

# Deterministic Algorithms for Low Individual Degree Factors of Sparse Polynomials

Somnath Bhattacharjee\*   Rishabh Kothary†   Shanthanu S. Rai‡   Shubhangi Saraf§

## Abstract

We study factoring algorithms for general sparse polynomials and sparse polynomials of bounded individual degree and prove the following results.

1. We give a deterministic polynomial-time algorithm which takes as input an  $n$ -variate  $s$ -sparse polynomial  $f$  of bounded individual degree  $d$  and outputs a list  $\mathcal{L}$  of circuits which contains all factors of  $f$ , although there might be additional spurious circuits in the list. The algorithm runs in time  $\text{poly}(n, s^d)$ . Additionally, we can ensure that every circuit in the list has constant depth. Our algorithm works over all fields of characteristic 0 or sufficiently large characteristic.

Our result generalizes a recent result of Chuyoon and Shpilka [CS26] that gives a  $\text{poly}(n, s^d)$ -time algorithm for recovering all *sparse* factors of  $f$  (without spurious factors). As a corollary, we can also recover *all* factors of  $f$  (without spurious factors) in time  $\text{poly}(n, s^{d^2 \log n})$ , and recover the algorithmic result of Bhargava, Saraf and Volkovich [BSV20] and its improvement by Chuyoon and Shpilka [CS26]. Both the above consequences follow from known interpolation and divisibility testing techniques.

2. We give a deterministic quasipolynomial-time algorithm which takes as input a general  $n$ -variate  $s$ -sparse polynomial  $f$  of (unbounded) individual degree  $D$  and outputs a list  $\mathcal{L}$  of polynomials which contains all factors of  $f$  that have bounded individual degree  $d$ . The algorithm runs in time  $\text{poly}(D^{d \log s}, s^{d^2 \log n})$  and works over arbitrary fields. The list may again contain spurious elements. Our result strengthens results of Dutta, Sinhababu and Thierauf [DST24] and Kumar, Ramanathan and Saptharishi [KRS24] which give algorithms to recover all factors of  $f$  of bounded total degree.

A consequence of our algorithm is a new upper bound on the total number of bounded individual degree factors of a sparse polynomial.

---

\*University of Toronto, Canada. Email: [somnath.bhattacharjee@mail.utoronto.ca](mailto:somnath.bhattacharjee@mail.utoronto.ca). Research partially supported by an NSERC Discovery Grant.

†University of Toronto, Canada. Email: [rishabh.kothary@mail.utoronto.ca](mailto:rishabh.kothary@mail.utoronto.ca). Research partially supported by an NSERC Discovery Grant.

‡Tata Institute of Fundamental Research, Mumbai, India. Email: [shanthanu.rai@tifr.res.in](mailto:shanthanu.rai@tifr.res.in). Research supported by the Department of Atomic Energy, Government of India, under project number RTI4014, TCS Research Fellowship and in part by research grants from Google Research, Premji Invest and SERB.

§University of Toronto, Canada. Email: [shubhangi.saraf@utoronto.ca](mailto:shubhangi.saraf@utoronto.ca). Research partially supported by an NSERC Discovery Grant and a McLean Award.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Our Results . . . . .	4
1.2	Proof Overview . . . . .	7
1.2.1	Sparse Polynomials of Bounded Individual Degree . . . . .	8
1.2.2	Bounded Individual Degree Factors of General Sparse Polynomials . . . . .	12
1.3	Organization . . . . .	14
<b>2</b>	<b>Preliminaries</b>	<b>14</b>
2.1	Notation . . . . .	15
2.2	Algebraic circuits . . . . .	16
2.3	Deterministic Univariate Factorization . . . . .	19
2.4	Sylvester Matrix, Resultant, and Discriminant . . . . .	19
2.5	Factors of Normalized and Reverse-Normalized Polynomials . . . . .	21
2.6	Power Series Roots of Monic Polynomials . . . . .	22
2.7	Sparse Polynomials . . . . .	25
<b>3</b>	<b>Factoring Normalized and Reverse Normalized Sparse Polynomials</b>	<b>26</b>
<b>4</b>	<b>Recovering <i>all</i> Factors of Sparse Polynomials of Bounded Individual Degree</b>	<b>32</b>
<b>5</b>	<b>Recovering Bounded Individual Degree Factors of General Sparse Polynomials</b>	<b>39</b>
<b>6</b>	<b>Future Directions and Open Questions</b>	<b>47</b>
<b>A</b>	<b>Omitted Proofs from Section 2.4</b>	<b>50</b>
<b>B</b>	<b>Omitted Proofs from Section 2.6</b>	<b>51</b>
B.1	Proof of Hensel’s Lemma . . . . .	51
B.2	Proof of Theorem 2.28 . . . . .	52
<b>C</b>	<b>Omitted Proofs from Section 2.7</b>	<b>55</b>

# 1 Introduction

Polynomial factorization is a central problem in computational algebra and algebraic complexity, with connections to coding theory, cryptography, pseudorandomness and derandomization. We refer the reader to the recent survey of Bhargav, Dwivedi, and Saxena [BDS25] for a detailed account.

Foundational work by Kaltofen [Kal89], and also Kaltofen and Trager [KT90], showed that multivariate polynomial factorization can be performed efficiently by randomized algorithms when the input is given by an algebraic circuit. These works also established the remarkable structural fact that factors of low-degree polynomials computed by small algebraic circuits themselves have small algebraic circuits. Even prior to these works, in the sparse setting, von zur Gathen and Kaltofen gave randomized algorithms for factoring sparse multivariate polynomials, with running time depending polynomially on the sparsity of the input and of the output factors [vzGK85]. Some very natural questions that arise are the following: can such algorithms be derandomized, and can one prove strong upper bounds on the complexity of factors of sparse polynomials?

By the work of Kopparty, Saraf and Shpilka [KSS15], we know that for general algebraic circuits, derandomizing multivariate factorization is essentially *equivalent* to derandomizing polynomial identity testing. Since this work, there has been significant progress on understanding if this equivalence extends to other subclasses of algebraic circuits, and this work will focus on the class of sparse polynomials.

In a recent work, Bhattacharjee, Kumar, Rai, Ramanathan, Saptharishi and Saraf proved closure under factorization for formulas and constant-depth circuits, and using this closure, extended the connection of derandomizing PIT and derandomizing factoring to these restricted but fundamental algebraic models [BKR<sup>+</sup>25b, BKR<sup>+</sup>26]. Thus, for several natural circuit classes, deterministic factorization and PIT are now known to be tightly linked. Despite this progress, we are still quite far from understanding whether this equivalence is true for sparse polynomials. On the one hand, sparse polynomials are among the simplest algebraic models, and efficient deterministic PIT and reconstruction algorithms for them have been known since the work of Klivans and Spielman [KS01] and related earlier works [GK87, BOT88, GKS90, CDGK91]. On the other hand, we are still far from obtaining a deterministic polynomial-time algorithm for factoring general sparse polynomials. There has nevertheless been significant progress on important special cases.

For constant-depth circuits, and in particular for sparse polynomials, Kumar, Ramanathan, Saptharishi and Volk gave a deterministic subexponential-time algorithm that outputs a list of candidate circuits containing all relevant irreducible factors, but the list may also contain spurious circuits [KRSV26]. This issue of having spurious circuits in the list was overcome in the constant-depth (and hence also sparse) setting by [BKR<sup>+</sup>25b], who gave a deterministic subexponential-time algorithm that outputs all irreducible factors of a constant-depth circuit, together with their multiplicities. Their algorithm yields, in particular, the first subexponential-time deterministic algorithm for factoring sparse polynomials, since sparse polynomials are depth-two circuits. The subsequent closure-under-factorization result of [BKR<sup>+</sup>26] gave a cleaner structural explanation and strengthened the connection between PIT and factorization for constant-depth circuits and formulas. However, these general constant-depth algorithms are still subexponential, and do not exploit the full special structure of sparse polynomials.

Kumar, Ramanathan and Saptharishi, and independently Dutta, Sinhababu and Thierauf, gave quasipolynomial-time deterministic algorithms for recovering all constant-degree factors of sparse polynomials [KRS24, DST24]<sup>1</sup>. However, to recover higher degree factors, very little is known.

---

<sup>1</sup>These works also do more. [KRS24] gave a subexponential-time algorithm for outputting all the constant-degree factors of constant-depth circuits. [DST24] showed that computing all the constant-degree factors of a polynomial in a class  $\mathcal{C}$  reduces to PIT for the class  $\mathcal{C}$  and divisibility tests of the polynomial by constant-degree polynomials.

Indeed, in the setting of sparse polynomials, other than subexponential algorithms of [BKR<sup>+</sup>25b, BKR<sup>+</sup>26], we know nothing better even for the task of finding multilinear factors. The following is an open question from [DST24]: *Can we find bounded individual degree factors of a sparse polynomial (that may not have bounded individual degree) in deterministic quasi-polynomial time?*

A more tractable and very natural setting is that of factoring sparse polynomials of bounded individual degree. Here the input polynomial is sparse, and in addition the exponent of each variable is at most some bounded parameter  $d$ . The cases  $d = 1$  and  $d = 2$  were handled by Shpilka and Volkovich and by Volkovich, respectively, who gave efficient deterministic factorization algorithms for sparse multilinear and sparse multiquadratic polynomials [SV10, Vol17]. For general constant  $d$ , Bhargava, Saraf and Volkovich gave the first quasipolynomial-time deterministic algorithm for factoring sparse polynomials of bounded individual degree [BSV20].

A key ingredient in their work was a structural theorem: if an  $s$ -sparse  $n$ -variate polynomial has individual degree at most  $d$ , then every factor has sparsity at most  $s^{O(d^2 \log n)}$ . The algorithmic framework by [BSV20] has the useful feature that an improved sparsity bound would immediately yield an improved factoring algorithm; in particular, a polynomial sparsity bound would imply a polynomial-time deterministic factorization algorithm in this setting.

Further evidence for this conjectural polynomial sparsity bound was given by Bisht and Volkovich, who obtained polynomial-time algorithms for several problems that would follow from such a bound, including identity testing for certain depth-four circuits and testing whether a sparse polynomial is an exact power [BV25].

Very recently, Chuyoon and Shpilka made further progress and gave deterministic polynomial-time algorithms for finding all sparse divisors of bounded-individual-degree sparse polynomials, and more generally developed algorithms for recovering sparse factors in several related settings [CS26]. This shows that, at least if one is only interested in factors that are themselves sparse, polynomial-time deterministic algorithms are possible in the bounded-individual-degree regime. However, the full factorization problem remains elusive, since the currently known bounds do not rule out factors of sparse polynomials (with bounded individual degree) having super-polynomially many monomials.

## 1.1 Our Results

In this work, we make progress toward deterministic polynomial-time factorization of sparse polynomials of bounded individual degree, as well as the problem of finding bounded individual degree factors of general sparse polynomials.

We first state our results informally.

1. Our first main result gives a deterministic polynomial-time algorithm which, on input an  $s$ -sparse polynomial of individual degree at most  $d$ , outputs a list of candidate factors that contains all factors of the input polynomial that are not divisible by a monomial. The list may also contain spurious candidates. Thus our result is in the spirit of the list-producing algorithm of Kumar, Ramanathan, Saptharishi and Volk [KRSV26] for constant-depth circuits, but in the bounded-individual-degree sparse setting we obtain a polynomial-time algorithm rather than a subexponential-time one.

Our algorithm also recovers and strengthens some of the earlier mentioned results. In the bounded-individual-degree setting, efficient reconstruction and divisibility testing for sparse polynomials allow us to recover the polynomial-time sparse-divisor algorithms of Chuyoon and Shpilka [CS26], as well as the quasipolynomial-time factorization algorithm of Bhargava, Saraf and Volkovich [BSV20].

2. In the setting of general sparse polynomials, we give a quasipolynomial-time deterministic algorithm that outputs a list containing all factors of individual degree at most  $d$ , again allowing the possibility of spurious candidates.

Thus we make progress towards the aforementioned open question of [DST24]. Our results essentially show that factoring in this setting reduces to divisibility testing. If we could efficiently test whether a sparse polynomial of bounded individual degree divides another sparse polynomial, then we would be able to prune out the list and output only the true factors.

When specialized to factors of constant total degree, known reconstruction and divisibility-testing tools [For15] allow the list to be pruned, thus recovering the quasipolynomial-time algorithms of Kumar, Ramanathan and Saptharishi [KRS24] and of Dutta, Sinhababu and Thierauf [DST24] for constant-degree factors of sparse polynomials.

We hope that our results are a useful step toward the long-standing goal of deterministic polynomial-time factorization of sparse polynomials. We now state our results more formally.

**Factoring sparse polynomials of bounded individual degree:** As our first result, we give a deterministic polynomial-time algorithm which takes as input an  $n$ -variate  $s$ -sparse polynomial  $f$  of bounded individual degree  $d$  and outputs a list  $\mathcal{L}$  of circuits which contains all factors of  $f$  that are not divisible by a monomial. The algorithm runs in time  $\text{poly}(n, s^d)$ .

The list may contain spurious circuits which do not correspond to factors of  $f$ . Additionally, we can ensure that every circuit in the list has constant depth.

**Theorem 1.1** (Main Theorem 1). *Let  $\mathbb{F}$  be any field such that  $\text{char}(\mathbb{F})$  is 0 or greater than  $d$ . Let  $f(X_1, \dots, X_n) \in \mathbb{F}[X_1, \dots, X_n]$  be an  $s$ -sparse polynomial with individual degree at most  $d$ . Then there is an algorithm that runs in time  $\text{poly}(s^d, d!, n)$  and outputs an  $s^d$ -size list of circuits  $\mathcal{L}$  such that every factor of  $f$  that is not divisible by a monomial is computed by some circuit in  $\mathcal{L}$ . The algorithm also outputs the maximal monomial divisor of  $f$ .*

*Moreover, each circuit in  $\mathcal{L}$  has size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ , and if  $|\mathbb{F}| > \text{poly}(n, s^d)$  the circuits are over  $\mathbb{F}$ .*

**Remark 1.2.** The above runtime suppresses polynomial dependence on the running time of factoring degree  $d$  univariate polynomials over  $\mathbb{F}$ . Over  $\mathbb{Q}$ , this is just polynomial in  $d$  and the input bit complexity. However, for fields of characteristic  $p$ , there is also a polynomial dependence on  $p$ . See Theorem 4.1 and Remark 4.2 for a more detailed statement.

**Remark 1.3.** If  $|\mathbb{F}| < \text{poly}(n, s^d)$ , we just work over a suitable field extension that makes the size sufficiently large for our computations. In this case, the output circuits will be over the field extension.

**Remark 1.4.** The algorithm outputs only factors not divisible by a monomial. This is unavoidable, since if the product of  $X_1^d \cdots X_n^d$  and  $g$  divides a sparse polynomial  $f$ , then  $X_1^{i_1} \cdots X_n^{i_n} g$  also divides  $f$  for all  $0 \leq i_1, \dots, i_n \leq d$ , yielding  $(d+1)^n$  factors. The omitted monomial part is easy to recover: for  $f = \sum_{j=1}^s c_j X_1^{\alpha_{j,1}} \cdots X_n^{\alpha_{j,n}}$ , the maximal monomial divisor is  $\prod_{i=1}^n X_i^{\min_{j \in [s]} \alpha_{j,i}}$ .

**Remark 1.5.** Our algorithm implicitly gives an upper bound on the total number of factors of an  $s$ -sparse polynomial of bounded individual degree that are not divisible by a monomial. Such an upper bound was also obtained by Chuyoon and Shpilka [CS26] using different techniques.

Given the above theorem, as a special case one can recover the recent result of Chuyoon and Shpilka [CS26]<sup>2</sup>, which gives a  $\text{poly}(s^d, d!, n)$ -time algorithm to return all  $s$ -sparse factors of an  $s$ -sparse polynomial with individual degree at most  $d$ . Indeed, given the list of circuits produced by our algorithm, we can run a sparse polynomial interpolation algorithm (say by [KS01]) on each circuit to recover the sparse polynomials computed by them. We then test each interpolated polynomial for divisibility against the input sparse polynomial using the divisibility testing algorithm designed by [Vol17]. This retains precisely the  $s$ -sparse divisors in the list and rejects all remaining candidates.

Another immediate corollary is a quasipolynomial-time deterministic algorithm to recover all factors of any  $s$ -sparse polynomial  $f$  with bounded individual degree  $d$ . [BSV20] showed that every factor of  $f$  has quasipolynomial sparsity  $\text{poly}(n, s^{d^2 \log n})$ . Using this, they obtained an algorithm that ran in time  $\text{poly}(n, s^{d^2 \log n})$ . This was improved to  $\text{poly}(n, s^{d^2 \log n})$  in [CS26].

We can recover this latter bound on running time as a corollary of Theorem 1.1. Indeed, by the quasipolynomial sparsity bound of [BSV20], every factor of  $f$  appears in the output list and has only  $\text{poly}(n, s^{d^2 \log n})$  monomials if expressed in the monomial representation. As in the sparse-factor case discussed above, we run (quasipolynomial) sparse polynomial interpolation on each circuit in the list, and then use divisibility testing to retain precisely those interpolated polynomials that divide the input polynomial  $f$ . This yields all factors of  $f$  within the same  $\text{poly}(n, s^{d^2 \log n})$  running time.

We formally prove the above corollaries in Corollary 4.5 (also see Remark 4.6).

**Factoring general sparse polynomials** Note that when the input polynomial  $f$  has bounded individual degree  $d$ , all factors of  $f$  automatically have bounded individual degree  $d$ . We also provide efficient factoring algorithms for the case of *general* sparse polynomials with no bound on individual degree. However, we only show how to recover bounded individual degree factors of  $f$ . As a first step, we show that number of factors of bounded individual degree of a general  $s$ -sparse polynomial cannot be too large.

In the theorem below, it will be useful to think of  $D$  as a large parameter, and  $d$  as a small fixed constant or slowly growing parameter. In the statement below, we use  $\binom{D}{\leq d}$  to denote the expression  $\sum_{i=0}^d \binom{D}{i}$ . Note that this quantity is upper bounded by  $D^{O(d)}$ .

**Theorem 1.6** (Divisor bound for general sparse polynomials). *Let  $f$  be any  $n$ -variate  $s$ -sparse polynomial of individual degree at most  $D$ . Then  $f$  can have at most  $\binom{D}{\leq d}^{\log s}$  factors of individual degree at most  $d$  that are not divisible by any monomial. Moreover, this bound is tight.*

We prove this theorem formally in Section 5. Prior to this work, no such nontrivial bound was known even for the number of multilinear factors.

We also give a deterministic quasipolynomial-time algorithm which takes as input a general multivariate  $s$ -sparse polynomial  $f$  of individual degree at most  $D$  and outputs a list  $\mathcal{L}$  of polynomials which contains all factors of  $f$  that have individual degree at most  $d$ . The list may contain additional spurious elements.

Again, in the theorem below, it will be useful to think of  $D$  as a large parameter, and  $d$  as a small fixed constant or slowly growing parameter.

**Theorem 1.7** (Main Theorem 2). *Let  $\mathbb{F}$  be an arbitrary field. Let  $f(X_1, \dots, X_n) \in \mathbb{F}[X_1, \dots, X_n]$  be an  $s$ -sparse polynomial with individual degree at most  $D$ . Then there is an algorithm that runs in time  $\text{poly}(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n)$  and outputs a  $\binom{D}{\leq d}^{\log s}$ -size list of polynomials  $\mathcal{L}$  such that every factor of  $f$  that is not divisible by a monomial and that has individual degree at most  $d$  lies in  $\mathcal{L}$ .*

<sup>2</sup>The [CS26] result holds over all fields.

**Remark 1.8.** The above runtime suppresses polynomial dependence on the running time of factoring degree  $D$  univariate polynomials over  $\mathbb{F}$ . Over  $\mathbb{Q}$ , this is just polynomial in  $D$  and the input bit complexity. However, for fields of characteristic  $p$ , there is also a polynomial dependence on  $p$ . See Theorem 5.1 and Remark 5.2 for a more detailed statement.

**Remark 1.9.** It follows from our algorithm that additionally each polynomial in  $\mathcal{L}$  has at most  $\text{poly}(s^{d^2 \log n})$  monomials.

**Remark 1.10.** If we only wanted to recover a list containing all  $s'$ -sparse factors of individual degree at most  $d$ , then our algorithm runtime can be improved to  $\text{poly}(s', \binom{D}{\leq d}^{\log s}, n)$ . By the factor sparsity bound of [BSV20], every factor of  $f$  of individual degree at most  $d$  has sparsity at most  $s^{\mathcal{O}(d^2 \log n)}$ . Thus, setting  $s' = s^{\mathcal{O}(d^2 \log n)}$  recovers the parameters in Main Theorem 2.

Theorem 1.7 makes progress towards an open problem by Dutta, Sinhababu and Thierauf [DST24], who asked if all *bounded individual degree* (sparse) factors of a general sparse polynomial can be found in deterministic quasipolynomial-time. We make progress on this problem by outputting a list of all bounded individual degree factors, but with the caveat that the list may contain spurious elements. Our result in particular also makes progress towards answering a question of Volkovich [Vol15] about finding multilinear factors of a general sparse polynomial.

Theorem 1.7 reduces the problem of computing all bounded-individual-degree factors of a sparse polynomial to divisibility testing. Indeed, if one can deterministically test whether an input sparse polynomial of bounded individual degree divides a general sparse polynomial, then one can run this test on every polynomial in the output list and prune the list down to exactly the true factors.

An interesting special case of the above theorem is the problem of recovering factors of constant total degree. In this setting, the quasipolynomial-time divisibility tests of Forbes [For15] allow us to prune the spurious candidates from our list and output exactly the constant-total-degree factors. This recovers the results of Dutta, Sinhababu and Thierauf [DST24] and Kumar, Ramanathan and Saptharishi [KRS24] in the setting of factoring sparse polynomials. Indeed, if the divisibility test can be done in polynomial-time, then a simple modification of our algorithm will give a polynomial-time factoring algorithm.

## 1.2 Proof Overview

Recall that [BSV20] gives a deterministic quasipolynomial-time algorithm for factoring  $n$ -variate  $s$ -sparse polynomials of bounded individual degree  $d$ , with running time  $\text{poly}(n, s^{d^7 \log n})$ . More recently, this was improved in [CS26] to  $\text{poly}(n, s^{d^2 \log n})$ . Moreover, and perhaps more significantly, the latter work gives a deterministic *polynomial-time* algorithm to output all  $s$ -sparse factors of such polynomials in time  $\text{poly}(s^d, d!, n)$ .

Since our algorithms are inspired by and build upon the algorithms in [BSV20] and [CS26], it will be instructive to see high-level sketches of these algorithms, and to understand where they fail for the more general task we are attempting. Thus through the proof overview we will discuss our techniques in conjunction with those from [BSV20] and [CS26].

The algorithms in [BSV20], [CS26], and the algorithms from our main theorems can all be viewed as a composition of the following steps: (0) a structural theorem, (1) normalization to obtain a monic polynomial, (2) factoring the normalized polynomial, (3) a recursive call to a simpler polynomial, and (4) recovering factors of the original polynomial.

Before we delve into a discussion of these steps, we first set up some notation.

We use  $\mathbf{X}_n$  to denote the tuple of variables  $(X_1, \dots, X_n)$  and often just abbreviate it to  $\mathbf{X}$ .

Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  be an  $s$ -sparse polynomial of individual degree  $d$  and suppose that

$$f(\mathbf{X}, Y) = f_k Y^k + f_{k-1} Y^{k-1} + \cdots + f_1 Y + f_0,$$

where for each  $i \in \{0, \dots, k\}$ , we have  $f_i \in \mathbb{F}[\mathbf{X}]$  and  $f_k \neq 0$ .

### 1.2.1 Sparse Polynomials of Bounded Individual Degree

We first give a proof sketch of our first main result (Theorem 1.1).

**Step 0: Structural theorem.** The starting point of [BSV20] is a structural theorem. [BSV20] shows, using tools from discrete geometry and the theory of Newton polytopes, that if an  $n$ -variate polynomial  $f$  of individual degree  $d$  has  $s$  monomials, then any factor of  $f$  can have at most  $s^{O(d^2 \log n)}$  monomials.

In [CS26], again using the theory of Newton polytopes, it is shown that if an  $n$ -variate polynomial  $f$  of individual degree  $d$  has  $s$  monomials, then the total number of factors (and in particular sparse factors) not divisible by a monomial is at most  $s^d$ . In order to efficiently output all sparse factors, it is crucial for the analysis in [CS26] to show that the number of sparse factors is polynomially bounded.

In our proof of Theorem 1.1, we will give a new and implicit proof of the [CS26] factor bound. The bound on the output list size and number of factors will follow naturally from the analysis of the algorithm.

As in the formal theorem statement, all factor bounds in this discussion are for factors that are not divisible by a monomial. This restriction is unavoidable if one wants a polynomial-size output list: monomial multiples of a fixed factor can already give exponentially many divisors. Algorithmically, we handle this by outputting the maximal monomial divisor of the input separately; the recursive factor-finding procedure only has to recover the non-monomial part.

Our analysis of Theorem 1.7 will also implicitly imply the divisor bound for general sparse polynomials, i.e., Theorem 1.6.

**Step 1: Normalization** A standard preprocessing step in the factorization literature is a normalization procedure that converts the input polynomial to one that is monic in some chosen variable, say  $Y$ , i.e., whose leading coefficient is 1 (or reverse-monic, i.e., whose constant term is 1).

Let the *normalization* of  $f(\mathbf{X}, Y)$  be the monic polynomial

$$\hat{f}(\mathbf{X}, Y) = f\left(\mathbf{X}, \frac{Y}{f_k}\right) f_k^{k-1}.$$

Assuming  $f_0 \neq 0$ , the *reverse normalization*<sup>3</sup> of  $f(\mathbf{X}, Y)$  is the reversed-monic polynomial

$$\tilde{f}(\mathbf{X}, Y) = \frac{f(\mathbf{X}, Y f_0)}{f_0}.$$

[BSV20] deals with normalization and [CS26] deals with reverse normalization, but either step has essentially the same effect: the resulting polynomial has no factor independent of  $Y$ . Since the degree in  $Y$  is at most  $d$  and  $\hat{f}$  is monic,  $\hat{f}$  has at most  $d$  irreducible factors (counted with multiplicity). So, all its factors can be obtained by multiplying subcollections of these irreducible

---

<sup>3</sup>This operation is called normalization in [CS26]. We call it reverse normalization to distinguish it from the monic normalization above: it scales the constant coefficient in  $Y$  to 1, rather than the leading coefficient.

factors. Though the entire analysis in [CS26] is presented with respect to reverse normalization, it could just as easily and equivalently have been presented with respect to normalization, and in the rest of the overview, for simplicity, we will assume [CS26] uses normalization.

A crucial aspect of our algorithms, both for Theorem 1.1 and for Theorem 1.7, is that we will make use of *both* normalization and reverse normalization.

**Step 2: Factoring the normalized polynomial.** The algorithm in [BSV20] shows how to completely factor  $\hat{f}(\mathbf{X}, Y)$  in quasipolynomial time. The algorithm in [CS26] aims only to recover those factors that correspond to *sparse* factors of the original polynomial  $f$ , and shows how to do this in polynomial time. There are some subtleties in making this formal, but we will not go into the details here.

Our algorithm (for Theorem 1.1) shows how to find a  $2^d$ -size list of circuits that contains *all* factors of  $\hat{f}(\mathbf{X}, Y)$  in polynomial time. One caveat is that the list may contain some circuits that do not represent factors, and unfortunately we do not know how to detect these spurious circuits and prune them out.

For now, just to get a bird's-eye view of the entire algorithm, let us not go into details of the monic factorization, but assume that it can be done. We first describe how this can be used to recover all factors of  $f(\mathbf{X}, Y)$  efficiently.

