynomial of bounded individual degree d. With this observation in mind, we are now ready to describe the algorithm for finding multilinear factors. Suppose $f = \prod_{i=1}^{\ell} \phi_i^{e_i} \in \mathbb{F}[x_0, \mathbf{x}]$ where ϕ_i are factors of $(n+1, s, d)$-sparse polynomials. Suppose further that we have blackbox access to f, and we want to recover all of its irreducible multilinear factors. [2] Our algorithm follows the following steps:

1.4.2.1 Step 1 – Normalization and Recursion

In this step, we would like to replace our input polynomial f with another polynomial $\tilde{f}$ that still captures all the information about the factors of f, but is normalized (in the sense that its free term is 1 as a polynomial in x_0). To do that, we use a standard method, employed for example in [5, 2]. Let $f = \sum_{i=0}^{\ell d} f_i x_0^i$ where $f_i \in \mathbb{F}[\mathbf{x}]$ and let $\tilde{f}(y, x_1, \ldots, x_n) = f(f_0 y, x_1, \ldots, x_n)/f_0$.[3] Note that as we have blackbox access to f, we also have blackbox access to f_0 and therefore to $\tilde{f}$. It is straightforward to see that each multilinear factor $h_i(\mathbf{x}) = a_i(\mathbf{x})x_0 + b_i(\mathbf{x})$ of f now corresponds to a multilinear factor $\frac{a_i f_0}{b_i} y + 1$ of $\tilde{f}$. Thus, one sees that:

1. If $a_i \neq 0$, then all the information about a_i and b_i is captured in $\tilde{f}$.
2. If $a_i = 0$, then the factor b_i cannot be studied from $\tilde{f}$. However, note that in this case $b_i | f_0$, and therefore it can be studied from a factorization of f_0.

Note that f_0 has the same structure of f – it is itself a product of factors of s-sparse polynomials of individual degree d. Therefore, by applying one recursive call, we can find all irreducible multilinear factors of f_0. Thus, to understand the factors with $a_i = 0$, we just need to understand, for any multilinear irreducible factor ϕ of f_0, how many times it divides f. We will show a way to do so in the next step. For now, the main benefit from this step is the normalization $\tilde{f}$ and the computation of the multilinear factors ϕ_i of f_0.

---

[2] The general algorithm finds the multiquadratic, sparse, irreducible factors of the input. To keep the presentation here as simple as possible, we consider only the multilinear factors.

[3] It is easy to reduce to the case $f_0 \neq 0$, so to keep the presentation as simple as possible we assume it to be the case.

1.4.2.2 Step 2 – Obtaining the factorization

As we saw in the first step, there are two types of factors – those with $a_i = 0$ and those with $a_i \neq 0$. Thus, we split the algorithm into two parts:

1. Recovering the factors for which $a_i = 0$. In this case, it holds that $b_i | f_0$. Therefore, as we already saw, it suffices to determine which irreducible factors of f_0 divide f (and to what power). To that end, note that for a given sparse irreducible factor ϕ of f_0, it follows from the second property of G and from the fact that f is a product of elements of $\mathcal{P}_{\text{Factors}}(n+1, s, d)$, that to calculate the multiplicity of ϕ as a factor of f it suffices to check how many times $\phi(G_j)$ divides $f(G_j)$, where $x_j \in \text{var}(\phi)$. This can be easily done, for example, by factoring both polynomials, which is possible as they have a constant number of variables and polynomial degree.
2. Recovering the factors for which $a_i \neq 0$. In this case, using the first step, we can get access to $\frac{a_i(G) f_0(G)}{b_i(G)}$, by factoring $\tilde{f}(y, G)$. Recovering a_i from such an expression is the rational interpolation problem that we earlier alluded to. Here we have the additional information that b_i is a multilinear factor of f_0. Thus, to find b_i, it suffices to understand, for each multilinear irreducible factor ϕ of f_0, what is the maximal power of ϕ that divides b_i. For any such factor ϕ, using the fact that a_i and b_i are coprime (as otherwise $a_i x_0 + b_i$ will not be irreducible), we have that $\phi^k | b_i$ if and only if $\phi^{t-k+1} \nmid \frac{a_i f_0}{b_i}$, where t is the multiplicity of ϕ as a factor of f_0. It follows that we just have to understand the multiplicity of ϕ as a factor of $\frac{a_i f_0}{b_i}$. Since a_i is sparse and f_0 is a product of polynomials in $\mathcal{P}_{\text{Factors}}(n, s, d)$, we have that $\frac{a_i f_0}{b_i}$ is a product of polynomials in $\mathcal{P}_{\text{Factors}}(n, s, d)$ as well, so we can finish by computing the multiplicity of $\phi(G_j)$ as a factor of $\frac{a_i(G_j) f_0(G_j)}{b_i(G_j)}$ (where $x_j \in \text{var}(\phi)$). This recovers b_i and subsequently a_i as well, completing the algorithm.

