Journal of Symbolic Computation 75 (2016) 4–24

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Faster sparse multivariate polynomial interpolation of straight-line programs

Andrew Arnold [a], Mark Giesbrecht [a], Daniel S. Roche [b]

[a] Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
[b] Computer Science Department, United States Naval Academy, Annapolis, MD, USA

ARTICLE INFO

Article history:
Received 9 December 2014
Accepted 10 July 2015
Available online 10 November 2015

Keywords:
Sparse interpolation
Complexity
Randomized algorithms
Straight-line program

ABSTRACT

Given a straight-line program whose output is a polynomial function of the inputs, we present a new algorithm to compute a concise representation of that unknown function. Our algorithm can handle any case where the unknown function is a multivariate polynomial, with coefficients in an arbitrary finite field, and with a reasonable number of nonzero terms but possibly very large degree. It is competitive with previously known sparse interpolation algorithms that work over an arbitrary finite field, and provides an improvement when there are a large number of variables.

Published by Elsevier Ltd.

1. Introduction

We consider the problem of interpolating a sparse multivariate polynomial F over $\mathbb{F}_q$, the finite field of size q:

$$F = \sum_{\ell=1}^{t} c_\ell z_1^{e_{\ell 1}} z_2^{e_{\ell 2}} \cdots z_n^{e_{\ell n}} \in \mathbb{F}_q[z_1, \ldots, z_n]. \quad (1)$$

We suppose F is given by a *Straight-Line Program* (SLP), a list of simple instructions performing operations $+$, $-$ and $\times$ on inputs and previously computed values, which evaluates the polynomial at any point. We further suppose we are given bounds $D > \max_j \deg_{z_j}(F)$ and $T \geq t$. It is expected that the

E-mail addresses: a4arnold@uwaterloo.ca (A. Arnold), mwg@uwaterloo.ca (M. Giesbrecht), roche@usna.edu (D.S. Roche).
URLs: http://www.AndrewArnold.ca (A. Arnold), http://www.uwaterloo.ca/~mwg (M. Giesbrecht), http://www.usna.edu/cs/roche (D.S. Roche).

http://dx.doi.org/10.1016/j.jsc.2015.11.005
0747-7171/Published by Elsevier Ltd.

bound T tells us that F is *sparse*, i.e., that $T \ll D^n$, the maximum number of terms. The goal of our interpolation algorithm is to obtain the t nonzero coefficients $c_\ell \in \mathbb{F}_q$ and corresponding exponents $\mathbf{e}_\ell = (e_{\ell_1}, \ldots, e_{\ell_n}) \in \mathbb{Z}^n$ of F. Our contribution is as follows.

Theorem 1. *Let $F \in \mathbb{F}_q[z_1, \ldots, z_n]$, and suppose we are given a division-free straight-line program $\mathcal{S}_F$ of length L which evaluates F, an upper bound $D \geq \max_j \deg_{z_j}(F)$, and an upper bound T on the number of nonzero terms t of F. There exists a probabilistic algorithm which interpolates F with probability at least $3/4$. The algorithm requires*

$$\widetilde{\mathcal{O}}\left(Ln(T \log D + n)(\log D + \log q) \log D + n^{\omega-1} T \log D + n^\omega \log D\right)$$

bit operations.[1,2]

This probability may be increased to $1 - \epsilon$ using standard techniques, with cost increased by a factor $\mathcal{O}(\log(\epsilon^{-1}))$.

The rest of this introductory section puts our work in context and defines the notation and problem definitions for the rest of the paper. The reader who is already familiar with the area may wish to glance at our list of notation in Appendix A, then skip to Section 2, where we give a high-level overview of the algorithm referred to by Theorem 1 and work out a small illustrative example in full detail. The end of Section 2 provides an outline for the remainder of the paper.

1.1. Background and related work

Polynomial interpolation is a fundamental problem of computational mathematics that dates back centuries to the classic work of Newton, Waring, and Lagrange. In such settings, given a list of $(n+1)$-dimensional points and some degree bounds, the coefficients of the unique n-variate polynomial interpolating those points is produced.

If the number of nonzero coefficients is relatively small, the unknown function can be treated as an exponential sum, and the task becomes that of finding the exponents and coefficients of only the nonzero terms. This is the *sparse interpolation* problem, and it differs crucially from other interpolation problems not only in the representation of the output, but also that of the input. Every efficient sparse interpolation algorithm of which we are aware requires some control over where the unknown function is sampled, and typically takes as input some procedure or black box that can evaluate the unknown sparse polynomial at any chosen point.

The sparse interpolation problem has received considerable interest over fields of characteristic zero. The classical Prony's method for exponential sums from 1795 (which can be regarded as the genesis of sparse interpolation) was later applied to sparse interpolation over the integers (Ben-Or and Tiwari, 1988; Kaltofen, 2010) and approximate complex numbers (Giesbrecht et al., 2009; Kaltofen et al., 2011). Compressive sensing is a different approach for approximate sparse interpolation which has the advantage of allowing the evaluation points to be chosen at random from a certain distribution (Candés et al., 2006; Donoho, 2006). Sparse Fourier and Hadamard–Walsh transforms allow for the interpolation of a sparse, complex-valued polynomial given by its discrete Fourier transform, and can find reasonable sparse approximations to non-sparse polynomials (Hassanieh et al., 2012; Kushilevitz and Mansour, 1993).

Straight-line programs are central to the study of algebraic complexity. One measure of the complexity of a rational function is the least size of a straight-line program that computes it. In 2003, in the celebrated paper (Kabanets and Impagliazzo, 2004) by Kabanets of Impagliazzo, it is shown that a deterministic polynomial-time algorithm for the identity testing of $F \in \mathbb{Z}[x]$ given by a straight-line

[1] For two functions ϕ, ψ, we say $\phi \in \widetilde{\mathcal{O}}(\psi)$ if and only if $\phi \in \mathcal{O}(\psi \log^c \psi)$ for some constant $c \geq 0$.

[2] The constant $\omega < 2.38$ is the exponent of matrix multiplication, meaning that the product of two $n \times n$ matrices can be computed in $O(n^\omega)$ field operations.

program would imply circuit lower bounds. This has encouraged efforts to derandomize identity testing of straight-line programs. We refer the reader to Shpilka and Yehudayoff (2010) for a detailed overview of recent developments.

Straight-line programs can model any polynomial resulting from a sequence of arithmetic operations. The aim of interpolation in this setting is to learn the monomial representation of this resulting polynomial F. Interpolation may be thought of as a natural generalization of polynomial identity testing. One can naively interpolate such a polynomial F by performing each arithmetic operation in the polynomial ring containing F. The polynomials resulting from intermediate operations in such an approach may be appreciably larger than the final result F. Sparse interpolation supposes that F comprises a small number of terms, i.e., that F is sparse, in which case we can reduce this intermediate expression swell by working with appropriately small homomorphic images of F. This is the approach taken in a number of previous sparse interpolation algorithms and the one presented herein.

As in the rest of this paper, define n, T, and D to be (respectively) the number of variables and known bounds on the number of terms and degree of the unknown polynomial. An information-theoretic lower bound on the complexity of univariate sparse interpolation is $\Omega(T(n \log D + \log q))$, the number of bits used to encode F in (1). This bound is (nearly) met by Prony's (Prony, 1795) algorithm (as adapted to the polynomial setting; see Kaltofen, 2010), which requires $O(T \log q + TL)$ bit operations when $n = 1$ and under the implicit assumption that $q > D$. Much of the complexity of the sparse interpolation problem appears to arise from the requirement to accommodate *any* finite field. Prony's algorithm is dominated by the cost of discrete logarithms in $\mathbb{F}_q$, for which no polynomial time algorithm is known in general. When there is a choice of fields (say, as might naturally arise in a modular scheme for interpolating integer or rational polynomials) more efficient interpolation methods have been developed. Kaltofen (1988) demonstrates a method for sparse interpolation over $\mathbb{F}_p$ for primes p such that $p-1$ is smooth; see (Kaltofen, 2010) for further exposition. In our notation, this algorithm would require $\widetilde{\mathcal{O}}(LnT \log D + n^2 T \log^2 D)$ bit operations.

Zippel (1990), Huang and Rao (1999), Javadi and Monagan (2010) developed multivariate sparse interpolation algorithms that work over arbitrary finite fields, but whose running time is polynomial in D, the degree. The running time of these methods is expressed in our terminology below for comparison, but in fairness we should point out that (Javadi and Monagan, 2010) is more general than our algorithm as it relies exclusively on classical polynomial arithmetic and requires only black-box access to the unknown function.

In extreme cases, the size of the sparse representation may not be polynomial in the degree D, but rather in $\log D$, as this sparse representation stores only the coefficients and exponents of nonzero terms. In such cases, black-box access to the unknown function may not even be sufficient, since the degree could be much larger than the number of field elements. Typically, algorithms in this setting take as input a straight-line program, which allows for evaluations in a field extension or modulo an ideal.

The earliest algorithm for sparse interpolation of a straight-line program over an arbitrary finite field whose cost is polynomial in n, T, and $\log D$ — that is, the sparse representation size — was presented by Garg and Schost (2009). A series of results (Arnold et al., 2013, 2014; Arnold and Roche, 2014; Giesbrecht and Roche, 2011) has made use of different randomizations to improve the complexity of this approach, in particular reducing the dependence on the sparsity T to quasi-linear.

The aforementioned algorithms for sparse interpolation of straight-line programs are essentially univariate algorithms, but can easily be extended to handle multivariate polynomials by use of the well-known Kronecker substitution. A separate paper from two co-authors at ISSAC 2014 (Arnold and Roche, 2014) presented a new randomization that achieves similar aims as the Kronecker substitution but with decreased degrees for sparse polynomials. This technique is sufficiently general that it can be combined with a wide variety of univariate interpolation algorithms to achieve faster multivariate interpolation.

Table 1 summarizes the complexity of the best known algorithms for multivariate sparse interpolation of straight-line programs over an arbitrary finite field. Our new algorithm in this paper uses many of the ideas in our ISSAC 2014 work (Arnold and Roche, 2014; Arnold et al., 2014), but synthesizes them in a novel way and reduces the amount of linear algebra required. The cost is similar to the combination of our ISSAC 2014 results, but will be faster when the number of variables n is

Table 1
Complexity of multivariate sparse interpolation algorithms.

Algorithm	Soft-O cost of SLP interpolation
Javadi and Monagan (2010)	$T^2 \log q(n+D) + LnT \log q$
Garg and Schost (2009)	$Ln^2T^4 \log^2 D \log q + n^4T^4 \log^3 D \log q$
Arnold et al. (2014) with Arnold and Roche (2014)	$LnT(\log D + \log q)\log^2 D + n^\omega T$
This paper (Thm. 1)	$Ln(T \log D + n)(\log D + \log q)\log D$ $+ n^{\omega-1}T(\log D) + n^\omega \log D$

sufficiently large that the cost of computing $O(T)$ matrix inverses dominates the complexity of the previous approach.

As a concrete example that may simplify the results of Table 1 and highlight the improvements here, suppose the size of the SLP is equal to the number of nonzero terms ($L = T$), which is in turn twice the number of variables ($T = 2n$), and the degree is bounded by a polynomial in the number of terms ($D < T^{\mathcal{O}(1)}$). Then the cost of the algorithm in Javadi and Monagan (2010) is at least $\widetilde{\mathcal{O}}(n^4)$, the algorithm obtained from our ISSAC 2014 results has cost $\widetilde{\mathcal{O}}(n^{\omega+1})$, and the new algorithm presented in this paper further reduces the complexity in this case to $\widetilde{\mathcal{O}}(n^3)$.