Note that for our analysis to work, we will also need to have an efficient way of finding a  $2^d$ -size list of circuits that contains *all* factors of  $\tilde{f}(\mathbf{X}, Y)$  (the reverse-normalized polynomial) in polynomial time, and let us for now assume that this can be done as well.

**Step 3: Recursively factoring  $f_k(\mathbf{X})$  or  $f_0(\mathbf{X})$ .** Both [BSV20] and [CS26] recursively factor  $f_k(\mathbf{X})$  and then show how these factors can be combined with the factors of  $\hat{f}(\mathbf{X}, Y)$  to obtain the factors of  $f(\mathbf{X}, Y)$ .

We will need to be able to factor *either*  $f_k(\mathbf{X})$  or  $f_0(\mathbf{X})$ , and show how these factors can be combined with the factors of  $\hat{f}(\mathbf{X}, Y)$  or  $\tilde{f}(\mathbf{X}, Y)$  to recover factors of  $f(\mathbf{X}, Y)$ . The reason for keeping both options is that one of  $f_k$  and  $f_0$  has sparsity at most half the sparsity of  $f$ , and recursing on the sparser one will keep the recursion depth to at most  $\log s$ .

**Step 4: From factors of  $\hat{f}(\mathbf{X}, Y)$  (or  $\tilde{f}(\mathbf{X}, Y)$ ) and  $f_k(\mathbf{X})$  (or  $f_0(\mathbf{X})$ ) to factors of  $f(\mathbf{X}, Y)$ .** We will use the fact that factors of  $f(\mathbf{X}, Y)$  that depend on the variable  $Y$  are closely related to factors of  $\hat{f}(\mathbf{X}, Y)$ . Indeed, for every factor  $g(\mathbf{X}, Y)$  of  $Y$ -degree  $r$  of  $f(\mathbf{X}, Y)$ , if

$$g(\mathbf{X}, Y) = g_r Y^r + g_{r-1} Y^{r-1} + \cdots + g_1 Y + g_0,$$

then the following polynomial

$$\hat{g}(\mathbf{X}, Y) = g \left( \mathbf{X}, \frac{Y}{f_k(\mathbf{X})} \right) \cdot \frac{f_k(\mathbf{X})^r}{g_r(\mathbf{X})}$$

is monic in  $Y$  and it can be shown by Gauss' Lemma that it is a factor of  $\hat{f}(\mathbf{X}, Y)$ . Moreover  $g_r(\mathbf{X})$  is a factor of  $f_k(\mathbf{X})$ , since the leading coefficient (in  $Y$ ) of  $g(\mathbf{X}, Y)$  divides the leading coefficient (in  $Y$ ) of  $f(\mathbf{X}, Y)$ .

Thus to recover all factors of  $f(\mathbf{X}, Y)$ , it suffices to learn factors of  $\hat{f}(\mathbf{X}, Y)$  and factors of  $f_k(\mathbf{X})$  and suitably *combine* them.

Indeed,

$$g(\mathbf{X}, Y) = \hat{g}(\mathbf{X}, Y \cdot f_k(\mathbf{X})) \cdot \frac{g_r(\mathbf{X})}{f_k(\mathbf{X})^r}$$

The factors of  $f_k(\mathbf{X})$  will contain all factors of  $f(\mathbf{X}, Y)$  that do not depend on  $Y$ , as well as factors that can be combined with factors of  $\hat{f}(\mathbf{X}, Y)$  to recover factors of  $f(\mathbf{X}, Y)$  that do depend on  $Y$ . In some sense this is what both [BSV20] and [CS26] also do.

However, one key step that both [BSV20] and [CS26] can perform is *testing* whether the purported factor returned by the combining step is indeed a true factor. Thus their algorithms can prune out all the spurious circuits and keep only the true factors. Indeed this notion of keeping only true factors can be attained at each step of the recursive procedure, and hence the final list size output by the algorithm is upper bounded by the total number of irreducible factors (in [BSV20]) and the total number of factors<sup>4</sup> (in [CS26]).

In our case, since the testing or pruning step is something we do not know how to do, *even if we knew an upper bound on the total number of factors*, following the above algorithmic scheme would not let us control the size of the output list. This is because there could potentially be  $n$  recursive calls, and at each recursive step the list size grows by a multiplicative factor of  $2^d$ . This would make the final list size output by the algorithm exponential in  $n$ .

This is where our ability to choose between normalization and reverse-normalization becomes useful: it lets us decide whether to recurse on  $f_k$  or on  $f_0$ .

In past algorithms, when the algorithm was recursively factoring  $f_k(\mathbf{X})$ , one reason  $f_k(\mathbf{X})$  was a simpler polynomial was that it depended on one fewer variable. One could have also carried out a very similar scheme and recursed on  $f_0(\mathbf{X})$ , again a polynomial on one fewer variable. However, notice that if  $f(\mathbf{X}, Y)$  has  $s$  monomials, then since  $f_k(\mathbf{X})$  and  $f_0(\mathbf{X})$  contribute to distinct monomials in  $f$ , one of them has sparsity at most  $s/2$ . Thus, if at each step the algorithm chooses to recurse on the polynomial with lower sparsity, then the recursion depth is at most  $\log s$ .

This also gives the informal divide-and-conquer bound on the list size. At each recursive level, the factoring of the normalized or reverse-normalized polynomial contributes a factor of at most  $2^d$  to the list size, while the sparsity of the polynomial on which we recurse drops by a factor of two. Thus, at the level of this overview, one should think of the recurrence as

$$L(s) \leq 2^d \cdot L(s/2),$$

and hence  $L(s) \leq (2^d)^{\log s} = s^d$ . This is the basic reason our algorithm produces an  $s^d$ -size list, and also why it implicitly recovers the corresponding bound on the number of non-monomial factors.

For completeness, we spell out the steps needed if the algorithm chooses to recurse on  $f_0(\mathbf{X})$ .

For this, note that factors of  $f(\mathbf{X}, Y)$  that depend on the variable  $Y$  are also closely related to factors of the reverse-monic polynomial  $\tilde{f}(\mathbf{X}, Y)$ . Indeed, for every factor  $g(\mathbf{X}, Y)$  of  $Y$ -degree  $r$  of  $f(\mathbf{X}, Y)$ , if

$$g(\mathbf{X}, Y) = g_r Y^r + g_{r-1} Y^{r-1} + \cdots + g_1 Y + g_0,$$

then the following polynomial

$$\tilde{g}(\mathbf{X}, Y) = g(\mathbf{X}, Y \cdot f_0(\mathbf{X})) \cdot \frac{1}{g_0(\mathbf{X})}$$

is reverse-monic in  $Y$  and it can be shown that it is a factor of  $\tilde{f}(\mathbf{X}, Y)$ . Moreover  $g_0(\mathbf{X})$  is a factor of  $f_0(\mathbf{X})$ .

Thus to recover all factors of  $f(\mathbf{X}, Y)$ , it suffices to learn factors of  $\tilde{f}(\mathbf{X}, Y)$  and factors of  $f_0(\mathbf{X})$  and suitably *combine* them.

Indeed,

$$g(\mathbf{X}, Y) = \tilde{g}\left(\mathbf{X}, \frac{Y}{f_0(\mathbf{X})}\right) \cdot g_0(\mathbf{X})$$

---

<sup>4</sup>More precisely, only those not divisible by a monomial.

The factors of  $f_0(\mathbf{X})$  will contain all factors of  $f(\mathbf{X}, Y)$  that do not depend on  $Y$ , as well as factors that can be combined with factors of  $\tilde{f}(\mathbf{X}, Y)$  to recover factors of  $f(\mathbf{X}, Y)$  that do depend on  $Y$ .

Strictly speaking, the formal algorithm first produces monomial-equivalent candidates: for every desired factor  $g$ , the recursive list contains some monomial multiple of  $g$ . A final post-processing step removes the largest monomial divisor from each candidate. This does not change the above list-size recurrence, but it is the reason the formal proof is phrased in terms of monomial-equivalent factors before the final output list is produced.

**Details of Step 2: Factoring the normalized polynomial** The algorithm in [BSV20] shows how to completely factor  $\hat{f}(\mathbf{X}, Y)$ . Recall that the algorithm from [BSV20] runs in quasipolynomial time and returns monomial representations of all factors. The proof in [BSV20] is in the style of the Kaltofen-Trager [KT90, Kal89] factoring algorithm and we omit the details. A crucial step is to find a projection to a univariate polynomial just in  $Y$  that keeps coprime factors of the original polynomial coprime even after the projection. For this, [BSV20] needs to set the  $\mathbf{X}$  variables to a value  $\alpha$  which keeps the resultant (in  $Y$ ) of every pair of irreducible factors nonzero. They show that this can be done in quasipolynomial time since the irreducible factors have sparsity at most  $s^{O(d^2 \log n)}$  and have  $Y$ -degree at most  $d$ , and the low degree of the  $Y$  variable and the sparsity of the factors imply that these resultants are also quasipolynomially sparse.

We would like to run in *polynomial time* and recover *all* factors, though possibly with some spurious factors. We show how, in polynomial time, to output a  $2^d$ -size list of constant-depth circuits such that each factor of  $\hat{f}(\mathbf{X}, Y)$  is a member of this list.

At a high level, our factoring algorithm first projects  $\hat{f}(\mathbf{X}, Y)$  to a univariate polynomial in  $Y$ , finds the roots of the univariate projection, lifts these roots to power series roots of the shifted polynomial, and then combines these roots to recover the various potential factors of  $\hat{f}(\mathbf{X}, Y)$ .

A well-known lemma, namely *Hensel's lemma*, says that if  $\hat{f}$  is monic in  $Y$  and satisfies some suitable conditions, then  $\hat{f}$  splits completely into power series roots (see Lemma 2.26).

Hensel's lemma (Lemma 2.26) essentially says that any polynomial  $P(\mathbf{X}, Y)$  that is monic in  $Y$  admits a factorization into power series roots if it satisfies the following two conditions:

1. For every irreducible factor  $g(\mathbf{X}, Y)$  of  $P(\mathbf{X}, Y)$ ,  $g(\mathbf{0}, Y)$  is square-free.
2. For every pair of distinct irreducible factors  $g_1(\mathbf{X}, Y)$  and  $g_2(\mathbf{X}, Y)$  of  $P$ ,  $g_1(\mathbf{0}, Y)$  and  $g_2(\mathbf{0}, Y)$  are coprime.

Our algorithm for factoring will first find an  $\alpha$  such that  $\hat{f}_\alpha(\mathbf{X}, Y) := \hat{f}(\mathbf{X} + \alpha, Y)$  satisfies the above conditions. For this, we essentially need an assignment  $\alpha \in \mathbb{F}^n$  such that, after shifting by  $\alpha$  and then setting  $\mathbf{X} = \mathbf{0}$ , all irreducible factors of  $\hat{f}$  remain square-free and pairwise coprime.

In the special case where  $\hat{f}(\mathbf{X}, Y)$  is square-free, it is well known that any  $\alpha$  which hits the discriminant (i.e., keeps it nonzero) of  $\hat{f}$  (see Definition 2.16) will work here. The discriminant in this case is  $(2d)!s^{d^2}$ -sparse, and hence such an  $\alpha$  can be found using PIT algorithms for sparse polynomials. In the general case, it turns out that it suffices to find an  $\alpha$  that *maintains the rank of the Sylvester matrix* of  $\hat{f}$  and  $\partial_Y \hat{f}$ . Naively this again leads to PIT for  $(2d)!s^{d^2}$ -sparse polynomials, but we use the relation between the Sylvester matrices of  $f$  and  $\hat{f}$  to reduce the relevant hitting-set computation to polynomials of sparsity  $(2d)!s^d$ . This is what keeps the running time polynomial in  $s^d$  (instead of  $s^{d^2}$ ). In a slightly different context (for divisibility testing of products of sparse polynomials), the underlying rank-preservation criterion was already observed in [CS26] and earlier in [BV25]; moreover, these works observed that finding such a projection can be reduced to finding a hitting set for a certain sparse polynomial.

Once we find this  $\alpha$ , we consider  $\hat{f}_\alpha(\mathbf{X}, Y) := \hat{f}(\mathbf{X} + \alpha, Y)$ . We will compute all factors of  $\hat{f}_\alpha(\mathbf{X}, Y)$  and then recover all factors of  $\hat{f}(\mathbf{X}, Y)$  by undoing the shift.

Note that this shifting will destroy the sparsity of  $\hat{f}$ , but to find the power series roots, we actually do not need the sparsity assumption any more; it was only needed to find the  $\alpha$ .

Once we have the shifted polynomial  $\hat{f}_\alpha$ , there are standard ways, such as Newton iteration, to compute these power series roots up to any desired degree precision. For our purposes, however, we use Furstenberg's theorem [Fur67], which was recently used by [BKR<sup>+</sup>26] in the context of factorization of constant-depth circuits. The advantage is that Furstenberg's theorem gives additional structure on the power series roots of  $\hat{f}(\mathbf{X}, Y)$ , which will be useful to conclude that our final output circuits have constant depth<sup>5</sup>.

We will now see why we get a list of  $2^d$  candidate factors.

First we note that  $\hat{f}_\alpha(\mathbf{X}, Y)$  has  $d$  ( $= \deg_Y(\hat{f})$ ) power series roots, counted with multiplicity. In other words, there are power series  $\varphi_1(\mathbf{X}), \dots, \varphi_d(\mathbf{X}) \in \overline{\mathbb{F}}[[\mathbf{X}]]$  such that we can write

$$\hat{f}_\alpha(\mathbf{X}, Y) = \prod_{i=1}^d (Y - \varphi_i(\mathbf{X})).$$

Moreover, any factor  $\hat{g}_\alpha(\mathbf{X}, Y)$  of  $\hat{f}_\alpha(\mathbf{X}, Y)$  is of the form  $\prod_{j \in S} (Y - \varphi_j(\mathbf{X}))$  for some subset  $S \subseteq [d]$ ,

where repeated roots are listed with their multiplicities. Hence to find any factor  $\hat{g}_\alpha$ , it is enough to find the set of power series  $\{\varphi_i\}$  up to sufficiently high precision, combine the appropriate subsets of these roots, and then truncate to the required degree. Since there are at most  $2^d$  such subsets, this gives the desired  $2^d$ -size list.

### 1.2.2 Bounded Individual Degree Factors of General Sparse Polynomials

We now describe the ideas that go into the algorithm for Theorem 1.7.

Recall that in this setting the input is a general  $n$ -variate  $s$ -sparse polynomial  $f(\mathbf{X}, Y)$  where each variable has individual degree  $D$ . We do not assume  $D$  is bounded or constant. However, the goal is to output all factors of  $f(\mathbf{X}, Y)$  that have bounded individual degree  $d$ .

**Step 0: Structural theorem.** In order to have an efficient algorithm for outputting all bounded-individual-degree factors (not divisible by a monomial) of a general sparse polynomial, one needs to show that the number of such factors is small. We show that any such polynomial  $f(\mathbf{X}, Y)$  can have at most  $\binom{D}{\leq d}^{\log s}$  such factors, and indeed this bound is tight. Instead of separately proving this bound, the bound will naturally follow from the structure of the algorithm.

**Step 1: Normalization** Just as before, we will have a normalization or reverse normalization step.

**Step 2: Factoring the normalized (or reverse-normalized) polynomial** The main issue here is that the monic polynomial  $\hat{f}(\mathbf{X}, Y)$  now might have large or unbounded  $Y$ -degree. In the previous algorithm, the boundedness of the  $Y$ -degree was crucial. It allowed us to argue about the sparsity of the discriminant of  $\hat{f}(\mathbf{X}, Y)$ , and this allowed us to efficiently find a projection of the  $\mathbf{X}$  variables that preserved the coprimality of factors. This is what made factoring via power-series

---

<sup>5</sup>We can obtain constant-depth circuits for each homogeneous component of the power series roots. One can obtain a general circuit by using standard Newton iteration.

methods, such as Newton iteration or Furstenberg’s theorem, effective. Another issue is that  $\hat{f}(\mathbf{X}, Y)$  may not even be sparse any more. These challenges are significant, and indeed we do not try to recover all factors of  $\hat{f}(\mathbf{X}, Y)$ . We aim for a more relaxed goal: recovering only those factors that are somewhat sparse (at most  $\text{poly}(s^{d^2 \log n})$  monomials) and have  $Y$ -degree at most  $d$ .

The reason we only need to focus on somewhat sparse polynomials is that by the sparsity bound of [BSV20], any factor of  $f(\mathbf{X}, Y)$  that has individual degree at most  $d$  can have at most  $s^{O(d^2 \log n)}$  monomials. Moreover, it is not hard to see that any monic (in  $Y$ ) polynomial of degree  $D$  can have at most  $\binom{D}{\leq d}$  factors of  $Y$ -degree at most  $d$ .

To achieve this relaxed goal, we use a key idea from the algorithm of [CS26]: applying the Klivans-Spielman generator [KS01] for sparse polynomials. We will use this generator in the parameter settings needed for PIT and reconstruction for polynomials of sparsity  $s^{O(d^2 \log n)}$ .

In essence, this is a variable-reduction procedure that has the net effect of “preserving” all the information in the sparse factors. We map  $\hat{f}(\mathbf{X}, Y)$  to  $\hat{f}(\mathcal{G}(\mathbf{Z}), Y)$ , a polynomial with a constant number of variables. We then factor the constant-variate polynomial  $\hat{f}(\mathcal{G}(\mathbf{Z}), Y)$  and “lift” the factors by interpolation to recover all potential *sparse* factors of  $\hat{f}(\mathbf{X}, Y)$ .

Note that since only the sparse factors are being recovered, we do not need any sophisticated factoring algorithm for large-variate polynomials. It suffices to factor constant-variate polynomials and have a way of interpolating and recovering any sparse factor  $\hat{g}(\mathbf{X}, Y)$  of  $\hat{f}(\mathbf{X}, Y)$  from the corresponding factor  $\hat{g}(\mathcal{G}(\mathbf{Z}), Y)$  of  $\hat{f}(\mathcal{G}(\mathbf{Z}), Y)$ .

There is a subtle issue with the above plan. To make the whole factoring scheme work, it will not be enough to recover sparse factors of  $\hat{f}(\mathbf{X}, Y)$ ; we will also need to do more. We elaborate on this in Step 4 of the algorithm.

**Step 3: Recursively factor  $f_k(\mathbf{X})$  or  $f_0(\mathbf{X})$ .** Just as before, we will either recursively factor  $f_k(\mathbf{X})$  or  $f_0(\mathbf{X})$  depending on their sparsity.

**Step 4: From factors of  $\hat{f}(\mathbf{X}, Y)$  (or  $\tilde{f}(\mathbf{X}, Y)$ ) and  $f_k(\mathbf{X})$  (or  $f_0(\mathbf{X})$ ) to factors of  $f(\mathbf{X}, Y)$ .** Let us assume for now that in Step 3, we recursed on  $f_k(\mathbf{X})$  and thus we need to find factors of  $\hat{f}(\mathbf{X}, Y)$ .

This step has most of the same components as in the previous algorithm, but with some crucial differences.

One significant difference is that in Step 2, we recover a larger list of  $\binom{D}{\leq d}$  potential factors of  $\hat{f}(\mathbf{X}, Y)$ , and this list is not guaranteed to contain all factors of  $\hat{f}(\mathbf{X}, Y)$ , but only those that are somewhat sparse (at most  $\text{poly}(s^{d^2 \log n})$  monomials) and have  $Y$ -degree at most  $d$ . When we recursively factor  $f_k(\mathbf{X})$ , we are guaranteed to recover all factors (not divisible by a monomial) that have individual degree at most  $d$ . We need to show that this suffices for recovering all factors of  $f(\mathbf{X}, Y)$  of individual degree at most  $d$ .

The issue is that a bounded-individual-degree factor  $g$  of  $f$  does not necessarily give rise to a sparse factor  $\hat{g}$  of the normalized polynomial, even when  $g$  itself is sparse.

Let  $g(\mathbf{X}, Y)$  be a factor of  $f(\mathbf{X}, Y)$  that has individual degree at most  $d$ . As a first step, by the sparsity bound of [BSV20],  $g(\mathbf{X}, Y)$  must be somewhat sparse: it can have at most  $s^{O(d^2 \log n)}$  monomials. If  $g(\mathbf{X}, Y)$  does not depend on  $Y$ , then it will be output by the recursive call on  $f_k(\mathbf{X})$ . Let us therefore assume that  $g(\mathbf{X}, Y)$  depends on  $Y$  and has  $Y$ -degree  $r$ , where  $r \leq d$ . If

$$g(\mathbf{X}, Y) = g_r Y^r + g_{r-1} Y^{r-1} + \cdots + g_1 Y + g_0,$$

then

$$\hat{g}(\mathbf{X}, Y) = g\left(\mathbf{X}, \frac{Y}{f_k(\mathbf{X})}\right) \cdot \frac{f_k(\mathbf{X})^r}{g_r(\mathbf{X})}$$

is monic in  $Y$ , and, as we argued before, is a factor of  $\hat{f}(\mathbf{X}, Y)$ .

The sparsity of  $\hat{g}(\mathbf{X}, Y)$  is controlled by the sparsity of  $g(\mathbf{X}, Y)$  except for the division by  $g_r(\mathbf{X})$ . Because of this division, we do not have a useful upper bound on the sparsity of  $\hat{g}(\mathbf{X}, Y)$ . Thus the algorithm cannot write out  $\hat{g}(\mathbf{X}, Y)$ , even though the use of generators gives us access to  $\hat{g}(\mathcal{G}(\mathbf{Z}), Y)$ .

Instead of interpolating and recovering  $\hat{g}(\mathbf{X}, Y)$ , we apply the combining step to  $\hat{g}(\mathcal{G}(\mathbf{Z}), Y)$  and  $g_r(\mathcal{G}(\mathbf{Z}))$  (by iterating over all factors of  $f_k(\mathbf{X})$ ) to recover  $g(\mathcal{G}(\mathbf{Z}), Y)$ . We then use interpolation to recover  $g(\mathbf{X}, Y)$ , which is guaranteed to be sparse.

Thus, instead of first recovering  $\hat{g}(\mathbf{X}, Y)$  and then recovering  $g(\mathbf{X}, Y)$  by the following combining step with  $g_r(\mathbf{X})$ ,

$$g(\mathbf{X}, Y) = \hat{g}(\mathbf{X}, Y \cdot f_k(\mathbf{X})) \cdot \frac{g_r(\mathbf{X})}{f_k(\mathbf{X})^r}$$

we combine  $\hat{g}(\mathcal{G}(\mathbf{Z}), Y)$  with  $g_r(\mathcal{G}(\mathbf{Z}))$  to obtain

$$g(\mathcal{G}(\mathbf{Z}), Y) = \hat{g}(\mathcal{G}(\mathbf{Z}), Y \cdot f_k(\mathcal{G}(\mathbf{Z}))) \cdot \frac{g_r(\mathcal{G}(\mathbf{Z}))}{f_k(\mathcal{G}(\mathbf{Z}))^r}$$

Once we have  $g(\mathcal{G}(\mathbf{Z}), Y)$ , since  $g(\mathbf{X}, Y)$  has sparsity at most  $s^{O(d^2 \log n)}$ , we can interpolate to recover  $g(\mathbf{X}, Y)$ . This idea of doing the combining step in conjunction with the use of generators is inspired by the algorithms in [CS26], although in a different setting of parameters.

Note that for the wrong choice of  $g_r(\mathbf{X})$ , we will recover a sparse polynomial which does not correspond to a true factor, and these will contribute to the spurious elements in our list.

There is one final issue caused by monomial factors. In the formal algorithm, we first output a list of monomial-equivalent candidates: for every desired factor  $g$ , the list contains some monomial multiple of  $g$ . A final post-processing step then removes the largest monomial divisor from each candidate. Since at each recursive step we recurse on the smaller of  $f_k$  and  $f_0$ , the recursion has depth at most  $\log s$ . Together with the  $\binom{D}{\leq d}$  possible bounded- $Y$ -degree factors considered at each level, this gives the claimed list-size bound, and hence also the divisor bound.

### 1.3 Organization

We begin in Section 2 by summarizing some well-known results on algebraic circuits, polynomial factorization, and sparse polynomials. Readers familiar with these notions may skip directly to Section 3 and return to Section 2 as needed. In Section 3, we show how to factor normalized and reverse-normalized polynomials. In Section 4, we present our first main algorithm, which recovers all factors of sparse polynomials of bounded individual degree. Finally, in Section 5, we present our second main algorithm, which recovers all bounded-individual-degree factors of general sparse polynomials.

## 2 Preliminaries

**Bit-complexity convention:** We will summarize several standard algorithms that take polynomials and circuits as input. Over fields of characteristic 0, the running time of these algorithms also depends polynomially on the bit complexity  $B$  of the coefficients and constants appearing in the input polynomials and circuits. Over finite fields, our algorithms have a polylogarithmic dependence on the field size. For brevity and ease of exposition, we suppress this dependence throughout.

**Convention on field size:** For the rest of the paper we will assume that the field we are working over is sufficiently large. If not, we work over a suitable extension.

## 2.1 Notation

Throughout this paper,  $[n]$  denotes the set  $\{1, 2, \dots, n\}$ . We use  $\mathbf{X}_n$  to denote the tuple of variables  $(X_1, \dots, X_n)$ . When the number of variables is clear from context, we abbreviate  $\mathbf{X}_n$  to  $\mathbf{X}$ . We say a polynomial  $f(\mathbf{X}_n)$  has individual degree  $d$  if for each  $i \in [n]$ , the degree in  $X_i$  is at most  $d$ .

For parameters  $a_1, \dots, a_k$ , we write  $\text{poly}(a_1, \dots, a_k)$  to denote a quantity bounded by  $(a_1 + \dots + a_k)^c$  for some absolute constant  $c$ .

For field elements or rational functions  $a_1, \dots, a_m$ , we write  $\text{Diag}(a_1, \dots, a_m)$  for the  $m \times m$  diagonal matrix with diagonal entries  $a_1, \dots, a_m$ .

When a polynomial is given explicitly as a sum of monomials, its *bit complexity* is the total number of bits needed to write down its coefficients together with the exponents appearing in its sparse representation.

We will often work with a list of algebraic circuits (defined in Section 2.2), say  $\mathcal{L}$ . For a polynomial  $g$ , we sometimes abuse notation and write  $g \in \mathcal{L}$  to mean that there exists a circuit in  $\mathcal{L}$  that computes  $g$ .

We will use  $\text{Esym}_r(S)$  for a finite subset  $S$  to represent the elementary symmetric polynomial of degree  $r$  evaluated on the elements of  $S$ .

For any polynomial (or formal power series)  $P(T)$ ,  $P(T) \text{ Trunc } T^{D+1}$  will denote the polynomial obtained by discarding all terms of degree at least  $D + 1$  in  $T$ .

**Definition 2.1.** Let  $g, h \in \mathbb{F}[\mathbf{X}]$  and  $r \in \mathbb{N}$ . We write  $g \prec_r h$  if there exists a monomial  $M$  whose individual degree is at most  $r$  such that  $h = gM$ . In this case, we say that  $h$  is *r-monomial equivalent* to  $g$ . When the parameter  $r$  is clear from context, we simply say that  $h$  is monomial equivalent to  $g$ .

Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  and suppose

$$f(\mathbf{X}, Y) = f_d Y^d + f_{d-1} Y^{d-1} + \dots + f_1 Y + f_0,$$

where for each  $i \in \{0, \dots, d\}$ , we have  $f_i \in \mathbb{F}[\mathbf{X}]$  and  $f_d \neq 0$ .