In the above, we actually solved a rational interpolation problem. That is, we had access to $\frac{af}{b}$, where $b | f$, and needed to recover a and b from this information. We claim that in fact, the algorithm described above works in greater generality, as given by Theorem 34.

We conclude with the following remarks. First, we note that it is possible to modify our algorithm to recover all the multiquadratic irreducible factors of f, rather than just the multilinear ones. We further note that the algorithm described above still works when the assumption on the field $\mathbb{F}$ is removed, albeit with a slightly worse running time.

1.4.3 Finding Sparse Divisors of Sparse Polynomials

We now describe the ideas behind our main result (Theorem 9). Recall that our goal is to output all the sparse divisors of a given (n, s, d)-sparse polynomial f. That is, we wish to efficiently find those divisors of f that can be "written down" efficiently.

To obtain such an algorithm, a natural thing to do is to follow the lines of the algorithm presented in Section 1.4.2. The first step of this algorithm is performing a normalization and then applying a recursive call for factoring the free term f_0. But now, a problem arises – this recursive call only returns the irreducible sparse factors of f_0, while the free term of each factor of f (b_i in the language of Section 1.4.2) might be a *reducible* sparse divisor of f_0 whose irreducible factors are not sparse; in this case, the recursive call did not provide us with any valuable information about b_i. In particular, we cannot reconstruct b_i by studying how many times each sparse irreducible factor of f_0 divides it as we did in Section 1.4.2. In fact, the true reason the algorithm presented in Section 1.4.2 works is that in that context, b_i is sparse multilinear (or sparse multiquadratic in the more general case), and therefore so are its irreducible factors; hence, we indeed get information about b_i from the recursive call. In other words, the free terms there belong to a class of sparse polynomials that is closed under factorization.

This motivates considering the problem of finding all the *sparse divisors* of a sparse polynomial. This way, we will still have information about the free terms of the factors of f – the recursive call will actually compute them. Indeed, sparse divisors of sparse divisors of (n, s, d)-sparse polynomials are themselves sparse divisors of (n, s, d)-sparse polynomials, so this class is closed under factorization. This enables us to recover all these sparse divisors in deterministic $\text{poly}(n, s^{\text{poly}(d)})$-time. Indeed, the recursive call returns all the sparse divisors of the free term f_0 rather than just the set of irreducible factors of it, so we can directly brute-force over the list of these factors to find b_i. This yields Theorem 9.

At first glance, this approach may seem surprising, as the existence of such an algorithm suggests that the possible number of outputs is bounded above by $\text{poly}(n, s^{\text{poly}(d)})$. Indeed, Theorem 8 ensures that s-sparse polynomials of bounded individual degree d have no more than $d \log s$ irreducible factors (that are not monomials), and therefore, no more than s^d divisors (not divisible by a monomial) overall. To prove Theorem 8, we show that if $f \in \mathcal{P}(n, s, d)$ is not divisible by any monomial, then one can identify a set of $\log s$ variables that satisfy the property that every irreducible factor of f depends on at least one of them. From this claim it easily follows that f has at most $d \log s$ irreducible factors, and therefore at most $2^{d \log s} = s^d$ divisors overall. To prove that such a set of variables exist, we analyze the Newton polytope of f, and show that if the minimal set of variables satisfying the property is of size k, then the Newton polytope of f has at least 2^k vertices. This is somewhat intuitive, as each additional variable in the set effectively adds a dimension to the polytope and therefore doubles the number of its vertices.

▶ Remark 18. Theorem 8 is easily seen to be tight, for example by considering $f(x_1, \ldots, x_n) = \prod_{i=1}^{\log s}({x_i}^d - 1)$ over $\mathbb{C}$.

We note that Theorem 9 is not implied by the methods of [2]; indeed, there the bound on the sparsity of *all irreducible factors* of f is needed to hit all resultants of different factors; therefore, applying their methods to this problem must result in a running time of at least $s^{O(d^2 \log n)}$, which is their bound on the sparsity of the factors of f.

One main drawback of our algorithm is that it does not tell us what are the *irreducible* sparse divisors of the input polynomial, as we are not familiar with a way of efficiently identifying the irreducible polynomials from the outputs we get. On the other hand, if we are guaranteed that f is a product of irreducible sparse factors then we can easily find the irreducible factors among all the sparse divisors. This solves Question 3 for the special case of (n, s, d)-sparse polynomials.

These ideas are also used in the proof of Theorem 10. Recall that Question 4 asks for a deterministic, polynomial-time algorithm that given blackbox access to a product of (n, s, d)-sparse polynomials returns the factors. While we do not know a polynomial-time algorithm when the number of terms is unbounded, we show an algorithm that runs in quasi-polynomial-time in their number, making the first step towards a resolution of the problem. The algorithm is the natural, straightforward generalization of the algorithm described above. Unfortunately, it requires quasi-polynomial-time as we actually find all the s-sparse divisors of bounded individual degree d of f, and there are (possibly) quasi-polynomially many of them, as can be deduced from Theorem 8 to obtain the following corollary.