1.2. Conventions and notation

The technical nature of our results unfortunately necessitates a fair bit of notation. In an effort to unburden the reader, we define the most important facets of our notation here, and provide a reference table in Appendix A that contains the names and conventions used throughout the paper.

We will write vectors in boldface and vector entries in standard typeface (with subscripts). Column vectors will be written as comma-separated tuples. That is, we will write a length-n column vector as $\mathbf{v} = (v_1, \ldots, v_n)$, whereas $\mathbf{v}_1, \ldots, \mathbf{v}_\ell$ denotes a list of ℓ vectors where each $\mathbf{v}_i = (v_{i1}, \ldots, v_{in})$.

A slight exception to this is that we will let $\mathcal{Z}$ be the vector of indeterminates $(z_1, \ldots, z_n)$, and hence $\mathbb{F}_q[z_1, \ldots, z_n]$ is written $\mathbb{F}_q[\mathcal{Z}]$. For any exponent vector $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{Z}^n_{\geq 0}$ we write $\mathcal{Z}^{\mathbf{e}}$ for the term $z_1^{e_1} \cdots z_n^{e_n}$. Similarly for any vector $\mathbf{a} = (a_1, \ldots, a_n)$, we write $\mathbf{a}^{\mathbf{e}}$ for the product $a_1^{e_1} \cdots a_n^{e_n}$. For a single indeterminate x, the notation $x^{\mathbf{e}}$ means $x^{e_1} \cdots x^{e_n} = x^{e_1+\cdots+e_n}$.

We assume a polynomial image $G(x)$ mod $H(x)$ is represented in its reduced form. That is, we store the image as $R(x)$, where $G = HQ + R$, $\deg_x R < \deg_x H$. Similarly, an integer s reduced modulo $t \in \mathbb{Z}_{>0}$ is represented by r such that $s = tq + r$, where $q, r \in \mathbb{Z}$ and $0 \leq r < t$.

We use "soft-oh" notation for our cost analysis. For an integer constant k and two functions $\phi, \psi : \mathbb{R}^k_{>0} \to \mathbb{R}_{>0}$, we say $\phi \in \widetilde{\mathcal{O}}(\psi)$ if and only if $\phi \in \mathcal{O}(\psi \log^c \psi)$ for some constant $c > 0$. Our algorithm will depend on asymptotically fast matrix multiplication. By Le Gall (2014), the cost of multiplying two $n \times n$ matrices over a field F entails $\mathcal{O}(n^\omega)$ arithmetic operations in F, where we can take $\omega = 2.3728639$.

For $q = p^v$, p prime, we suppose that elements of $\mathbb{F}_q$ are represented as $\mathbb{Z}_p[x]/\langle \Phi(x) \rangle$, where Φ is a degree-v irreducible polynomial over $\mathbb{Z}_p$. We further identify $\mathbb{F}_{q^u}$ with $\mathbb{F}_q[y]/\langle \Psi(y) \rangle$, for Ψ an irreducible polynomial over $\mathbb{F}_q$. Under such representation, arithmetic operations in $\mathbb{F}_{q^u}$ may be computed in $\widetilde{\mathcal{O}}(u)$ arithmetic operations in $\mathbb{F}_q$, each of which in turn may be computed in $\widetilde{\mathcal{O}}(v)$ arithmetic operations in $\mathbb{Z}_p$ (Cantor and Kaltofen, 1991).[3] Multiplication in $\mathbb{Z}_p$ may be performed with $\widetilde{\mathcal{O}}(\log p)$ bit operations (e.g., refer to Fürer (2009) and Thm. 9.8 of Gathen and Gerhard (2003)). It follows that one can perform an arithmetic operation in $\mathbb{F}_{q^u}$ in $\widetilde{\mathcal{O}}(u \log q)$ bit operations.

Our algorithm will require randomness. We assume that we may obtain a random bit with bit-cost $\mathcal{O}(1)$, and that the cost of choosing x from a set S uniformly at random admits a bit-cost of $\mathcal{O}(\log |S|)$. The amount of randomness required for our algorithm is stated in Lemma 10.

The polynomial F we are interpolating will be assumed to possess the following description and features throughout the remainder of this article:

[3] See, e.g., Thm. 9.6 of Gathen and Gerhard (2003) for a cost analysis for division operations.

$$
\begin{aligned}
F &= \sum_{\ell=1}^{t} c_\ell \mathcal{Z}^{\mathbf{e}_\ell} \in \mathbb{F}_q[\mathcal{Z}],\\
D &\geq \max_{j\in[n]} \deg_{z_j}(F),\\
T &\geq t,\\
f^{(\ell)} &= c_\ell \mathcal{Z}^{\mathbf{e}_\ell},\\
\mathcal{I}_i &= \langle z_1^{p_i} - 1, \ldots, z_n^{p_i} - 1\rangle,\\
F_i &= F \bmod \mathcal{I}_{p_i} \in \mathbb{F}_q[\mathcal{Z}]/\mathcal{I}_{p_i},\\
F_{ij} &= F(x^{\mathbf{v}_{ij}}) \bmod (x^{p_i} - 1) \in \mathbb{F}_q[x]/\langle x^{p_i} - 1\rangle,\\
F_{ijk} &= F(\mathbf{a}_k x^{\mathbf{v}_{ij}}) \bmod (x^{p_i} - 1) \in \mathbb{F}_{q^u}[x]/\langle x^{p_i} - 1\rangle.
\end{aligned}
$$

1.3. Straight-line programs

A Straight-Line Program (SLP) is a branchless sequence of arithmetic instructions that may represent a rational function.

Definition 2. A *Straight-Line Program* (SLP) over a ring R with inputs $z_1, \ldots, z_n$, is a sequence of arithmetic instructions $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_L)$ of the form $\mathcal{S}_i = (\beta_i \leftarrow \alpha \star \alpha')$, where $\star \in \{+, -, \cdot, \div\}$ and

$$\alpha, \alpha' \in \{z_1, \ldots, z_n\} \cup \{\beta_j \mid j < i\} \cup \mathsf{R}.$$

We say β_L (if well-defined) is the output for a choice of inputs $z_1, \ldots, z_n$.

We say $\mathcal{S}$ is *n-variate* if it accepts n inputs. We let L denote the length, or number of instructions, of an SLP. We henceforth restrict our attention to *division-free* SLPs over R, so as to avoid possible division by zero.

Since an SLP gives us a list of arithmetic instructions, we may choose inputs z_i from R, homomorphic images of R, or ring extensions thereof. For instance, we may treat z_i as indeterminates, in which case, the resulting outputs β_i each are polynomials from $\mathsf{R}[\mathcal{Z}]$. We say $\mathcal{S}$ *computes* $F \in \mathsf{R}[\mathcal{Z}]$ if it outputs $\beta_L = F$ given indeterminate inputs $z_1, \ldots, z_n$. We write $\mathcal{S}_F$ to denote an SLP that computes F. Every division-free SLP over R computes some $F \in \mathsf{R}[\mathcal{Z}]$.

The aim of *sparse interpolation*, given an SLP $\mathcal{S}_F$, is to construct a *sparse representation* of F: a list of nonzero terms of distinct degree comprising F. For instance, $F = 5z_1^6 + 7z_1^2z_2^3$ admits a sparse representation $((5, (6, 0)), (7, (2, 3)))$.

One could naively interpolate $\mathcal{S}_F$ by treating inputs $z_1, \ldots, z_n$ as indeterminates and performing each arithmetic instruction as an arithmetic operation in $\mathsf{R}[\mathcal{Z}]$. A caveat of such an approach is that the number of terms and degree of intermediate results β_i can be arbitrarily large compared to the size of F. Such an approach, in the worst case, has cost exponential in L.

Instead, we will use $\mathcal{S}_F$ to compute homomorphic images of F. We construct images of the form $F(a_1x^{v_1}, \ldots, a_nx^{v_n}) \bmod (x^p - 1)$, for appropriate choices of $p \in \mathbb{Z}_{>0}$. Computing images of this form is preferable because it bounds the cost of executing each arithmetic instruction of $\mathcal{S}_F$. Namely, we perform arithmetic in $\mathsf{R}[x]/\langle x^p - 1\rangle$. Using FFT-based techniques, one can perform arithmetic operations in $\mathsf{R}[x]/\langle x^p - 1\rangle$, R a ring, in $\widetilde{\mathcal{O}}(p)$ arithmetic operations in R (Cantor and Kaltofen, 1991).

In our setting $\mathsf{R} = \mathbb{F}_q$ usually, but sometimes we will have to choose a_i from a ring extension $\mathbb{F}_{q^u}$, in which case we construct an image of $F \in \mathbb{F}_{q^u}[x]/\langle x^p - 1\rangle$. Per the previous discussion of finite field arithmetic in section 1.2, this gives the following cost, which we state as a claim.

Claim 3. *Given a length-L division-free SLP $\mathcal{S}_F$ computing $F \in \mathbb{F}_q[\mathcal{Z}]$, one can construct an image of F in $\mathbb{F}_{q^u}[x]/\langle x^p - 1\rangle$ in $\widetilde{\mathcal{O}}(Lpu)$ operations in $\mathbb{F}_q$, or $\widetilde{\mathcal{O}}(Lpu \log q)$ bit operations.*

2. Overview

Our algorithm works by using three types of randomized homomorphisms to compress the size of the problem, while still maintaining sufficient information for reconstruction. The randomization ensures that there is not too much "information collapse" and we can correlate the different homomorphic images with terms of the sparse polynomial we are attempting to recover. A complete description of the algorithm is given later in Procedure SparseInterpolate, along with a complete analysis, but we present the main ideas here.

2.1. A high-level description of the algorithm

We employ three distinct types of homomorphism. The first reduces the degree of each variable, the second transforms the problem into a univariate problem while the final one ensures that all the terms are distinct (and so can be identified during reconstruction).

For the first type of homomorphism, we truncate, or wrap, each of the variables z_j modulo a prime p_i. In fact, we construct this same homomorphism m times, with a collection of random primes $p_1, \ldots, p_m$, selected once for the entire computation. Specifically, let F_i be the polynomial obtained by replacing $z_j \in \mathbb{F}_q[\mathcal{Z}]$ by $z_j \bmod (z_j^{p_i} - 1)$ for all $j \in [n]$. Letting $\mathcal{I}_{p_i}$ be the ideal $\langle z_1^{p_i} - 1, \ldots, z_n^{p_i} - 1\rangle$, we define

$$\begin{aligned}\Phi_i : \quad & \mathbb{F}_q[\mathcal{Z}] \to \mathbb{F}_q[\mathcal{Z}]/\mathcal{I}_i \\ & z_1^{e_1} \cdots z_n^{e_n} \mapsto z_1^{e_1 \bmod p_i} \cdots z_n^{e_n \bmod p_i},\end{aligned}$$

so that $F_i = \Phi_i(F)$. We say two terms $c\mathcal{Z}^{\mathbf{e}}$ and $c'\mathcal{Z}^{\mathbf{e}'}$ (alternatively, their exponents) *collide* under this homomorphism if $\Phi_i(z^{\mathbf{e}}) = \Phi(z^{\mathbf{e}'})$. This happens precisely when $\mathbf{e} \equiv \mathbf{e}' \bmod p_i$, and we say that terms $c\mathcal{Z}^{\mathbf{e}}$ and $c\mathcal{Z}^{\mathbf{e}'}$ collide in F_i in this case. We define a *collision* to be a set of size at least two comprising all the terms of F whose exponents all agree under Φ_i.