- $f(\mathbf{X}, Y)$  is said to be *monic in Y* if  $f_d = 1$ .
- $f(\mathbf{X}, Y)$  is said to be *reversed-monic in Y* if  $f_0 = 1$ .
- The *normalization* of  $f(\mathbf{X}, Y)$  is the monic polynomial

$$\hat{f}(\mathbf{X}, Y) = f\left(\mathbf{X}, \frac{Y}{f_d}\right) f_d^{d-1}.$$

- Assuming  $f_0 \neq 0$ , the *reverse normalization* of  $f(\mathbf{X}, Y)$  is the reversed-monic polynomial

$$\tilde{f}(\mathbf{X}, Y) = \frac{f(\mathbf{X}, Y f_0)}{f_0}.$$

- The *reversal* of  $f$  in  $Y$ , denoted by  $\text{rev}_Y(f)$ , is defined as

$$\text{rev}_Y(f) = Y^d f(\mathbf{X}, 1/Y) = f_0 Y^d + f_1 Y^{d-1} + \dots + f_d.$$

We note below that the reversal of a product equals the product of the reversals. Under the mild assumption that the constant term in  $Y$  is non-zero, this lets us transfer irreducibility and factorization statements between a polynomial and its reversal.

**Lemma 2.2.** *Let  $f(\mathbf{X}, Y), g(\mathbf{X}, Y), h(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$ . If  $f = gh$ , then*

$$\text{rev}_Y(f) = \text{rev}_Y(g)\text{rev}_Y(h).$$

**Corollary 2.3.** *Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$ , viewed as a polynomial in  $\mathbb{F}(\mathbf{X})[Y]$ , and assume  $f_0 \neq 0$ . Then  $f$  is irreducible in  $\mathbb{F}(\mathbf{X})[Y]$  if and only if  $\text{rev}_Y(f)$  is irreducible in  $\mathbb{F}(\mathbf{X})[Y]$ .*

**Corollary 2.4.** *Let  $f = g_1^{e_1} g_2^{e_2} \cdots g_k^{e_k}$  be the irreducible factorization of  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  in  $\mathbb{F}(\mathbf{X})[Y]$ . If each factor  $g_i$  has non-zero constant term in  $Y$ , then*

$$\text{rev}_Y(f) = \text{rev}_Y(g_1)^{e_1} \text{rev}_Y(g_2)^{e_2} \cdots \text{rev}_Y(g_k)^{e_k}$$

*is the irreducible factorization of  $\text{rev}_Y(f)$ .*

## 2.2 Algebraic circuits

Algebraic circuits are a standard way to compactly represent multivariate polynomials in algebraic complexity theory.

**Definition 2.5.** An *algebraic circuit*  $C$  over a field  $\mathbb{F}$  is a directed acyclic graph whose internal gates are labeled by  $+$  or  $\times$ , and whose leaves are labeled by variables or elements of  $\mathbb{F}$ . We allow gates to have arbitrary fan-in. The circuit computes the polynomial obtained in the natural way by evaluating gates from the leaves to the output gate.

The *size* of  $C$ , denoted  $\text{size}(C)$ , is the number of wires in  $C$ . The *depth* of  $C$ , denoted  $\text{depth}(C)$ , is the length of the longest directed path from an input leaf to the output gate.

A *circuit with division gates* is defined similarly, except that internal gates may also be labeled by  $\div$ . Such a circuit computes a rational function over  $\mathbb{F}$ ; in some applications below, this rational function will be promised to equal a polynomial.

We will also use interpolation to recover coefficients with respect to a single variable while preserving efficient circuit representations. The following standard lemma is due to Michael Ben-Or.

**Lemma 2.6** (Interpolation). *Let  $C$  be an algebraic circuit of size  $m$  and depth  $\Delta$  computing a polynomial*

$$f(\mathbf{X}, Y) = \sum_{i=0}^d f_i(\mathbf{X})Y^i,$$

*where each  $f_i(\mathbf{X}) \in \mathbb{F}[\mathbf{X}]$ . Suppose  $|\mathbb{F}| > d$ . Then, given  $C$ , one can construct circuits  $C_0, \dots, C_d$  computing  $f_0, \dots, f_d$ , respectively, each of size  $\text{poly}(m, d)$  and depth at most  $\Delta + 2$ . Moreover, these circuits can be constructed in time  $\text{poly}(m, d)$ .*

We will routinely truncate a polynomial computed by a circuit by discarding its higher-degree terms in one of the variables. We formalize this notation below and record that such truncations can be computed efficiently.

**Definition 2.7** (Truncation). For a polynomial  $f(\mathbf{X}, Z) \in \mathbb{F}[\mathbf{X}, Z]$  and an integer  $D \geq 0$ , we denote by  $f \text{ Trunc } Z^D$  the polynomial obtained from  $f$  by discarding all monomials whose degree in  $Z$  is at least  $D$ . Equivalently, if

$$f(\mathbf{X}, Z) = \sum_{i \geq 0} f_i(\mathbf{X})Z^i,$$

then

$$f \text{ Trunc } Z^D = \sum_{0 \leq i < D} f_i(\mathbf{X})Z^i.$$

The following lemma shows that, given a circuit computing  $f(\mathbf{X}, Z)$ , one can efficiently compute a circuit for the truncation of  $f$  without significantly increasing the size or depth.

**Lemma 2.8.** *Let  $C$  be an algebraic circuit of size  $m$  and depth  $\Delta$  over  $\mathbb{F}$  computing a polynomial  $f(\mathbf{X}, Z) \in \mathbb{F}[\mathbf{X}, Z]$  with  $\deg_Z(f) = d$ . Suppose  $|\mathbb{F}| > d$ . Then, for every  $D \in \mathbb{N}$ , one can construct, in time  $\text{poly}(m, d)$ , a circuit of size  $\text{poly}(m, d)$  and depth at most  $\Delta + 4$  computing  $f \text{ Trunc } Z^D$ .*

*Proof.* Write

$$f(\mathbf{X}, Z) = \sum_{i=0}^d f_i(\mathbf{X})Z^i.$$

By Lemma 2.6, given  $C$ , we can construct circuits  $C_0, \dots, C_d$  computing  $f_0, \dots, f_d$ , respectively, each of size  $\text{poly}(m, d)$  and depth at most  $\Delta + 2$ , in time  $\text{poly}(m, d)$ . Taking the sum

$$\sum_{0 \leq i < D} C_i(\mathbf{X})Z^i$$

therefore gives a circuit computing  $f \text{ Trunc } Z^D$ . This adds one multiplication layer to form the terms  $C_i(\mathbf{X})Z^i$  and one addition layer to sum them. Hence the resulting circuit has size  $\text{poly}(m, d)$  and depth at most  $\Delta + 4$ .  $\blacksquare$

We will later need to recover factors of the original polynomial from factors obtained from the corresponding normalized and reverse-normalized polynomials. At the level of circuits, this leads to substitutions of the form  $Y \mapsto Y \cdot f$  and  $Y \mapsto Y/f$ . A direct implementation by composing the two input circuits would stack their depths, which is too large for our applications. The following lemma shows that, using interpolation (Lemma 2.6) with respect to  $Y$ , these substitutions can instead be carried out with only a constant additive overhead over the larger of the two input depths.

**Lemma 2.9.** *Let  $g(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$ , let  $f(\mathbf{Z}) \in \mathbb{F}[\mathbf{Z}]$ , and let  $d = \deg_Y(g)$ . Suppose  $g$  is computable by a circuit  $C_g$  of size  $s_g$  and depth  $\Delta_g$ , and  $f$  is computable by a circuit  $C_f$  of size  $s_f$  and depth  $\Delta_f$ . Assume  $|\mathbb{F}| > d$ . Then one can construct circuits with division gates computing*

$$g(\mathbf{X}, Y \cdot f) \quad \text{and} \quad g(\mathbf{X}, Y/f),$$

*each of size  $\text{poly}(s_g, s_f, d)$  and depth at most  $\max\{\Delta_g, \Delta_f\} + 6$ . Moreover, each output circuit may be arranged to have a multiplication gate at the top.*

*Proof.* Write

$$g(\mathbf{X}, Y) = \sum_{i=0}^d g_i(\mathbf{X})Y^i.$$

By Lemma 2.6, applied to  $C_g$  with respect to  $Y$ , we can construct circuits  $C_0, \dots, C_d$  computing  $g_0, \dots, g_d$ , respectively. Each  $C_i$  has size  $\text{poly}(s_g, d)$  and depth at most  $\Delta_g + 2$ .

To compute  $g(\mathbf{X}, Y \cdot f)$ , for each  $0 \leq i \leq d$  construct a circuit for

$$g_i(\mathbf{X})Y^i f^i$$

by multiplying the output of  $C_i$ ,  $i$  copies of the input  $Y$ , and  $i$  copies of  $C_f$ . Summing these circuits over all  $i$  gives

$$\sum_{i=0}^d g_i(\mathbf{X})Y^i f^i = g(\mathbf{X}, Y \cdot f).$$

The size is  $\text{poly}(s_g, s_f, d)$ , and the depth is at most  $\max\{\Delta_g, \Delta_f\} + 5$  after adding, if necessary, a multiplication-by-1 gate at the output.

The construction for  $g(\mathbf{X}, Y/f)$  is analogous. For each  $0 \leq i \leq d$ , compute

$$\frac{g_i(\mathbf{X})Y^i}{f^i}$$

using a division gate, where the numerator is the product of the output of  $C_i$  with  $i$  copies of  $Y$ , and the denominator is the product of  $i$  copies of  $C_f$ . Summing these rational functions gives

$$\sum_{i=0}^d \frac{g_i(\mathbf{X})Y^i}{f^i} = g(\mathbf{X}, Y/f).$$

Again the size is  $\text{poly}(s_g, s_f, d)$ , and the depth is at most  $\max\{\Delta_g, \Delta_f\} + 6$  after adding, if necessary, a multiplication-by-1 gate at the output. ■

Finally, we will use the following lemma on symmetric polynomials to ensure that the algorithm remains over the ground field. Later, when recovering factors from the normalized and reverse-normalized polynomials, it will be straightforward to describe the relevant factors after passing to an extension field containing the roots of a suitable univariate polynomial. However, our algorithm needs to output circuits over the original field  $\mathbb{F}$ . The lemma below allows us to rewrite the required symmetric expressions over the base field: elementary symmetric polynomials in rational functions evaluated at these roots admit small circuits over the base field.

Andrews and Wigderson [AW24] proved such a statement over fields of characteristic zero or sufficiently large characteristic, and [BKR<sup>+</sup>25a] extended it to all sufficiently large fields, with no restriction on the characteristic. The proof is constructive and yields the stated polynomial-time construction.

**Lemma 2.10** ([AW24, BKR<sup>+</sup>25a]). *Let  $\mathbb{F}$  be a sufficiently large field, say  $|\mathbb{F}| \geq (mdn)^c$  for some absolute constant  $c > 0$ . Let  $q(Y) \in \mathbb{F}[Y]$  and  $g(\mathbf{X}_n, Y), h(\mathbf{X}_n, Y) \in \mathbb{F}[\mathbf{X}_n, Y]$  be monic in  $Y$ , of degrees  $d_q, d_g, d_h$ , respectively. Suppose that  $q, g, h$  are given by circuits of size at most  $m$  and depth at most  $\Delta$ . Assume moreover that  $h(\mathbf{X}_n, \sigma_i)$  is nonzero as a polynomial over  $\overline{\mathbb{F}}$  for every root  $\sigma_i$  of  $q$ . Let  $d = \max\{d_q, d_g, d_h\}$ . If  $\sigma_1, \dots, \sigma_{d_q}$  are the roots of  $q$  over  $\overline{\mathbb{F}}$ , counted with multiplicity, then for every  $1 \leq r \leq d_q$ , one can construct, in time  $\text{poly}(m, n, d)$ , circuits  $A(\mathbf{X})$  and  $B(\mathbf{X})$  over  $\mathbb{F}$ , each of size  $\text{poly}(m, n, d)$  and depth  $\Delta + \mathcal{O}(1)$ , such that*

$$\text{Esym}_r \left( \left\{ \frac{g(\mathbf{X}, \sigma_i)}{h(\mathbf{X}, \sigma_i)} : i \in [d_q] \right\} \right) = \frac{A(\mathbf{X})}{B(\mathbf{X})}.$$

### 2.3 Deterministic Univariate Factorization

We briefly summarize some standard results on deterministic algorithms for univariate polynomial factorization.

**Definition 2.11.** Let  $f \in \mathbb{F}[X]$  be an univariate polynomial of degree  $d$ . We write  $\mathcal{T}(\mathbb{F}, d)$  for the time taken to factor  $f$  over  $\mathbb{F}$ .

Univariate factorization over the rational numbers can be performed in deterministic polynomial time using the celebrated LLL algorithm of Lenstra, Lenstra, and Lovász [LLL82].

**Theorem 2.12** (Univariate factorization over rationals [LLL82]). *There is a deterministic algorithm that, given a polynomial  $f \in \mathbb{Q}[x]$  of degree  $d$  and coefficient bit complexity at most  $B$ , computes the irreducible factorization of  $f$  over  $\mathbb{Q}$  in time  $\text{poly}(d, B)$ . Thus, over  $\mathbb{Q}$ ,  $\mathcal{T}(\mathbb{Q}, d)$  is polynomial in  $d$  and in the coefficient bit complexity of the input.*

Polynomial factorization over finite fields has been extensively studied; see von zur Gathen and Gerhard [vzGG13, Chapter 14] for more details. The following standard deterministic bound is sufficient for our purposes.

**Theorem 2.13** ([Ber70, Sho90]). *There is a deterministic algorithm that, given a polynomial  $f \in \mathbb{F}_{p^m}[x]$  of degree  $d$ , computes the irreducible factorization of  $f$  over  $\mathbb{F}_{p^m}$  in time  $\text{poly}(d, m, p)$ . In other words,  $\mathcal{T}(\mathbb{F}_{p^m}, d) = \text{poly}(d, m, p)$ .*

Since specifying the input polynomial  $f$  requires only  $\mathcal{O}(dm \log p)$  bits, the running time in Theorem 2.13 can be super-polynomial in the bit complexity of the input when the characteristic  $p$  is large. Finally, we state the time complexity of the following folklore result on factorization of constant-variate polynomials.

**Lemma 2.14.** *Let  $f(\mathbf{X}, Y)$  be a  $(c + 1)$ -variate polynomial of degree  $D$  over  $\mathbb{F}$ . Suppose  $f$  is monic or reversed-monic in  $Y$ . Then all factors of  $f$  of  $Y$ -degree at most  $d$  can be computed in time  $\text{poly}(D^{cd}, \mathcal{T}(\mathbb{F}, D))$ .*

### 2.4 Sylvester Matrix, Resultant, and Discriminant

We recall the definitions of the Sylvester matrix, the resultant, and the discriminant of univariate polynomials. A more thorough treatment of this material can be found in [vzGG13].

Resultants give an algebraic way to detect when two univariate polynomials have a common root.

**Definition 2.15.** Let  $f(X) = \sum_{i=0}^{d_1} f_i X^i$  and  $g(X) = \sum_{i=0}^{d_2} g_i X^i$  be polynomials in  $\mathbb{F}[X]$  of degrees  $d_1$  and  $d_2$ , respectively. The *Sylvester matrix* of  $f$  and  $g$  is the  $(d_1 + d_2) \times (d_1 + d_2)$  matrix given



1.  $\text{Syl}_Y(f, g)(\alpha) = \text{Syl}(f(\alpha, Y), g(\alpha, Y))$  and  $\text{Res}_Y(f, g)(\alpha) = \text{Res}(f(\alpha, Y), g(\alpha, Y))$ .
2. If  $f(\alpha, Y)$  and  $g(\alpha, Y)$  are coprime, then  $\text{Res}_Y(f, g)(\alpha) \neq 0$ .
3. If  $f(\alpha, Y)$  is square-free, then  $\text{Disc}_Y(f)(\alpha) \neq 0$ .

We next record how the Sylvester matrix changes under the substitution  $Y \mapsto Y \cdot g(\mathbf{X})$ .

**Lemma 2.20.** *Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  satisfy  $\deg_Y(f) = k$ , and let  $g(\mathbf{X}) \in \mathbb{F}(\mathbf{X})$  be non-zero. Define  $\bar{f}(\mathbf{X}, Y) := f(\mathbf{X}, Y \cdot g(\mathbf{X}))$ . Then*

$$\text{Syl}_Y(\bar{f}, \bar{f}') = \text{Diag}\left(1/g, 1/g^2, \dots, 1/g^{2k-1}\right) \cdot \text{Syl}_Y(f, f') \cdot \text{Diag}\left(g^{k+1}, g^{k+2}, \dots, g^{2k-1}, g^{k+1}, g^{k+2}, \dots, g^{2k}\right).$$

*Proof.* Write

$$f(\mathbf{X}, Y) = \sum_{i=0}^k f_i(\mathbf{X})Y^i$$

with  $f_k \neq 0$ . Then

$$\bar{f}(\mathbf{X}, Y) = \sum_{i=0}^k f_i(\mathbf{X})g(\mathbf{X})^i Y^i.$$

Since  $\deg_Y(f') = k - 1$ , the matrices  $\text{Syl}_Y(f, f')$  and  $\text{Syl}_Y(\bar{f}, \bar{f}')$  have size  $(2k - 1) \times (2k - 1)$ . For  $i, j \in [2k - 1]$ , we have

$$\text{Syl}_Y(f, f')_{i,j} = \begin{cases} f_{k-(i-j)} & j \in [k - 1] \text{ and } 0 \leq i - j \leq k, \\ (j - i + 1)f_{j-i+1} & k \leq j \leq 2k - 1 \text{ and } 0 \leq j - i \leq k - 1, \\ 0 & \text{otherwise,} \end{cases}$$

$$\text{Syl}_Y(\bar{f}, \bar{f}')_{i,j} = \begin{cases} f_{k-(i-j)}g^{k-(i-j)} & j \in [k - 1] \text{ and } 0 \leq i - j \leq k, \\ (j - i + 1)f_{j-i+1}g^{j-i+1} & k \leq j \leq 2k - 1 \text{ and } 0 \leq j - i \leq k - 1, \\ 0 & \text{otherwise.} \end{cases}$$

These entrywise identities can be rewritten as

$$\text{Syl}_Y(\bar{f}, \bar{f}') = \text{Diag}\left(1/g, 1/g^2, \dots, 1/g^{2k-1}\right) \cdot \text{Syl}_Y(f, f') \cdot \text{Diag}\left(g^{k+1}, g^{k+2}, \dots, g^{2k-1}, g^{k+1}, g^{k+2}, \dots, g^{2k}\right).$$

This proves the lemma. ■

**Remark 2.21.** Since the lemma allows  $g(\mathbf{X})$  to be an element of  $\mathbb{F}(\mathbf{X})$ , it also applies to substitutions of the form  $Y \mapsto Y/h(\mathbf{X})$  for non-zero  $h(\mathbf{X}) \in \mathbb{F}[\mathbf{X}]$ , by taking  $g(\mathbf{X}) = 1/h(\mathbf{X})$ .

## 2.5 Factors of Normalized and Reverse-Normalized Polynomials

We record two simple facts showing that factors behave well under the normalization and reverse-normalization operations defined above. In both cases, divisibility is immediate over the rational function field  $\mathbb{F}(\mathbf{X})$ ; the point is that, after the chosen scaling, the relevant factor lies in  $\mathbb{F}[\mathbf{X}, Y]$  and is monic or reversed-monic in  $Y$ . We will use the following elementary form of Gauss's lemma.

**Lemma 2.22** (Gauss's lemma). *Let  $\mathbb{A}$  be a unique factorization domain with fraction field  $\mathbb{K}$ . If  $p(Y), q(Y) \in \mathbb{A}[Y]$  and  $p(Y)$  is primitive as a polynomial over  $\mathbb{A}$ , then*

$$p(Y) \mid q(Y) \text{ in } \mathbb{K}[Y] \quad \implies \quad p(Y) \mid q(Y) \text{ in } \mathbb{A}[Y].$$

*In particular, the conclusion holds if  $p(Y)$  is monic or reversed-monic.*

**Lemma 2.23.** *Let*

$$f(\mathbf{X}, Y) = \sum_{i=0}^d f_i(\mathbf{X})Y^i, \quad g(\mathbf{X}, Y) = \sum_{i=0}^t g_i(\mathbf{X})Y^i$$

be polynomials in  $\mathbb{F}[\mathbf{X}, Y]$  with  $f_d \neq 0$  and  $g_t \neq 0$ . Suppose  $g \mid f$  in  $\mathbb{F}[\mathbf{X}, Y]$ , and define

$$\hat{f}(\mathbf{X}, Y) = f(\mathbf{X}, Y/f_d) f_d^{d-1}, \quad \hat{g}(\mathbf{X}, Y) = g(\mathbf{X}, Y/f_d) \frac{f_d^t}{g_t}.$$

Then  $\hat{g} \in \mathbb{F}[\mathbf{X}, Y]$  and  $\hat{g} \mid \hat{f}$  in  $\mathbb{F}[\mathbf{X}, Y]$ .

*Proof.* Since  $g \mid f$ , the leading coefficient  $g_t$  divides  $f_d$  in  $\mathbb{F}[\mathbf{X}]$ . Hence

$$\hat{g}(\mathbf{X}, Y) = \sum_{i=0}^t g_i(\mathbf{X})Y^i \frac{f_d^{t-i}}{g_t}$$

lies in  $\mathbb{F}[\mathbf{X}, Y]$ , and it is monic in  $Y$ . Moreover, over  $\mathbb{F}(\mathbf{X})[Y]$ , the substitution  $Y \mapsto Y/f_d$  preserves divisibility, and hence  $\hat{g} \mid \hat{f}$  in  $\mathbb{F}(\mathbf{X})[Y]$ . Since  $\hat{g}$  is monic, by Lemma 2.22,  $\hat{g} \mid \hat{f}$  in  $\mathbb{F}[\mathbf{X}, Y]$ .  $\blacksquare$

**Lemma 2.24.** *Let*

$$f(\mathbf{X}, Y) = \sum_{i=0}^d f_i(\mathbf{X})Y^i, \quad g(\mathbf{X}, Y) = \sum_{i=0}^t g_i(\mathbf{X})Y^i$$

be polynomials in  $\mathbb{F}[\mathbf{X}, Y]$  with  $f_0 \neq 0$  and  $g_0 \neq 0$ . Suppose  $g \mid f$  in  $\mathbb{F}[\mathbf{X}, Y]$ , and define

$$\tilde{f}(\mathbf{X}, Y) = \frac{f(\mathbf{X}, Y f_0)}{f_0}, \quad \tilde{g}(\mathbf{X}, Y) = \frac{g(\mathbf{X}, Y f_0)}{g_0}.$$

Then  $\tilde{g} \in \mathbb{F}[\mathbf{X}, Y]$  and  $\tilde{g} \mid \tilde{f}$  in  $\mathbb{F}[\mathbf{X}, Y]$ .

*Proof.* Since  $g \mid f$ , the constant coefficient  $g_0$  divides  $f_0$  in  $\mathbb{F}[\mathbf{X}]$ . Thus

$$\tilde{g}(\mathbf{X}, Y) = \sum_{i=0}^t g_i(\mathbf{X})Y^i \frac{f_0^i}{g_0}$$

lies in  $\mathbb{F}[\mathbf{X}, Y]$ . Moreover, over  $\mathbb{F}(\mathbf{X})[Y]$ , the substitution  $Y \mapsto Y f_0$  preserves divisibility, and hence  $\tilde{g} \mid \tilde{f}$  in  $\mathbb{F}(\mathbf{X})[Y]$ . Since  $\tilde{g}$  is reversed-monic, by Lemma 2.22,  $\tilde{g} \mid \tilde{f}$  in  $\mathbb{F}[\mathbf{X}, Y]$ .  $\blacksquare$

## 2.6 Power Series Roots of Monic Polynomials

In this section, we discuss the factorization of monic polynomials into power series roots. To guarantee the existence of such power series roots, we require the polynomial to satisfy a suitable non-degeneracy condition, formalized in the following definition. Once this condition holds, we will describe how to compute the corresponding power series roots algorithmically.

**Definition 2.25.** Let  $f(\mathbf{X}_n, Y) \in \mathbb{F}[\mathbf{X}_n, Y]$  and suppose  $f(\mathbf{X}_n, Y) = f_d Y^d + f_{d-1} Y^{d-1} + \cdots + f_1 Y + f_0$  where  $f_0, \dots, f_d \in \mathbb{F}[\mathbf{X}_n]$  and  $f_d \neq 0$ . Let the irreducible factorization of  $f$  be  $f = h_1^{e_1} h_2^{e_2} \cdots h_r^{e_r}$ . We call  $f$  **Hensel-ready** at  $\mathbf{X}_n = \boldsymbol{\alpha} \in \mathbb{F}^n$  if  $\text{Res}_Y(h_i, h_j)(\boldsymbol{\alpha}) \neq 0$  for all  $i \neq j$  and  $\text{Disc}_Y(h_i)(\boldsymbol{\alpha}) \neq 0$  for all  $i \in [r]$ .

Once we find such a point  $\alpha$ , we may apply the shift  $\mathbf{X}_n \mapsto \mathbf{X}_n + \alpha$ . The resulting polynomial is Hensel-ready at  $\mathbf{X}_n = \mathbf{0}$ . We can therefore factor the shifted polynomial around the origin, and finally undo the shift to recover the corresponding factorization of the original polynomial. Thus, without loss of generality, we will assume throughout this section that the polynomial under consideration is Hensel-ready at  $\mathbf{X}_n = \mathbf{0}$ .

Suppose  $f$  is a monic polynomial that is Hensel-ready at  $\mathbf{0}$ , we have

$$f(\mathbf{0}, Y) = h_1(\mathbf{0}, Y)^{e_1} h_2(\mathbf{0}, Y)^{e_2} \dots h_r(\mathbf{0}, Y)^{e_r}$$

then for each  $i \neq j \in [r]$ ,  $\gcd(h_i(\mathbf{0}, Y), h_j(\mathbf{0}, Y)) = 1$  and  $h_j(\mathbf{0}, Y)$  is a square-free polynomial. Let  $\mathcal{B} := \{\beta_1, \dots, \beta_d\}$  be the multiset of roots of  $f(\mathbf{0}, Y)$  over a field extension of  $\mathbb{F}$ . The following well-known lemma states that each such univariate root can be uniquely lifted to a power series root of  $f(\mathbf{X}, Y)$  in  $Y$ . This lemma appeared in many forms in the literature of commutative algebra and polynomial factorization; for references, see [Eis95, Rui93, BDS25]. For completeness, we include a proof of the lemma in the appendix (see Section B.1).

**Lemma 2.26** (Hensel's lemma). *Let  $f(\mathbf{X}, Y)$  be a monic polynomial in  $Y$  with  $\deg_Y(f) = d$ , and let the irreducible factorization of  $f$  be  $f(\mathbf{X}, Y) = h_1^{e_1}(\mathbf{X}, Y) h_2^{e_2}(\mathbf{X}, Y) \dots h_r^{e_r}(\mathbf{X}, Y)$ . Suppose  $f$  is Hensel-ready at  $\mathbf{X} = \mathbf{0}$  and  $\mathcal{B} = \{\beta_1, \dots, \beta_d\}$  is the multiset of roots of  $f(\mathbf{0}, Y)$ . Then*

$$f(\mathbf{X}, Y) = \prod_{i=1}^d (Y - \Phi_i(\mathbf{X}))$$

where  $\Phi_i(\mathbf{X}) \in \mathbb{F}(\beta_i)[[\mathbf{X}]]$ ,  $\Phi_i(\mathbf{0}) = \beta_i$ , and  $\Phi_i = \Phi_j$  if and only if  $\beta_i = \beta_j$ .

The preceding lemma shows that each power series root of  $f(\mathbf{X}, Y)$  is uniquely determined by its constant term, which is a root of  $f(\mathbf{0}, Y)$ . We will therefore write  $\Phi_\beta$  for the power series root corresponding to the root  $\beta$  of  $f(\mathbf{0}, Y)$ . Moreover, since  $\overline{\mathbb{F}}[[\mathbf{X}]]$  is a UFD, the above factorization is the unique factorization of  $f$  into linear factors in  $\overline{\mathbb{F}}[[\mathbf{X}]][[Y]]$ .