▶ **Corollary 19.** *Let f be a product of ℓ elements of $\mathcal{P}_{Factors}(n, s, d)$. The number of (n, s, d)-sparse divisors of f that are not divisible by a monomial is at most*

$$\sum_{i=0}^{d \log s} \binom{\ell d \log s}{i} \leq s^{d(2+\log \ell)}.$$

1.4.4 Factorization for Sparse Polynomials

Lastly, we discuss our algorithms for factoring (n, s, d)-sparse polynomials and their products. These algorithms are essentially the same as the algorithms we already presented; the only difference is that now, when dealing with elements of $\mathcal{P}_{\text{Factors}}(n, s, d)$, we view them as elements of $\mathcal{P}(n, s^{O(d^2 \log n)}, d)$.

Consider for example our algorithm for finding all the sparse divisors of an (n, s, d)-sparse polynomial. With a slight modification, we can make it output all the $(n, s^{O(d^2 \log n)}, d)$-sparse divisors of the input polynomial in $\text{poly}(n, s^{d^2 \log n})$-time. By [2], these are all the possible divisors. Identifying the irreducible factors among these can be easily done. For example, we can check for any two divisors ϕ and ψ if $\psi|\phi$ by trying to sparse-interpolate their quotient – which should be an $s^{O(d^2 \log n)}$-sparse polynomial.

As for our algorithm for factoring products of $\mathcal{P}_{\text{Factors}}(n, s, d)$ elements, it is identical to the algorithm described in Section 1.4.2, now taking into account that all irreducible factors of interest are $s^{O(d^2 \log n)}$-sparse, rather than s-sparse and multiquadratic.

1.5 Organization

In Section 2 we give some basic definitions and known results. Section 2.4 contains the construction of our genrator. Our meta algorithm is given in Section 3. Due to space constraints all the proofs are defered to the full version [6].

2 Preliminaries

We use $\mathbf{x}$ to denote the n-variate vector of variables, $\mathbf{x} = (x_1, \ldots, x_n)$. In general, bold-face letters such as $\mathbf{x}, \mathbf{e}$ will be used to denote n-tuples, where the domain of the n-tuple will always be clear from the context.

We say that two nonzero polynomials f, g are associate if one is a scalar multiple of the other. We also state the following special case of Gauss' lemma.

▶ **Definition 20.** *A polynomial $f \in \mathbb{F}[x_0, \mathbf{x}]$ is said to be reversed-monic with respect to x_0 if its free term as a polynomial in $\mathbb{F}(\mathbf{x})[x_0]$ is 1. Equivalently, f is reversed-monic with respect to x_0 if it is of the form $f = 1 + \sum_{i \geq 1} f_i x_0^i$ where $f_i \in \mathbb{F}[\mathbf{x}]$.*

▷ Claim 21. Let $f \in \mathbb{F}[x_0, \mathbf{x}]$ be a reversed-monic polynomial with respect to x_0. Then, the number of irreducible factors of f is at most $\deg_{x_0}(f)$.

▶ **Definition 22** (Normalization). *Let $f(x_0, \mathbf{x}) = \sum_{i=0}^{d} f_i(\mathbf{x}) x_0^i \in \mathbb{F}[x_0, \mathbf{x}]$, such that $x_0 \nmid f$. The normalization of f with respect to x_0 is the reverse monic polynomial*

$$\tilde{f}(y, \mathbf{x}) = f(y f_0, x_1, \ldots, x_n)/f_0 = 1 + \sum_{i=1}^{d} f_i(\mathbf{x}) f_0^{i-1}(\mathbf{x}) y^i. \tag{1}$$

▷ Claim 23. Let f as in Definition 22. Suppose $f = \prod_{i=0}^{N} h_i \in \mathbb{F}[x_0, \mathbf{x}]$ is the factorization of f into irreducible factors h_i, and denote each factor by $h_i = \sum_{j=0}^{d_i} c_{i,j} {x_0}^j$ (where $c_{i,j} \in \mathbb{F}[\mathbf{x}]$). Then, the factorization of $\tilde{f}$ is given by

$$\tilde{f} = \prod_{i=0}^{N} \tilde{h}_i \text{ , where } \quad \tilde{h}_i = \sum_{j=0}^{d_i} \frac{c_{i,j}}{c_{i,0}} {f_0}^j y^j \ .$$

Moreover, every factor (not necessarily irreducible) of f of the form $\sum_{j=0}^{d} \zeta_j {x_0}^j$ (where $\zeta_j \in \mathbb{F}[\mathbf{x}]$) corresponds to a factor of $\tilde{f}$ of the form $\sum_{j=0}^{d} \frac{\zeta_j}{\zeta_0} {f_0}^j y^j$. In particular, every multiquadratic factor of f of the form $a_2 x_0^2 + a_1 x_0 + b$ corresponds to a factor of $\tilde{f}$ of the form $1 + \frac{a_1 f_0}{b} y + \frac{a_2 f_0^2}{b} y^2$. In addition, we have blackbox access to $\tilde{f}$ from blackbox access to f.