The second homomorphism builds on the first type, by not only truncating the degrees but also reducing the number of variables to a new one x. For each prime p_i, we choose n random vectors $\mathbf{v}_{i1}, \ldots, \mathbf{v}_{in} \in \mathbb{Z}_p^n$. The homomorphism Ψ_{ij} is then defined by

$$\begin{aligned}\Psi_{ij} : \mathbb{F}_q[\mathcal{Z}] &\to \mathbb{F}_q[x]/\langle x^p - 1\rangle \\ z_k &\mapsto x^{v_{ijk} \bmod p_i}.\end{aligned}$$

This further reduces the size of images we need to compute by an exponential factor in n.

Consider the term $f = cz^{\mathbf{e}} = cz_1^{e_1} \cdots z_n^{e_n}$ of F, which maps under Ψ_{ij} to

$$\Psi_{ij}(f) = cx^{(e_1 v_{ij1} + \cdots + e_{ijn} v_{ijn}) \bmod p_i} = cx^{\mathbf{v}_{ij} \cdot \mathbf{e}} \bmod (x^{p_i} - 1),$$

and suppose its image has degree $d_{ij} < p_i$. Then, combining the results from n such homomorphisms, $(e_1, \ldots, e_n) = \mathbf{e} \bmod p_i$ is the solution to the linear system

$$\underbrace{\begin{bmatrix} v_{i11} & \cdots & v_{i1n} \\ \vdots & & \\ v_{in1} & \cdots & v_{inn} \end{bmatrix}}_{V_i} \underbrace{\begin{bmatrix} e_1 \\ \vdots \\ e_n \end{bmatrix}}_{\mathbf{e}} \equiv \underbrace{\begin{bmatrix} d_{i1} \\ \vdots \\ d_{in} \end{bmatrix}}_{\mathbf{d}_i} \bmod p_i. \tag{2}$$

The basic idea of our algorithm is then to first make a selection of random primes $p_1, \ldots, p_m$, and for each prime choose a random nonsingular matrix $V_i \in \mathbb{Z}_{p_i}^{n \times n}$ and compute its inverse. Now for each term $f = c\mathcal{Z}^{\mathbf{e}}$ of F, and for each prime p_i, we obtain a vector of degrees $\mathbf{d}_i$. Using linear algebra as above, we can determine the degree vector $(e_{i1}, \ldots, e_{in}) = \mathbf{e} \bmod p_i$.

In order for us to identify such a vector $\mathbf{d}_i$, we require that *(i)* f is not in a collision in any of the images F_{ij}, for any $j \in [n]$, and *(ii)* all the other terms in images F_{ij} have a distinct coefficient from that of f. The first criterion is probabilistically ensured by our choice of images F_{ij}. In order to obtain

the latter, we need to employ a third type of homomorphism which randomizes coefficients of terms of F, as well as coefficients arising from collisions of terms. This technique is called *diversification* after its introduction in Giesbrecht and Roche (2011).

Specifically, *before* applying the aforementioned mappings to F, we first choose a small number s of random vectors $\mathbf{a}_1, \ldots, \mathbf{a}_s$ over a field extension $\mathbb{F}_{q^u}$, each of which defines a mapping

$$(z_1, \ldots, z_n) \mapsto (a_{k1}z_1, \ldots, a_{kn}z_n).$$

We then apply the latter homomorphisms and construct

$$F_{ijk} \stackrel{\text{def}}{=} \sum_{\ell=1}^{t} c_\ell \mathbf{a}_k^{\mathbf{e}_\ell} \mathcal{Z}^{\mathbf{e}_\ell \mathbf{v}_j \bmod p_i},$$

for $(i, j, k) \in [m, n, s]$.

Now observe that the collisions only occur according to the choices of primes p_i and exponent vectors $\mathbf{v}_j$; the different choices for $\mathbf{a}_k$ affect the coefficients in each image but not the corresponding exponents. So for each pair $(i, j) \in [m, n]$, we have a sequence of degrees of nonzero terms that appear in any F_{ijk} for $k \in [s]$. These are exactly the degrees of nonzero terms in the polynomials determined by the second homomorphism above, $F_{ij} = \Psi_{ij}(F)$.

In order to recover the complete multivariate exponents of the terms from these images, we need to correlate these degrees in different images F_{ij} and $F_{i'j'}$, according to which term in F they correspond to. This is where the third homomorphism (diversification) is used: the random choice of $\mathbf{a}_k$'s guarantees that, with high probability, if the degree-d term of F_{ij} does *not* correspond to the degree-d' term of $F_{i'j'}$, there exists at least one $\mathbf{a}_k$ such that the degree-d term of F_{ijk} has a different coefficient than the degree-d' term of $F_{i'j'k}$. Hence the degrees in different images F_{ij} can be grouped according to their coefficients in *all* of the images F_{ijk}, and we obtain a *vector* of coefficients for each term in F_{ij}.

In our algorithm, this grouping is facilitated by a dictionary, for each prime p_i, mapping coefficient vectors that appear in images F_{ij} to their degrees. Whenever the same coefficient vector appears for every vector $\mathbf{v}_j$, $j \in [n]$, there is enough information to set up a linear system as in (2) and recover that term's exponents modulo p.

For each term $f = c\mathcal{Z}^{\mathbf{e}}$ of F, the probability of actually obtaining c and $\mathbf{e}$ mod p_i for some $i \in [m]$ will be shown to exceed 9/10. By choosing sufficiently many primes p_i, we can then guarantee that we will have enough information to construct all the terms of f, and to detect all images of collisions, with probability 3/4.

2.2. An illustrative example

We consider an example which explains how we could use the suggested homomorphic images in order to construct terms of F, as well as obstacles to our approach. Suppose we are given an SLP that computes

$$F = \underbrace{z_1 z_2}_{f^{(1)}} + \underbrace{z_1^6 z_2^6}_{f^{(2)}} + \underbrace{2z^4 z^{10}}_{f^{(3)}} + \underbrace{4z_1^3 z_2^{20}}_{f^{(4)}} \in \mathbb{F}_{13}[z_1, z_2],$$

and we are given that $\max_{j\in[2]} \deg_{z_j} F < D = 21$, and that F has at most $T = 4$ nonzero terms. Suppose we choose primes $p_1 = 5$ and $p_2 = 7$, such that

$$\begin{aligned} F_1 = F \bmod (z_1^5 - 1, z_2^5 - 1) &= \overbrace{2z_1 z_2}^{\Phi_1(f^{(1)}+f^{(2)})} + \overbrace{2z_1^4}^{\Phi_1(f^{(3)})} + \overbrace{4z_1^3}^{\Phi_1(f^{(4)})}, \\ F_2 = F \bmod (z_1^7 - 1, z_2^7 - 1) &= \underbrace{z_1 z_2}_{\Phi_2(f^{(1)})} + \underbrace{z_1^6 z_2^6}_{\Phi_2(f^{(2)})} + \underbrace{2z_1^4 z_2^3}_{\Phi_2(f^{(3)})} + \underbrace{4z_1^3 z_2^6}_{\Phi_2(f^{(4)})}. \end{aligned}$$

Under the homomorphism Φ_1, the terms $f^{(1)}$ and $f^{(2)}$ *collided*. We call this type of collision an *exponent collision*. Keep in mind that we do not know F, F_1, or F_2. We will construct images of F_1 and F_2 in order to recover terms of F.

We choose $\mathbf{v}_{11} = (4, 1), \mathbf{v}_{12} = (2, 0) \in \mathbb{Z}_5^2$, such that

$$\begin{aligned} F_{11} = F(x^4, x) \bmod (x^5 - 1) &= \overbrace{2}^{\Psi_{11}(f^{(1)}+f^{(2)})} + \overbrace{2x}^{\Psi_{11}(f^{(3)})} + \overbrace{4x^2}^{\Psi_{11}(f^{(4)})}, \\ F_{12} = F(x^2, 1) \bmod (x^5 - 1) &= \underbrace{2x^2}_{\Psi_{12}(f^{(1)}+f^{(2)})} + \underbrace{2x^3}_{\Psi_{12}(f^{(3)})} + \underbrace{4x.}_{\Psi_{12}(f^{(4)})} \end{aligned}$$

We similarly choose $\mathbf{v}_{21} = (2, 4), \mathbf{v}_{22} = (1, 6) \in \mathbb{Z}_7^2$, such that

$$\begin{aligned} F_{21} = F(x^2, x^4) \bmod (x^7 - 1) &= \overbrace{3x^6}^{\Psi_{21}(f^{(1)}+f^{(3)})} + \overbrace{x}^{\Psi_{22}(f^{(2)})} + \overbrace{4x^2}^{\Psi_{21}(f^{(4)})}, \\ F_{22} = F(x, x^6) \bmod (x^7 - 1) &= \underbrace{2}_{\Psi_{22}(f^{(1)}+f^{(2)})} + \underbrace{2x}_{\Psi_{22}(f^{(3)})} + \underbrace{4x^4.}_{\Psi_{22}(f^{(4)})} \end{aligned}$$

If we constructed F_{11} and F_{12}, we could suppose (correctly) that their terms with coefficient 4 are images of the same single term f of F. We could then construct the exponent $\mathbf{e}$ of $\Phi_1(f)$ as the solution to the linear system

$$\underbrace{\begin{bmatrix} 4 & 1 \\ 2 & 0 \end{bmatrix}}_{V_1} \mathbf{e} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \bmod 5,$$

which gives $\mathbf{e} = (3, 0)$, from which we recover the term $\Phi_1(f^{(4)}) = 4z_1^3$ of F_1. While $f^{(3)}$ also does not collide in either image F_{11} or F_{12}, we cannot construct $\Phi_1(f^{(3)})$ in the same fashion because both F_{11} and F_{12} have two terms with coefficient 2. We call a pair of distinct terms (f, f') over all pairs of nonzero terms from the images $F_{ij}, (i, j) \in [2, 2]$, a *deceptive pair* if f and f' share the same coefficient but are *not* images of the same sum of terms of F. For instance the terms 2 of F_{11} and $2x$ of F_{22} form a deceptive pair, as the former is an image of $f^{(1)} + f^{(2)}$, whereas the latter is an image of $f^{(3)}$. Similarly, we say that the two terms with coefficient 2 in F_{11} form a deceptive pair; even though in this case they are not so "deceptive" as they appear distinctly in the same image.

Now if we have images F_{21}, F_{22} we can reconstruct the exponent $\mathbf{e}$ of $\Phi_2(f^{(4)})$ by way of the linear system

$$\underbrace{\begin{bmatrix} 2 & 4 \\ 1 & 6 \end{bmatrix}}_{V_2} \mathbf{e} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \bmod 7,$$

to get a solution $\mathbf{e} = (3, 6)$, which gives us the term $\Phi_2(f^{(4)}) = 4z_1^3 z_2^6$ of F_2. We cannot construct $\Phi_2(f)$ for any other term f of F in this fashion, because $f^{(1)}$ and $f^{(3)}$ collide in image F_{21}, and $f^{(1)}$ and $f^{(2)}$ collide in image F_{22}. We call these types of collision that depends on our choices of $\mathbf{v}_{21}$ and $\mathbf{v}_{22}$ *substitution collisions*. If V_i is invertible, it gives a linear injective map, in which case it follows that any distinct terms of F_i will not collide for at least one vector $\mathbf{v}_{ij}$, $j \in [n]$.

If we (correctly) suppose our recovered terms $4z_1^3 \in \mathbb{F}_{13}[\mathcal{Z}]/\mathcal{I}_1$ and $4z_1^3 z_2^5 \in \mathbb{F}_{13}[\mathcal{Z}]/\mathcal{I}_2$ are images of the same single term of F, then we can reconstruct the exponent $\mathbf{e}$ of the term $f^{(4)}$ of F by way of the system of congruences

$$\begin{aligned} \mathbf{e} \bmod 5 &= (3, 0), \\ \mathbf{e} \bmod 7 &= (3, 6). \end{aligned}$$

We use Chinese Remaindering to the solve the system to get $\mathbf{e} = (3, 20)$, from which we recover term $4z_1^3 z_2^{20}$ of F. Our algorithm will choose sufficiently many primes such that, with high probability, we will be able to construct the exponents of *every* term of F.