The following corollary shows that every monic factor of  $f(\mathbf{X}, Y)$  arises from a factor of  $f(\mathbf{0}, Y)$ , by lifting the corresponding sub-multiset of roots of  $f(\mathbf{0}, Y)$  to their power series roots.

**Corollary 2.27.** *Let  $f(\mathbf{X}, Y)$  be a monic polynomial with  $\deg_Y(f) = d$  and such that  $f$  is Hensel-ready at  $\mathbf{X} = \mathbf{0}$ . Let  $\mathcal{B} = \{\beta_1, \dots, \beta_d\}$  be the multiset of univariate roots of  $f(\mathbf{0}, Y)$ . Let  $g(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  be monic in  $Y$ , with  $\deg_Y(g) = k$ , and suppose that  $g$  divides  $f$ . Then there exists a multiset  $S \subseteq \mathcal{B}$  with  $|S| = k$  such that  $g = \prod_{\beta \in S} (Y - \Phi_\beta)$  and  $\text{Esym}_j(S) \in \mathbb{F}$  for all  $j \in [k]$ .*

We now explain how to compute the power series  $\Phi_\beta$  algorithmically. We first reduce the task to computing power series roots of bivariate polynomials. Introduce a new variable  $T$  and replace each  $X_i$  by  $TX_i$ . Define  $F(T, Y) := f(T\mathbf{X}, Y)$ , which we view as a polynomial in  $T$  and  $Y$  over the coefficient ring  $\mathbb{F}[\mathbf{X}]$ . Since  $F(0, Y) = f(\mathbf{0}, Y)$ , Hensel's lemma applies to  $F$  as well. Moreover, suppose the power series root of  $f(\mathbf{X}, Y)$  corresponding to  $\beta$  has the homogeneous decomposition  $\Phi_\beta(\mathbf{X}) = \sum_{r \geq 0} \Phi_\beta^{(r)}(\mathbf{X})$ , where each  $\Phi_\beta^{(r)}(\mathbf{X})$  is homogeneous of degree  $r$ . Then the corresponding power series root of  $F(T, Y)$  is  $\Phi_\beta(T) = \sum_{r \geq 0} \Phi_\beta^{(r)}(\mathbf{X}) T^r$ . Thus, the coefficient of  $T^r$  in  $\Phi_\beta(T)$  recovers precisely the degree- $r$  homogeneous component of  $\Phi_\beta(\mathbf{X})$ . By a slight abuse of notation, we will also write  $\Phi_\beta(T)$  for the power series root of  $F(T, Y)$  corresponding to  $\beta$ .

At this point, any standard polynomial-time algorithm for computing power series roots (such as Newton iteration or Hensel lifting) would work. Obtaining the additional properties we need

requires more work: the coefficients of the roots, and hence eventually the factors, should have constant-depth circuits, and the output circuits computing the factors should use only constants from the base field  $\mathbb{F}$ . For this, we use some results from the recent works [BKR<sup>+</sup>26, BKR<sup>+</sup>25a] on factoring constant-depth circuits, which we summarize below.

For the rest of the section, assume that  $\text{char}(\mathbb{F}) = 0$  or  $\text{char}(\mathbb{F}) > d$ . Let  $F(T, Y) \in \mathbb{F}[\mathbf{X}][T, Y]$  be monic in  $Y$ , with  $\deg_Y(F) = d$  and  $\deg_T(F) = D$ , and suppose that  $F$  is Hensel-ready at  $T = 0$ . Let  $\mathcal{B}$  be the set of roots of  $F(0, Y)$ , and let  $\beta$  be a root of  $F(0, Y)$  of multiplicity  $e$ . By Lemma 2.26,  $F(T, Y)$  has a power series root  $\Phi_\beta$  of multiplicity  $e$ . We now give an explicit formula for  $\Phi_\beta(T)$ . Such a formula was obtained in [BKR<sup>+</sup>26] when  $e = 1^6$ , and over fields of characteristic dividing  $e$ , it used Hasse derivatives to obtain an expression. Here, we use standard derivatives to obtain a variation of the formula in [BKR<sup>+</sup>26] for general  $e$ . The proof follows by a close adaptation of the arguments of [BKR<sup>+</sup>26]; for completeness, we include the details in the appendix.

Note that, since the multiplicity of  $\beta$  in  $F(0, Y)$  is  $e$ ,  $\partial_{Y^{e-1}}F(0, Y)$  is a non-zero polynomial in  $Y$ . Define the Laurent series  $R_e \in \mathbb{F}[\mathbf{X}][T]((Z))$  in  $Z$

$$R_e(Z) := Z + \sum_{m \geq 0}^{2e(D+1)} [Y^{em+e-2}] \left\{ \left( \frac{e!}{\partial_{Y^e}F(0, Z)} \right)^{m+1} \frac{\partial_Y F(T, Y+Z)}{e} \left( \frac{\partial_{Y^e}F(0, Z)}{e!} Y^{e-F(T, Y+Z)} \right)^m \right\}$$

Here  $[Y^r]\{P(Y)\}$  denotes the coefficient of  $Y^r$  in  $P(Y)$ .

One can observe that if we set  $e = 1$  in the above definition, we get the  $R(Z)$  defined in [BKR<sup>+</sup>26, Theorem 4.3]. For more on the background and motivation of the origins of the above expression, the reader is encouraged to look at [BKR<sup>+</sup>26]. The exact expression for  $R_e(Z)$  is provided for completeness, but what we really need for the algorithm is the following two properties of  $R_e(Z)$ :

1. If  $F(T, Y) = f(T\mathbf{X}, Y)$ , then  $R_e(Z)$  can be written as a ratio of  $A(\mathbf{X}, T, Z)$  and  $B(Z)$ , where both polynomials are computable by polynomial-size constant-depth circuits (since  $f$  is sparse).
2.  $R_e(\beta)$  computes a truncated version of  $\Phi_\beta$ .

So in particular, we are able to obtain constant-depth circuits (with division gates) that compute suitable truncations of the power series  $\Phi_\beta$ . We formally state this fact below (and prove it in the appendix, Section B.2). In the setting of  $e = 1$ , such a result was also provided in [BKR<sup>+</sup>26].

**Theorem 2.28** ([BKR<sup>+</sup>26]). *Let  $\mathbb{F}$  be a field of characteristic 0 or greater than  $d$ . Let  $F(T, Y) \in \mathbb{F}[\mathbf{X}][T, Y]$  be monic in  $Y$  and Hensel-ready at  $T = 0$ , furthermore assume  $F(T, Y)$  can be computed by a circuit of size  $s'$  and depth  $\Delta$ . Let  $\beta$  be a root of  $F(0, Y)$  with multiplicity  $e$ , let  $\deg_Y(F) = d$  and  $\deg_T(F) = D$ . Let  $R_e$  be as defined above. Then*

1.  $R_e(Z)$  can be expressed as a ratio of two polynomials  $A(\mathbf{X}, T, Z)$  and  $B(Z)$  where  $A(\mathbf{X}, T, Z) \in \mathbb{F}[\mathbf{X}, T, Z]$  and  $B(Z) \in \mathbb{F}[Z]$ . Furthermore both  $A(\mathbf{X}, T, Z)$  and  $B(Z)$  can be computed by circuits of size  $\text{poly}(n, s', d, D)$  and depth  $\Delta + \mathcal{O}(1)$ .
2.  $B(\beta) \neq 0$ .
3.  $\Phi_\beta(T) \text{ Trunc } T^{D+1} = R_e(\beta) \text{ Trunc } T^{D+1}$

---

<sup>6</sup>The factoring algorithm of [BKR<sup>+</sup>26] first shows how to reduce to the case  $e = 1$  over characteristic zero fields and fields where the characteristic does not divide  $e$ , and hence does not have to consider expressions for power series roots for larger values of  $e$ .

## 2.7 Sparse Polynomials

We say that a polynomial  $f(\mathbf{X}) \in \mathbb{F}[\mathbf{X}]$  is  $s$ -sparse if it has at most  $s$  monomials with non-zero coefficients. We denote the sparsity of  $f$  by  $\|f\|$ . Thus, if  $f$  has exactly  $s$  non-zero monomials, then  $\|f\| = s$ .

Klivans and Spielman [KS01] gave a beautiful deterministic hitting-set construction for sparse polynomials. We will use the following form of their result.

**Theorem 2.29** (PIT for sparse polynomials [KS01]). *Let  $n, s, d \in \mathbb{N}$  and let  $\mathbb{F}$  be a field with  $|\mathbb{F}| \geq \text{poly}(n, s, d)$ . There exists an explicit hitting set  $S_{n,s,d} \subseteq \mathbb{F}^n$  of size  $\text{poly}(n, s, d)$  for the class of  $s$ -sparse polynomials in  $\mathbb{F}[\mathbf{X}_n]$  of individual degree at most  $d$ . That is, every non-zero polynomial in this class evaluates to a non-zero value at some point of  $S_{n,s,d}$ .*

*As a consequence, there exists a hitting set  $S_{n,s,d,l} \subseteq \mathbb{F}^n$  of size  $\text{poly}(n, s, d, l)$  for the class of products of  $l$   $s$ -sparse polynomials, each with individual degree at most  $d$ . Moreover, such a hitting set can be constructed deterministically in time  $\text{poly}(n, s, d, l)$ .*

The Klivans–Spielman construction can also be viewed as giving a low-variate generator for sparse polynomials, and this yields a black-box reconstruction algorithm. A generator is a tuple

$$\mathcal{G}(\mathbf{Z}) = (\mathcal{G}_1(\mathbf{Z}), \dots, \mathcal{G}_n(\mathbf{Z})),$$

where  $\mathbf{Z} = (Z_1, \dots, Z_c)$  and each  $\mathcal{G}_i(\mathbf{Z}) \in \mathbb{F}[\mathbf{Z}]$ . For sparse polynomial reconstruction, one wants  $\mathcal{G}$  to have low degree and to preserve enough information about every  $s$ -sparse polynomial  $P \in \mathbb{F}[\mathbf{X}_n]$  of individual degree at most  $d$ , so that  $P$  can be recovered from black-box access to the low-variate polynomial  $P \circ \mathcal{G}$ . The construction in [KS01] provides such a generator. The following theorem, which is essentially [CS26, Theorem 3.2], states the resulting reconstruction guarantee.

**Theorem 2.30** (Sparse reconstruction via the Klivans–Spielman generator). *Let  $n, s, d \in \mathbb{N}$ , and let  $S \subseteq \mathbb{F}$  be a set<sup>7</sup> of size  $|S| =: m \geq \text{poly}(n, s, d)$ . There exists a generator  $\mathcal{G}_{(s,d,n)}^{KS}(Z_1, Z_2, Z_3, Z_4) = (\mathcal{G}_1(\mathbf{Z}_4), \dots, \mathcal{G}_n(\mathbf{Z}_4))$ , such that each  $\mathcal{G}_i \in \mathbb{F}[\mathbf{Z}_4]$  has degree at most  $\text{poly}(n, s, d)$ , with the following property. Given the evaluations of  $P \circ \mathcal{G}_{(s,d,n)}^{KS}$  on all points of  $S^4$ , any  $s$ -sparse polynomial  $P \in \mathbb{F}[\mathbf{X}_n]$  of individual degree at most  $d$  can be reconstructed in monomial representation in time  $\text{poly}(n, s, d)$ . Moreover,  $\mathcal{G}_{(s,d,n)}^{KS}$  can be constructed in time  $\text{poly}(n, s, d)$ .*

In Section 4, we will first give an algorithm that produces a list of candidate circuits (with division gates) that are monomial equivalent to factors of the input polynomial  $f$ . In the post-processing step, we will remove the suitable monomial divisors from these candidates and eliminate division gates. For this purpose, we record the following simple consequences of the Klivans–Spielman hitting-set and reconstruction results above. For brevity, we shift their proofs to Section C in the appendix.

**Corollary 2.31.** *Let  $\mathbb{F}$  be a field with  $|\mathbb{F}| \geq \text{poly}(n, s, d)$ , and let  $i \in [n]$ . Suppose  $f(\mathbf{X}_n)$  is a factor of a non-zero  $s$ -sparse polynomial of individual degree at most  $d$ , and suppose additionally that  $f$  is not divisible by a monomial. Let  $C$  be an algebraic circuit of size  $m$  and degree  $D$  computing a non-zero polynomial  $g(\mathbf{X}_n) \in \mathbb{F}[\mathbf{X}_n]$  such that  $f \prec_d g$ . Then there exists a deterministic algorithm that, given  $C$ , computes the largest power of  $X_i$  dividing  $g$  in time  $\text{poly}(n, s, d, D, m)$ .*

**Lemma 2.32** (Division elimination). *Let  $C$  be a circuit of size  $m$  and depth  $\Delta$  with division gates over  $\mathbb{F}$ , computing a polynomial  $f \in \mathbb{F}[\mathbf{X}_n]$  of individual degree at most  $d$ . Suppose that every*

<sup>7</sup>We canonically choose  $S$  to be the first  $m$  elements in the field, according to some fixed standard enumeration.

denominator appearing at a division gate of  $C$  is a product of  $s$ -sparse polynomials of individual degree at most  $d$ . Then there is a deterministic algorithm that runs in  $\text{poly}(n, s, m, d)$  time and outputs a circuit without division gates, of size  $\text{poly}(m, d)$  and depth at most  $\Delta + \mathcal{O}(1)$ , computing  $f$ .

Our algorithms will output a list of candidate factors of sparse polynomials. In general, this list may contain spurious candidates that do not correspond to actual factors of the input polynomial. In some settings, such as the problem of outputting all sparse factors considered in [CS26], these spurious candidates can be removed using divisibility tests. We record below a result of Volkovich [Vol17], which provides an efficient deterministic divisibility test in the bounded individual degree regime.

**Theorem 2.33** ([Vol17]). *Let  $f, g \in \mathbb{F}[\mathbf{X}_n]$  be  $s$ -sparse polynomials of individual degree at most  $d$ , and suppose that  $\text{char}(\mathbb{F}) = 0$  or  $\text{char}(\mathbb{F}) > d$ . Then there exists a deterministic algorithm that, given black-box access to  $f$  and  $g$ , decides whether  $g$  divides  $f$  in time  $\text{poly}(n, s^d, d!)$ .*

We can also use the above theorem to prune out the list and output *all* the factors in the bounded individual degree regime, in quasi-polynomial time. For this, we will need the following result of Bhargava, Saraf and Volkovich [BSV20]: bounded individual degree factors of sparse polynomials have at most quasi-polynomial sparsity.

**Theorem 2.34** (Sparsity Bound). [BSV20, Theorem 4.1] *Let  $f \in \mathbb{F}[\mathbf{X}_n]$  be a  $s$ -sparse polynomial. Suppose  $g$  is a factor of  $f$  where  $g$  has individual degree at most  $d$ . Then, the sparsity of  $g$  is at most  $s^{4d^2 \log n}$ .*

**Remark 2.35.** [BSV20, Theorem 4.1] is stated when  $f$  has individual degree  $d$ . But on inspecting their proof, it is clear that it extends to the setting of Theorem 2.34.

Finally in Section 5, we use the divisibility test by Forbes [For15] and prune out the list to obtain all constant degree factors of general sparse polynomials. This recovers the results of Kumar, Ramanathan and Saptharishi and Dutta, Sinhababu and Thierauf [KRS24, DST24] for sparse polynomials.

**Theorem 2.36** ([For15]). *Let  $\mathbb{F}$  be a field of characteristic 0 or characteristic at least  $\text{poly}(D)$ . Let  $f \in \mathbb{F}[\mathbf{X}_n]$  be an  $n$ -variate  $s$ -sparse polynomial of degree at most  $D$ , and let  $g \in \mathbb{F}[\mathbf{X}_n]$  be a polynomial of constant degree. Then there is a deterministic algorithm that computes the multiplicity of  $g$  as a factor of  $f$  in time  $\text{poly}(s, n, D)^{\mathcal{O}(\log s)}$ .*

### 3 Factoring Normalized and Reverse Normalized Sparse Polynomials

In this section, we prove the factorization subroutine used by our main algorithm. The input is a sparse polynomial

$$f(\mathbf{X}, Y) = f_d(\mathbf{X})Y^d + \cdots + f_0(\mathbf{X})$$

of bounded individual degree, given in its monomial representation. We first handle the normalization  $\hat{f}$ , which is monic in  $Y$ , and later reduce the reverse-normalized case to the same setting using polynomial reversals. We assume throughout this section that  $\text{char}(\mathbb{F}) = 0$  or  $\text{char}(\mathbb{F}) > d$ .

The goal is to produce a small list of constant-depth circuits that contains every non-constant factor of the normalized polynomial  $\hat{f}$ , while allowing spurious candidates.

**Lemma 3.1.** *Let  $\mathbb{F}$  be a field of characteristic 0 or greater than  $d$ . Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  be an  $n$ -variate polynomial with sparsity at most  $s$  and individual degree at most  $d$ , and let  $\hat{f}(\mathbf{X}, Y)$  be its normalization with respect to  $Y$ . Then there is an algorithm running in time  $\text{poly}(s^d, d!, n, \mathcal{T}(\mathbb{F}, d))$  that outputs a list  $\mathcal{L}$  such that*

1.  $|\mathcal{L}| \leq 2^d - 1$
2. every element of  $\mathcal{L}$  is a  $\text{poly}(n, s, d)$ -size,  $\mathcal{O}(1)$ -depth circuit over  $\mathbb{F}$
3. every non-constant factor  $\hat{g}(\mathbf{X}, Y)$  of  $\hat{f}(\mathbf{X}, Y)$  is computed by some circuit in  $\mathcal{L}$

The list  $\mathcal{L}$  may contain spurious candidates. Moreover, the algorithm can compute the  $Y$ -degree of the polynomial computed by every circuit in  $\mathcal{L}$ .

Before proving the lemma, we establish the main preprocessing step: finding a point  $\alpha \in \mathbb{F}^n$  at which  $\hat{f}(\mathbf{X}, Y)$  is Hensel-ready, in the sense of Definition 2.25. After shifting  $\mathbf{X}$  by this point, the polynomial can be factored through its power-series roots around the origin.

**Finding a Hensel-ready shift.** Let us first consider the easier case where  $\hat{f}$  is square-free. By Lemma 2.17, this is equivalent to  $\text{Disc}_Y(\hat{f}) \neq 0$ , or equivalently to  $\text{Syl}_Y(\hat{f}, \partial_Y \hat{f})$  having full rank. Since the normalization step can increase sparsity from  $s$  to  $s^d$ , every entry of this Sylvester matrix is  $s^d$ -sparse, and hence  $\text{Disc}_Y(\hat{f})$  is  $(2d)!s^{d^2}$ -sparse. Therefore, any point  $\alpha \in \mathbb{F}^n$  for which  $\text{Disc}_Y(\hat{f})(\alpha) \neq 0$  makes  $\hat{f}$  Hensel-ready at  $\mathbf{X} = \alpha$ .

In general,  $\hat{f}$  need not be square-free, so  $\text{Disc}_Y(\hat{f})$  may vanish identically. In this case, we choose  $\alpha$  so that the rank of the Sylvester matrix  $\text{Syl}_Y(\hat{f}, \partial_Y \hat{f})$  is preserved after substituting  $\mathbf{X} = \alpha$ . To be precise, as observed in [CS26, Lemma 4.19], it is enough to find a point  $\alpha$  such that a nonzero maximum-rank minor of  $\text{Syl}_Y(\hat{f}, \partial_Y \hat{f})$  remains nonzero after substituting  $\mathbf{X} = \alpha$ . Such a minor is again  $(2d)!s^{d^2}$ -sparse.

Naively, this would lead to a running time polynomial in  $s^{d^2}$ . The key point is that the relevant rank-preservation test can be carried out before normalization. Let  $\bar{f}(\mathbf{X}, Y) = f(\mathbf{X}, Y/f_d)$ . By Lemma 2.20, the minors of  $\text{Syl}_Y(\bar{f}, \partial_Y \bar{f})$  are related, up to diagonal row and column scalings, to the corresponding minors of  $\text{Syl}_Y(f, \partial_Y f)$ . Since  $\hat{f} = f_d^{d-1} \bar{f}$  and  $f_d$  is independent of  $Y$ , preserving rank for  $\bar{f}$  is equivalent to preserving rank for  $\hat{f}$  at points where  $f_d$  is nonzero. Thus the point  $\alpha$  can be found by working with the Sylvester matrix of the original sparse polynomial  $f$ , which is what keeps the running time polynomial in  $s^d$ .

We formalize this preprocessing step in the next lemma. It is a variant of [CS26, Lemma 4.19], adapted to our notation and parameter setting. We include the proof because the lemma is used here in a slightly different context.

**Lemma 3.2.** *Let  $\mathbb{F}$  be a field of characteristic 0 or greater than  $d$ . Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  be an  $s$ -sparse polynomial of individual degree  $d$ , and let  $\hat{f}(\mathbf{X}, Y)$  be the normalization of  $f$  with respect to the variable  $Y$ . Then there is a deterministic algorithm that runs in  $\text{poly}(n, d!, s^d)$  time and finds  $\alpha \in \mathbb{F}^n$  such that  $\hat{f}$  is Hensel-ready at  $\mathbf{X} = \alpha$ .*

*Proof.* The matrix  $\text{Syl}_Y(f, \partial_Y f)$  has size at most  $(2d - 1) \times (2d - 1)$ . Hence it has at most  $2^{O(d)}$  minors, and each minor expands to a polynomial with sparsity at most  $(2d)!s^{O(d)}$ . The algorithm will find  $\alpha \in \mathbb{F}^n$  that keeps every nonzero minor of  $\text{Syl}_Y(f, \partial_Y f)$  nonzero after substituting  $\mathbf{X} = \alpha$ , and moreover keeps  $f_d(\alpha)$  nonzero. By Theorem 2.29, such a point can be found in  $\text{poly}(d!s^d, n)$  time. Let  $\bar{f}(\mathbf{X}, Y) = f(\mathbf{X}, Y/f_d)$ . By Lemma 2.20, the choice of  $\alpha$  preserves the rank of  $\text{Syl}_Y(\bar{f}, \partial_Y \bar{f})$ . Since  $\hat{f} = f_d^{d-1} \bar{f}$  and  $f_d$  is independent of  $Y$ , the Sylvester matrices  $\text{Syl}_Y(\hat{f}, \partial_Y \hat{f})$  and  $\text{Syl}_Y(\bar{f}, \partial_Y \bar{f})$

differ only by multiplication by powers of  $f_d$ . Thus, over  $\mathbb{F}(\mathbf{X})$ , they have the same rank, and after substituting  $\mathbf{X} = \boldsymbol{\alpha}$ , where  $f_d(\boldsymbol{\alpha}) \neq 0$ , rank preservation for  $\bar{f}$  implies rank preservation for  $\hat{f}$ . In particular, the rank of  $\text{Syl}_Y(f, \partial_Y \hat{f})$  over  $\mathbb{F}(\mathbf{X})$  is the same as the rank of  $\text{Syl}_Y(\hat{f}, \partial_Y \hat{f})(\boldsymbol{\alpha})$ .

We now show that  $\hat{f}$  is Hensel-ready at  $\mathbf{X} = \boldsymbol{\alpha}$ . Let us assume the irreducible factorization of  $\hat{f}$  is  $\hat{f} = h_1(\mathbf{X}, Y)^{e_1} \dots h_k(\mathbf{X}, Y)^{e_k}$  and  $\deg_Y(h_i) = r_i$ . We choose each  $h_i$  to be monic in  $Y$ . Then from Corollary 2.18 we get that

$$\dim(\ker(\text{Syl}_Y(\hat{f}, \partial_Y \hat{f}))) = \sum_{i=1}^k (e_i - 1)r_i =: R.$$

Since  $\boldsymbol{\alpha}$  preserves the rank of the Sylvester matrix, we have  $\dim(\ker(\text{Syl}_Y(\hat{f}(\boldsymbol{\alpha}, Y), \partial_Y \hat{f}(\boldsymbol{\alpha}, Y)))) = R$ . Assume the irreducible factorization of  $\hat{f}(\boldsymbol{\alpha}, Y)$  is  $p_1(Y)^{l_1} \dots p_r(Y)^{l_r}$  and  $\deg(p_i) = b_i$ . Then again from Corollary 2.18 we have  $R = \sum_{j=1}^r (l_j - 1)b_j$ .

We will now compare the above expression for the kernel dimension  $R$  with the way the irreducible factors  $p_j(Y)$  distribute among the substituted factors  $h_i(\boldsymbol{\alpha}, Y)$ . This will show that no  $p_j(Y)$  appears in two different substituted factors  $h_i(\boldsymbol{\alpha}, Y)$ , and that it appears with multiplicity at most one inside whichever  $h_i(\boldsymbol{\alpha}, Y)$  contains it. In other words, each  $h_i(\boldsymbol{\alpha}, Y)$  is square-free, and for  $i \neq j$ , the polynomials  $h_i(\boldsymbol{\alpha}, Y)$  and  $h_j(\boldsymbol{\alpha}, Y)$  are coprime.

Let the irreducible factorization of each  $h_i(\boldsymbol{\alpha}, Y)$  be

$$h_i(\boldsymbol{\alpha}, Y) = p_1(Y)^{l_{i,1}} \dots p_r(Y)^{l_{i,r}}.$$

Since the  $h_i$  are monic in  $Y$ , substituting  $\mathbf{X} = \boldsymbol{\alpha}$  preserves their  $Y$ -degrees. Clearly  $\sum_{i=1}^k e_i l_{i,j} = l_j$  and  $r_i = \sum_{j=1}^r l_{i,j} b_j$ . So

$$\begin{aligned} R &= \sum_{j=1}^r (l_j - 1)b_j \\ &= \sum_{j=1}^r ((\sum_{i=1}^k e_i l_{i,j}) - 1)b_j \\ &\geq \sum_{j=1}^r (\sum_{i=1}^k (e_i - 1)l_{i,j})b_j && \text{(as } \sum_i l_{i,j} \geq 1) \\ &= \sum_{i=1}^k (e_i - 1) \sum_{j=1}^r l_{i,j} b_j \\ &= \sum_{i=1}^k (e_i - 1)r_i = R \end{aligned}$$

Hence the inequality must be an equality, and in particular  $\sum_i l_{i,j} = 1$  for every  $j$ . This means that for all  $j$ , there is only one  $i$  such that  $l_{i,j} = 1$ , and all other  $l_{i',j} = 0$ . In other words,  $p_j(Y)$  divides only one  $h_i(\boldsymbol{\alpha}, Y)$ , and it does so with multiplicity one. Thus the polynomials  $h_i(\boldsymbol{\alpha}, Y)$  are pairwise coprime, and each  $h_i(\boldsymbol{\alpha}, Y)$  is square-free. By the standard resultant and discriminant criteria from Lemma 2.19, this implies that for all  $i \neq j$ ,  $\text{Res}_Y(h_i, h_j)(\boldsymbol{\alpha}) \neq 0$ , and for every  $i$ ,  $\text{Disc}_Y(h_i)(\boldsymbol{\alpha}) \neq 0$ . These are precisely the conditions for  $\hat{f}$  to be Hensel-ready at  $\mathbf{X} = \boldsymbol{\alpha}$ .  $\blacksquare$

To apply Hensel's lemma (Lemma 2.26), we need the polynomial to be Hensel-ready at  $\mathbf{X} = \mathbf{0}$ . We therefore shift  $\mathbf{X} \mapsto \mathbf{X} + \boldsymbol{\alpha}$  in  $\hat{f}$ . This shift may destroy sparsity, but sparsity is only needed to find a Hensel-ready point. Once the shift has been found, the power-series factorization no longer uses sparsity. After computing the potential factors of the shifted polynomial, we will shift back by  $\mathbf{X} \mapsto \mathbf{X} - \boldsymbol{\alpha}$  to recover the corresponding factors of the original polynomial.