2.1 Factorization and Interpolation of Polynomials

▶ **Theorem 24** (Univariate factorization over $\mathbb{Q}$). *Let $f \in \mathbb{Q}[x]$ have degree d and bit complexity B. Then, there is a deterministic algorithm that outputs all irreducible factors of f whose running time is* $\mathrm{poly}(d, B)$.

For the following theorem see e.g., [29, Section 9].

▶ **Theorem 25** (Univariate factorization over Finite Fields). *Let $\mathbb{F}_q$ be a field of size $q = p^k$ and characteristic $p > 0$. Let $f(x) \in \mathbb{F}_q[x]$ be a polynomial of degree d. Then, there is a deterministic algorithm that outputs all irreducible factors of f in time* $\mathrm{poly}(p, \log q, d)$.

We next state a theorem concerning deterministic multivariate factorization in the multivariate case. See e.g., [20, 17]. These results determine the dependence on the field, its characteristic and the bit complexity of the input polynomial, in all our algorithms that invoke polynomial factorization.

▶ **Theorem 26.** *There exists a deterministic algorithm that given an n-variate polynomial $f \in \mathbb{F}[\mathbf{x}]$ of total degree d, and bit complexity B, outputs its factorization into irreducible polynomials. Its running time is* $\mathrm{poly}(d^n) \cdot T(d, B, \mathbb{F})$, *where $T(d, B, \mathbb{F})$ is the running time of univariate factorization of degree d polynomials, with bit complexity B, over $\mathbb{F}$.*

▶ **Lemma 27.** *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be an n-variate polynomial of total degree d. Then, there is a deterministic polynomial-time algorithm that given blackbox access to f returns its monomial representation. Its running time is* $\mathrm{poly}((d+1)^n)$.

▶ **Theorem 28** ([2, Theorem 1]). $\mathcal{P}_{Factors}(n, s, d) \subset \mathcal{P}(n, s^{O(d^2 \log n)}, d)$. *In other words, every factor of an s-sparse polynomial of bounded individual degree d in n-variables is $s^{O(d^2 \log n)}$-sparse.*

▶ **Theorem 29** ([2, Theorem 2]). *There is a deterministic* $\mathrm{poly}(n, d^d, s^{d^7 \log n})$*-time algorithm, that given an (n, s, d)-sparse polynomial, returns its factorization into irreducible factors.*

▶ **Corollary 30** (Finding monomial divisors). *There is a deterministic* $\mathrm{poly}(n, d, s, \ell)$*-time algorithm that, given blackbox access to a product f of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d) \subset \mathbb{F}[\mathbf{x}]$, returns the multiplicity k_i of x_i as a factor of f. Moreover, if M is the largest monomial divisor of f, such that $f = Mg$, then we can obtain blackbox access to g from blackbox access to f.*

2.2 Primitive Divisors

▶ **Definition 31.** *Let $\mathcal{P} \subset \mathbb{F}[\mathbf{x}]$ be a class of polynomials. We say that a polynomial $g \in \mathbb{F}[\mathbf{x}]$ is a primitive divisor for $\mathcal{P}$ if it satisfies the following two conditions.*

1. *For every $f \in \mathcal{P}$ and every $t \in \mathbb{N}$, the polynomial* $\gcd(f, g^t)$ *is a power of g. Equivalently, for every $f \in \mathcal{P}$, there exists $k \geq 0$ and a polynomial $\tilde{g}$ coprime to g such that $f = g^k \cdot \tilde{g}$.*
2. *No non-scalar multiple of g satisfies the first condition.*

▶ **Definition 32.** *Let $f \in \mathcal{P}$, and let g be a primitive divisor for $\mathcal{P}$. The multiplicity of g as a factor of f, denoted by $v_g(f)$, is the power of g appearing in the factorization of f to primitive divisors.*

▶ **Lemma 33.** *There is a deterministic $\mathrm{poly}(n, (d^2)!, s^{d^3}, \ell)$-time algorithm that, given blackbox access to a product f of ℓ (n, s, d)-sparse polynomials, and blackbox access to a primitive divisor ϕ for $\mathcal{P}(n, s, d)$, returns the multiplicity of ϕ as a factor of f (in the sense of Definition 32). In fact, the algorithm only needs to know the values of f on the image of the generator $G_{(m)}$, for some explicit value $m = \mathrm{poly}(n, (d^2)!, s^{d^3})$. Moreover, if ϕ is an irreducible s-sparse polynomial, this algorithm works if f is a product of polynomials in $\mathcal{P}_{Factors}(n, s, d)$ (rather than $\mathcal{P}(n, s, d)$); if furthermore ϕ is multiquadratic, taking $m = \mathrm{poly}(n, (d^2)!, s^{d^2})$ suffices, and the running time reduces to $\mathrm{poly}(n, (d^2)!, s^{d^2}, \ell)$.*