We were unable to recover some of the terms of F because of deceptive pairs. We would like that any terms in a deceptive pair are somehow distinguishable as images of distinct sums of terms of F. To that end we can choose $\mathbf{a}_1 = (6, 8) \in \mathbb{F}_{13}$ and construct the corresponding images of $F(6z_1, 8z_2)$. Note that

$$\begin{aligned} F(6z_1, 8z_2) &= 9z_1z_2 + z_1^6 z_2^6 + 8z_1^4 z_2^{10} + 6z_1^3 z_2^{20}, \\ F(6z_1, 8z_2) \bmod (z_1^5 - 1, z_2^5 - 1) &= 10z_1z_2 + 8z_1^4 + 6z_1^3, \\ F(6z_1, 8z_2) \bmod (z_1^7 - 1, z_2^7 - 1) &= 9z_1z_2 + z_1^6 z_2^6 + 8z_1^4 z_2^3 + 6z_1^3 z_2^6. \end{aligned}$$

Again we do not directly compute the images listed above, but rather we construct

$$\begin{aligned} F_{111} &= F(6x^4, 8x) \bmod (x^5 - 1) = 10 + 8x + 6x^2, \\ F_{121} &= F(6x^2, 8) \bmod (x^5 - 1) = 10x^2 + 8x^3 + 6x, \\ F_{211} &= F(6x^2, 8x^4) \bmod (x^7 - 1) = 4x^6 + x + 6x^2, \\ F_{221} &= F(6x, 8x^6) \bmod (x^7 - 1) = 10 + 8x + 6x^4. \end{aligned}$$

Note that the terms of F_{ij1} have the same exponents as those of F_{ij} for $(i, j) \in [2, 2]$. Only the coefficients are affected by this additional homomorphism. However, now we can observe, for instance, that the degree-0 term of F_{111} differs from the degree-3 term of F_{121}, such that we now can tell the degree-0 term of F_{11} and the degree-3 term of F_{12} form a deceptive pair. We say $\mathbf{a}_1$ *reveals* the deceptive pair. We leave it to the reader to check that $\mathbf{a}_1$ reveals every deceptive pair of terms from $F_{ij}, (i, j) \in [2, 2]$. We may have to choose multiple vectors $\mathbf{a}_k$ in order to reveal all deceptive pairs. We may also have to choose $\mathbf{a}_k$ over a sufficiently large field extension $\mathbb{F}_{q^u}$.

If all deceptive pairs are revealed, then we can collect all terms f of images F_{ij} according to their coefficients and the coefficients of their corresponding terms in the images F_{ijk}. Any such collection of terms consists of images of the same sum of terms of F.

For instance, we can collect the terms with coefficient 2 in F_{11}, F_{12} and F_{22} whose corresponding terms in F_{111}, F_{121}, and F_{221}, respectively, have coefficient 8. These are exactly the terms of F_{ij} that are images of $f^{(3)}$. Those terms from F_{11} and F_{12} allow us to construct an exponent $\mathbf{e} = V_1^{-1}(1, 2) \bmod 5 = (4, 0)$ of a term of F_1. This gives $\mathbf{e} = (4, 0)$. The corresponding coefficient 2. This gives the term $\Phi_1(f^{(3)}) = 2z_1^4$ of F_1.

Unfortunately, in this same fashion we can now collect terms $2, 2x^2$, and 2 of F_{11}, F_{12} and F_{22} respectively, all images of $f^{(1)} + f^{(2)}$. Considering the pair of such terms in F_{11} and F_{12}, we can construct an exponent $\mathbf{e} = V_1^{-1}(0, 2) \bmod 5 = (1, 1)$, which gives a term $2x_1x_2 \in \mathbb{F}_{13}[z_1, z_2]/\langle z_1^5 - 1, z_2^5 - 1\rangle$. If $f^{(1)}$ and $f^{(2)}$ had exponent collisions for too many primes p_i, and we did not detect that the resulting terms of F_i produced were images of a sum of terms of F, then naively we might use Chinese Remaindering to then construct an exponent that is *not* a term of F.

As the partial degrees of F are at most 20, any exponents $\mathbf{e} \neq \mathbf{e}'$ of F cannot be identical modulo 5 and 7. Thus, were *every* deceptive pair revealed and we collected terms in the manner prescribed, any collection that results in a recovered term in both F_1 and F_2 could not be an image of a sum of multiple terms of F. We will use this principle in our algorithm in order to distinguish between terms of F_{ij} that are images of single terms of F, and those that are images of a sum of terms of F.

2.3. Outline of the paper

The remainder contains a detailed description and analysis of our algorithm. Sections 3–5 give the details and proofs relating to the three randomizations described above: the degree-reducing primes p_i, the univariate substitution vectors $\mathbf{v}_{ij}$, and the diversification vectors $\mathbf{a}_k$. Section 6 then

gives a full and complete description of the algorithm; a reader uninterested in the probability analysis may safely skip to this part. Finally, the proofs of (probabilistic) correctness and running time for our algorithm are presented in Sections 7 and 8.

3. Truncating the degree of every variable of F

In order to interpolate a sparse univariate polynomial F given by an SLP, the usual method is to compute images of the form $F' = F \bmod (x^p - 1)$, where p, typically prime, is considerably smaller than D (see Arnold et al., 2013; Garg and Schost, 2009; Giesbrecht and Roche, 2011). This allows us to truncate potential intermediate expression swell over the execution of the straight-line program. We typically choose p such that, with high probability, the number of terms in collisions in each image F' is either zero or bounded by some fixed proportion ρ of T. From this requirement, it follows that most of the terms of F' are images of single terms of F. This is desirable because then most of the terms of F' contain "good" information about the sparse representation of F. We say two terms of F *collide* in the image F' if their respective images appearing in F are terms of the same degree. Given a means of sifting good information from "bad" information (collisions), we can rebuild the sparse representation of F from a sufficiently large set of images.

In the multivariate case, we will more generally consider truncated images

$$F' = F(\mathcal{Z}) \bmod (\mathcal{Z}^p - 1) \stackrel{\text{def}}{=} f(z_1, \ldots, z_n) \bmod (z_1^p - 1, \ldots, z_n^p - 1). \tag{3}$$

We let an *exponent collision* denote a set of two or more terms of F whose exponents agree under the mapping from F to F' given by (3). This occurs when there exist exponent vectors $\mathbf{e} \neq \mathbf{e}'$ such that

$$(\mathbf{e} - \mathbf{e}') \bmod p = \mathbf{0}. \tag{4}$$

Lemma 4. *Let $F \in \mathbb{F}_q[\mathcal{Z}]$ be an n-variate polynomial with $t \leq T$ terms and partial degrees $\deg_{z_j}(F) < D$, for $j \in [n]$. Let $\mu \in (0, 1)$, and let*

$$\lambda \geq \max\left(21, \tfrac{5}{3}(T-1)\ln D/\mu\right). \tag{5}$$

Choose a prime p uniformly at random from $(\lambda, 2\lambda]$. The probability that a fixed term of F is not *in an exponent collision in $F(\mathcal{Z}) \bmod (\mathcal{Z}^p - 1)$ is at least $1 - \mu$.*

The lemma is a generalization of Lemma 2.1 of Giesbrecht and Roche (2011) and the proof follows similarly.

Proof of Lemma 4. Without loss of generality, we consider the probability that the term $f^{(1)} = c_1 \mathcal{Z}^{\mathbf{e}_1}$ is not in any collision. Let $\mathcal{B}$ comprise the set of all "bad" primes $p \in [\lambda, 2\lambda]$ such that $f^{(1)}$ collides with another term of F in the image $f(\mathcal{Z}) \bmod (\mathcal{Z}^p - 1)$. If $f^{(1)}$ and $f^{(\ell)}$ collide modulo $\mathcal{Z}^p - 1$, then p divides $e_{1j} - e_{\ell j}$ for every $j \in [n]$. Furthermore, as $f^{(1)}$ and $f^{(\ell)}$ are distinct terms, there must be some variable z_{j_i} with a different exponent, i.e., $e_{1j} - e_{\ell j_i} \neq 0$. It follows that, for each $p \in \mathcal{B}$, p divides the nonzero integer

$$\prod_{i=2}^{t} (e_{1j} - e_{ij_i}). \tag{6}$$

Thus

$$\lambda^{|\mathcal{B}|} \leq \prod_{p \in \mathcal{B}} p \leq \prod_{i=2}^{t} |e_{1j} - e_{ij_i}| \leq D^{(T-1)}, \tag{7}$$

which gives us $|\mathcal{B}| \leq (T-1)\ln D / \ln \lambda$. By Corollary 3 of Rosser and Schoenfeld (1962), the total number of primes in the range $(\lambda, 2\lambda]$ is at least $3\lambda/(5\ln\lambda)$ for $\lambda \geq 21$. As

$$|\mathcal{B}| \leq \tfrac{(T-1)\ln D}{\ln \lambda} \leq \mu \tfrac{3\lambda}{5 \ln \lambda},$$

this completes the proof. □

In the subsequent sections, we will show how we can compute a term of $F(\mathcal{Z})$ modulo $\langle z_1^p - 1, \ldots, z_n^p - 1\rangle$ with probability at least 9/10. Therefore our algorithm will select $m = \lceil 2 \log D \rceil$ primes p_i with corresponding images F_i, so that any fixed term f of F that is found in at least half of the images F_i can be recovered in its entirety.

4. Substitution vectors and substitution collisions

In this section we will construct a set of images F_{ij} which will allow us to reconstruct *some* terms of F_i. One means of interpolating F_i is via *Kronecker substitution*, whereby we use a univariate interpolation algorithm to obtain an image

$$F_i' = F_i(x, x^D, \ldots, x^{D^{n-1}}) \in \mathbb{F}_q[x].$$

An advantage of this map is that it is collision-free, meaning that we can obtain every term of F_i from its image in F_i'. A term of F_i with exponent $\mathbf{e}$ will result in a term with exponent $e = \sum_{j=1}^n e_j D^{j-1}$ in F_i', and hence $\mathbf{e}$ is given by the base-D expansion of e.

But the Kronecker map causes considerable degree swell, as $\deg(F_i')$ can be on the order of $\deg(F_i)^n$, exponentially larger than our target degree. We instead will construct n univariate images of F_i,

$$F_{ij} = F_i(x^{\mathbf{v}_{ij}}) \bmod (x^{p_i} - 1) \in \mathbb{F}_q[x]/\langle x^{p_i} - 1\rangle, \tag{8}$$

where the $\mathbf{v}_{ij}$ are randomly chosen vectors from $\mathbb{F}_q^n$, for each $(i, j) \in [m, n]$.

We say that two terms $f = c\mathcal{Z}^{\mathbf{e}}$ and $f' = c'\mathcal{Z}^{\mathbf{e}'}$ of F_i are in a *substitution collision* if $(\mathbf{e} - \mathbf{e}') \bmod p_i \neq \mathbf{0}$, but $(\mathbf{e} - \mathbf{e}') \cdot \mathbf{v}_{ij} \bmod p_i = \mathbf{0}$. In which case both terms have an image of degree $\mathbf{e} \cdot \mathbf{v}_{ij} \bmod p$ in F_{ij}. If f does not collide with any other terms of F in the images F_{ij}, for any $j \in [n]$, then we can construct $\mathbf{e}$ as the solution to the linear system

$$\begin{bmatrix} \mathbf{v}_{i1} \\ \vdots \\ \mathbf{v}_{in} \end{bmatrix} \mathbf{e} = \mathbf{d} \bmod p_i, \tag{9}$$

provided $V_i = \left[v_{ijk}\right]_{j,k\in[n]}$ is invertible.