**Lifting factors from power-series roots.** Let  $\hat{f}_\alpha(\mathbf{X}, Y) = \hat{f}(\mathbf{X} + \boldsymbol{\alpha}, Y)$ . Then  $\hat{f}_\alpha$  is monic in  $Y$ , has  $\deg_Y(\hat{f}_\alpha) = d$ , has total  $\mathbf{X}$ -degree at most  $D \leq nd^2$ , and is Hensel-ready at  $\mathbf{X} = \mathbf{0}$ . Moreover, we can construct a constant-depth circuit for  $\hat{f}_\alpha$ , since  $f$  is sparse and the operations used to obtain  $\hat{f}_\alpha$  from  $f$  can be implemented in constant depth.

Let  $\mathcal{B}$  be the multiset of roots of  $\hat{f}_\alpha(0, Y)$ . By Corollary 2.27, every factor of  $\hat{f}_\alpha(\mathbf{X}, Y)$  arises by lifting a factor of  $\hat{f}_\alpha(0, Y)$ . More concretely, if  $S \subseteq \mathcal{B}$  is a sub-multiset and  $q(Y) = \prod_{\beta \in S} (Y - \beta)$  is a factor of  $\hat{f}_\alpha(0, Y)$ , then  $\prod_{\beta \in S} (Y - \Phi_\beta)$ , truncated up to  $\mathbf{X}$ -degree  $D$ , is a potential factor of  $\hat{f}_\alpha$ ; here  $\Phi_\beta$  is the unique power-series root of  $\hat{f}_\alpha(\mathbf{X}, Y)$  with constant term  $\beta$  given by Lemma 2.26. This potential factor is a true factor precisely when the corresponding product of power series,  $\prod_{\beta \in S} (Y - \Phi_\beta)$ , is in fact a polynomial. We now show how to construct small constant-depth circuits for all such potential factors.

[BKR<sup>+</sup>26] showed how to compute the homogeneous components of  $\Phi_\beta$  in constant depth. Following their approach, we introduce a fresh variable  $T$  and replace each  $X_i$  by  $TX_i$ . We then view the resulting polynomial as a bivariate polynomial in  $T$  and  $Y$ , with coefficients in  $\mathbb{F}[\mathbf{X}]$ . We refer to Section 2.6 for a more complete discussion. The key point is that the Laurent series  $R_e(Z)$  from Theorem 2.28, viewed as a series in  $Z$  whose coefficients depend on the variables  $\mathbf{X}$  and  $T$ , can be represented as a ratio of two constant-depth circuits. Substituting  $Z = \beta$  gives the truncated power-series root  $\Phi_\beta(T)$ , and then setting  $T = 1$  gives  $\Phi_\beta(\mathbf{X})$  truncated up to degree  $D$ .

Define  $F(T, Y) = \hat{f}_\alpha(TX_1, TX_2, \dots, TX_n, Y) \in \mathbb{F}[\mathbf{X}][T, Y]$ . Since  $F(0, Y) = \hat{f}_\alpha(0, Y)$ , the multiset of roots of  $F(0, Y)$  is exactly  $\mathcal{B}$ . For  $\beta \in \mathcal{B}$ , let  $\Phi_\beta(T) \in \mathbb{F}(\beta)[\mathbf{X}][[T]]$  denote the power-series root of  $F(T, Y)$  corresponding to  $\beta$ . Then the set  $\{\Phi_\beta(1) : \beta \in \mathcal{B}\}$  is exactly the set of power-series roots of  $\hat{f}_\alpha(\mathbf{X}, Y)$ .

Let  $q(Y)$  be a factor of  $F(0, Y)$ . As discussed above,  $q(Y)$  potentially lifts to a factor of  $F(T, Y)$ . We formalize this correspondence in the following definition.

**Definition 3.3.** Let  $q(Y)$  be a factor of  $F(0, Y)$  and  $S$  be the multiset of roots of  $q(Y)$ . We define the *factor-lift* of  $q(Y)$  to be  $\prod_{\beta \in S} (Y - \Phi_\beta(T)) \text{ Trunc } T^{D+1}$ .

The next lemma records that true factors are recovered by this lifting operation.

**Lemma 3.4.** *If  $g(\mathbf{X}, Y)$  is a factor of  $\hat{f}_\alpha(\mathbf{X}, Y)$ , then  $g(\mathbf{X}, Y)$  is obtained from the factor-lift of  $g(0, Y)$  by setting  $T = 1$ .*

*Proof.* By Corollary 2.27, if  $S$  is the multiset of roots of  $g(0, Y)$ , then  $g(T\mathbf{X}, Y) = \prod_{\beta \in S} (Y - \Phi_\beta(T))$  as a power series in  $T$ . Since  $g(\mathbf{X}, Y)$  is a polynomial of total  $\mathbf{X}$ -degree at most  $D$ , the polynomial  $g(T\mathbf{X}, Y)$  has  $T$ -degree at most  $D$ . Therefore

$$\begin{aligned} g(T\mathbf{X}, Y) &= \prod_{\beta \in S} (Y - \Phi_\beta(T)) \\ &= \prod_{\beta \in S} (Y - \Phi_\beta(T)) \text{ Trunc } T^{D+1} \end{aligned}$$

Setting  $T = 1$  proves the lemma. ■

For any  $\beta \in S$ , let  $e(\beta)$  be the multiplicity of  $\beta$  in  $F(0, Y)$ . Using Theorem 2.28, we can rewrite the factor-lift as

$$\begin{aligned} \prod_{\beta \in S} (Y - \Phi_\beta(T)) \operatorname{Trunc} T^{D+1} &= \prod_{\beta \in S} (Y - (\Phi_\beta(T) \operatorname{Trunc} T^{D+1})) \operatorname{Trunc} T^{D+1} \\ &= \prod_{\beta \in S} (Y - (R_{e(\beta)}(\beta) \operatorname{Trunc} T^{D+1})) \operatorname{Trunc} T^{D+1} \\ &= \prod_{\beta \in S} (Y - R_{e(\beta)}(\beta)) \operatorname{Trunc} T^{D+1} \end{aligned}$$

The rational function  $R_e(Z)$  has a representation as a ratio of constant-depth circuits in the variables  $\mathbf{X}, T, Z$ , where  $Z$  is the main variable and  $\mathbf{X}, T$  are retained as formal variables. Moreover, its denominator depends only on  $Z$ . Hence, after substituting  $Z = \beta$ , the denominator becomes a constant in the field extension containing  $\beta$ . This gives a constant-depth representation of  $R_e(\beta)$  as a series in  $\mathbf{X}$  and  $T$ . Since multiplication and truncation can also be carried out in constant depth, every factor-lift of a factor of  $F(0, Y)$  has a constant-depth representation over a suitable field extension. The remaining issue is that these circuits may use constants such as  $\beta$  outside the base field  $\mathbb{F}$ . To remove these extension-field constants, we use ideas from [BKR<sup>+</sup>26], and in particular Lemma 2.10, to obtain constant-depth representations of the same factor-lifts using only constants from  $\mathbb{F}$ .

**Ensuring output circuits are over the base field.** We now formalize the base-field conversion step. The input to this step is a factor  $q(Y)$  of  $F(0, Y)$ , together with its factorization over  $\mathbb{F}$ . Although the factor-lift is naturally described using the roots of  $q(Y)$  over an extension field, the following lemma shows that it can be computed by a constant-depth circuit over the original field  $\mathbb{F}$ .

**Lemma 3.5.** *Let  $F(T, Y)$  be as above, with  $\deg_Y(F) = d$ . Let  $q(Y) \in \mathbb{F}[Y]$  be a factor of  $F(0, Y)$ , and suppose its irreducible factorization over  $\mathbb{F}$  is  $q(Y) = q_1(Y)^{l_1} \dots q_r(Y)^{l_r}$ . Then there is an algorithm that takes as input a size  $s'$  constant-depth circuit for  $F(T, Y)$  and the irreducible factorization of  $q(Y)$ , and outputs a constant-depth circuit over  $\mathbb{F}$  for the factor-lift of  $q(Y)$  in  $\operatorname{poly}(n, s', d)$  time.*

*Proof.* We first express the factor-lift in terms of the power-series root formula from Theorem 2.28, and then explain why the resulting expression can be written over the base field.

For each  $i \in [r]$ , let  $d_i = \deg(q_i)$ , let  $S_i$  be the set of roots of  $q_i$  over  $\overline{\mathbb{F}}$ , and let  $e_i$  be the multiplicity of  $q_i$  as a factor of  $F(0, Y)$ . Since  $q_i$  is irreducible, every  $\beta \in S_i$  has multiplicity  $e_i$  as a root of  $F(0, Y)$ . As  $q_i$  appears in  $q$  with multiplicity  $l_i$ , the factor-lift of  $q(Y)$  is

$$\prod_{i=1}^r \left( \prod_{\beta \in S_i} (Y - R_{e_i}(\beta)) \right)^{l_i} \operatorname{Trunc} T^{D+1}.$$

Fix  $i \in [r]$ . By Theorem 2.28,  $R_{e_i}(Z)$  can be written as a rational function  $A_i(\mathbf{X}, T, Z)/B_i(Z)$ , where  $A_i$  and  $B_i$  have  $\operatorname{poly}(n, s', d)$ -size,  $\mathcal{O}(1)$ -depth circuits constructible in  $\operatorname{poly}(n, s', d)$  time, and  $B_i(Z)$  depends only on the main variable  $Z$ . Moreover, Theorem 2.28 guarantees that  $B_i(\beta) \neq 0$  for every  $\beta \in S_i$ , since each such  $\beta$  has multiplicity exactly  $e_i$  in  $F(0, Y)$ . The product  $\prod_{\beta \in S_i} (Y - R_{e_i}(\beta))$  is symmetric in the roots in  $S_i$ . Expanding it gives

$$\prod_{\beta \in S_i} (Y - R_{e_i}(\beta)) = Y^{d_i} + \sum_{a=1}^{d_i} (-1)^a \operatorname{Esym}_a \left\{ R_{e_i}(\beta) \mid \beta \in S_i \right\} Y^{d_i-a}.$$

Applying Lemma 2.10 to the irreducible polynomial  $q_i$  and the rational function  $A_i/B_i$ , we can construct, for every  $a \in [d_i]$ , a constant-depth circuit (with division gate at the top) over  $\mathbb{F}$  for the elementary symmetric expression  $\text{Esym}_a\{R_{e_i}(\beta) \mid \beta \in S_i\}$ . To make the denominator clearing explicit, let  $P_i = \prod_{\beta \in S_i} B_i(\beta)$ . Multiplying  $\text{Esym}_a\{R_{e_i}(\beta) \mid \beta \in S_i\}$  by  $P_i$  clears all denominators in each summand, and the resulting expression is symmetric in the roots of the monic polynomial  $q_i$ . Hence, by the fundamental theorem of symmetric polynomials, it lies in  $\mathbb{F}[\mathbf{X}, T]$ . Also  $P_i = \text{Res}(q_i, B_i)$  (by the Poisson formula for resultants) is a nonzero element of  $\mathbb{F}$ . We can therefore absorb  $P_i^{-1}$  into the circuit constants. It follows that the coefficients of  $\prod_{\beta \in S_i} (Y - R_{e_i}(\beta))$  have constant-depth circuits over  $\mathbb{F}$ , constructible in  $\text{poly}(n, s', d)$  time.

We apply this construction for every irreducible factor  $q_i$ , raise the corresponding lifted factor to the power  $l_i$ , multiply these expressions, and truncate at  $T^{D+1}$ . These operations preserve constant depth and can be carried out in  $\text{poly}(n, s', d)$  time (using product gates for powers and products, and Lemma 2.8 for the final truncation), giving the desired circuit for the factor-lift of  $q(Y)$ . ■

We remark that another way to prove the above lemma is to make use of the fact that the factor-lift of  $q(Y)$  is a multi-symmetric polynomial in its roots. Although the factor-lift of  $q(Y)$  need not be symmetric in all roots of  $q(Y)$  at once, it is symmetric separately in the roots of each irreducible factor  $q_i(Y)$ . Thus Theorem 3.6 of [BKR<sup>+</sup>25a], which generalizes Lemma 2.10 to multi-symmetric polynomials, gives an alternative route to the same constant-depth construction over  $\mathbb{F}$ .

Now we are ready to prove Lemma 3.1.

*Proof of Lemma 3.1.* Given  $f(\mathbf{X}, Y)$  in monomial representation, we first compute the normalization  $\hat{f}(\mathbf{X}, Y)$ . This has at most  $s^d$  monomials and can be computed efficiently. We then apply Lemma 3.2 to find  $\alpha \in \mathbb{F}^n$  such that  $\hat{f}_\alpha(\mathbf{X}, Y) = \hat{f}(\mathbf{X} + \alpha, Y)$  is Hensel-ready at  $\mathbf{X} = \mathbf{0}$ . As discussed above, we can construct a constant-depth circuit for  $\hat{f}_\alpha$ , and hence also for  $F(T, Y) = \hat{f}_\alpha(T\mathbf{X}, Y)$ .

We factor the univariate polynomial  $F(0, Y)$  using the deterministic univariate factorization algorithm summarized in Section 2.3. Starting with an empty list  $\mathcal{L}$ , for every non-constant factor  $q(Y)$  of  $F(0, Y)$ , we compute the factor-lift  $\hat{q}(\mathbf{X}, T, Y)$  using Lemma 3.5, and add  $\hat{q}(\mathbf{X} - \alpha, 1, Y)$  to  $\mathcal{L}$ . Since  $F(0, Y)$  has degree at most  $d$ , it has at most  $2^d - 1$  non-constant factors. Thus  $|\mathcal{L}| \leq 2^d - 1$ , and the whole procedure runs in time  $\text{poly}(n, d!, s^d, \mathcal{T}(\mathbb{F}, d))$ .

Finally, let  $g(\mathbf{X}, Y)$  be any non-constant factor of  $\hat{f}(\mathbf{X}, Y)$ . Then  $g_\alpha(\mathbf{X}, Y) = g(\mathbf{X} + \alpha, Y)$  is a factor of  $\hat{f}_\alpha(\mathbf{X}, Y)$ . By Lemma 3.4,  $g_\alpha$  is obtained by setting  $T = 1$  in the factor-lift of the univariate factor  $g_\alpha(0, Y)$  of  $F(0, Y)$ . Therefore, when the algorithm processes  $g_\alpha(0, Y)$ , it adds  $g_\alpha(\mathbf{X} - \alpha, Y) = g(\mathbf{X}, Y)$  to the list. This proves correctness. ■

We now show how factoring reverse-normalized polynomials can be reduced to factoring normalized polynomials, and thus show how to recover an analog of Lemma 3.1 for the case of reverse-normalized polynomials.

**Corollary 3.6.** *Let  $\mathbb{F}$  be a field of characteristic 0 or greater than  $d$ . Let  $f(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  be an  $n$ -variate polynomial with sparsity at most  $s$  and individual degree at most  $d$ . Suppose  $\tilde{f}(\mathbf{X}, Y) \in \mathbb{F}[\mathbf{X}, Y]$  is the reverse-normalized form of  $f$  with respect to  $Y$ . Then there is an algorithm running in time  $\text{poly}(s^d, d!, n, \mathcal{T}(\mathbb{F}, d))$  that outputs a list  $\mathcal{L}$  such that*

1.  $|\mathcal{L}| \leq 2^d - 1$
2. every element of  $\mathcal{L}$  is a  $\text{poly}(n, s, d)$ -size,  $\mathcal{O}(1)$ -depth circuit over  $\mathbb{F}$
3. every non-constant factor  $\tilde{g}(\mathbf{X}, Y)$  of  $\tilde{f}(\mathbf{X}, Y)$  is computed by some circuit in  $\mathcal{L}$

The list  $\mathcal{L}$  may contain spurious candidates. Moreover, the algorithm can compute the  $Y$ -degree of the polynomial computed by every circuit in  $\mathcal{L}$ .

*Proof.* Let  $g = \text{rev}_Y(f)$ . The normalization of  $g$  is exactly  $\text{rev}_Y(\tilde{f})$ . Since  $g = \text{rev}_Y(f)$  is still  $s$ -sparse and has individual degree at most  $d$ , we can apply Lemma 3.1 to  $g$  and obtain a list of circuits containing all non-constant factors of  $\text{rev}_Y(\tilde{f})$ . By Lemma 2.2, reversing these candidate circuits gives a list containing all non-constant factors of  $\tilde{f}$ . Reversal preserves the list size, constant depth, and the claimed size bound (using coefficient extraction in  $Y$ , as in Lemma 2.6), and the  $Y$ -degree of each reversed candidate is computable from the corresponding candidate for  $\text{rev}_Y(\tilde{f})$ . ■

## 4 Recovering *all* Factors of Sparse Polynomials of Bounded Individual Degree

In this section, we prove Theorem 1.1. In particular, we obtain a deterministic polynomial-time algorithm that recovers *all* factors  $g$  (not divisible by a monomial) of an  $n$ -variate  $s$ -sparse polynomial  $f \in \mathbb{F}[\mathbf{X}_n]$  of bounded individual degree  $d$ . The algorithm outputs an  $s^d$ -size list  $\mathcal{L}$  of constant-depth circuits such that each factor  $g$  is computed by some circuit in  $\mathcal{L}$ .  $\mathcal{L}$  might contain spurious circuits that do not compute any factor of  $f$ . Unfortunately, we are unable to prune out the spurious elements, since we do not know of efficient deterministic algorithms to check if a constant-depth circuit divides  $f$  (see Section 6). Note that the restriction of outputting factors not divisible by a monomial is unavoidable (see Remark 1.4). Since we are given white-box access to the polynomial  $f$ , we compensate for this restriction by outputting the maximal degree monomial factor dividing  $f$ . Throughout this section, when a list consists of circuits, we often identify a circuit in the list with the polynomial it computes.

**Theorem 4.1** (Main Theorem 1). *Let  $f(\mathbf{X}_n) \in \mathbb{F}[\mathbf{X}_n]$  be an  $s$ -sparse polynomial with individual degree at most  $d$ , where  $\text{char}(\mathbb{F})$  is 0 or greater than  $d$ . Then, there is an algorithm that runs in time  $\text{poly}(s^d, d!, n)$  and outputs an  $s^d$ -size list of circuits  $\mathcal{L}$  such that every factor of  $f$  that is not divisible by a monomial is computed by some circuit in  $\mathcal{L}$ . The algorithm also outputs the monomial of maximal degree dividing  $f$ . Moreover, each circuit in  $\mathcal{L}$  is over  $\mathbb{F}$  and has size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ .*

**Remark 4.2.** In the above theorem, the displayed running time suppresses the dependence on the time taken to perform univariate polynomial factorization. More precisely, over a field  $\mathbb{F}$ , the algorithm runs in time  $\text{poly}(s^d, d!, n) \cdot \mathcal{T}(\mathbb{F}, d)$  (see Definition 2.11). In particular, over  $\mathbb{Q}$ , if the input polynomial has bit complexity  $B$ , then the algorithm runs in time  $\text{poly}(s^d, d!, n, B)$ . If  $\mathbb{F}$  is a finite field with characteristic  $p$  (greater than  $d$ ), the algorithm runs in time  $\text{poly}(s^d, d!, n, p, \log |\mathbb{F}|)$ .

The algorithm runs in two stages, the main recursive Algorithm 1 and the post-processing Algorithm 2. Algorithm 1 outputs a set  $\mathcal{L}_{\text{mon}}$  of *monomial equivalent* factors of  $f$  as constant-depth circuits (with division gates). More precisely, for every factor  $g$  of  $f$ , there exists a polynomial  $h$  computed by some circuit in  $\mathcal{L}_{\text{mon}}$  such that  $g \prec_d h$  (see Definition 2.1). So for a factor  $g$  with no monomial divisor,  $h = gM$  will be computed by some circuit  $C$  in  $\mathcal{L}_{\text{mon}}$ , where  $M$  is a monomial with individual degree at most  $d$ . Algorithm 2 strips out this monomial  $M$  from circuit  $C$  by iteratively using Klivans-Spielman PIT on factors of sparse polynomials (see Corollary 2.31), thus obtaining the promised list  $\mathcal{L}$  containing factors<sup>8</sup>. Additionally, Algorithm 2 will eliminate all division gates.

<sup>8</sup>On members of the list that do not correspond to true factors, we do not guarantee what might come out of this process, but it does not affect our run time or the guarantee on true factors.

We formally argue the correctness and the runtime of Algorithm 1 below. We refer the reader to Section 1.2 for a high-level overview of the key ideas in the algorithm.

**Theorem 4.3** (Theorem for the Recursive Algorithm). *Let  $f(\mathbf{X}_n) \in \mathbb{F}[\mathbf{X}_n]$  be an  $s$ -sparse polynomial with individual degree at most  $d$ , where  $\text{char}(\mathbb{F})$  is 0 or greater than  $d$ . There is an algorithm that runs in time  $\text{poly}(s^d, d!, n)$  and outputs an  $s^d$ -size list  $\mathcal{L}_{\text{mon}}$  of circuits with division gates. For every factor  $g$  of  $f$ , there exists a polynomial  $h$  computed by some circuit in  $\mathcal{L}_{\text{mon}}$  such that  $g \prec_d h$ . Moreover, each circuit in  $\mathcal{L}_{\text{mon}}$  is over  $\mathbb{F}$  and has size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ .*

*Proof.* We will show that Algorithm 1 outputs the required list of circuits. We prove the theorem by induction on the sparsity of  $f$ . Algorithm 1 initially strips off the maximal monomial divisor of  $f$ . Thus, from this point on, we work with an  $f$  that has no nonconstant monomial divisor.

**Base case** ( $\|f\| \leq 1$ ): Then  $f$  is a constant, since we have stripped out the maximal monomial divisor. Algorithm 1 outputs  $\{f \cdot (X_1 \cdot X_2 \cdots X_n)^d\}$ , and thus gives a  $d$ -monomial equivalent polynomial to the constant factor 1.

**Induction case** ( $\|f\| \leq s$ ): Without loss of generality,  $f$  depends on the variable  $X_n$  (Algorithm 1 ensures this by renaming variables). Let  $k = \deg_{X_n}(f)$ , and

$$f = \sum_{i=0}^k f_i(\mathbf{X}_{n-1}) X_n^i$$

where  $f_i(\mathbf{X}_{n-1}) \in \mathbb{F}[\mathbf{X}_{n-1}]$ . Since we stripped off the maximal monomial divisor,  $f_0(\mathbf{X}_{n-1})$  and  $f_k(\mathbf{X}_{n-1})$  are both non-zero. By averaging, either  $\|f_0(\mathbf{X}_{n-1})\|$  or  $\|f_k(\mathbf{X}_{n-1})\|$  is at most  $s/2$ . We first deal with the case where  $\|f_k(\mathbf{X}_{n-1})\|$  is at most  $s/2$ .

**Sparsity of  $f_k \leq s/2$ :** Algorithm 1 normalizes  $f$  to get  $\hat{f}(\mathbf{X}_n) = f(\mathbf{X}_{n-1}, X_n/f_k) \cdot f_k^{k-1}$ , which is monic in  $X_n$  and has sparsity at most  $s^d$ . The algorithm invokes Lemma 3.1 to compute a  $(2^d - 1)$ -size list  $\mathcal{L}_{\hat{f}}$  of circuits such that every factor of  $\hat{f}$  is computed by some circuit in  $\mathcal{L}_{\hat{f}}$ .

$$|\mathcal{L}_{\hat{f}}| \leq 2^d - 1 \tag{4.1}$$

Moreover, this computation takes  $\text{poly}(d!, s^d, n)$  time and each circuit in  $\mathcal{L}_{\hat{f}}$  is over  $\mathbb{F}$  and is of size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ .

Next it recursively computes the list  $\mathcal{L}_{f_k, \text{mon}}$  of circuits corresponding to monomial equivalent factors of  $f_k$ . For every factor  $g$  of  $f_k$ , there exists a polynomial  $h$  computed by some circuit in  $\mathcal{L}_{f_k, \text{mon}}$  such that  $g \prec_d h$ . By our induction hypothesis, since  $\|f_k\| \leq s/2$ , we have  $|\mathcal{L}_{f_k, \text{mon}}| \leq (s/2)^d$  and we can compute this list in time  $\text{poly}(d!, (s/2)^d, n)$ .

$$|\mathcal{L}_{f_k, \text{mon}}| \leq (s/2)^d \tag{4.2}$$

Moreover, each circuit in  $\mathcal{L}_{f_k, \text{mon}}$  is over  $\mathbb{F}$  and has size  $\text{poly}((s/2)^d, n)$  and depth  $\mathcal{O}(1)$ .

Finally, the algorithm uses the lists  $\mathcal{L}_{\hat{f}}$  and  $\mathcal{L}_{f_k, \text{mon}}$  to construct the list  $\mathcal{L}_{\text{mon}}$ . It defines  $\mathcal{L}_{\text{mon}}$  as follows.

$$\mathcal{L}_{\text{mon}} = \left\{ \hat{g}(\mathbf{X}_{n-1}, X_n f_k) \cdot \frac{h}{f_k^{\deg_{X_n}(\hat{g})}} \mid \hat{g} \in \mathcal{L}_{\hat{f}}, h \in \mathcal{L}_{f_k, \text{mon}} \right\} \cup \mathcal{L}_{f_k, \text{mon}}$$

where  $\deg_{X_n}(\hat{g})$  is obtained from Lemma 3.1. We will argue next that for any factor of  $f$ , there is a  $d$ -monomial equivalent polynomial in  $\mathcal{L}_{mon}$ .

Let  $g$  be a factor of  $f$ . Suppose  $g$  does not depend on the variable  $X_n$ . Then  $g$  divides  $f_k$ . Hence, there exists a  $g'$  in  $\mathcal{L}_{f_k, mon} \subset \mathcal{L}_{mon}$  such that  $g \prec_d g'$ .

Now suppose  $g$  depends on  $X_n$ . Let  $g$  be of the following form,

$$g(\mathbf{X}_n) = \sum_{i=0}^t g_i(\mathbf{X}_{n-1}) X_n^i$$

where  $g_t \neq 0$ . Then by Lemma 2.23,  $\hat{g}|f$  where  $\hat{g}$  is defined by

$$\hat{g}(\mathbf{X}_n) = g(\mathbf{X}_{n-1}, X_n/f_k) \frac{f_k^t}{g_t}.$$

In other words,

$$g(\mathbf{X}_n) = \hat{g}(\mathbf{X}_{n-1}, X_n f_k) \frac{g_t}{f_k^t}.$$

Note that  $g_t|f_k$ , which implies there exists a  $g'_t$  in  $\mathcal{L}_{f_k, mon}$  such that  $g_t \prec_d g'_t$ . Hence, the polynomial  $g'(\mathbf{X}_n)$  defined as

$$g'(\mathbf{X}_n) = \hat{g}(\mathbf{X}_{n-1}, X_n f_k) \cdot \frac{g'_t}{f_k^t}$$

lies in  $\mathcal{L}_{mon}$  since  $\hat{g} \in \mathcal{L}_{\hat{f}}$ . Moreover,  $g \prec_d g'$ , which finishes the proof that  $\mathcal{L}_{mon}$  contains the claimed circuits for this case.

Next, we prove the bound on the size of  $\mathcal{L}_{mon}$ , the runtime of the algorithm, and the size and depth of the circuits.

Note that

$$\begin{aligned} |\mathcal{L}_{mon}| &= |\mathcal{L}_{\hat{f}}| \cdot |\mathcal{L}_{f_k, mon}| + |\mathcal{L}_{f_k, mon}| \\ &\leq (2^d - 1) \cdot |\mathcal{L}_{f_k, mon}| + |\mathcal{L}_{f_k, mon}| && \text{(Using Eq. (4.1))} \\ &\leq 2^d \cdot \left(\frac{s}{2}\right)^d && \text{(Using Eq. (4.2))} \\ &= s^d \end{aligned}$$

which proves the list bound.