2.3 Rational Interpolation Theorem

▶ **Theorem 34.** *Suppose we have blackbox access to a set of rational functions, defined over a field $\mathbb{F}$, of the form $\frac{a_j f^j}{b}$, for $1 \le j \le D$, where:*

1. *a_j is an (n, s, d)-sparse polynomial for every $1 \le j \le D$.*
2. *f is a product of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d)$ to which we have blackbox access.*
3. *We have a set $\mathcal{F}$ of s-sparse irreducible factors of f, call them ϕ_i.*
4. *b is an s-sparse factor of f whose irreducible factors are all elements of $\mathcal{F}$.*
5. *$\gcd(a_1, \ldots, a_D, b) = 1$.*

Then, there is an algorithm that recovers the a_j-s and b from evaluations of $\frac{a_j f^j}{b}$ on the image of the generator $G_{(m)}$, for some explicit m (depending on $\mathrm{char}(\mathbb{F})$), such that the running time is $\mathrm{poly}(m, \ell, D)$, where:

- *If $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > 2d$, one can take $m = \mathrm{poly}(n, d!, s^d, D)$.*
- *For arbitrary fields we have $m = \mathrm{poly}(n, (d^2)!, s^{d^3}, D)$. If we further assume that all elements of $\mathcal{F}$ are multiquadratic, then we can take $m = \mathrm{poly}(n, (d^2)!, s^{d^2}, D)$.*

2.4 Generator Construction

In this section, we construct our generator, which is a map $G : \mathbb{F}^6 \to \mathbb{F}^n$ that satisfies the properties described in Section 1.4.1.

Fix the following sets:

1. A set of integers $\mathcal{I} = \{1, \ldots, m\}$ of size m, that contains m different residues mod q.
2. A set of field elements $\mathcal{A} = \{\alpha_1, \ldots, \alpha_m\} \subset \mathbb{F}$ of size m. For every $1 \le i \le m$ let A_i be the Lagrange interpolating polynomials for $\mathcal{A}$.
3. A set of field elements $\mathcal{B} = \{\beta_1, \ldots, \beta_n\} \subset \mathbb{F}$ of size n. For every $1 \le i \le n$ let B_i be the Lagrange interpolating polynomials for $\mathcal{B}$.
4. A set of field elements $\mathcal{C} = \{\gamma_1, \ldots, \gamma_n\} \subset \mathbb{F}$ of size n. For every $1 \le i \le n$ let C_i be the Lagrange interpolating polynomials for $\mathcal{C}$.
5. A set of field elements $\mathcal{T} = \{\delta_1, \ldots, \delta_{ndq}\} \subset \mathbb{F}$ of size ndq.

To define our generator we first define the generator of Klivans and Spielman and state some of its properties.

▶ **Definition 35** (The KS-generator)**.** *In the notations above, the KS-generator is defined as*

$$G^{KS}_{(m)}(x, y, z, w) = \sum_{i=1}^{m} A_i(y) \left((1 + B_1(z)(w-1))x^{i \bmod q}, \ldots, (1 + B_n(z)(w-1))x^{i^n \bmod q} \right). \quad (2)$$

▶ **Theorem 36** ([18, Theorem 11]). *Let $f \in \mathbb{F}[\mathbf{x}]$ be an (n,s,d)-sparse polynomial. Let $m = (nds)^2$. Then, f can be reconstructed in time* $\mathrm{poly}(n,s,d)$ *from the following set of evaluations:*

$$\left\{ f\left(G^{KS}_{(m)}(x,y,z,w)\right) \mid \ y \in \mathcal{A},\ z \in \mathcal{B},\ x,w \in \mathcal{T} \right\}.$$

Another tool that we will use is the generator defined in [23], which is given by the formula

$$G^{\mathrm{SV}}(u,v) = (C_1(v)u, \ldots, C_n(v)u) \ . \tag{3}$$

The basic property of this generator is that it makes it possible to "revive" a variable of our choice. E.g., by setting $v = \gamma_j$ we get the vector that has zeros in all coordinates except of the j-th coordinate where we have the variable u.

We are now ready to define our generator.

▶ **Definition 37** (The generator $G_{(m)}$). *The generator $G_{(m)} : \mathbb{F}^6 \to \mathbb{F}^n$ is defined as*

$$G_{(m)}(x,y,z,w,u,v) = G^{KS}_{(m)}(x,y,z,w) + G^{SV}(u,v) \ . \tag{4}$$

We further define the generators obtained from reviving a variable. Denote with $G^{KS}_{(m)}(x,y,z,w)_i$ the i-th coordinate of $G^{KS}_{(m)}(x,y,z,w)$.