Lemma 5. *Let $p \geq 23$ be prime and consider a pair of exponents $\mathbf{e} \neq \mathbf{e}' \in \mathbb{Z}_p^n$. Let $\mathbf{v}_1, \ldots \mathbf{v}_n \neq \mathbf{0}$ be chosen at random from $\mathbb{Z}_p^n$. Then, with probability exceeding $1 - \frac{n}{p}$, the inequality $\mathbf{e} \cdot \mathbf{v}_i \neq \mathbf{e}' \cdot \mathbf{v}_i \bmod p$ holds for each $i \in [n]$.*

Proof. As $\mathbf{e} \neq \mathbf{e}'$, there exists some $j \in [n]$ for which $e_j \neq e_j'$. Without loss of generality assume $e_n \neq e_n'$. Consider a single substitution vector $\mathbf{v}_i \in \mathbb{F}_p^n$. Note, for any choice of $v_{i1}, \ldots, v_{i,n-1}$, solving for v_{in} in

$$(\mathbf{e} - \mathbf{e}') \cdot \mathbf{v}_i = 0 \bmod p$$

gives

$$v_{in} = \frac{\sum_{j=1}^{n-1} (e_j - e_j') v_{ij}}{e_n' - e_n} \bmod p.$$

That is, v_{in} is uniquely determined given any choices for the other elements in $\mathbf{v}_i$. Thus precisely a proportion $1/p$ of choices of substitution vectors $\mathbf{v} \in \mathbb{Z}_p^n$ will cause a substitution collision between

the terms with exponents $\mathbf{e}$ and $\mathbf{e}'$. By the union bound, the probability that any one of a random choice of n vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ results in a substitution collision between $\mathbf{e}$ and $\mathbf{e}'$ is at most n/p. □

We also require that the matrix V is invertible. In a practical setting, if V is not invertible, we might reasonably just choose n^2 new random entries for V and try again. But for the purpose of obtaining fast deterministic running time in a Monte Carlo setting, if V is singular, our algorithm will merely ignore the images F_{ij}. By Dickson (1901, Part II, Chapter 1, Theorem 99), we have that the probability that a matrix chosen at random from $\mathbb{Z}_p^{n\times n}$ is invertible is $\prod_{i=1}^{n}(1-1/p^i)$. For $p \geq 23$,

$$\prod_{i=1}^{n}(1-1/p^i) \geq \prod_{i\geq 1}(1-1/23^i) \approx 0.95463 \geq 19/20.$$

By the union bound we get the following lemma.

Lemma 6. *Consider a pair of exponents $\mathbf{e} \neq \mathbf{e}' \in \mathbb{Z}_p^n$. Let $\mathbf{v}_1, \ldots \mathbf{v}_n \in \mathbb{Z}_p^n$ be chosen uniformly at random, where $p \geq 23$. Let V be the matrix whose rows are given by $\mathbf{v}_1, \ldots, \mathbf{v}_n$, chosen at random from $\mathbb{Z}_p^n$. Then with probability at least $1 - n/p - 1/20$, V is invertible and $\mathbf{e}\cdot\mathbf{v}_j \neq \mathbf{e}'\cdot\mathbf{v}_j \bmod p$ for every $j \in [n]$.*

We can use Lemmata 4 and 6 to bound the probability of a term of F being involved in either an exponent collision or a substitution collision by $1-\mu/2$ and $\frac{19}{20}-\mu/2$, respectively, with an appropriate choice of λ. By the union bound this gives the following Corollary.

Corollary 7. *Let p_i be chosen at random from $(\lambda, 2\lambda]$, where*

$$\lambda = \max\left(21, \tfrac{5}{6}(T-1)\ln D/\mu, 2n/\mu\right). \tag{10}$$

Let $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{Z}_{p_i}^n$, be chosen uniformly at random, and let V be the matrix whose rows are given by the $\mathbf{v}_j$. Fix a term f of F. Then, with probability exceeding $\frac{19}{20}-\mu$, V is invertible and f does not collide with any other term of F in the images F_{ij}.

In other words, with probability greater than $19/20-\mu$ we can obtain c and $\mathbf{e} \bmod p_i$ for a term $f = c\mathcal{Z}^{\mathbf{e}}$ of F, from n images F_{ij}. This is provided that we can identify the terms from the images F_{ij} that correspond to f. We accomplish this task in the next section, by showing how to group terms that are images of the same term, or sum of terms, of F.

5. Diversification

We need a means of collecting terms amongst the images $F_{ij} = F(x^{\mathbf{v}_{ij}}) \bmod (x^{p_i}-1)$ which are images of the same term, or sum of terms, of F. Consider a pair of images

$$F^{(1)} = F(x^{\mathbf{v}'_1}) \bmod (x^{p'_1}-1), \qquad F^{(2)} = F(x^{\mathbf{v}'_2}) \bmod (x^{p'_2}-1),$$

where p'_k is prime and $\mathbf{v}'_k \in \mathbb{Z}_{p'_k}$ for $k = 1, 2$. Suppose $F^{(k)}$ has a term $f_k = bx^{d_k}$, for $k = 1, 2$. As these terms share the same coefficient b, it is possible that they are images of the same term of F. However, they could also be images of two different terms of F that happen to have the same coefficient. Moreover, it is possible that one or both are images of a *sum* of terms from F. In particular, f_k is image of the sum of all terms $c\mathcal{Z}^{\mathbf{e}}$ of F such that $\mathbf{e}\cdot\mathbf{v}'_k \bmod p'_k = d_k$, for $k = 1, 2$. Let h_1, h_2 be the respective sums of such terms, i.e.,

$$h_k = \sum_{\substack{\ell\in[t] \\ \mathbf{e}_\ell\cdot\mathbf{v}'_k \bmod p'_k = d_k}} c_\ell \mathcal{Z}^{\mathbf{e}_\ell}, \quad k = 1, 2,$$

and let $h = h_1 - h_2$. The coefficient of the x^{d_k} term of $F^{(k)}$ is $h_k(1, 1, \ldots, 1)$, $k = 1, 2$. These terms share the same coefficient if and only if $h(1, 1, \ldots, 1) = 0$. If h is not the zero polynomial but $h(1, \ldots, 1) = 0$,

then we might erroneously believe that f_1 and f_2 are images of the same term or sum of terms of F. We call such an unordered pair of terms $\{f_1, f_2\}$ a *deceptive pair*.

Now consider $\mathbf{a} \in \left(\mathbb{F}_{q^u}^*\right)^n$ chosen uniformly at random, where $u \geq 1$, and the images

$$\widetilde{F}^{(1)} = F(\mathbf{a}x^{\mathbf{v}'_1}) \bmod (x^{p'_1} - 1), \qquad \widetilde{F}^{(2)} = F(\mathbf{a}x^{\mathbf{v}'_2}) \bmod (x^{p'_2} - 1).$$

Now the degree-d_k term of $\widetilde{F}^{(k)}$ is $h_k(\mathbf{a})$. If we observe $h(\mathbf{a}) \neq 0$, then we can conclude that f_1 and f_2 were not images of the same sum of terms of F. In this instance we say $\mathbf{a}$ *reveals* the deceptive pair $\{f_1, f_2\}$. We will choose nonzero entries for $\mathbf{a} \neq \mathbf{0}$ from a field extension $\mathbb{F}_{q^u}$, where $u = \lceil \log_q(2nD + 1) \rceil$. As the total degree of h is less than nD, then by the Schwartz–Zippel Lemma (Schwartz, 1980), the probability that $h(\mathbf{a}) = 0$ is at most nD/q^u. By our choice of u the probability that $h(\mathbf{a}) = 0$ is at most $\frac{1}{2}$.

If we choose s random vectors $\mathbf{a}_1, \ldots, \mathbf{a}_s \in (\mathbb{F}_{q^u}^*)^n$ independently and uniformly at random, then the probability that none of the $\mathbf{a}_i$ reveal a given deceptive pair is less than 2^{-s}. Choosing

$$s = \lceil \log \tfrac{1}{\mu} + 2 \log \ell + 2 \log n + 2 \log T \rceil,$$

gives a probability bound of

$$\left(\tfrac{1}{2}\right)^{\lceil \log \frac{1}{\mu} + 2 \log m + 2 \log n + 2 \log T \rceil} = \tfrac{1}{\mu}(m^2 n^2 T^2)^{-1}.$$

The images F_{ij}, $(i, j) \in [m, n]$, have collectively at most mnT terms, thus fewer than $m^2 n^2 T^2$ deceptive pairs may occur. By the union bound we get the following lemma, which shows that the vectors $\mathbf{a}_k, k \in [s]$ reveal *all* possible deceptive pairs with high probability.

Lemma 8. *Let* $\mu \in (0, 1)$, $u = \lceil \log_q(2nD + 1) \rceil$, *and* $s = \lceil \log \frac{1}{\mu} + 2 \log m + 2 \log n + 2 \log T \rceil$. *Choose* $\mathbf{a}_1, \ldots, \mathbf{a}_s$ *independently and uniformly at random from* $\left(\mathbb{F}_{q^u}^*\right)^n$. *Then* $\mathbf{a}_1, \ldots, \mathbf{a}_m$ *reveal every deceptive pair amongst all pairs of terms from images* F_{ij}, $(i, j) \in [m, n]$, *with probability greater than* $1 - \mu$.

With this lemma we now have the tools required in order to reconstruct the terms of F.

6. Description of the algorithm

Our algorithm is given by Procedure SparseInterpolate on page 17. It is divided into two parts: the precomputation phase which defines various parameters and chooses all necessary random values, and the actual algorithm which performs the evaluations of the images to eventually reconstruct F. We note that the precomputation phase does *not* dominate the cost of the algorithm, and hence could be considered simply "computation" with no asymptotic effects.

After the precomputation, we use the straight-line program to construct images

$$F_{ij} = F(x^{\mathbf{v}_j}) \bmod (x^{p_i} - 1), \quad \text{and}$$
$$F_{ijk} = F(\mathbf{a}_k x^{\mathbf{v}_j}) \bmod (x^{p_i} - 1), \quad (i, j, k) \in [m, n, s],$$

for every i such that V_i is invertible. If V_i is not invertible, it will be impossible to uniquely recover terms of F_i, so we continue (line 14). Our analysis will show that the diversification vectors $\mathbf{a}_k, k \in [s]$ are sufficient to reveal all "deceptive pairs" of colliding terms in these images, in the language of the previous sections.

We use these images to construct congruences of the form $(\mathbf{e} \bmod p_i)$, $\mathbf{e} \in \mathbb{Z}_{p_i}^n$ (lines 12–25). Each congruence $(\mathbf{e} \bmod p_i)$ is constructed as a solution to a linear system $V_i \mathbf{e} = \mathbf{d} \bmod p_i$. These congruences are each uniquely associated with a vector of coefficients $\mathbf{b} = (b_0, \ldots, b_s) \in \mathbb{F}_{q^u}^{s+1}$, where $b_0 \in \mathbb{F}_q$ is the coefficient in the base field that appeared in F_{ij}. Then, for any $\mathbf{b}$ that has a sufficiently large set of congruences, we can recover the actual exponent vector $\mathbf{e} \in \mathbb{Z}^n$ by way of Chinese Remaindering and add $b_0 \mathcal{Z}^{\mathbf{e}}$ to F^*, a sparse representation for F (lines 26–30).