The total time taken is dominated by the time taken to compute the lists  $\mathcal{L}_{\hat{f}}$  and  $\mathcal{L}_{f_k, mon}$  and to combine them to compute  $\mathcal{L}_{mon}$ , which takes  $\text{poly}(d!, s^d, n)$  time.

We conclude by showing the size and depth bounds for circuits in  $\mathcal{L}_{mon}$ . The circuits in  $\mathcal{L}_{\hat{f}}$  are of size  $\text{poly}(s^d, n)$  and depth  $\Delta$ . Here  $\Delta$  is an absolute constant (by Lemma 3.1). Without loss of generality, the top gate of these circuits is a multiplication gate.

We will inductively show that the circuits in  $\mathcal{L}_{mon}$  have  $\text{poly}(s^d, n)$  size and  $\Delta + 6$  depth with a multiplication gate on top. The recursively inherited circuits in  $\mathcal{L}_{f_k, mon}$  satisfy these bounds by the induction hypothesis. It remains to consider the newly constructed circuits, which are of the form

$$\hat{g}(\mathbf{X}_{n-1}, X_n f_k) \cdot \frac{h}{f_k^{\deg_{X_n}(\hat{g})}}$$

where  $\hat{g} \in \mathcal{L}_{\hat{f}}$  and  $h \in \mathcal{L}_{f_k, mon}$ . By Lemma 2.9,  $\hat{g}(\mathbf{X}_{n-1}, X_n f_k)$  is computable by a circuit  $C_1$  of size  $\text{poly}(s^d, n)$  and depth  $\Delta + 6$  (with a multiplication gate on top).  $1/f_k^{\deg_{X_n}(\hat{g})}$  is computable

by a circuit  $C_2$  (recall that we are allowing division gates) of size  $\text{poly}(s, n)$  and depth 5 (with a multiplication gate on top). By induction,  $h$  is computable by a circuit  $C_3$  of size  $\text{poly}(s^d, n)$  and depth  $\Delta + 6$  (with a multiplication gate on top). We multiply all these circuits together by putting a multiplication gate on top. The resulting circuit will have size  $\text{poly}(s^d, n)$  and depth  $\Delta + 6$  (we merge the multiplication gates of  $C_1, C_2, C_3$  into one).

We now address the case where sparsity of  $f_0 < s/2$ . Many of the ingredients are common to the previous case, but the normalization and combining procedures to obtain  $\mathcal{L}_{mon}$  are different.

**Sparsity of  $f_0 < s/2$ :** Algorithm 1 reverse-normalizes  $f$  to get  $\tilde{f}(\mathbf{X}_n) = f(\mathbf{X}_{n-1}, X_n f_0)/f_0$ , which is reversed-monic in  $X_n$  and has sparsity at most  $s^d$ . The algorithm invokes Corollary 3.6 to compute a  $(2^d - 1)$ -size list  $\mathcal{L}_{\tilde{f}}$  of circuits such that every factor of  $\tilde{f}$  is computed by some circuit in  $\mathcal{L}_{\tilde{f}}$ . Moreover, this computation takes  $\text{poly}(d!, s^d, n)$  time and each circuit in  $\mathcal{L}_{\tilde{f}}$  is over  $\mathbb{F}$  and is of size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ .

Next it recursively computes the list  $\mathcal{L}_{f_0, mon}$  of circuits. For every factor  $g$  of  $f_0$ , there exists a polynomial  $h$  computed by some circuit in  $\mathcal{L}_{f_0, mon}$  such that  $g \prec_d h$ . By our induction hypothesis, since  $\|f_0\| \leq s/2$ , we have  $|\mathcal{L}_{f_0, mon}| \leq (s/2)^d$  and we can compute this list in time  $\text{poly}(d!, (s/2)^d, n)$ . Moreover, each circuit in  $\mathcal{L}_{f_0, mon}$  is over  $\mathbb{F}$  and has size  $\text{poly}((s/2)^d, n)$  and depth  $\mathcal{O}(1)$ .

Finally, the algorithm uses the lists  $\mathcal{L}_{\tilde{f}}$  and  $\mathcal{L}_{f_0, mon}$  to construct the list  $\mathcal{L}_{mon}$ . It defines  $\mathcal{L}_{mon}$  as follows.

$$\mathcal{L}_{mon} = \left\{ \tilde{g}(\mathbf{X}_{n-1}, X_n/f_0) \cdot h \mid \tilde{g} \in \mathcal{L}_{\tilde{f}}, h \in \mathcal{L}_{f_0, mon} \right\} \cup \mathcal{L}_{f_0, mon}$$

We will argue next that for any factor of  $f$ , there is a  $d$ -monomial equivalent polynomial in  $\mathcal{L}_{mon}$ .

Let  $g$  be a factor of  $f$ . Suppose  $g$  does not depend on the variable  $X_n$ . Then  $g$  divides  $f_0$ . Hence, there exists a  $g'$  in  $\mathcal{L}_{f_0, mon} \subset \mathcal{L}_{mon}$  such that  $g \prec_d g'$ .

Now suppose  $g$  depends on  $X_n$ . Let  $g$  be of the following form,

$$g(\mathbf{X}_n) = \sum_{i=0}^t g_i(\mathbf{X}_{n-1}) X_n^i$$

where  $g_0 \neq 0$  (since we had stripped out the maximal monomial divisor initially). Then by Lemma 2.24,  $\tilde{g} \mid f$  where  $\tilde{g}$  is defined by

$$\tilde{g}(\mathbf{X}_n) = \frac{g(\mathbf{X}_{n-1}, X_n f_0)}{g_0}.$$

In other words,

$$g(\mathbf{X}_n) = \tilde{g}(\mathbf{X}_{n-1}, X_n/f_0) \cdot g_0.$$

Note that  $g_0 \mid f_0$ , which implies there exists a  $g'_0$  in  $\mathcal{L}_{f_0, mon}$  such that  $g_0 \prec_d g'_0$ . Hence, the polynomial  $g'(\mathbf{X}_n)$  defined as

$$g'(\mathbf{X}_n) = \tilde{g}(\mathbf{X}_{n-1}, X_n/f_0) \cdot g'_0$$

lies in  $\mathcal{L}_{mon}$  since  $\tilde{g} \in \mathcal{L}_{\tilde{f}}$ . Moreover,  $g \prec_d g'$ , which finishes the proof of correctness for this case.

The arguments for the bounds on the size of  $\mathcal{L}_{mon}$  and the runtime of the algorithm are the same as in the previous case. Thus, the size of  $\mathcal{L}_{mon}$  is at most  $s^d$  and the total runtime is  $\text{poly}(d!, s^d, n)$ .

We conclude by showing the size and depth bounds for circuits in  $\mathcal{L}_{mon}$ . The circuits in  $\mathcal{L}_{\tilde{f}}$  are of size  $\text{poly}(s^d, n)$  and depth  $\Delta$ . Here  $\Delta$  is an absolute constant (see Corollary 3.6). Without loss of generality, the top gate of these circuits is a multiplication gate.

We will inductively show that the circuits in  $\mathcal{L}_{mon}$  have  $\text{poly}(s^d, n)$  size and  $\Delta + 6$  depth with a multiplication gate on top. The recursively inherited circuits in  $\mathcal{L}_{f_0, mon}$  satisfy these bounds by the induction hypothesis. It remains to consider the newly constructed circuits, which are of the form

$$\tilde{g}(\mathbf{X}_{n-1}, X_n/f_0) \cdot h$$

where  $\tilde{g} \in \mathcal{L}_{\tilde{f}}$  and  $h \in \mathcal{L}_{f_0, mon}$ . By Lemma 2.9,  $\tilde{g}(\mathbf{X}_{n-1}, X_n/f_0)$  is computable by a circuit  $C_1$  (in this case  $C_1$  also has division gates) of size  $\text{poly}(s^d, n)$  and depth  $\Delta + 6$  (with a multiplication gate on top). By induction,  $h$  is computable by a circuit  $C_2$  of size  $\text{poly}(s^d, n)$  and depth  $\Delta + 6$  (with a multiplication gate on top). We multiply all these circuits together by putting a multiplication gate on top. The resulting circuit will have size  $\text{poly}(s^d, n)$  and depth  $\Delta + 6$  (we merge the multiplication gates of  $C_1, C_2$  into one).

Note that in our construction of the circuits (in both cases) in  $\mathcal{L}_{mon}$ , the denominators of the division gates are products of  $s$ -sparse polynomials. We record this fact as a claim below.

**Claim 4.4.** The denominators of the division gates in the circuits in  $\mathcal{L}_{mon}$  are products of  $s$ -sparse polynomials.

Indeed, the recursive calls only introduce denominators with this form by induction. In the case  $\|f_k\| \leq s/2$ , the newly constructed circuits introduce only powers of  $f_k$ , and in the case  $\|f_0\| \leq s/2$ , the newly constructed circuits introduce only powers of  $f_0$  through the substitution  $X_n/f_0$ . Since both  $f_k$  and  $f_0$  are coefficients of the  $s$ -sparse polynomial  $f$ , they are  $s$ -sparse. ■

At this point, Algorithm 1 outputs a list  $\mathcal{L}_{mon}$  containing monomial equivalent factors of  $f$ . Algorithm 2 will perform post-processing on this list to obtain a new list  $\mathcal{L}$  that contains all factors (not divisible by a monomial) of  $f$ . We show this formally by proving Theorem 4.1 below.

*Proof of Theorem 4.1.* We will show that Algorithm 2 outputs the required list of circuits. The algorithm first removes and records the maximal monomial divisor of the input polynomial  $f$ ; this gives the monomial output promised in the theorem. It then computes the list of circuits  $\mathcal{L}_{mon}$  using Algorithm 1. By Claim 4.4, the denominators of all division gates in these circuits are products of  $s$ -sparse polynomials. Thus, before removing monomial factors, Algorithm 2 applies Lemma 2.32 to every circuit in  $\mathcal{L}_{mon}$ , replacing  $\mathcal{L}_{mon}$  by an equivalent list of division-free circuits. It then removes the largest monomial factor from each resulting circuit using Corollary 2.31, adds the result to  $\mathcal{L}$ , and finally eliminates the monomial divisions introduced in this last step.

Let  $g$  be a factor of  $f$  such that no monomial divides  $g$ . By Theorem 4.3, there exists  $h \in \mathcal{L}_{mon}$  such that  $g \prec_d h$ . By removing the largest monomial factor from  $h$ , we ensure that  $g \in \mathcal{L}$ . Thus, every factor of  $f$  that is not divisible by a monomial is computed by some circuit in  $\mathcal{L}$ .

Note that  $|\mathcal{L}| \leq |\mathcal{L}_{mon}| = s^d$ , and the runtime is dominated by Algorithm 1, which takes time  $\text{poly}(s^d, d!, n)$ . The first division-elimination step gives division-free circuits of size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ . Removing the largest monomial factor then only introduces monomial denominators, so a second application of Lemma 2.32 removes these divisions with polynomial blow-up and only a constant increase in depth. Thus, the circuits in  $\mathcal{L}$  are of size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ .

Moreover, for circuits in  $\mathcal{L}_{mon}$  that do not correspond to true monomial equivalent factors of  $f$ , the above post-processing will add circuits in  $\mathcal{L}$  that do not correspond to true factors of  $f$  (in  $\text{poly}(s^d, d!, n)$ -time). These spurious circuits are also of size  $\text{poly}(s^d, n)$  and depth  $\mathcal{O}(1)$ . ■

As a consequence of Theorem 1.1, we recover the following corollary. This recovers a result of Chuyoon and Shpilka [CS26], as well as the algorithmic result of Bhargava et al. [BSV20] and its improvement in [CS26].

**Corollary 4.5.** *Let  $f(X_1, \dots, X_n) \in \mathbb{F}[X_1, \dots, X_n]$  be an  $s$ -sparse polynomial with individual degree at most  $d$ , where  $\text{char}(\mathbb{F})$  is 0 or greater than  $d$ . Then there is an algorithm that runs in time  $\text{poly}(s^d, d!, n, s')$  and outputs all  $s'$ -sparse factors of  $f$  not divisible by a monomial.*

*Proof.* Applying Theorem 4.1 to  $f$ , we obtain a list of circuits  $\mathcal{L}$  such that every factor of  $f$  that is not divisible by a monomial is computed by some circuit in  $\mathcal{L}$ . In particular,  $\mathcal{L}$  could contain spurious circuits that do not correspond to factors of  $f$ . We apply  $s'$ -sparse polynomial interpolation (Theorem 2.30) to each circuit  $C$  in  $\mathcal{L}$ . For each circuit where interpolation succeeds, let  $g_C$  be the corresponding polynomial. We use Theorem 2.33 with sparsity parameter  $s'$  to check whether  $g_C$  divides  $f$ . We prune out the polynomials  $g_C$  that do not divide  $f$  and get a final list of all  $s'$ -sparse factors of  $f$  that are not divisible by any monomial. ■

**Remark 4.6.** By setting  $s' = s$  in Corollary 4.5, we obtain a  $\text{poly}(s^d, d!, n)$ -time algorithm to output all  $s$ -sparse factors of  $f$  not divisible by a monomial. Thus, we recover [CS26, Theorem 1.9] for fields of characteristic zero or greater than  $d$ .

By Theorem 5.4, all factors of  $f$  have sparsity at most  $s^{4d^2 \log n}$ . Thus, by setting  $s' = s^{4d^2 \log n}$  in Corollary 4.5, we obtain a  $\text{poly}(s^{d^2 \log n}, d!, n)$ -time algorithm to output all factors of  $f$  not divisible by a monomial. This generalizes [BSV20, Theorem 2] and matches the improved runtime obtained in [CS26].

---

**Algorithm 1:** List of monomial equivalent factors of sparse polynomials with bounded individual degree (possibly with spurious entries)

---

**Data:**  $f \in \mathbb{F}[\mathbf{X}_n]$ , sparsity parameter  $s$ , individual degree parameter  $d$

**Result:** List  $\mathcal{L}_{mon}$  of circuits over  $\mathbb{F}$  with division gates

// Pre-processing

Remove the maximal monomial divisor from  $f$  and rename the polynomial  $f$ .

// Base Case

**if**  $f$  is a constant **then**

$\mathcal{L}_{mon} = \{f \cdot (X_1 \cdot X_2 \cdots X_n)^d\}$

**return**  $\mathcal{L}_{mon}$

**end**

Without loss of generality assume  $f$  depends on variable  $X_n$ , otherwise rename the variables appropriately.

Let  $k = \deg_{X_n}(f)$ , and find  $f_0, \dots, f_k \in \mathbb{F}[\mathbf{X}_{n-1}]$  such that  $f = \sum_{i=0}^k f_i(\mathbf{X}_{n-1})X_n^i$

// Normalize  $f$

**if**  $\|f_k\| \leq s/2$  **then**

    Compute  $\hat{f} = f(\mathbf{X}_{n-1}, X_n/f_k) \cdot f_k^{k-1}$

    Compute  $\mathcal{L}_{\hat{f}} = \{\text{Factors of } \hat{f}\}$  using monic factorization (Lemma 3.1)

$\mathcal{L}_{f_k, mon} = \text{Algorithm 1}(f_k, s/2, d)$  // Monomial equivalent factors of  $f_k$

$\mathcal{L}_{mon} = \left\{ \hat{g}(\mathbf{X}_{n-1}, X_n f_k) \cdot \frac{h}{f_k^{\deg_{X_n}(\hat{g})}} \mid \hat{g} \in \mathcal{L}_{\hat{f}}, h \in \mathcal{L}_{f_k, mon} \right\} \cup \mathcal{L}_{f_k, mon}$

    // We know  $\deg_{X_n}(\hat{g})$  by Lemma 3.1

**end**

// Reverse Normalize  $f$

**else**

    Compute  $\tilde{f} = f(\mathbf{X}_{n-1}, X_n f_0)/f_0$

    Compute  $\mathcal{L}_{\tilde{f}} = \{\text{Factors of } \tilde{f}\}$  using reversed-monic factorization (Corollary 3.6)

$\mathcal{L}_{f_0, mon} = \text{Algorithm 1}(f_0, s/2, d)$  // Monomial equivalent factors of  $f_0$

$\mathcal{L}_{mon} = \left\{ \tilde{g}(\mathbf{X}_{n-1}, X_n/f_0) \cdot h \mid \tilde{g} \in \mathcal{L}_{\tilde{f}}, h \in \mathcal{L}_{f_0, mon} \right\} \cup \mathcal{L}_{f_0, mon}$

**end**

**return**  $\mathcal{L}_{mon}$

---

---

**Algorithm 2:** List of factors not divisible by a monomial of bounded individual degree sparse polynomials (possibly with spurious entries)

---

**Data:**  $f \in \mathbb{F}[\mathbf{X}_n]$ , sparsity parameter  $s$ , individual degree parameter  $d$

**Result:** List  $\mathcal{L}$  of circuits over  $\mathbb{F}$  and the maximal monomial divisor  $M$  of  $f$

Let  $M$  be the maximal monomial divisor of  $f$ , and replace  $f$  by  $f/M$ .

$\mathcal{L}_{mon} = \text{Algorithm 1}(f, s, d)$

$\mathcal{L} = \emptyset$

// Eliminating division gates

Use Lemma 2.32 to eliminate division gates in circuits in  $\mathcal{L}_{mon}$

// Removing some spurious elements from the list

```

for  $g \in \mathcal{L}_{mon}$  do
  for  $i \in [n]$  do
    if  $X_i^{d+1} | g$  (Using Corollary 2.31) then
      Remove  $g$  from  $\mathcal{L}_{mon}$ 
      Break
    end
  end
end

```

// Removing monomial divisors from true factors

```

for  $g \in \mathcal{L}_{mon}$  do
  for  $i \in [n]$  do
    Find largest power  $d_i$  such that  $X_i^{d_i} | g$  (Using Corollary 2.31)
  end
   $\mathcal{L} = \mathcal{L} \cup \left\{ \frac{g}{X_1^{d_1} \cdots X_n^{d_n}} \right\}$ 
end

```

Use Lemma 2.32 to eliminate monomial division gates in circuits in  $\mathcal{L}$

**return**  $(\mathcal{L}, M)$

---

## 5 Recovering Bounded Individual Degree Factors of General Sparse Polynomials

In this section, we prove Theorem 1.7. In particular, we obtain a deterministic quasipolynomial-time algorithm that recovers all factors  $g$  that have bounded individual degree  $d$  (and are not divisible by a monomial) of an  $n$ -variate  $s$ -sparse polynomial  $f \in \mathbb{F}[\mathbf{X}_n]$  of (unbounded) individual degree  $D$ . The algorithm outputs a  $\binom{D}{\leq d}^{\log s}$ -size list  $\mathcal{L}$  such that each such factor  $g$  is in  $\mathcal{L}$ .  $\mathcal{L}$  might contain spurious polynomials that do not correspond to any factor of  $f$ . Unfortunately, we are unable to prune out the spurious elements since we do not know of efficient deterministic algorithms to check if a bounded individual degree polynomial  $g$  divides  $f$  (see Section 6). Indeed, if we could do this divisibility testing efficiently, then we would be able to clean the list and output only true factors. Note that the restriction to outputting factors not divisible by a monomial is unavoidable (see

Remark 1.4). Since we are given white-box access to the polynomial  $f$ , we compensate for this restriction by outputting the maximal degree monomial factor dividing  $f$ .

We restate our Theorem 1.7 (slightly more formally) below.

**Theorem 5.1** (Main Theorem 2). *Let  $f(\mathbf{X}_n) \in \mathbb{F}[\mathbf{X}_n]$  be an  $s$ -sparse polynomial with individual degree at most  $D$ . Then, there is an algorithm that runs in time  $\text{poly}\left(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n\right)$  and outputs a  $\binom{D}{\leq d}^{\log s}$ -size list of  $s^{4d^2 \log n}$ -sparse polynomials  $\mathcal{L}$ . For every factor  $g$  of  $f$ , where the individual degree of  $g$  is at most  $d$  and  $g$  is not divisible by any monomial,  $g$  lies in  $\mathcal{L}$ .*

**Remark 5.2.** In the above theorem, the displayed running time suppresses the dependence on the time taken to perform univariate polynomial factorization. More precisely, over a field  $\mathbb{F}$ , the algorithm runs in time  $\text{poly}\left(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n\right) \cdot \mathcal{T}(\mathbb{F}, D)$  (see Definition 2.11). In particular, over  $\mathbb{Q}$ , if the input polynomial has bit complexity  $B$ , then the algorithm runs in time  $\text{poly}\left(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n, B\right)$ . If  $\mathbb{F}$  is a finite field with characteristic  $p$  (greater than  $d$ ), the algorithm runs in time  $\text{poly}\left(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n, p, \log |\mathbb{F}|\right)$ .

At a high level, this algorithm also runs in two stages as in Section 4: the main recursive Algorithm 3 and the post-processing Algorithm 4. The main recursive Algorithm 3 uses a similar divide-and-conquer strategy to that in Algorithm 1. Algorithm 3 outputs a set  $\mathcal{L}_{\text{mon}}$  of *monomial equivalent* factors of  $f$ . More precisely, for every factor  $g$  (of individual degree at most  $d$ ) of  $f$ , there exists a polynomial  $h$  in  $\mathcal{L}_{\text{mon}}$  such that  $g \prec_d h$  (see Definition 2.1). Thus, for such a factor  $g$  with no monomial divisor,  $h = gM$  will be in  $\mathcal{L}_{\text{mon}}$ , where  $M$  is a monomial with individual degree at most  $d$ . Algorithm 4 strips out this monomial  $M$  from circuit  $C$ , thus obtaining the promised list  $\mathcal{L}$  containing the factors<sup>9</sup>.

We formally argue the correctness and the runtime of Algorithm 3 below. We refer the reader to Section 1.2.2 for a high-level overview of the key ideas in the algorithm.

**Theorem 5.3** (Recursive algorithm for factoring general sparse polynomials). *Let  $f(\mathbf{X}_n) \in \mathbb{F}[\mathbf{X}_n]$  be an  $s$ -sparse polynomial with individual degree at most  $D$ . Then, there is an algorithm that runs in time  $\text{poly}\left(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n\right)$  and outputs a  $\binom{D}{\leq d}^{\log s}$ -size list of  $s^{4d^2 \log n}$ -sparse polynomials  $\mathcal{L}_{\text{mon}}$ . For every factor  $g$  of  $f$ , where the individual degree of  $g$  is at most  $d$ , there exists a polynomial  $h$  in  $\mathcal{L}_{\text{mon}}$  such that  $g \prec_d h$ .*

*Proof.* We will show that Algorithm 3 outputs the required list of circuits. We prove the theorem by induction on the sparsity of  $f$ . Algorithm 3 initially strips off the maximal monomial divisor of  $f$ . Thus, from this point on, we work with an  $f$  that has no nonconstant monomial divisor.

**Base case** ( $\|f\| \leq 1$ ): Then  $f$  is a constant, since we have stripped off the maximal monomial divisor. Algorithm 3 outputs  $\{f \cdot (X_1 \cdot X_2 \cdots X_n)^d\}$ , and thus gives a  $d$ -monomial equivalent polynomial to  $f$ .

**Induction case** ( $\|f\| \leq s$ ): Without loss of generality,  $f$  depends on the variable  $X_n$  (Algorithm 3 ensures this by renaming variables). The algorithm instantiates a  $(n-1)$ -variate generator  $\mathcal{G} := \mathcal{G}_{(s', 2d, n-1)}^{KS}$  from Theorem 2.30, where  $s' = s^{4d^2 \log n}$  (as we will see later, the value of  $s'$  comes from

<sup>9</sup>On members of the list that do not correspond to true factors, we do not guarantee what might come out of this process, but it does not affect our run time or the guarantee on true factors.

Theorem 2.34). By Theorem 2.30, it takes  $\text{poly}(s^{4d^2 \log n}, n, d)$  time to construct the generator  $\mathcal{G}$ . Let  $k = \deg_{X_n}(f)$ , and

$$f = \sum_{i=0}^k f_i(\mathbf{X}_{n-1}) X_n^i$$

where  $f_i(\mathbf{X}_{n-1}) \in \mathbb{F}[\mathbf{X}_{n-1}]$ . Since we stripped off the maximal monomial divisor,  $f_0(\mathbf{X}_{n-1})$  and  $f_k(\mathbf{X}_{n-1})$  are both non-zero. By averaging, either  $\|f_0(\mathbf{X}_{n-1})\|$  or  $\|f_k(\mathbf{X}_{n-1})\|$  is at most  $s/2$ . We first deal with the case where  $\|f_k(\mathbf{X}_{n-1})\|$  is at most  $s/2$ .

**Sparsity of  $f_k \leq s/2$ :** Algorithm 3 normalizes  $f$  to get  $\hat{f}(\mathbf{X}_n) = f(\mathbf{X}_{n-1}, X_n/f_k) \cdot f_k^{k-1}$ , which is monic in  $X_n$ . Note that since  $\hat{f}$  is  $\mathcal{O}(s^D)$ -sparse, we do not have the budget to explicitly store its monomial representation. Hence, we represent it by a circuit  $C$ . Using  $\hat{f}$ , the algorithm computes the five-variate polynomial

$$\hat{f}_{\mathcal{G}}(\mathbf{Z}_4, X_n) = \hat{f}(\mathcal{G}(\mathbf{Z}_4), X_n)$$

in its monomial representation. We can compute the monomial representation (using interpolation) in time  $\text{poly}(n, s', d)$  since  $\deg(\hat{f}_{\mathcal{G}}(\mathbf{Z}_4, X_n)) = \text{poly}(n, s', d)$ . The algorithm invokes Lemma 2.14 to compute the list  $\mathcal{L}_{\hat{f}_{\mathcal{G}}}$  such that every nonconstant factor of  $\hat{f}_{\mathcal{G}}$  with  $X_n$ -degree at most  $d$  is in  $\mathcal{L}_{\hat{f}_{\mathcal{G}}}$ . Since  $\hat{f}_{\mathcal{G}}$  is monic in  $X_n$  and degree at most  $D$ , it follows that

$$|\mathcal{L}_{\hat{f}_{\mathcal{G}}}| \leq \binom{D}{\leq d} - 1 \quad (5.1)$$

Moreover, this computation takes  $\text{poly}(s', \binom{D}{\leq d}, n)$  time and each polynomial in  $\mathcal{L}_{\hat{f}_{\mathcal{G}}}$  is  $\text{poly}(s', n)$ -sparse.

Next it recursively computes the list  $\mathcal{L}_{f_k, \text{mon}}$ . For every factor  $g$  of  $f_k$ , where the individual degree of  $g$  is at most  $d$ , there exists a polynomial  $h$  in  $\mathcal{L}_{f_k, \text{mon}}$  such that  $g \prec_d h$ . By our induction hypothesis, since  $\|f_k\| \leq s/2$ , we have  $|\mathcal{L}_{f_k, \text{mon}}| \leq \binom{D}{\leq d}^{\log s-1}$  and we can compute this list in time  $\text{poly}((s/2)^{d^2 \log n}, \binom{D}{\leq d}^{\log s-1}, n)$ .

$$|\mathcal{L}_{f_k, \text{mon}}| \leq \binom{D}{\leq d}^{\log s-1} \quad (5.2)$$

Moreover, each polynomial in  $\mathcal{L}_{f_k, \text{mon}}$  is  $(s/2)^{4d^2 \log n}$ -sparse.