For any $1 \le i \le n$, let $G_{(m,i)}$ be the generator obtained from $G_{(m)}$ by restricting to $u = u - G^{KS}_{(m)}(x,y,z,w)_i$ and $v = \gamma_i$. That is,

$$G_{(m,i)}(x,y,z,w,u) = \left(G^{KS}_{(m)}(x,y,z,w)_1, \ldots, u, \ldots, G^{KS}_{(m)}(x,y,z,w)_n\right) \ . \tag{5}$$

Let's explain this definition. The first part $G^{\mathrm{KS}}_{(m)}$ enables us to interpolate (n,s,d)-sparse polynomials (Theorem 36).

3 Meta-Algorithm

In this section we present a meta-algorithm that will be used, with some variations, in the proofs of Theorems 9, 10, 11, 12, 13 and 14, which state our algorithmic results. Since Theorem 10 follows as a corollary of Theorem 11, we will not discuss it here.

Each of these algorithms follows the outline of the meta-algorithm, and in their proofs we will refer to steps of the meta-algorithm and describe how they are implemented in the relevant setting.

1. **Setting:**
   We assume that we want to learn properties of a polynomial f such that f is either
   a. (n,s,d)-sparse (Theorem 9, Theorem 12)
   b. a product of polynomials in $\mathcal{P}_{\mathrm{Factors}}(n,s,d)$ (Theorem 11, Theorem 13, Theorem 14)
   and we want to find a list $\mathcal{F}(f)$ of divisors of f having certain properties. For every variable x_i, the polynomial $f|_{x_i=0}$ is also of the same type as f. This is clearly true when f is (n,s,d)-sparse. When f is a product of polynomials in $\mathcal{P}_{\mathrm{Factors}}(n,s,d)$, this follows from Lemma 39, whose formulation and proof are given at the end of this section.
   In what follows, for the sake of convenience, we assume that f is an $n+1$-variate polynomial and is an element of $\mathbb{F}[x_0, \mathbf{x}]$.
2. **Preprocessing:**
   a. Remove from f its largest monomial divisor (using Corollary 30).

b. Normalize f to get $\tilde{f}(y, \mathbf{x}) = f(yf_0, x_1, \ldots, x_n)/f_0$. Set $\hat{f} = f(y, G)$ for an appropriate generator G. Observe that $\tilde{f}(y, \mathbf{x})$ is reverse monic with respect to y, and hence so is $\hat{f}$. By Claim 21 it then follows that $\hat{f}$ has at most $\deg_y(\hat{f}) \leq d$ irreducible factors if f is (n, s, d)-sparse, and at most $\deg_y(\hat{f}) \leq \ell d$ if f is a product of ℓ polynomials in $\mathcal{P}_{\text{Factors}}(n, s, d)$.

c. Use Lemma 27 to interpolate $\hat{f}$ as a polynomial in $O(1)$ many variables. Then apply Theorem 26 to obtain its factorization into irreducible factors.

3. **Recursive call to solve for f_0:** Recursively solve the problem for $f_0 = f|_{x_0=0}$, obtaining a list $\mathcal{F}(f_0)$ of divisors not divisible by a monomial or irreducible factors, according to the problem.
4. **Generating a list of candidate solutions:**
   By Claim 23, every divisor $h = \sum_{i=0}^{d} c_i x_0^i$ of f gives rise to a divisor $\sum_{i=0}^{d} \frac{c_i}{c_0} f_0^i y^i$ of $\tilde{f}$ and hence to a divisor $\hat{h} = \sum_{i=0}^{d} \frac{c_i(G)}{c_0(G)} f_0^i y^i$ of $\hat{f}$. We would like to get access to $c_i(G)$ in order to interpolate them, and through that obtain access to h. This leads us to the rational interpolation problem, where we have some information about c_0 (or even c_0 itself) coming from $\mathcal{F}(f_0)$.

   a. In Theorem 9, Theorem 11 we will have that $c_0 \in \mathcal{F}(f_0)$. In Theorem 12 c_0 is a product of elements of $\mathcal{F}(f_0)$, but we have enough time to enumerate all relevant products so we can pretend that $c_0 \in \mathcal{F}(f_0)$.
   In fact, the above statement is not entirely accurate. The recursive call returns the set $\mathcal{F}(f_0)$ of all divisors of f_0 that are not divisible by a monomial, together with the largest monomial dividing f_0. It might be the case that c_0 is a divisor of f_0 that *is* divisible by a monomial. In this case, c_0 will not belong to $\mathcal{F}(f_0)$. However, c_0 does lie in $\mathcal{F}(f_0)$ *up to a multiplication by a monomial*. As we shall see, this suffices for our purposes.

   b. In Theorem 13 and Theorem 14 the situation is a bit different. Here, c_0 may be a product of elements from $\mathcal{F}(f_0)$, and we cannot go over all possible ways to obtain c_0. However, in this case we will have that $\mathcal{F}(f_0)$ contains all the irreducible factors of c_0, which is what Theorem 34 requires.