Procedure SparseInterpolate($\mathcal{S}_F, D, T$).

Input: $\mathcal{S}_F$, an SLP computing $F = \sum_{\ell=1}^{t} c_\ell \mathcal{Z}^{\mathbf{e}_\ell} \in \mathbb{F}_q[z_1, \ldots, z_n]$; $D > \max_{j\in[n]} \deg_{z_j} F$; $T \geq t$
Output: F^*, a sparse representation of F, with probability $\geq 3/4$

1 **precomputation**
2 $\mathcal{D} \leftarrow$ `create_dictionary()` ;
3 $m \leftarrow \max(6, 2\lceil \log D \rceil, \lceil (25/8)\ln(4T) \rceil)$;
4 $\lambda \leftarrow \max(21, \frac{95}{3}(T-1)\ln D, 80n, \frac{10}{3} m \ln m)$;
5 $s \leftarrow \lceil \log 40 + 2\log m + 2\log n + 2\log T \rceil$;
6 $u \leftarrow \lceil \log_q(2nD+1) \rceil$;
7 Choose the following independently and uniformly at random:
8 • p_i, a prime in $(\lambda, 2\lambda]$ for $i \in [m]$;
9 • $\mathbf{v}_{ij} \in \mathbb{Z}_{p_i}^n$, for $i, j \in [m, n]$;
10 • $\mathbf{a}_k \in \left(\mathbb{F}_{q^u}^*\right)^n$, for $k \in [s]$;
11 **begin**
12 **for** $i \in [m]$ **do**
13 $V_i \leftarrow [v_{ijk}]_{j,k=1}^n \in \mathbb{F}_q^{n\times n}$;
14 **if** V_i *is invertible* **then** Compute V_i^{-1} **else continue**;
15 $\mathcal{L} \leftarrow$ `create_dictionary()`;
16 **for** $j \in [n]$ **do**
17 $F_{ij} \leftarrow F(x^{\mathbf{e}\cdot\mathbf{v}_{ij}}) \bmod (x^{p_i} - 1)$;
18 **for** $k \in [s]$ **do** $F_{ijk} \leftarrow F(\mathbf{a}_k x^{\mathbf{e}\cdot\mathbf{v}_{ij}})$;
19 **foreach** *nonzero term* $b_0 x^d$ *of* F_{ij} **do**
20 **for** $k \in [s]$ **do** $b_k \leftarrow$ coefficient of the degree-d term of F_{ijk} ;
21 $\mathcal{L}$.`append_to`$(\mathbf{b}, d)$;
22 **for** $(\mathbf{b}, \mathbf{d}) \in \mathcal{L}$.`get_items()` **do**
23 **if** $|\mathbf{d}| \neq n$ **then continue**;
24 $\mathbf{e} \leftarrow V_i^{-1}\mathbf{d} \bmod p_i$;
25 $\mathcal{D}$.`append_to`$(\mathbf{b}, (\mathbf{e} \bmod p_i))$;
26 $F^* \leftarrow 0 \in \mathbb{F}_q[\mathcal{Z}]$;
27 **for** $(\mathbf{b}, \mathbf{c}) \in \mathcal{D}$.`get_items()` **do**
28 **if** $|\mathbf{c}| < m/2$ **then continue**;
29 $\mathbf{e} \leftarrow$ solution to set of congruences $\mathbf{c}$;
30 $F^* \leftarrow F^* + b_0 \mathcal{Z}^{\mathbf{e}}$;
31 **return** F^*;

In the i-th iteration of the for loop starting on line 12, we build the set of tuples $\mathcal{T}_i$ comprised of all $(\mathbf{b}, \mathbf{d})$, $\mathbf{b} = (b_0, \ldots, b_s) \in \mathbb{F}_{q^u}^{s+1}$, such that F_{ij} has a term $b_0 x^{d_j}$ and F_{ijk} has a term $b_k x^{d_j}$ for all $(j, k) \in [n, s]$. For each such tuple, we construct a congruence $(\mathbf{e} \bmod \mathbf{p}_i)$, where $\mathbf{e} \in \mathbb{Z}_{p_i}^n$ is the solution to the linear system

$$V_i \mathbf{e} = \mathbf{d} \bmod p_i.$$

We build this set of tuples using a dictionary $\mathcal{L}$, whose keys are $\mathbf{b} \in \mathbb{F}_{q^u}^{s+1}$ and whose values are degree vectors $\mathbf{d} \in \mathbb{Z}_{p_i}^n$. We build the degree vectors $\mathbf{d}$ iteratively. During j-th iteration of the for loop on line 16, we construct all the tuples $\mathbf{b}$ such that F_{ij} contains a nonzero term $b_0 x^d$ and F_{ijk} has a term $b_k x^d$ for $k \in [s]$.

Each iteration adds one more entry to the degree vector $\mathbf{d}$, so that for any term which appeared uncollided in every image F_{ij}, $j \in [n]$, the corresponding degree vector $\mathbf{d}$ will have length exactly n by the time the loop is finished after line 21. Denote by $\mathcal{T}_i$ the set of all such $(\mathbf{b}, \mathbf{d})$ tuples for which $|\mathbf{d}| = n$.

For each $(\mathbf{b}, \mathbf{d}) \in \mathcal{T}_i$, we recover $\mathbf{e} = V_i^{-1}\mathbf{d} \bmod p_i, \mathbf{e} \in \mathbb{Z}_{p_i}^n$ (line 24). For any key $\mathbf{b}$ we may recover up to m such congruences, one for each $i \in [m]$. For each $\mathbf{b}$ we store the set $\mathbf{c}$ of such congruences in a second dictionary $\mathcal{D}$ (line 25).

The two dictionaries $\mathcal{L}$ and $\mathcal{D}$ are maps from tuples $\mathbf{b} \in \mathbb{F}_{q^u}^{s+1}$ to lists of values. Under the reasonable assumption that there is a consistent and computable ordering on the base field $\mathbb{F}_q$, these

dictionaries can be implemented as any balanced search tree, such as an AVL tree or red-black tree.

In particular, the dictionary data structures should support the following operations. The running times are expressed in bit cost, which is affected by the fact that keys in $\mathbb{F}_{q^u}^{s+1}$ have $O(su \log q)$ bits each.

- `create_dictionary()` constructs a new, empty dictionary. The running time is $O(1)$.
- $\mathcal{D}$.`append_to(key, value)` first searches to see if `key` is already in the tree. If it is, `value` is appended to the end of the list associated with `key`. Otherwise, `key` is added to the tree and associated with a new list containing `value`. The bit cost of this operation, for keys in $\mathbb{F}_{q^u}^{s+1}$, is $O(\log|\mathcal{D}| \cdot su \log q)$.
- $\mathcal{D}$.`get_items()` iterates over all (`key, lst`) pairs of keys and lists of values that are stored in the tree. The bit cost of this operation is linear in the total size of the dictionary and its keys, $O(|\mathcal{D}| \cdot su \log q)$.

Provided every deceptive pair is revealed, every key $\mathbf{b}$ of $\mathcal{D}$ uniquely corresponds to a fixed, nonempty sum of terms of F. We are interested in those corresponding to exactly one term of F. With high probability, for any term f of F, we can construct the image of f in F_i for at least half of the $i \in [m]$. In other words, with high probability, any $\mathbf{b}$ corresponding to a term f of F should be associated in $\mathcal{D}$ with a set of at least $m/2$ congruences. By setting $m \geq 2\lceil \log D \rceil$, any $\mathbf{b}$ corresponding to a collision of terms of F will produce a set of less than $m/2$ congruences. Otherwise, this collision would contain terms $c\mathcal{Z}^{\mathbf{e}} \neq c'\mathcal{Z}^{\mathbf{e}'}$ of F such that $(\mathbf{e} - \mathbf{e}') \bmod p_i = 0$, for over $m/2$ primes p_i. This gives a contradiction as the product of such primes is at least D but the partial degrees of F are less than D.

For any key $\mathbf{b}$ associated with a set of at least $m/2$ congruences ($\mathbf{e} \bmod p_i$), we reconstruct $\mathbf{e} \in \mathbb{Z}_D^n$ by way of Chinese Remaindering (line 29). We then add a term $b_0\mathcal{Z}^{\mathbf{e}}$ to a sparse polynomial F^* (line 30). Provided each probabilistic step of the algorithm succeeded, this sum of such terms then comprises the sparse representation of F.

7. Probability analysis

The Procedure SparseInterpolate sets the following four parameters in order to guarantee Monte Carlo-type correctness:

$$m = \max(6, 2\lceil \log D \rceil, \lceil \tfrac{25}{8} \ln(4T) \rceil), \tag{11}$$

$$\lambda = \max(21, \tfrac{100}{3}(T-1)\ln D, 80n, \tfrac{10}{3} m \ln m), \tag{12}$$

$$s = \lceil \log 40 + 2\log m + 2\log n + 2\log T \rceil, \tag{13}$$

$$u = \lceil \log_q(2nD+1) \rceil. \tag{14}$$

The setting of these parameters is explained by the following lemma.

Lemma 9. *Procedure SparseInterpolate correctly outputs a sparse representation of F with probability at least $\frac{3}{4}$.*

Proof. We first require that the interval $(\lambda, 2\lambda]$ contains at least m primes. By Rosser and Schoenfeld (1962), the total number of primes in $(\lambda, 2\lambda]$ is at least $3\lambda/(5\ln\lambda)$. Because $\lambda > \frac{10}{3} m \ln m$,

$$3\lambda/(5\ln\lambda) > m\ln(m^2)/\ln((10/3)m\ln m) \geq m.$$

The last inequality above holds whenever $(10/3)m\ln m \leq m^2$, which is true for all $m \geq 6$.

Next, recall that in our notation, the polynomial we wish to interpolate is written term-wise as $F = \sum_{\ell \in [t]} c_\ell \mathcal{Z}^{\mathbf{e}_\ell}$. The algorithm works by computing m images $F_i = F \bmod (\mathcal{Z}^{p_i} - 1)$.

Fix a single term $c_\ell \mathcal{Z}^{\mathbf{e}_\ell}$, for an arbitrary $\ell \in [t]$. If (1) the random coefficient vectors $\mathbf{a}_k$ reveal all deceptive terms as in Lemma 8; (2) the matrix of exponent substitutions $V_i = (v_{ijk})_{j,k\in[n,n]}$ is invertible; and (3) the term $c_\ell \mathcal{Z}^{\mathbf{e}_\ell}$ does not collide with any other terms modulo p_i, then the exponent vector $\mathbf{e}_\ell$ will be recovered modulo p_i at that step.

Lemma 8 guarantees condition (1) with probability at least $1 - \mu$, and Corollary 7 guarantees (2) and (3), for a fixed term $c_\ell \mathcal{Z}^{\mathbf{e}_\ell}$ and prime p_i, with probability at least $\frac{19}{20} - \mu$. By the union bound, the probability that all three conditions hold, and therefore that we recover the exponents of this fixed term modulo any given prime p_i, is at least $\frac{19}{20} - 2\mu$. By setting $\mu = \frac{1}{40}$, the probability of failure for any fixed term index ℓ and prime index i is at most $\frac{1}{10}$.

Observe that the number m of primes p_i is at least $2\log_2 D$, so that if the exponent vector $\mathbf{e}_\ell$ is recovered modulo any fraction $m/2$ of the primes, then there is sufficient information to recover the actual exponents $\mathbf{e}_\ell$ over $\mathbb{Z}$ at the end of the algorithm. The preceding paragraph shows that the *expected* number of primes p_i for which we can recover the exponent vector modulo p_i is at least $9m/10$. Hoeffding's inequality provides a way to bound the probability that the actual number of primes that allow us to recover $\mathbf{e}_\ell$ is less than $m/2$, much smaller than the expected value of $9m/10$.