Next, the algorithm uses the lists  $\mathcal{L}_{\hat{f}_{\mathcal{G}}}$  and  $\mathcal{L}_{f_k, \text{mon}}$  to construct the list  $\mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$ . It defines  $\mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$  as follows.

$$\mathcal{L}_{f_{\mathcal{G}}, \text{mon}} = \left\{ \hat{g}(\mathbf{Z}_4, X_n \cdot (f_k \circ \mathcal{G}(\mathbf{Z}_4))) \cdot \frac{h \circ \mathcal{G}(\mathbf{Z}_4)}{(f_k \circ \mathcal{G}(\mathbf{Z}_4))^{\deg_{X_n}(\hat{g})}} \mid \hat{g} \in \mathcal{L}_{\hat{f}_{\mathcal{G}}}, h \in \mathcal{L}_{f_k, \text{mon}} \right\}.$$

The algorithm then uses  $\mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$  and  $\mathcal{L}_{f_k, \text{mon}}$  to construct the list  $\mathcal{L}_{\text{mon}}$ . For any  $g'(\mathbf{Z}_4, X_n) \in \mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$ , the algorithm applies the reconstruction algorithm from Theorem 2.30 to find a polynomial  $g(\mathbf{X}_n)$  of sparsity<sup>10</sup> at most  $s'$ , such that  $g(\mathcal{G}(\mathbf{Z}_4), X_n) = g'(\mathbf{Z}_4, X_n)$ <sup>11</sup>. We further include  $\mathcal{L}_{f_k, \text{mon}}$ , and thus  $\mathcal{L}_{\text{mon}}$  is defined as

$$\mathcal{L}_{\text{mon}} = \{ \text{Reconstruct } g(\mathbf{X}_n) \text{ from } g'(\mathbf{Z}_4, X_n) \mid g' \in \mathcal{L}_{f_{\mathcal{G}}, \text{mon}} \} \cup \mathcal{L}_{f_k, \text{mon}}.$$

<sup>10</sup>If the reconstruction algorithm outputs a polynomial of sparsity greater than  $s'$ , we simply remove it from  $\mathcal{L}_{\text{mon}}$ . Since the reconstruction algorithm runs in time  $\text{poly}(s', D, n)$ , the sparsity of the polynomial output by it is at most  $\text{poly}(s', D, n)$ , which makes it feasible to prune the list this way.

<sup>11</sup>If the reconstruction algorithm outputs a polynomial  $g$  such that  $g(\mathcal{G}(\mathbf{Z}_4), X_n) \neq g'(\mathbf{Z}_4, X_n)$ , then we remove it from  $\mathcal{L}_{\text{mon}}$ . Checking whether  $g(\mathcal{G}(\mathbf{Z}_4), X_n) \neq g'(\mathbf{Z}_4, X_n)$  takes time at most  $\text{poly}(s', D, n)$ , so this pruning is feasible.

Constructing  $\mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$  takes time polynomial in  $|\mathcal{L}_{\hat{f}_{\mathcal{G}}}| \cdot |\mathcal{L}_{f_k, \text{mon}}|$ ,  $s'$ ,  $D$ , and  $n$ . Since each reconstruction and pruning step takes  $\text{poly}(s', D, n)$  time, the list  $\mathcal{L}_{\text{mon}}$  is constructed in time  $\text{poly}\left(s', \binom{D}{\leq d}^{\log s}, n\right)$ .

We will argue next that for any factor  $g$  of  $f$  where the individual degree of  $g$  is at most  $d$ , there is a  $d$ -monomial equivalent polynomial  $h$  in  $\mathcal{L}_{\text{mon}}$ . Let  $g$  be such a factor. Suppose  $g$  does not depend on the variable  $X_n$ . Then  $g$  divides  $f_k$ . Hence, there exists a  $g'$  in  $\mathcal{L}_{f_k, \text{mon}} \subset \mathcal{L}_{\text{mon}}$  such that  $g \prec_d g'$ .

Now suppose  $g$  depends on  $X_n$ . Let  $g$  be of the following form,

$$g(\mathbf{X}_n) = \sum_{i=0}^t g_i(\mathbf{X}_{n-1}) X_n^i$$

where  $g_t \neq 0$ . Then by Lemma 2.23,  $\hat{g}|f$  where  $\hat{g}$  is defined by

$$\hat{g}(\mathbf{X}_n) = g(\mathbf{X}_{n-1}, X_n/f_k) \frac{f_k^t}{g_t}$$

In other words,

$$g(\mathbf{X}_n) = \hat{g}(\mathbf{X}_{n-1}, X_n f_k) \frac{g_t}{f_k^t}$$

Note that  $g_t|f_k$ , and  $g_t$  has individual degree at most  $d$ , which implies that there exists  $g'_t$  in  $\mathcal{L}_{f_k, \text{mon}}$  such that  $g_t \prec_d g'_t$ . We will show that the polynomial  $g'(\mathbf{X}_n)$  defined by

$$g'(\mathbf{X}_n) = \hat{g}(\mathbf{X}_{n-1}, X_n f_k) \cdot \frac{g'_t}{f_k^t}$$

lies in  $\mathcal{L}_{\text{mon}}$ . Since  $\hat{g}(\mathbf{X}_n) | f(\mathbf{X}_n)$ , we have  $\hat{g}(\mathcal{G}(\mathbf{Z}_4), X_n) | f(\mathcal{G}(\mathbf{Z}_4), X_n)$ . Furthermore,  $\deg_{X_n}(\hat{g}(\mathcal{G}(\mathbf{Z}_4), X_n))$  is at most  $d$ , and hence  $\hat{g}(\mathcal{G}(\mathbf{Z}_4), X_n) \in \mathcal{L}_{\hat{f}_{\mathcal{G}}}$ . By definition of  $\mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$ , since  $\hat{g}(\mathcal{G}(\mathbf{Z}_4), X_n) \in \mathcal{L}_{\hat{f}_{\mathcal{G}}}$  and  $g'_t \in \mathcal{L}_{f_k, \text{mon}}$ , we have  $g'(\mathcal{G}(\mathbf{Z}_4), X_n) \in \mathcal{L}_{f_{\mathcal{G}}, \text{mon}}$ . Let  $g'(\mathbf{X}_n)$  be of the form

$$g'(\mathbf{X}_n) = \sum_{i=0}^d g'_i(\mathbf{X}_{n-1}) X_n^i$$

then

$$g'(\mathcal{G}(\mathbf{Z}_4), X_n) = \sum_{i=0}^d g'_i(\mathcal{G}(\mathbf{Z}_4)) X_n^i$$

Since  $g$  has individual degree at most  $d$  and  $g | f$ , by Theorem 2.34, we know that  $g$  has sparsity at most  $s^{4d^2 \log n}$ . Note that the sparsity of  $g'$  is equal to the sparsity of  $g$ , which is  $s^{4d^2 \log n}$ , and its individual degree is at most  $2d$ . Thus, for all  $i$ ,  $g'_i(\mathbf{X}_{n-1})$  is  $s^{4d^2 \log n}$ -sparse and has individual degree at most  $2d$ . Since we have access to the monomial representation of  $g'(\mathcal{G}(\mathbf{Z}_4), X_n)$ , we also have access to the monomial representation of  $g'_i(\mathcal{G}(\mathbf{Z}_4))$ . Thus, by using Theorem 2.30, we reconstruct  $g'_i(\mathbf{X}_{n-1})$  from  $g'_i(\mathcal{G}(\mathbf{Z}_4))$  for  $0 \leq i \leq d$ , thereby reconstructing  $g'$ . This finishes the proof that  $\mathcal{L}_{\text{mon}}$  contains the claimed polynomials for this case.

Next, we prove the bound on the size of  $\mathcal{L}_{\text{mon}}$ , the runtime of the algorithm, and the sparsity of the polynomials in it.

Observe that

$$\begin{aligned}
|\mathcal{L}_{mon}| &\leq |\mathcal{L}_{f_{\mathcal{G}},mon}| + |\mathcal{L}_{f_k,mon}| && \text{(Since } \mathcal{L}_{mon} = \mathcal{L}_{f_{\mathcal{G}},mon} \cup \mathcal{L}_{f_k,mon}\text{)} \\
&\leq |\mathcal{L}_{\tilde{f}_{\mathcal{G}}}| \cdot |\mathcal{L}_{f_k,mon}| + |\mathcal{L}_{f_k,mon}| && \text{(Since } |\mathcal{L}_{f_{\mathcal{G}},mon}| \leq |\mathcal{L}_{\tilde{f}_{\mathcal{G}}}| \cdot |\mathcal{L}_{f_k,mon}|\text{)} \\
&\leq \left( \binom{D}{\leq d} - 1 \right) \cdot |\mathcal{L}_{f_k,mon}| + |\mathcal{L}_{f_k,mon}| && \text{(Using Eq. (5.1))} \\
&\leq \binom{D}{\leq d}^{\log s} && \text{(Using Eq. (5.2))}
\end{aligned}$$

which proves the required list size bound.

The total time taken is dominated by the time taken to compute the lists  $\mathcal{L}_{\tilde{f}_{\mathcal{G}}}$ ,  $\mathcal{L}_{f_k,mon}$ ,  $\mathcal{L}_{f_{\mathcal{G}},mon}$ , and  $\mathcal{L}_{mon}$ , which takes  $\text{poly}\left(\binom{D}{\leq d}^{\log s}, s^{d^2 \log n}, n\right)$  time.

Observe that every element in  $\mathcal{L}_{mon}$  is obtained from an application of  $s^{4d^2 \log n}$ -sparse polynomial reconstruction (by Theorem 2.30). Thus, every polynomial in  $\mathcal{L}_{mon}$ , including spurious elements not corresponding to factors, is  $s^{4d^2 \log n}$ -sparse.

We now address the other case where the sparsity of  $f_0$  is less than  $s/2$ . Many of the ingredients are common to the previous case.

**Sparsity of  $f_0 < s/2$ :** Algorithm 3 reverse-normalizes  $f$  to get  $\tilde{f}(\mathbf{X}_n) = f(\mathbf{X}_{n-1}, X_n \cdot f_0)/f_0$ , which is reversed-monic in  $X_n$ . Note that since  $\tilde{f}$  is  $\mathcal{O}(s^D)$ -sparse, we do not have the budget to explicitly store its monomial representation. Hence, we represent it by a circuit  $C$ . Using  $\tilde{f}$ , the algorithm computes the five-variate polynomial

$$\tilde{f}_{\mathcal{G}}(\mathbf{Z}_4, X_n) = \tilde{f}(\mathcal{G}(\mathbf{Z}_4), X_n)$$

The algorithm invokes Lemma 2.14 to compute a  $\binom{D}{\leq d} - 1$ -size list  $\mathcal{L}_{\tilde{f}_{\mathcal{G}}}$  such that every non-constant factor of  $\tilde{f}_{\mathcal{G}}$  with  $X_n$ -degree at most  $d$  is in  $\mathcal{L}_{\tilde{f}_{\mathcal{G}}}$ . Moreover, this computation takes  $\text{poly}(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n)$  time and each polynomial in  $\mathcal{L}_{\tilde{f}_{\mathcal{G}}}$  is  $\text{poly}(s^{d^2 \log n}, n)$ -sparse.

Next it recursively computes the list  $\mathcal{L}_{f_0,mon}$ . For every factor  $g$  of  $f_0$ , where the individual degree of  $g$  is at most  $d$ , there exists a polynomial  $h$  in  $\mathcal{L}_{f_0,mon}$  such that  $g \prec_d h$ . By our induction hypothesis, since  $\|f_0\| \leq s/2$ , we have  $|\mathcal{L}_{f_0,mon}| \leq \binom{D}{\leq d}^{\log s-1}$  and we can compute this list in time  $\text{poly}((s/2)^{d^2 \log n}, \binom{D}{\leq d}^{\log s-1}, n)$ . Moreover, each polynomial in  $\mathcal{L}_{f_0,mon}$  is  $(s/2)^{4d^2 \log n}$ -sparse.

Next, the algorithm uses the lists  $\mathcal{L}_{\tilde{f}_{\mathcal{G}}}$  and  $\mathcal{L}_{f_0,mon}$  to construct the list  $\mathcal{L}_{f_{\mathcal{G}},mon}$ . It defines  $\mathcal{L}_{f_{\mathcal{G}},mon}$  as follows.

$$\mathcal{L}_{f_{\mathcal{G}},mon} = \left\{ \tilde{g}(\mathbf{Z}_4, X_n / (f_0 \circ \mathcal{G}(\mathbf{Z}_4))) \cdot h \circ \mathcal{G}(\mathbf{Z}_4) \mid \tilde{g} \in \mathcal{L}_{\tilde{f}_{\mathcal{G}}}, h \in \mathcal{L}_{f_0,mon} \right\}.$$

The algorithm then uses  $\mathcal{L}_{f_{\mathcal{G}},mon}$  and  $\mathcal{L}_{f_0,mon}$  to construct the list  $\mathcal{L}_{mon}$ . The list  $\mathcal{L}_{mon}$  is defined as

$$\mathcal{L}_{mon} = \left\{ \text{Reconstruct } g(\mathbf{X}_{n-1}, X_n) \text{ from } g'(\mathbf{Z}_4, X_n) \mid g' \in \mathcal{L}_{f_{\mathcal{G}},mon} \right\} \cup \mathcal{L}_{f_0,mon}.$$

where the reconstruction is similar to the previous case.

We will argue next that for any factor  $g$  of  $f$  where the individual degree of  $g$  is at most  $d$ , there is a  $d$ -monomial equivalent polynomial  $h$  in  $\mathcal{L}_{mon}$ . Let  $g$  be such a factor. Suppose  $g$  does not depend on the variable  $X_n$ . Then  $g$  divides  $f_0$ . Hence, there exists a  $g'$  in  $\mathcal{L}_{f_0,mon} \subset \mathcal{L}_{mon}$  such that  $g \prec_d g'$ .

Now suppose  $g$  depends on  $X_n$ . Let  $g$  be of the following form,

$$g(\mathbf{X}_n) = \sum_{i=0}^t g_i(\mathbf{X}_{n-1}) X_n^i$$

where  $g_0 \neq 0$  (since we had stripped out the maximal monomial divisor initially). Then by Lemma 2.24,  $\tilde{g} | f$  where  $\tilde{g}$  is defined by

$$\tilde{g}(\mathbf{X}_n) = \frac{g(\mathbf{X}_{n-1}, X_n f_0)}{g_0}.$$

In other words,

$$g(\mathbf{X}_n) = \tilde{g}(\mathbf{X}_{n-1}, X_n / f_0) \cdot g_0.$$

Note that  $g_0 | f_0$ , which implies that there exists  $g'_0$  in  $\mathcal{L}_{f_0, mon}$  such that  $g_0 \prec_d g'_0$ . We will now show that the polynomial  $g'(\mathbf{X}_n)$  defined by

$$g'(\mathbf{X}_n) = \tilde{g}(\mathbf{X}_{n-1}, X_n / f_0) \cdot g'_0$$

lies in  $\mathcal{L}_{mon}$ . Since  $\tilde{g}(\mathbf{X}_n) | f(\mathbf{X}_n)$ , we have  $\tilde{g}(\mathcal{G}(\mathbf{Z}_4), X_n) | f(\mathcal{G}(\mathbf{Z}_4), X_n)$ . Furthermore,  $\deg_{X_n}(\tilde{g}(\mathcal{G}(\mathbf{Z}_4), X_n))$  is at most  $d$ , and hence  $\tilde{g}(\mathcal{G}(\mathbf{Z}_4), X_n) \in \mathcal{L}_{\tilde{f}_G}$ . By definition of  $\mathcal{L}_{f_G, mon}$ , since  $\tilde{g}(\mathcal{G}(\mathbf{Z}_4), X_n) \in \mathcal{L}_{\tilde{f}_G}$  and  $g'_0 \in \mathcal{L}_{f_0, mon}$ , we have  $g'(\mathcal{G}(\mathbf{Z}_4), X_n) \in \mathcal{L}_{f_G, mon}$ . By Theorem 2.34, we know that  $g$  has sparsity at most  $s^{4d^2 \log n}$ , and the sparsity of  $g'$  is equal to the sparsity of  $g$ . Since  $g'$  is  $s^{4d^2 \log n}$ -sparse and has individual degree at most  $2d$ , by Theorem 2.30,  $g' \in \mathcal{L}_{mon}$ . This finishes the proof that  $\mathcal{L}_{mon}$  contains the claimed polynomials for this case.

The arguments for the bounds on the size of  $\mathcal{L}_{mon}$ , the runtime of the algorithm, and the sparsity of the polynomials in  $\mathcal{L}_{mon}$  are the same as in the previous case. Thus, the size of  $\mathcal{L}_{mon}$  is at most  $\binom{D}{\leq d}^{\log s}$ , the total runtime is poly  $\left( \binom{D}{\leq d}^{\log s}, s^{d^2 \log n}, n \right)$ , and every polynomial in  $\mathcal{L}_{mon}$  is  $s^{4d^2 \log n}$ -sparse. ■

At this point, Algorithm 3 outputs a list  $\mathcal{L}_{mon}$  containing monomial equivalent factors of  $f$ . Algorithm 4 performs post-processing on this list to obtain a new list  $\mathcal{L}$  that contains all factors (not divisible by a monomial) of  $f$ . We show this formally by proving Theorem 5.1 below.

*Proof of Theorem 5.1.* We will show that Algorithm 4 outputs the required list of polynomials. Algorithm 4 first computes the list  $\mathcal{L}_{mon}$  of  $s^{4d^2 \log n}$ -sparse polynomials using Algorithm 3. For any  $h \in \mathcal{L}_{mon}$ , Algorithm 4 then removes the largest monomial factor of  $h$  and adds the result to  $\mathcal{L}$ . Note that after removing the largest monomial factor of  $h$ , the sparsity does not change. We prove the correctness and runtime of Algorithm 4 below.

Let  $g | f$ , where  $g$  has individual degree at most  $d$  and no monomial divides  $g$ . We will show that  $g \in \mathcal{L}$ . By Theorem 5.3, there exists  $h \in \mathcal{L}_{mon}$  such that  $g \prec_d h$ . By removing the largest monomial factor of  $h$ , we ensure  $g \in \mathcal{L}$ . Note that  $|\mathcal{L}| \leq |\mathcal{L}_{mon}| \leq \binom{D}{\leq d}^{\log s}$ , which proves the required bound on list size. Furthermore, the time complexity of the algorithm is dominated by the computation of  $\mathcal{L}_{mon}$ , which takes time poly  $(s^{d^2 \log n}, \binom{D}{\leq d}^{\log s}, n)$  by Theorem 5.3. ■

Along with an algorithm for factoring general sparse polynomials, Theorem 5.1 also gives a bound on the number of factors of individual degree at most  $d$  (not divisible by a monomial) of general sparse polynomials. In fact, the example from [CS26, Remark 8.2] shows that this bound is tight.

**Theorem 5.4** (Divisor bound for general sparse polynomials). *Let  $f$  be any  $n$ -variate  $s$ -sparse polynomial of individual degree  $D$ . Then  $f$  can have at most  $\binom{D}{\leq d}^{\log s}$  factors of individual degree at most  $d$  that are not divisible by any monomial. Moreover, this bound is tight.*

*Proof.* Theorem 5.1 gives a  $\binom{D}{\leq d}^{\log s}$ -size list  $\mathcal{L}$  that contains all factors of individual degree at most  $d$  (not divisible by a monomial) of  $f$ . This shows the required divisor bound.

To prove that the bound is tight, consider the polynomial  $f(\mathbf{X}) = \prod_{i=1}^{\log s} (X_i^D - 1)$  over  $\mathbb{C}$ . Observe that

$$f(\mathbf{X}) = \prod_{i=1}^{\log s} (X_i^D - 1) = \prod_{i=1}^{\log s} \prod_{j=0}^{D-1} (X_i - \omega^j),$$

where  $\omega \in \mathbb{C}$  is the  $D$ th primitive root of unity. Thus, the number of factors of  $f$  of individual degree at most  $d$  is exactly  $\binom{D}{\leq d}^{\log s}$ . ■

As an immediate consequence of our algorithm, we can find all constant-degree factors of an  $s$ -sparse polynomial in quasipolynomial time by pruning the list using the divisibility testing algorithm of [For15]. In doing so, we recover some results from [KRS24, DST24]. We state the result formally in the following corollary.

**Corollary 5.5.** *Let  $f \in \mathbb{F}[\mathbf{X}_n]$  be an  $s$ -sparse polynomial of individual degree at most  $D$ . Let  $\text{char}(\mathbb{F}) \geq \text{poly}(D)$ . We can find all factors  $g$  of  $f$  such that  $\deg(g) = \mathcal{O}(1)$  in time  $\text{poly}(s, n, D)^{\mathcal{O}(\log s)}$ .*

*Proof.* Using Theorem 5.1, we can find all factors  $g$  of  $f$  such that the individual degree of  $g$  is  $\mathcal{O}(1)$  and no monomial divides it. We can prune the list of factors to only contain polynomials of degree  $\mathcal{O}(1)$ . Every constant-degree factor of  $f$  can be written as a monomial times such a polynomial, and there are only  $\text{poly}(n, D)$  monomials of degree  $\mathcal{O}(1)$  to try. Then, by trying all these monomial multiples (which keeps degree  $\mathcal{O}(1)$ ) and using Theorem 2.36, we can find the list of all factors of  $f$  of constant degree in time  $\text{poly}(s, n, D)^{\mathcal{O}(\log s)}$ . ■

---

**Algorithm 3:** List of monomial equivalent bounded individual degree factors of general sparse polynomials (possibly with spurious entries)

---

**Data:**  $f \in \mathbb{F}[\mathbf{X}_n]$ , sparsity parameter  $s$ , individual degree parameter  $d$

**Result:**  $\mathcal{L}_{mon} \subset \mathbb{F}[\mathbf{X}_n]$ , list of circuits over  $\mathbb{F}$  with division gates

// Pre-processing

// Base Case

**if**  $f$  is a constant polynomial **then**

$\mathcal{L}_{mon} = \{f \cdot (X_1 \cdot X_2 \cdots X_n)^d\}$

**return**  $\mathcal{L}_{mon}$

**end**

Without loss of generality, assume  $f$  depends on the variable  $X_n$ ; otherwise, rename the variables appropriately.

Let  $s' = s^{4d^2 \log n}$

Instantiate  $(n-1)$ -variate generator  $\mathcal{G}(\mathbf{Z}_4) := \mathcal{G}_{(s', 2d, n-1)}^{KS}(Z_1, Z_2, Z_3, Z_4)$  from Theorem 2.30.

Let  $k = \deg_{X_n}(f)$ , and find  $f_0, \dots, f_k \in \mathbb{F}[\mathbf{X}_{n-1}]$  such that  $f = \sum_{i=0}^k f_i(\mathbf{X}_{n-1})X_n^i$

// Normalize  $f$

**if**  $\|f_k\| \leq s/2$  **then**

    Compute  $\hat{f} = f(\mathbf{X}_{n-1}, X_n/f_k) \cdot f_k^{k-1}$  as a circuit

    Compute  $\hat{f}_{\mathcal{G}} = \hat{f}(\mathcal{G}(\mathbf{Z}_4), X_n)$  as a sparse polynomial

    Compute  $\mathcal{L}_{\hat{f}_{\mathcal{G}}} = \{\text{All nonconstant factors of } \hat{f}_{\mathcal{G}} \text{ with } X_n\text{-degree at most } d\}$  using Lemma 2.14

    // Monomial equivalent individual degree  $d$  factors of  $f_k$

$\mathcal{L}_{f_k, mon} = \text{Algorithm 3}(f_k, s/2, d)$

$\mathcal{L}_{f_{\mathcal{G}}, mon} = \left\{ \hat{g}(\mathbf{Z}_4, X_n \cdot (f_k \circ \mathcal{G}(\mathbf{Z}_4))) \cdot \frac{h \circ \mathcal{G}(\mathbf{Z}_4)}{(f_k \circ \mathcal{G}(\mathbf{Z}_4))^{\deg_{X_n}(\hat{g})}} \mid \hat{g} \in \mathcal{L}_{\hat{f}_{\mathcal{G}}}, h \in \mathcal{L}_{f_k, mon} \right\}$

    // Use Theorem 2.30 to reconstruct sparse polynomials below

$\mathcal{L}_{mon} = \{\text{Reconstruct } g(\mathbf{X}_{n-1}, X_n) \text{ from } g'(\mathbf{Z}_4, X_n) \mid g' \in \mathcal{L}_{f_{\mathcal{G}}, mon}\} \cup \mathcal{L}_{f_k, mon}$

**end**

---

---

**Algorithm 3:** List of monomial equivalent bounded individual degree factors of general sparse polynomials (possibly with spurious entries) (Continued)

---

```

// Reverse-normalize  $f$ 
else
  Compute  $\tilde{f} = f(\mathbf{X}_{n-1}, X_n f_0) / f_0$  as a circuit
  Compute  $\tilde{f}_G = \tilde{f}(\mathcal{G}(\mathbf{Z}_4), X_n)$  as a sparse polynomial
  Compute  $\mathcal{L}_{\tilde{f}_G} = \{\text{All nonconstant factors of } \tilde{f}_G \text{ with } X_n\text{-degree at most } d\}$  using
    Lemma 2.14

  // Monomial equivalent individual degree  $d$  factors of  $f_0$ 
   $\mathcal{L}_{f_0, \text{mon}} = \text{Algorithm 3}(f_0, s/2, d)$ 
   $\mathcal{L}_{f_G, \text{mon}} = \left\{ \tilde{g}(\mathbf{Z}_4, X_n / (f_0 \circ \mathcal{G}(\mathbf{Z}_4))) \cdot h \circ \mathcal{G}(\mathbf{Z}_4) \mid \tilde{g} \in \mathcal{L}_{\tilde{f}_G}, h \in \mathcal{L}_{f_0, \text{mon}} \right\}$ 

  // Use Theorem 2.30 to reconstruct sparse polynomials below
   $\mathcal{L}_{\text{mon}} = \{\text{Reconstruct } g(\mathbf{X}_{n-1}, X_n) \text{ from } g'(\mathbf{Z}_4, X_n) \mid g' \in \mathcal{L}_{f_G, \text{mon}}\} \cup \mathcal{L}_{f_0, \text{mon}}$ 
end

return  $\mathcal{L}_{\text{mon}}$ 

```

---

**Algorithm 4:** List of non-monomial bounded individual degree factors of general sparse polynomials (possibly with spurious entries)

---

**Data:**  $f \in \mathbb{F}[\mathbf{X}_n]$ , sparsity parameter  $s$ , individual degree parameter  $d$

**Result:**  $\mathcal{L} \subset \mathbb{F}[\mathbf{X}_n]$ , list of circuits over  $\mathbb{F}$

$\mathcal{L}_{\text{mon}} = \text{Algorithm 3}(f, s, d)$

$\mathcal{L} = \emptyset$

// Removing monomial factors from true factors

for  $g \in \mathcal{L}_{\text{mon}}$  do

  Find the largest monomial factor  $M$  that divides  $g$

$\mathcal{L} = \mathcal{L} \cup \{g/M\}$

end

return  $\mathcal{L}$

---

## 6 Future Directions and Open Questions

Some interesting and natural open problems are the following.

1. The most interesting open question is to prune our output lists and return only the true factors. This question arises for both our factorization algorithms. Consider the following problem: given an  $s$ -sparse polynomial  $f$ , find all its multilinear factors. In the setting of Theorem 1.7, we show that we can output a list of quasipolynomially sparse multilinear polynomials containing all such factors. To prune the list, we would need a divisibility test for checking if a given multilinear quasipolynomially sparse polynomial  $g$  divides another sparse polynomial  $f$ . It can be shown that this can be reduced to PIT for constant-depth circuits, and hence has subexponential-time PIT algorithms. However, it would be interesting to get something more efficient in this case, for instance, a quasipolynomial-time algorithm.

2. As of now, Theorem 1.1 only works over fields of characteristic 0 or characteristic greater than  $d$ . It would be nice to have the result over all fields.

**Acknowledgements:** Somnath is grateful to Pranjal Dutta (NTU), Saswata Mukherjee (NUS) and Amit Sinhababu (CMI) for the helpful discussions on polynomial factorization. Shanthanu is grateful to Mrinal Kumar (TIFR) and Pratik Shastri (IMSc) for insightful discussions about sparse polynomials and, in particular, sparse polynomial factorization algorithms.