   Accordingly, for each factor $\hat{h} = \sum_{i=0}^{d} e_i y^i$ we compose a list $\mathcal{F}_{\hat{h}}$ as follows:

   a. When we have a list containing (up to multiplication by a monomial) the correct c_0 (Theorem 9, Theorem 11, Theorem 12), we iterate over all possible candidates c_0 in that list. For each such candidate, we interpolate the polynomials of the form $\left((x_1 \cdots x_n)^d c_0 / f_0^i\right)(G)\, e_i$ in order to recover the coefficients c_i. We then add to $\mathcal{F}_{\hat{h}}$ those polynomials $\sum_{i=0}^{d} c_i x_0^i$ that are valid candidates for the problem, after dividing by their largest monomial divisor:
      i. In Theorem 9, Theorem 11 we keep all (n, s, d)-sparse polynomials.
      ii. In Theorem 12 we keep all (n, S, d)-sparse polynomials, where $S = s^{O(d^2 \log n)}$ is the upper bound on sparsity of factors of (n, s, d)-sparse polynomials obtained in Theorem 28.

      The term $(x_1 \cdots x_n)^d$ is needed to deal with the fact that c_0 is in $\mathcal{F}(f_0)$ only up to division by a monomial. We shall explain this point in full detail in the proof of Theorem 9.

   b. If we only have a list of irreducible factors of c_0 (Theorem 13 and Theorem 14), apply Theorem 34 on the coefficients of $\hat{h}$ to recover potential c_0 and $c_1, \ldots, c_d$. We put the resulting polynomial $\sum_{i=0}^{d} c_i x_0^i$ in the list $\mathcal{F}_{\hat{h}}$ if:
      i. For Theorem 13 it is (n, S, d)-sparse.
      ii. For Theorem 14, it is multiquadratic and (n, s, d)-sparse.

      We note that, in both cases, we set $\mathcal{F}_{\hat{h}} = \mathcal{F}(f_0)$ for $\hat{h} = 1$.

5. **Pruning the list:**
   Depending on the application we either have to keep divisors or irreducible divisors. We do that in two steps – we first find the divisors in the list of candidates, and then, if needed, identify the irreducible polynomials among them. The implementation of the two steps vary in the different algorithms; we give here a solution that works for all cases, though not always optimal, and provide a better implementation in the proofs of the algorithms.
   In all cases we apply Theorem 16 to remove from the list those polynomials that do not divide f. This already suffices for Theorem 9 and Theorem 11.
   To obtain irreducible divisors as required in Theorem 12, Theorem 13, Theorem 14, we apply Theorem 16 to each two polynomials g, h in the list of divisors to check whether g divides h, and keep only those divisors that are not divisible by any other divisor.
6. **Computing multiplicities:**
   To compute multiplicities, where needed, we apply Lemma 40 or Lemma 33, depending on the characteristic. Here as well there are better solutions in some cases; these are described in the proofs of the algorithms.

The analysis of the running time of the algorithm depends on the exact setting. However, in all settings we perform exactly one recursive call so the overall complexity is determined by the cost of the different steps of the algorithm, and it does not blow up because of the recursive call.

▶ Remark 38. The described meta-algorithm provides the outline that all our algorithms follow. It is, however, *not* an algorithm that *generalizes* all of them. It is possible to formulate two concrete algorithms which all our results follow – one generalizing Theorems 9, 11 and 12, and one generalizing Theorems 13 and 14. However, the precise formulations of these general algorithms are rather complicated and not particularly intuitive; hence, we have chosen not to include them in the present paper.

We close this section by stating the following lemma.

▶ **Lemma 39.** *Let f be a product of ℓ polynomials in $\mathcal{P}_{Factors}(n+1, s, d) \subset \mathbb{F}[x_0, \mathbf{x}]$. Suppose further that $x_0 \nmid f$, and let f_0 be the free term of f as a polynomial in x_0. Then, f_0 is a product of ℓ elements of $\mathcal{P}_{Factors}(n, s, d) \subset \mathbb{F}[\mathbf{x}]$.*

▶ **Lemma 40.** *Let $\mathbb{F}$ be a field of characteristic zero, or of characteristic larger than $2d$. There is a deterministic $\mathrm{poly}(n, d!, s^d, \ell)$-time algorithm that, given blackbox access to a product, f, of ℓ polynomials in $\mathcal{P}_{Factors}(n, s, d)$ and blackbox access to an irreducible polynomial ϕ in $\mathcal{P}_{Factors}(n, s, d)$, both defined over $\mathbb{F}$, returns the multiplicity of ϕ as a factor of f. In fact, the algorithm only needs to know the values of f and ϕ on the image of the generator $G_{(m)}$ for some explicit $m = \mathrm{poly}(n, d!, s^d)$.*

References

1 Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309, 1988.