Define the random variable X_i to be 1 if the exponent vector $\mathbf{e}_\ell$ is recovered modulo p_i, and 0 otherwise. Hence $\mathbb{E}[X_i] = \frac{9}{10}$. We want to know the probability that $\sum_{i\in m} X_i \geq \frac{m}{2}$. Since the primes p_i are chosen uniformly at random, without replacement, Theorems 1 and 4 from Hoeffding (1963) tell us that

$$\Pr\left[\sum_{i\in m} X_i < \frac{m}{2}\right] < \exp\left(\frac{-8m}{25}\right).$$

As there are T total terms, the union bound tells us that the probability of recovering *every* term in at least $m/2$ of the images F_i is at least $1 - T\exp(-8m/25)$, which is at least $\frac{3}{4}$ since we choose $m \geq (25/8)\ln(4T)$. □

Note that the dependence $m \geq 2\lceil \log D \rceil$ could be reduced by roughly $\log\lambda$, but this would complicate the algorithm and analysis, without considerable benefit, so we do not consider it.

We employ a meta-algorithm in order to interpolate F with an arbitrarily small probability of failure $\epsilon < 1$. One iteration of the algorithm succeeds with probability at least 3/4. Thus, if we run the algorithm r times producing outputs $F_1^*, \ldots, F_r^*$, then by Hoeffding's inequality (Hoeffding, 1963, Theorem 1), the probability that $F_i \neq F$ for at least half of the F_i^*, $i \in [r]$, is less than $e^{-r/8}$. Setting $r = 8\ln(1/\epsilon)$ makes this probability less than ϵ. We thus merely run the algorithm $\lceil 8\ln\frac{1}{\epsilon}\rceil$ times and return the polynomial F that appears most frequently. If no such F appears more than half the time, the meta-algorithm fails.

8. Cost analysis

We give a "soft-oh" cost analysis of the algorithm, whereby we ignore possible additional logarithmic factors in the cost. As the final cost is polynomial in $\log D, T, n$, and $\log q$, we ignore poly-logarithmic factors of these values. We let κ denote any term that is poly-logarithmic in $nT\log D\log q$, i.e., any term that is bounded by $(\log(nT\log D\log q))^{\mathcal{O}(1)}$, to simplify the cost analysis of intermediate steps.

Equations (11)–(14) give us

$$\begin{aligned}
m &\in \mathcal{O}(\log D + \log T) \subseteq \widetilde{\mathcal{O}}(\kappa \log D),\\
\lambda &\in \mathcal{O}\left((T + \log\log D)\log D + n\right) \subseteq \widetilde{\mathcal{O}}(T\log D + n),\\
s &\in \mathcal{O}(\log\log D + \log n + \log T) \subseteq \widetilde{\mathcal{O}}(\kappa),\\
u &\in \mathcal{O}(1 + (\log D + \log n)/\log q) \subseteq \widetilde{\mathcal{O}}(1 + \kappa\log D/\log q).
\end{aligned}$$

Observe also that each of $\log m, \log\lambda, \log s, \log u$ is $\widetilde{\mathcal{O}}(\kappa)$, and therefore does not affect the overall soft-oh analysis.

8.1. Cost of precomputation

The precomputation steps involve setting up the fields and constants that the algorithm uses, as well as making all the necessary random choices. We will repeatedly need to choose a single item uniformly at random from a finite set S. Observe that such a choice can be made in $\mathcal{O}(\log|S|)$ time and using $\mathcal{O}(\log|S|)$ random bits by, for example, defining an enumeration of the set and choosing a random index between 0 and $|S|-1$.

8.1.1. Cost of generating primes

The algorithm requires selecting uniformly at random a list of m primes p_i in the range $(\lambda, 2\lambda]$. It is possible to generate all primes in $(\lambda, 2\lambda]$ with

$$\widetilde{\mathcal{O}}(\lambda) = \widetilde{\mathcal{O}}(T\log D + n) \tag{15}$$

bit operations using a sieve method, e.g., the wheel sieve (Pritchard, 1982). From the previous discussion, choosing m of these at random from the generated list would cost $\mathcal{O}(m\log\lambda)$, which is $\widetilde{\mathcal{O}}(\kappa\log D)$ and thus dominated by the cost of sieving.

A more practical approach might be to select each such prime probabilistically, with expected bit-cost poly-logarithmic in λ, by selecting integers p at random from $(\lambda, 2\lambda]$, p perhaps not a multiple of a small prime, and then running probabilistic primality test on it, e.g. Miller–Rabin. But as that would complicate our probabilistic analysis (namely by increasing the probability of failure), we will assume for the purposes of analysis that the sieving method is used to choose primes.

8.1.2. Cost of constructing a field extension $\mathbb{F}_{q^u}$

In order to select vectors $\mathbf{a} \in \mathbb{F}_{q^u}$ at random, we first need to construct a representation for $\mathbb{F}_{q^u}$. In particular, we need to construct an irreducible polynomial of degree u over $\mathbb{F}_q$. Per Shoup (1994), this may be done in $\widetilde{\mathcal{O}}(u^2 + u\log q)$ operations in $\mathbb{F}_q$, for a total bit complexity of $\widetilde{\mathcal{O}}(u\log q(u+\log q))$.

Because $u \in \widetilde{\mathcal{O}}(1 + \kappa\log D/\log q)$, we can see that $u\log q \in \widetilde{\mathcal{O}}(\log q + \kappa\log D)$. Furthermore, we only work in an extension provided $q \le 2nD$, which means that in this case we always have $\log q \in \widetilde{\mathcal{O}}(\kappa\log D)$. Hence $u\log q$ and $(u+\log q)$ are both $\widetilde{\mathcal{O}}(\kappa\log D)$, and the entire cost of this step is less than $\widetilde{\mathcal{O}}(\kappa\log^2 D)$.

8.1.3. Cost of selecting $\mathbf{a}_k$ and $\mathbf{v}_{ij}$

As $\mathbb{F}_{q^u}$ is a finite set of size q^u, the cost of selecting a single element from that field is $\mathcal{O}(u\log q)$, which is $\widetilde{\mathcal{O}}(\log q + \kappa\log D)$. Our algorithm requires s length-n vectors $\mathbf{a}_k \in \mathbb{F}_{q^u}^n$, for a total cost of

$$\mathcal{O}(snu\log q) \subseteq \widetilde{\mathcal{O}}\left(\kappa n\left(\log D + \log q\right)\right) \tag{16}$$

bit operations.

The algorithm also requires choosing mn size-n vectors $\mathbf{v}_{ij} \in \mathbb{Z}_{p_i}^n$, where each $p_i \in \mathcal{O}(\lambda)$. The cost of selecting these vectors is

$$\mathcal{O}(mn^2\log\lambda) \subseteq \widetilde{\mathcal{O}}(\kappa n^2\log D). \tag{17}$$

Summing up all the precomputation costs above and removing extraneous factors of κ gives a total of

$$\widetilde{\mathcal{O}}\left(\left(T + n^2 + \log D\right)\log D + n\log q\right) \tag{18}$$

bit operations.

Furthermore, considering the costs (15), (16), and (17) of selecting randomly chosen primes and vectors, we can also bound the bits of randomness required by Procedure SparseInterpolate by

$$\mathcal{O}(m\log\lambda + snu\log q + mn^2\log\lambda) = \widetilde{\mathcal{O}}\left(n\log T\left(n\log D + n\log T + \log q\right)\right).$$

We give this as a lemma.

Lemma 10. *Procedure SparseInterpolate requires* $\widetilde{\mathcal{O}}\left(n \log T \left(n \log D + n \log T + \log q\right)\right)$ *bits of randomness.*

8.2. Cost of producing images

The algorithm produces $\mathcal{O}(mns)$ images in rings $\mathbb{F}_{q^u}[x]/(x^p - 1)$, $p \in \mathcal{O}(\lambda)$. Per Claim 3, this cost is $\widetilde{\mathcal{O}}(mns \cdot Lpu \log q)$ bit operations, or

$$\widetilde{\mathcal{O}}(Ln(T \log D + n)(\log D + \log q) \log D). \tag{19}$$

This dominates the cost of precomputation.

8.3. Cost of constructing and accessing dictionaries

Observe that as every image F_{ij} has at most T nonzero terms, we run the for loop on line 19 of Procedure SparseInterpolate at most T times consecutively. It follows that $|\mathcal{L}| \leq nT$ at any point of the algorithm. Counting loop iterations, this implies that Procedure SparseInterpolate runs $\mathcal{L}$.`append_to` at most mnT times, in addition to running $\mathcal{L}$.`create_dictionary` and $\mathcal{L}$.`get_items` m times.

It follows from the discussion in section 6 that the cost of these operations will total

$$\widetilde{\mathcal{O}}(mnT \log |\mathcal{L}| su \log q + m|\mathcal{L}| su \log q) = \widetilde{\mathcal{O}}(mnT su \log q).$$

As $|\mathcal{L}| \leq nT$, we run $\mathcal{D}$.`append_to` at most nT times for every iteration of the outer for loop beginning on line 19. It follows that $|\mathcal{D}| \leq mnT$ over the execution of the algorithm, and that $\mathcal{D}$.`append_to` is run at most mnT times. These operations yield a cost of $\widetilde{\mathcal{O}}(mnT \log |\mathcal{D}| su \log q) = \widetilde{\mathcal{O}}(mnT su \log q)$. The soft-oh cost due to running $\mathcal{D}$.`get_items` once is similar. Thus the cost for all dictionary operations becomes

$$\widetilde{\mathcal{O}}(mnT su \log q) = \widetilde{\mathcal{O}}(nT(\log D + \log q) \log D).$$

This cost is also absorbed by the cost (19) of constructing images.

8.4. Cost of solving linear systems

For each of the m primes p_i, we need to invert an $n \times n$ matrix V_i with entries in $\mathbb{Z}_{p_i}$. This requires $\mathcal{O}(n^\omega)$ operations in $\mathbb{Z}_{p_i}$, where $p_i \in \mathcal{O}(\lambda)$ (see, e.g., Prop. 16.6 in Bürgisser et al. (1997)). This bit-operation cost becomes

$$\widetilde{\mathcal{O}}(\log(\lambda) mn^\omega) = \widetilde{\mathcal{O}}(\kappa(\log D) n^\omega). \tag{20}$$

In addition, for every $i \in [n]$ we have to compute some number of vectors $\mathbf{e} \bmod p_i$ as products $V_i^{-1}\mathbf{d} \bmod p_i$ for various $\mathbf{d} \in \mathbb{Z}_{p_i}^n$ on line 24. As $\mathcal{L}$.`append_to` is run at most nT times in a single iteration of the outer for loop starting on line 12, then $\mathcal{L}$ can only have at most T entries with values $\mathbf{d}$ with length n.

These linear system solutions $V_i^{-1}\mathbf{d} \bmod p_i$ can be performed more efficiently via blocking, whereby we multiply V_i^{-1} by n vectors of $\mathbf{d}$ at a time using fast matrix multiplication. This entails $\mathcal{O}\left(n^\omega \left\lceil \frac{T}{n} \right\rceil\right)$ arithmetic operations in $\mathbb{Z}_{p_i}$, $p_i \in \mathcal{O}(\lambda)$. Thus, using this blocking strategy the total cost of the linear system solving over every iteration of the outer for-loop becomes

$$\begin{aligned} \mathcal{O}\left(n^\omega m \left\lceil \frac{T}{n} \right\rceil\right) &\subseteq \mathcal{O}(mn^\omega(T/n + 1)) = \mathcal{O}(n^{\omega-1}mT + n^\omega m) \\ &\subseteq \widetilde{\mathcal{O}}(n^{\omega-1}T \log D + n^\omega \log D), \end{aligned}$$

which dominates the cost (20) of computing the inverses.