## References

- [AW24] Robert Andrews and Avi Wigderson. Constant-depth arithmetic circuits for linear algebra problems. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2367–2386. IEEE, 2024.
- [BDS25] C. S. Bhargav, Prateek Dwivedi, and Nitin Saxena. A primer on the closure of algebraic complexity classes under factoring. *arXiv preprint arXiv:2506.19604*, 2025.
- [Ber70] E.R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713 – 735, 1970. Cited by: 381; All Open Access, Bronze Open Access.
- [BKR<sup>+</sup>25a] Somnath Bhattacharjee, Mrinal Kumar, Shanthanu S. Rai, Varun Ramanathan, Ramprasad Satharishi, and Shubhangi Saraf. Constant-depth circuits for polynomial GCD over any characteristic. *CoRR*, abs/2506.23220, 2025.
- [BKR<sup>+</sup>25b] Somnath Bhattacharjee, Mrinal Kumar, Varun Ramanathan, Ramprasad Satharishi, and Shubhangi Saraf. Deterministic factorization of constant-depth algebraic circuits in subexponential time. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 684–709, 2025.
- [BKR<sup>+</sup>26] Somnath Bhattacharjee, Mrinal Kumar, Shanthanu S. Rai, Varun Ramanathan, Ramprasad Satharishi, and Shubhangi Saraf. Closure under factorization from a result of furstenberg. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, STOC '26*, page 174–185, New York, NY, USA, 2026. Association for Computing Machinery.
- [BOT88] Michael Ben-Or and Prasoona 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.
- [BSV20] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *J. ACM*, 67(2), May 2020.
- [BV25] Pranav Bisht and Ilya Volkovich. On solving sparse polynomial factorization related problems. *computational complexity*, 34(1):7, 2025.
- [CDGK91] 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.

- [CS26] Aminadav Chuyoon and Amir Shpilka. On factorization of sparse polynomials of bounded individual degree. *arXiv preprint arXiv:2603.07589*, 2026.
- [DST24] 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)*, volume 317 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [Eis95] David Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995.
- [For15] Michael A. Forbes. Deterministic divisibility testing via shifted partial derivatives. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 451–465, 2015.
- [Fur67] Harry Furstenberg. Algebraic functions over finite fields. *Journal of Algebra*, 7(2):271–277, 1967.
- [GK87] Dima Yu Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in nc. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 166–172. IEEE, 1987.
- [GKS90] 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.
- [Kal89] Erich Kaltofen. Factorization of Polynomials Given by Straight-Line Programs. In *Randomness and Computation*, pages 375–412. JAI Press, 1989.
- [KRS24] Mrinal Kumar, Varun Ramanathan, and Ramprasad Saptharishi. *Deterministic Algorithms for Low Degree Factors of Constant Depth Circuits*, pages 3901–3918. SIAM, 2024.
- [KRSV26] Mrinal Kumar, Varun Ramanathan, Ramprasad Saptharishi, and Ben Lee Volk. Towards deterministic algorithms for constant-depth factors of constant-depth circuits. *ACM Trans. Comput. Theory*, 18(1), February 2026.
- [KS01] Adam R. Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, page 216–223, New York, NY, USA, 2001. Association for Computing Machinery.
- [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. Computational algebraic complexity editorial.

- [LLL82] A. K. Lenstra, H. W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
- [Rui93] Jesús M. Ruiz. *The Basic Theory of Power Series*. Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden, 1993.
- [Sho90] V. Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Inf. Process. Lett.*, 33(5):261–267, January 1990.
- [Str73] Volker Strassen. Vermeidung von Divisionen. *J. Reine Angew. Math.*, 264:184–202, 1973.
- [SV10] Amir Shpilka and Ilya Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming*, pages 408–419, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [Vol15] Ilya Volkovich. Deterministically Factoring Sparse Polynomials into Multilinear Factors and Sums of Univariate Polynomials. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 943–958, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [Vol17] Ilya Volkovich. On Some Computations on Sparse Polynomials. In Klaus Jansen, José D. P. Rolim, David P. Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*, volume 81 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013.
- [vzGK85] Joachim von zur Gathen and Erich Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, 1985.

## A Omitted Proofs from Section 2.4

In this section, we give the omitted proofs of some lemmas from Section 2.4.

*Proof of Lemma 2.17.* Let  $\deg(f) = d_1$  and  $\deg(g) = d_2$ . Define  $h := \gcd(f, g)$  and  $D = \deg(h)$ . By the definition of the Sylvester matrix,

$$\ker(\text{Syl}(f, g)) \cong \{(w_1, w_2) \in \mathbb{F}[X] \times \mathbb{F}[X] \mid \deg(w_1) < d_2, \deg(w_2) < d_1, \text{ and } w_1f + w_2g = 0\}.$$

Let  $\hat{f} = f/h$  and  $\hat{g} = g/h$ . Then for any  $(w_1, w_2) \in \ker(\text{Syl}(f, g))$ ,

$$w_1\hat{f} + w_2\hat{g} = 0.$$

Since  $\gcd(\hat{f}, \hat{g}) = 1$ , it follows that  $\hat{g} \mid w_1$  and  $\hat{f} \mid w_2$ . Thus, there is a polynomial  $e(X) \in \mathbb{F}[X]$  such that

$$w_1 = \hat{g} \cdot e \quad w_2 = -\hat{f} \cdot e$$

The degree constraints imply  $\deg(e) < D$ , because  $\deg(\hat{g}) = d_2 - D$  and  $\deg(\hat{f}) = d_1 - D$ .

Conversely, every polynomial  $e \in \mathbb{F}[X]$  with  $\deg(e) < D$  gives a kernel element  $(\hat{g}e, -\hat{f}e)$ , since  $\deg(\hat{g}e) < d_2$ ,  $\deg(\hat{f}e) < d_1$ , and  $(\hat{g}e)f - (\hat{f}e)g = 0$ . Hence

$$\ker(\text{Syl}(f, g)) \cong \{(\hat{g}e, -\hat{f}e) \in \mathbb{F}[X] \times \mathbb{F}[X] \mid e \in \mathbb{F}[X] \text{ and } \deg(e) < D\}.$$

Therefore  $\ker(\text{Syl}(f, g))$  is isomorphic to the vector space of polynomials of degree less than  $D$ , and hence

$$\dim \ker(\text{Syl}(f, g)) = D = \deg(\gcd(f, g)).$$

■

*Proof of Corollary 2.18.* We have

$$\prod_i h_i^{e_i-1} \mid f \quad \text{and} \quad \prod_i h_i^{e_i-1} \mid f',$$

and hence  $\prod_i h_i^{e_i-1} \mid \gcd(f, f')$ . Therefore

$$\deg(\gcd(f, f')) \geq \sum_i (e_i - 1) \deg(h_i).$$

By Lemma 2.17,

$$\dim \ker(\text{Syl}(f, f')) = \deg(\gcd(f, f')) \geq \sum_i (e_i - 1) \deg(h_i).$$

Now suppose the  $h_i$  are distinct pairwise coprime irreducible polynomials. Since  $\text{char}(\mathbb{F}) = 0$  or  $\text{char}(\mathbb{F}) > d$ , each  $h_i$  is coprime to its derivative. Thus the only common factors of  $f$  and  $f'$  come from repeated powers of the  $h_i$ , and

$$\gcd(f, f') = \prod_i h_i^{e_i-1}.$$

Applying Lemma 2.17 again gives equality.

■

## B Omitted Proofs from Section 2.6

Here, we add the proof of the version of Hensel's lemma (Lemma 2.26) we use. We also provide the proof of Theorem 2.28.

### B.1 Proof of Hensel's Lemma

*Proof of Lemma 2.26.* Note that it is enough to prove the above theorem for each  $h_i(\mathbf{X}, Y)$ , as the polynomials  $h_i(0, Y)$  are pairwise coprime since  $f(\mathbf{X}, Y)$  is Hensel-ready at  $\mathbf{X} = 0$ . Furthermore, each  $h_i(0, Y)$  is square-free. We first prove this for  $n = 1$ , i.e.,  $\mathbf{X} = \{X\}$ .

Let  $\deg_Y(h_i) = d$  and  $\mathcal{B} = \{\beta_1, \dots, \beta_d\}$  be the set of  $d$  distinct roots of  $h_i(0, Y)$  in  $\overline{\mathbb{F}}$ . Pick any  $\beta \in \mathcal{B}$ . We apply Newton iteration (aka Newton-Raphson Method) starting at  $\beta$ . Suppose, for some  $r \geq 0$ , we have  $a_0 = \beta, a_1, \dots, a_r \in \mathbb{F}(\beta)$  such that for  $a^{(r)}(X) := a_0 + a_1X + \dots + a_rX^r$ ,  $h_i(X, a^{(r)}(X)) = 0 \pmod{X^{r+1}}$ . Then we will recursively compute  $a_{r+1}$ .

**Claim B.1.** There is a unique  $a_{r+1} \in \mathbb{F}(\beta)$  such that, for  $a^{(r+1)}(X) := a^{(r)}(X) + a_{r+1}X^{r+1}$ ,  $h_i(X, a^{(r+1)}(X)) = 0 \pmod{X^{r+2}}$ .

*Proof.* Since  $h_i(0, Y)$  is square-free,  $\partial_Y h_i(0, \beta) \neq 0$ . Now we have

$$\begin{aligned} h_i(X, a^{(r+1)}(X)) &= 0 \pmod{X^{r+2}} \\ \iff h_i(X, a^{(r)}(X) + a_{r+1}X^{r+1}) &= 0 \pmod{X^{r+2}} \\ \iff h_i(X, a^{(r)}(X)) + a_{r+1}X^{r+1}\partial_Y h_i(X, a^{(r)}(X)) &= 0 \pmod{X^{r+2}} \\ &\quad \text{(Taylor expansion around } (X, a^{(r)}(X))) \end{aligned}$$

Note  $h_i(X, a^{(r)}(X)) = 0 \pmod{X^{r+1}}$ , hence  $h_i(X, a^{(r)}(X)) = cX^{r+1} \pmod{X^{r+2}}$  for some  $c \in \mathbb{F}(\beta)$ . Also note that, modulo  $X^{r+2}$ ,

$$a_{r+1}X^{r+1}\partial_Y h_i(X, a^{(r)}(X)) = a_{r+1}\partial_Y h_i(0, \beta)X^{r+1}.$$

Hence we can conclude  $h_i(X, a^{(r+1)}(X)) = 0 \pmod{X^{r+2}} \iff a_{r+1} = -\frac{c}{\partial_Y h_i(0, \beta)}$ . ■

Hence we can set  $\Phi_\beta(X) = \sum_r a_r X^r$  and we get the unique power series root with constant term  $\beta$ . Since  $\overline{\mathbb{F}}[[X]][Y]$  is a UFD, we have the unique factorization  $h_i(X, Y) = \prod_\beta (Y - \Phi_\beta(X))$ . We also remark that there is another nice proof using the fixed-point theorem in [Rui93].

For the case when  $n \geq 2$ , introduce a new variable  $T$  and consider

$$H_i(T, Y) := h_i(TX_1, \dots, TX_n, Y) \in \mathbb{F}[\mathbf{X}][T, Y].$$

Applying the one-variable argument to  $H_i$  as a polynomial in  $T$  gives a unique root

$$\Psi_\beta(T) = \sum_{r \geq 0} a_r(\mathbf{X})T^r \in \mathbb{F}(\beta)[\mathbf{X}][[T]]$$

with  $\Psi_\beta(0) = \beta$  and  $H_i(T, \Psi_\beta(T)) = 0$ . Since the coefficient of  $T^r$  in  $H_i$  is homogeneous of  $\mathbf{X}$ -degree  $r$ , uniqueness implies that each  $a_r(\mathbf{X})$  is homogeneous of degree  $r$ . Setting  $T = 1$  gives

$$\Phi_\beta(\mathbf{X}) := \Psi_\beta(1) = \sum_{r \geq 0} a_r(\mathbf{X}) \in \mathbb{F}(\beta)[[\mathbf{X}]],$$

and this is the unique power series root of  $h_i(\mathbf{X}, Y)$  with constant term  $\beta$ . ■

## B.2 Proof of Theorem 2.28

Let  $F(T, Y)$  be a monic polynomial in  $Y$  and Hensel-ready at  $T = 0$ . For a root  $\beta$  of  $F(0, Y)$  with multiplicity  $e$ , define

$$R_e(Z) := Z + \sum_{m \geq 0}^{2e(D+1)} [Y^{em+e-2}] \left\{ \left( \frac{e!}{\partial_{Y^e} F(0, Z)} \right)^{m+1} \frac{\partial_Y F(T, Y+Z)}{e} \left( \frac{\partial_{Y^e} F(0, Z)}{e!} Y^e - F(T, Y+Z) \right)^m \right\}$$

Here  $[Y^r]\{P(Y)\}$  denotes the coefficient of  $Y^r$  in  $P(Y)$ .

For a bivariate power series  $\varphi(X, Y)$ , define the diagonal operator

$$\mathcal{D}(\varphi)(Z) := \sum_{r \geq 0} ([X^r Y^r]\{\varphi(X, Y)\})Z^r.$$

We will use the following consequence of Furstenberg's theorem<sup>12</sup>.

<sup>12</sup>[Fur67] gave the exact form when  $e = 1$

**Lemma B.2** (Theorem 3.1 in [BKR<sup>+</sup>26]). *Assume that 0 is a root of  $F(0, Y)$  of multiplicity  $e$ , and let  $\varphi(T)$  be the unique power series root of  $F(T, Y)$  corresponding to 0. Then*

$$\varphi(T) = \mathcal{D}\left(\frac{Y^2 \cdot \partial_Y F(TY, Y)}{e \cdot F(TY, Y)}\right).$$

Before we prove Theorem 2.28, we first show how to express the above diagonal expression as a power series and do some initial calculations which will be helpful in the proof.

Now, since 0 is a root of  $F(0, Y)$  with multiplicity  $e$ , we have

$$c := \frac{\partial_{Y^e} F(0, 0)}{e!} \neq 0.$$

Moreover,  $1 - F(TY, Y)/(cY^e)$  has zero constant coefficient as a Laurent series in  $Y$  whose coefficients are power series in  $T$ , so the following geometric expansion is valid coefficient-wise:

$$F(TY, Y) = cY^e \left[ 1 - \left( 1 - \frac{F(TY, Y)}{cY^e} \right) \right]$$

we use  $\frac{1}{1-z} = \sum_{m \geq 0} z^m$  with  $z = 1 - F(TY, Y)/(cY^e)$ . This gives

$$\begin{aligned} \varphi(T) &= \mathcal{D}\left[\frac{Y^2 \partial_Y F(TY, Y)}{e \cdot F(TY, Y)}\right] \\ &= \mathcal{D}\left[\frac{Y \partial_Y F(TY, Y) / Y^{e-1}}{e \cdot c \left[1 - (1 - F(TY, Y)/(c \cdot Y^e))\right]}\right] \\ &= \mathcal{D}\left[\frac{Y \partial_Y F(TY, Y)}{Y^{e-1} \cdot e \cdot c} \sum_{m \geq 0} \left(1 - \frac{F(TY, Y)}{c \cdot Y^e}\right)^m\right] \\ &= \mathcal{D}\left[\sum_{m \geq 0} \left(1 - \frac{F(TY, Y)}{c \cdot Y^e}\right)^m \frac{Y \partial_Y F(TY, Y)}{Y^{e-1} \cdot e \cdot c}\right]. \end{aligned}$$

For  $m \geq 0$ , expand

$$\left(1 - \frac{F(TY, Y)}{c \cdot Y^e}\right)^m = \frac{(c \cdot Y^e - F(TY, Y))^m}{c^m \cdot Y^{em}},$$

so the  $m$ -th summand becomes

$$\frac{(c \cdot Y^e - F(TY, Y))^m \partial_Y F(TY, Y)}{e \cdot c^{m+1} \cdot Y^{em+e-2}}.$$

Therefore,

$$\begin{aligned} [T^n](\varphi) &= [T^n Y^n] \left\{ \sum_{m \geq 0} \left(1 - \frac{F(TY, Y)}{c \cdot Y^e}\right)^m \frac{Y \partial_Y F(TY, Y)}{Y^{e-1} \cdot e \cdot c} \right\} \\ &= \frac{1}{e} \sum_{m \geq 0} \frac{1}{c^{m+1}} [T^n Y^{n+em+e-2}] \left\{ (c \cdot Y^e - F(TY, Y))^m \partial_Y F(TY, Y) \right\}. \end{aligned}$$

Finally, for any Laurent series  $L(T, Y)$ , we have  $[T^i Y^j]\{L(T, Y)\} = [T^i Y^{i+j}]\{L(TY, Y)\}$ . Hence, we have

$$\begin{aligned}\varphi(T) &= \frac{1}{e} \sum_{n \geq 0} \sum_{m \geq 0} \frac{1}{c^{m+1}} [T^n Y^{em+e-2}] \left\{ (c \cdot Y^e - F(T, Y))^m \partial_Y F(T, Y) \right\} \cdot T^n \\ &= \frac{1}{e} \sum_{m \geq 0} \frac{1}{c^{m+1}} [Y^{em+e-2}] \left\{ (c \cdot Y^e - F(T, Y))^m \partial_Y F(T, Y) \right\}\end{aligned}$$

Finally setting  $c = \frac{\partial_{Y^e} F(0, 0)}{e!}$  we get

$$\varphi(T) = \sum_{m \geq 0} [Y^{em+e-2}] \left\{ \left( \frac{e!}{\partial_{Y^e} F(0, 0)} \right)^{m+1} \frac{\partial_Y F(T, Y)}{e} \cdot \left( \frac{\partial_{Y^e} F(0, 0)}{e!} Y^e - F(T, Y) \right)^m \right\}$$

We now justify the finite cutoff used in the definition of  $R_e$ . Let  $Q(T, Y) := cY^e - F(T, Y)$ . Since 0 is a root of  $F(0, Y)$  of multiplicity  $e$ , every  $T^0$ -term of  $Q$  has  $Y$ -degree at least  $e + 1$ , while every remaining term of  $Q$  has positive  $T$ -degree. Consider a monomial contributing to

$$[Y^{em+e-2}]\{Q(T, Y)^m \partial_Y F(T, Y)\}.$$

If  $s$  of the  $m$  factors of  $Q$  contribute positive  $T$ -degree terms, then the other  $m - s$  factors contribute  $Y$ -degree at least  $e + 1$ . Hence, for the total  $Y$ -degree to be at most  $em + e - 2$ , we must have

$$(m - s)(e + 1) \leq em + e - 2,$$

and therefore  $s \geq (m - e + 2)/(e + 1)$ . Thus, if the corresponding term has  $T$ -degree at most  $D$ , then certainly  $s \leq D$  since each of those  $s$  factors already contributes at least one to the  $T$ -degree. This implies

$$m \leq (e + 1)D + e - 2 \leq 2e(D + 1).$$

Consequently, terms with  $m > 2e(D + 1)$  do not contribute to the truncation modulo  $T^{D+1}$ .

Note for the validity of the entire calculation above we need  $e! \neq 0$ , for this purpose we will use the fact that  $\text{char}(\mathbb{F}) = 0$  or greater than  $d$  and clearly  $d \geq e$ . Now we are ready to prove Theorem 2.28

*Proof of Theorem 2.28.* Item (1) follows from the definition of  $R_e(Z)$  and the fact that truncation and coefficient extraction can be carried out in constant depth (which follows from Lemma 2.8 and Lemma 2.6). Moreover, the denominator  $B(Z)$  can be chosen to be an appropriate power of  $\partial_{Y^e} F(0, Z)$ , and hence  $B(\beta) \neq 0$  because  $\beta$  has multiplicity exactly  $e$  in  $F(0, Y)$ . This proves item (2). We now prove item (3).

Let  $\hat{F}(T, Y) = F(T, Y + \beta)$ . Then  $\hat{F}(0, Y)$  has a root 0 with multiplicity  $e$ , and the unique power series root corresponding to 0 is  $\hat{\varphi}(T) := \Phi_\beta(T) - \beta$ .

Now from the above calculation we have

$$\begin{aligned}
& \Phi_\beta(T) \text{ Trunc } T^{D+1} \\
&= \hat{\varphi}(T) \text{ Trunc } T^{D+1} + \beta \\
&= \sum_{m \geq 0} [Y^{em+e-2}] \left\{ \left( \frac{e!}{\partial_{Y^e} \hat{F}(0,0)} \right)^{m+1} \frac{\partial_Y \hat{F}(T, Y)}{e} \left( \frac{\partial_{Y^e} \hat{F}(0,0)}{e!} Y^e - \hat{F}(T, Y) \right)^m \right\} \text{ Trunc } T^{D+1} + \beta \\
&= \beta + \sum_{m \geq 0} [Y^{em+e-2}] \left\{ \left( \frac{e!}{\partial_{Y^e} F(0, \beta)} \right)^{m+1} \frac{\partial_Y F(T, Y + \beta)}{e} \left( \frac{\partial_{Y^e} F(0, \beta)}{e!} Y^e - F(T, Y + \beta) \right)^m \right\} \text{ Trunc } T^{D+1} \\
&= \beta + \sum_{m \geq 0}^{2e(D+1)} [Y^{em+e-2}] \left\{ \left( \frac{e!}{\partial_{Y^e} F(0, \beta)} \right)^{m+1} \frac{\partial_Y F(T, Y + \beta)}{e} \left( \frac{\partial_{Y^e} F(0, \beta)}{e!} Y^e - F(T, Y + \beta) \right)^m \right\} \text{ Trunc } T^{D+1} \\
&= R_e(\beta) \text{ Trunc } T^{D+1}
\end{aligned}$$

■

## C Omitted Proofs from Section 2.7

We give the omitted proofs from Section 2.7 below.

*Proof of Corollary 2.31.* It will be convenient to introduce the following notation. Let  $P(n, s, d)$  denote the class of all polynomials  $f \in \mathbb{F}[\mathbf{X}_n]$  such that  $f$  divides a non-zero  $s$ -sparse polynomial in  $\mathbb{F}[\mathbf{X}_n]$  of individual degree at most  $d$ .

Since  $f \in P(n, s, d)$  and  $f \prec_d g$ , the individual degree of  $g$  is at most  $2d$ . Write  $g(\mathbf{X}_n) = \sum_{j=0}^{2d} g_j(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n) X_i^j$ . By Lemma 2.6, we can construct, in time  $\text{poly}(m, d)$ , circuits  $C_0, \dots, C_{2d}$  computing  $g_0, \dots, g_{2d}$  respectively.

Let  $e$  be the largest power of  $X_i$  dividing  $g$ . Then  $g_0 = g_1 = \dots = g_{e-1} = 0$  and  $g_e \neq 0$ . Moreover, since  $f \prec_d g$  and  $f$  is not divisible by a monomial,  $g_e$  is a monomial multiple of  $f|_{X_i=0}$ . We claim that  $g_e \in P(n-1, s, 2d)$ . Indeed, since  $f \in P(n, s, d)$ , there is a non-zero  $s$ -sparse polynomial  $F \in \mathbb{F}[\mathbf{X}_n]$  of individual degree at most  $d$  such that  $f \mid F$ . Write  $F = fH$ , and let  $H_r$  be the first non-zero coefficient of  $H$  when viewed as a polynomial in  $X_i$ . Since  $f$  is not divisible by  $X_i$ ,  $f|_{X_i=0}$  is non-zero, and the coefficient of  $X_i^r$  in  $F$  is  $f|_{X_i=0} \cdot H_r$ . This coefficient is a non-zero  $(n-1)$ -variate  $s$ -sparse polynomial of individual degree at most  $d$  divisible by  $f|_{X_i=0}$ . Since  $g_e$  differs from  $f|_{X_i=0}$  by a monomial of individual degree at most  $d$ , it divides a non-zero  $s$ -sparse polynomial of individual degree at most  $2d$ .

We now test the coefficient polynomials in order. Let  $S_{n-1, s, 2d}$  be the Klivans–Spielman hitting set from Theorem 2.29. For each  $j = 0, \dots, 2d$ , we evaluate  $C_j$  on all points of  $S_{n-1, s, 2d}$ . If all these evaluations are zero, we continue to the next value of  $j$ ; otherwise, we output  $j$ .

This procedure outputs  $e$ . Indeed, for every  $j < e$ , we have  $g_j = 0$ , so all evaluations of  $g_j$  vanish. On the other hand,  $g_e$  is a non-zero polynomial in  $P(n-1, s, 2d)$ , and therefore Theorem 2.29 implies that  $g_e$  is non-zero on some point of  $S_{n-1, s, 2d}$ . Thus  $e$  is the first index for which the test finds a non-zero evaluation.

The running time is  $\text{poly}(n, s, d, m)$ : there are at most  $2d+1$  coefficient polynomials to test, the circuits for them are obtained in time  $\text{poly}(m, d)$ , and each hitting-set construction and evaluation uses only  $\text{poly}(n, s, d, m)$  time by Theorem 2.29. Hence the largest power of  $X_i$  dividing  $g$  can be computed deterministically in  $\text{poly}(n, s, d, D, m)$  time. ■

*Proof of Lemma 2.32.* We use Strassen’s division elimination [Str73], together with the Klivans–Spielman hitting set from Theorem 2.29, to first reduce to the case where the denominator has a non-zero constant term.

By the standard first step in Strassen’s argument, we may move all divisions to the top. Thus, in deterministic  $\text{poly}(m, d)$  time, we obtain division-free circuits  $C_1$  and  $C_2$  of size  $\text{poly}(m, d)$  and depth at most  $\Delta + \mathcal{O}(1)$  such that  $C$  computes  $C_1/C_2$ . Moreover,  $C_2$  is a product of the denominator polynomials appearing in the division gates of  $C$ . In particular, by assumption,  $C_2$  is a product of  $s$ -sparse polynomials of individual degree at most  $d$ .

We now find a shift  $\alpha \in \mathbb{F}^n$  such that  $C_2(\alpha) \neq 0$ . If  $C_2 = Q_1 \cdots Q_t$  is the product decomposition of  $C_2$  into sparse denominator factors, then  $t \leq m$ , and  $C_2$  belongs to the class of products of at most  $m$  many  $s$ -sparse polynomials of individual degree at most  $d$ . By Theorem 2.29, the hitting set  $S_{n,s,d,m}$  contains a point  $\alpha$  such that  $C_2(\alpha) \neq 0$ .

Consider the shifted quotient  $C'(\mathbf{X}_n) = C(\mathbf{X}_n + \alpha) = C_1(\mathbf{X}_n + \alpha)/C_2(\mathbf{X}_n + \alpha)$ . Its denominator has non-zero constant term, since  $C_2(\alpha) \neq 0$ . Therefore, after normalizing the denominator by its constant term, we may write it as  $1 + h(\mathbf{X}_n)$ , where  $h$  has zero constant term. Since  $C'$  computes the polynomial  $f(\mathbf{X}_n + \alpha)$ , whose total degree is at most  $nd$ , Strassen’s truncation argument applies: the inverse of  $1 + h$  is needed only up to total degree  $nd$ , and hence can be replaced by the truncated geometric series  $1 - h + h^2 - \cdots + (-h)^{nd}$ . Using Lemma 2.8, this truncated product, namely  $C_1(\mathbf{X}_n + \alpha) \cdot (1 - h + h^2 - \cdots + (-h)^{nd})$ , can be computed by a division-free circuit of size  $\text{poly}(m, d)$  and depth increased by only  $\mathcal{O}(1)$ . Thus we obtain a division-free circuit for  $f(\mathbf{X}_n + \alpha)$ .

Finally, substituting  $\mathbf{X}_n - \alpha$  for  $\mathbf{X}_n$  in this division-free circuit gives a division-free circuit computing  $f(\mathbf{X}_n)$ . This substitution increases the size only polynomially and does not increase the depth by more than a constant. Therefore the resulting circuit has size  $\text{poly}(m, d)$  and depth at most  $\Delta + \mathcal{O}(1)$ , and the whole construction runs in  $\text{poly}(n, s, m, d)$  time. ■