2 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. doi:10.1145/3365667.

3 Somnath Bhattacharjee, Mrinal Kumar, Shanthanu S Rai, Varun Ramanathan, Ramprasad Saptharishi, and Shubhangi Saraf. Closure under factorization from a result of Furstenberg. *arXiv preprint arXiv:2506.23214*, 2025. doi:10.48550/arXiv.2506.23214.

4 Pranav Bisht and Nitin Saxena. Derandomization via symmetric polytopes: Poly-time factorization of certain sparse polynomials. *ACM Transactions on Computation Theory*, 17(2):1–20, 2025. doi:10.1145/3719022.

5 Pranav Bisht and Ilya Volkovich. On solving sparse polynomial factorization related problems. *Computational Complexity*, 34(1):7, 2025. doi:10.1007/S00037-025-00268-5.

6 Aminadav Chuyoon and Amir Shpilka. On factorization of sparse polynomials of bounded individual degree. *CoRR*, abs/2603.07589, 2026. doi:10.48550/arXiv.2603.07589.

7 Michael Clausen, Andreas Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theoretical Computer Science*, 84(2):151–164, 1991. doi:10.1016/0304-3975(91)90157-W.

8 Annie Cuyt and Wen-shin Lee. Sparse interpolation of multivariate rational functions. *Theoretical Computer Science*, 412(16):1445–1456, 2011. doi:10.1016/J.TCS.2010.11.050.

9 Pranjal Dutta and Sumanta Ghosh. Sigact news complexity theory column 121. *SIGACT News*, 55(2):53–88, June 2024. doi:10.1145/3674159.3674165.

10 Pranjal Dutta, Amit Sinhababu, and Thomas Thierauf. Derandomizing multivariate polynomial factoring for low degree factors. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, London School of Economics, London, UK, August 28-30, 2024*, volume 317 of *LIPIcs*, pages 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.75.

11 Michael A Forbes. Deterministic divisibility testing via shifted partial derivatives. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 451–465. IEEE, 2015. doi:10.1109/FOCS.2015.35.

12 Michael A Forbes and Amir Shpilka. Complexity theory column 88: Challenges in polynomial factorization1. *ACM SIGACT News*, 46(4):32–49, 2015. doi:10.1145/2852040.2852051.

13 Dima Grigoriev, Marek Karpinski, and Michael F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.*, 19(6):1059–1063, 1990. doi:10.1137/0219073.

14 Dima Grigoriev, Marek Karpinski, and Michael F Singer. Computational complexity of sparse rational interpolation. *SIAM Journal on Computing*, 23(1):1–11, 1994. doi:10.1137/S0097539791194069.

15 Ankit Gupta, Neeraj Kayal, and Satyanarayana V. Lokam. Reconstruction of depth-4 multilinear circuits with top fan-in 2. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 625–642. ACM, 2012. doi:10.1145/2213977.2214035.

16 Erich Kaltofen. Computing with polynomials given by straight-line programs I: greatest common divisors. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 131–142, 1985. doi:10.1145/22145.22160.

17 Erich Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. Comput.*, 14(2):469–489, 1985. doi:10.1137/0214035.

18 Adam R. Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, STOC '01, pages 216–223, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/380752.380801.

19 Mrinal Kumar, Varun Ramanathan, and Ramprasad Saptharishi. Deterministic algorithms for low degree factors of constant depth circuits. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3901–3918. SIAM, 2024. doi:10.1137/1.9781611977912.137.

20 Arjen Klaas Lenstra. Factoring multivariate polynomials over finite fields. *J. Comput. System Sci.*, 30(2):235–248, 1985. doi:10.1016/0022-0000(85)90016-9.

21 Nitin Saxena. Progress on polynomial identity testing. *Bull. EATCS*, 99:49–79, 2009.

22 Nitin Saxena. Progress on polynomial identity testing-ii. In *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 131–146. Springer, 2014.

23 Amir Shpilka and Ilya Volkovich. Improved polynomial identity testing for read-once formulas. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 700–713. Springer, 2009. doi:10.1007/978-3-642-03685-9_52.

24 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010. doi:10.1561/0400000039.

25 Joris van der Hoeven and Grégoire Lecerf. On sparse interpolation of rational functions and gcds. *ACM Commun. Comput. Algebra*, 55(1):1–12, 2021. doi:10.1145/3466895.3466896.

26 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)*, pages 943–958. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.943.

27 Ilya Volkovich. On some computations on sparse polynomials. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*, pages 48–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.48.

28 Joachim Von Zur Gathen and Erich Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, 1985. doi:10.1016/0022-0000(85)90044-3.

29 Joachim von zur Gathen and Victor Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2(3):187–224, 1992. doi:10.1007/BF01272074.