8.5. Cost of constructing terms

To construct the terms of F, we have to construct T exponents $\mathbf{e} \in \mathbb{Z}_D^n$, from sets of at most m congruences. Constructing one entry $e_j \in [0, D)$ of one exponent $\mathbf{e}$ by the Chinese Remainder algorithm entails $\mathcal{O}(\log^2 D)$ bit operations (Gathen and Gerhard, 2003, Thm. 5.8). Doing this for at most Tn vector entries yields a total cost of $\mathcal{O}(nT \log^2 D)$. Again this cost is absorbed by the cost (19) of constructing images.

From the previous subsections, we see that the total cost of Procedure SparseInterpolate is dominated by the cost of constructing the images of F, and the cost of solving linear systems. We state the total bit-cost of the algorithm as a lemma.

Lemma 11. *Procedure SparseInterpolate entails a bit-operation cost of*

$$\widetilde{\mathcal{O}}\left(Ln(T \log D + n)(\log D + \log q) \log D + n^{\omega-1} T \log D + n^{\omega} \log D\right).$$

Combining Lemmata 9 and 11 gives Theorem 1.

Acknowledgements

We thank Mustafa Elsheikh for helpful discussion related to section 4, Lutz Kämmerer for pointing out an error in Lemma 4, and the helpful comments of the referees. The first author acknowledges the support of the National Sciences and Engineering Research Council of Canada (NSERC). The third author is supported by National Science Foundation (NSF) award no. 1319994, "AF: Small: RUI: Faster Arithmetic for Sparse Polynomials and Integers".

Appendix A. Notation

Mathematical objects	
$m, n, p, q, r, s, t, u \in \mathbb{Z}_{>0}$	
$\epsilon, \mu \in (0, 1)$	probabilities
$\lambda \in \mathbb{R}_{>0}$	constant
$\mathbb{Z}_p$	ring of integers modulo p
$\mathbb{F}_q$	finite field of size q
$[n]$	$\stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$
$[n_1, \ldots, n_s]$	$\stackrel{\text{def}}{=} \{(i_1, \ldots, i_s) \mid i_k \in [n_k], k \in [s]\}$
$(i, j, k, \ell) \in [m, n, s, t]$	indices
$x, z_j, j \in [n]$	indeterminates
$\mathbb{F}_q[\mathcal{Z}] \stackrel{\text{def}}{=} \mathbb{F}_q[z_1, \ldots, z_n]$	polynomial ring with n indeterminates
$F \in \mathbb{F}_q[\mathcal{Z}]$	polynomial
$\mathcal{S}_F$	straight-line program computing $F \in \mathbb{F}_q[\mathcal{Z}]$
L	length of $\mathcal{S}_F$
$c, c_\ell \in \mathbb{F}_q, \ell \in [t], b \in \mathbb{F}_{q^u}$	coefficients
$\mathbf{b} \in \mathbb{F}_{q^u}^{s+1}$	vector of coefficients from a field extension
$\mathbf{e}, \mathbf{e}_\ell \in \mathbb{Z}^n$	exponent vectors
$p_i, i \in [m]$	randomly selected primes
$\mathbf{v}, \mathbf{v}_{ij} \in \mathbb{Z}_{p_i}^n, (i, j) \in [m, n]$	randomly selected vectors
$V_i = [v_{ijk}] \in \mathbb{Z}_{p_i}^{n \times n}$	matrix whose rows are $\mathbf{v}_{ij}$, $j \in [n]$
$\mathbf{a}, \mathbf{a}_k \in \mathbb{F}_{q^u}, k \in [s]$	randomly selected vectors

Polynomial notation	
$\mathbf{e} \bmod p$	$\stackrel{\text{def}}{=} (e_1 \bmod p, \ldots, e_n \bmod p)$
$\mathbf{a}^{\mathbf{e}}$	$\stackrel{\text{def}}{=} a_1^{e_1} \cdots a_n^{e_n}$
$c\mathcal{Z}^e$	$\stackrel{\text{def}}{=} c z_1^{e_1} \cdots z_t^{e_t}$
$F(\mathcal{Z})$	$\stackrel{\text{def}}{=} F(z_1, \ldots, z_n)$
$F(\mathbf{a}\mathcal{Z})$	$\stackrel{\text{def}}{=} F(a_1 z_1, \ldots, a_n z_n)$
$F(x^{\mathbf{v}})$	$\stackrel{\text{def}}{=} F(x^{v_1}, \ldots, x^{v_n})$
$F(\mathbf{a}x^{\mathbf{v}})$	$\stackrel{\text{def}}{=} F(a_1 x^{v_1}, \ldots, a_n x^{v_n})$
$F \bmod (\mathcal{Z}^p - 1)$	$\stackrel{\text{def}}{=} F \bmod (z_1^p - 1, \ldots, z_n^p - 1)$.

Defined constants
$m = \max(6, 2\lceil \log D \rceil, \lceil \frac{25}{8} \ln(4T) \rceil)$,
$\lambda = \max(21, \frac{100}{3}(T-1)\ln D, 80n, \frac{10}{3} m \ln m)$,
$s = \lceil \log 40 + 2\log m + 2\log n + 2\log T \rceil$,
$u = \lceil \log_q (2nD+1) \rceil$.

References

Arnold, A., Giesbrecht, M., Roche, D.S., 2013. Faster sparse interpolation of straight-line programs. In: Proc. Computer Algebra in Scientific Computing. CASC 2013. In: Lecture Notes in Computer Science, vol. 8136, pp. 61–74.

Arnold, A., Giesbrecht, M., Roche, D.S., 2014. Sparse interpolation over finite fields via low-order roots of unity. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. ISSAC '14. ACM, New York, NY, USA, pp. 27–34.

Arnold, A., Roche, D.S., 2014. Multivariate sparse interpolation using randomized Kronecker substitutions. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. ISSAC '14. ACM, New York, NY, USA, pp. 35–42.

Ben-Or, M., Tiwari, P., 1988. A deterministic algorithm for sparse multivariate polynomial interpolation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. STOC '88. ACM, New York, NY, USA, pp. 301–309.

Bürgisser, P., Clausen, M., Shokrollahi, M.A., 1997. Algebraic Complexity Theory, vol. 315. Springer.

Candés, E.J., Romberg, J.K., Tao, T., 2006. Stable signal recovery from incomplete and inaccurate measurements. Commun. Pure Appl. Math. 59 (8), 1207–1223. http://dx.doi.org/10.1002/cpa.20124.

Cantor, D.G., Kaltofen, E., 1991. On fast multiplication of polynomials over arbitrary algebras. Acta Inform. 28 (7), 693–701. http://dx.doi.org/10.1007/BF01178683.

Dickson, L.E., 1901. Linear Groups with an Exposition of the Galois Field Theory. B.G. Teubner, Leipzig. http://www.biodiversitylibrary.org/bibliography/22174. http://www.biodiversitylibrary.org/item/62667.

Donoho, D., 2006. Compressed sensing. IEEE Trans. Inf. Theory 52 (4), 1289–1306. http://dx.doi.org/10.1109/TIT.2006.871582.

Fürer, M., 2009. Faster integer multiplication. SIAM J. Comput. 39 (3), 979–1005. http://dx.doi.org/10.1137/070711761.

Garg, S., Schost, E., 2009. Interpolation of polynomials given by straight-line programs. Theor. Comput. Sci. 410 (27–29), 2659–2662. http://dx.doi.org/10.1016/j.tcs.2009.03.030.

Gathen, J.V.Z., Gerhard, J., 2003. Modern Computer Algebra, 2nd edition. Cambridge University Press, New York, NY, USA.

Giesbrecht, M., Labahn, G., Lee, W., 2009. Symbolic–numeric sparse interpolation of multivariate polynomials. J. Symb. Comput. 44 (8), 943–959. http://dx.doi.org/10.1016/j.jsc.2008.11.003.

Giesbrecht, M., Roche, D.S., 2011. Diversification improves interpolation. In: ISSAC'11, pp. 123–130.

Hassanieh, H., Indyk, P., Katabi, D., Price, E., 2012. Nearly optimal sparse Fourier transform. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing. STOC '12. ACM, New York, NY, USA, pp. 563–578. http://doi.acm.org/10.1145/2213977.2214029.

Hoeffding, W., 1963. Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. 58 (301), 13–30. http://dx.doi.org/10.1080/01621459.1963.10500830.

Huang, M.-D.A., Rao, A.J., 1999. Interpolation of sparse multivariate polynomials over large finite fields with applications. J. Algorithms 33 (2), 204–228. http://dx.doi.org/10.1006/jagm.1999.1045.

Javadi, S.M.M., Monagan, M., 2010. Parallel sparse polynomial interpolation over finite fields. In: Proceedings of the 4th International Workshop on Parallel and Symbolic Computation. PASCO '10. ACM, New York, NY, USA, pp. 160–168.

Kabanets, V., Impagliazzo, R., 2004. Derandomizing polynomial identity tests means proving circuit lower bounds. Comput. Complex. 13 (1–2), 1–46. http://dx.doi.org/10.1007/s00037-004-0182-6.

Kaltofen, E.L., 1988. Unpublished article fragment. http://www.math.ncsu.edu/~kaltofen/bibliography/88/Ka88_ratint.pdf.

Kaltofen, E.L., 2010. Fifteen years after DSC and WLSS2 what parallel computations I do today. In: Proc. International Workshop on Parallel Symbolic Computation. PASCO 2010, pp. 10–17.

Kaltofen, E.L., Lee, W.-s., Yang, Z., 2011. Fast estimates of Hankel matrix condition numbers and numeric sparse interpolation. In: Proceedings of the 2011 International Workshop on Symbolic–Numeric Computation. SNC '11. ACM, New York, NY, USA, pp. 130–136.

Kushilevitz, E., Mansour, Y., 1993. Learning decision trees using the Fourier spectrum. SIAM J. Comput. 22 (6), 1331–1348. http://dx.doi.org/10.1137/0222080.

Le Gall, F., 2014. Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation. ISSAC '14. ACM, New York, NY, USA, pp. 296–303. http://doi.acm.org/10.1145/2608628.2608664.

Pritchard, P., 1982. Explaining the wheel sieve. Acta Inform. 17 (4), 477–485. http://dx.doi.org/10.1007/BF00264164.

Prony, Gaspard-Clair-François-Marie Riche, B.d., 1795. Essai expérimental et analytique sur les lois de la Dilatabilité des fluides élastique et sur celles de la Force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures. J. Éc. Polytech. 1, 24–76.

Rosser, J.B., Schoenfeld, L., 1962. Approximate formulas for some functions of prime numbers. Ill. J. Math. 6, 64–94. http://projecteuclid.org/euclid.ijm/1255631807.

Schwartz, J.T., 1980. Fast probabilistic algorithms for verification of polynomial identities. J. ACM 27 (4), 701–717. http://doi.acm.org/10.1145/322217.322225.

Shoup, V., 1994. Fast construction of irreducible polynomials over finite fields. J. Symb. Comput. 17 (5), 371–391. http://dx.doi.org/10.1006/jsco.1994.1025.

Shpilka, A., Yehudayoff, A., 2010. Arithmetic circuits: a survey of recent results and open questions. Foundations and Trends in Theoretical Computer Science 5 (3–4), 207–388.

Zippel, R., 1990. Interpolating polynomials from their values. J. Symb. Comput. 9 (3), 375–403. http://dx.doi.org/10.1016/S0747-7171(08)80018-1, computational algebraic complexity editorial.